

AD A213322

**ON ALGORITHMS FOR GENERATING COMPUTATIONALLY
SIMPLE PIECEWISE LINEAR CLASSIFIERS**

by

HANS CHRISTIAN PALM

NDRE/PUBL-89/1001

ISSN 0800-4412

Accession For	
NTIS GBA&I	☒
DTIC TAB	☒
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

FORSVARETS FORSKNING SINSTITUTT

NORWEGIAN DEFENCE RESEARCH ESTABLISHMENT

P O Box 25 - N-2007 Kjeller, Norway

May 1989

ON ALGORITHMS FOR GENERATING COMPUTATIONALLY SIMPLE PIECEWISE LINEAR CLASSIFIERS

by

HANS CHRISTIAN PALM

NDRE/PUBL-89/1001

ISSN 0800-4412

Accession For	
NTIS - GPO	☒
DTIC	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist and/or	
Dist	Special
A-1	

FORSVARETS FORSKNING SINSTITUTT

NORWEGIAN DEFENCE RESEARCH ESTABLISHMENT

P O Box 25 - N-2007 Kjeller, Norway

May 1989

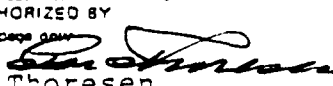
NORWEGIAN DEFENCE RESEARCH ESTABLISHMENT (NDRE)
FORSVARETS FORSKNING SINSTITUTT (FFI)

POST OFFICE BOX 25
N-2007 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER NDRE/PUBL-89/1001 1a) JOB REFERENCE 557-VM/130	2) SECURITY CLASSIFICATION Unclassified 2a) DECLASSIFICATION/DOWNGRADING SCHEDULE	3) NUMBER OF PAGES 120
4) TITLE ON ALGORITHMS FOR GENERATING COMPUTATIONALLY SIMPLE PIECEWISE LINEAR CLASSIFIERS		
5) NAMES OF AUTHOR(S) IN FULL (surname first) PALM Hans Christian		
6) DISTRIBUTION STATEMENT Approved for public release (offentlig tilgjengelig)		
7) INDEXING TERMS IN ENGLISH: a) <u>Pattern recognition</u> b) <u>Piecewise linear</u> c) <u>discriminant function</u> d) <u>Supervised learning</u> e) <u>Cluster analysis</u> f) <u>Monte Carlo simulation</u> IN NORWEGIAN: a) <u>Mønstergjenkjenning</u> b) <u>Stykkevis lineære</u> c) <u>diskriminantfunksjoner</u> d) <u>Leder læring</u> e) <u>Klyngeanalyse</u> f) <u>Monte Carlo simulering</u>		
THESAURUS REFERENCE:		
8) ABSTRACT (continue on reverse side if necessary) Piecewise linear classifiers constitute a group of classifiers often used in real time pattern recognition. In most cases they are reliable and sufficiently fast. In this work, new algorithms for generating piecewise linear classifiers are developed, which can easily handle multi class problems. Using only a small number of discriminant functions, they attempt to create a classifier with low error rate. In brief, the concept consists of first splitting the sample space of a given class into two subsample spaces. The samples belonging to a given class which are lying in the same (sub)sample space are said to belong to the same subclass. Next, the (sub)classes are separated using linear classifiers. Then, a new (sub)class is split and a classifier based on this splitting, is created.		
9) DATE 24 May 1989	AUTHORIZED BY This page only  P Thoresen	POSITION Chief Scientist

UNCLASSIFIED

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

ABSTRACT (continued)

This process continues until the overall performance is not improved by further splitting. Our classifiers have been compared with other relevant classifiers such as Bayes classifier (for known distributions), Bayes classifier with estimated densities, the nearest neighbour rule as well as previously developed piecewise linear classifiers. So far tests have shown that our algorithms are working very well. They produce fast and reliable classifiers which in some cases have been found superior to the classifiers used for the comparison.

Accession For	
NTIS GPO-I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A-1	

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

CONTENTS	Page
1 INTRODUCTION	5
2 INTRODUCTION TO PATTERN RECOGNITION	7
2.1 Basic concepts	7
2.2 Linear classifiers	9
2.3 Quadratic classifiers	10
2.4 Piecewise linear classifiers	12
3 THE NEW CONCEPT	22
3.1 Finding a suitable (sub)sample space for splitting	23
3.2 How to split a given subsample space	25
3.3 Generation of piecewise linear classifiers	27
3.4 Termination of the algorithm and generation of the final classifier	28
4 DEVELOPMENT OF THE ALGORITHMS	29
4.1 Determination of a suitable subsample space to split	29
4.1.1 The hillclimbing approach	31
4.1.2 A contribution based splitting	32
4.1.3 Comparison of the two strategies	33
4.2 How to split a given subsamples space	44
4.2.1 Initial splitting procedures	45
4.2.1.1 Splitting based on the scatter matrix	45
4.2.1.2 Splitting based on starting samples	46
4.2.1.2.1 Determination of the starting samples	47
4.2.1.2.2 Splitting of a subclass when the staring samples are given	48
4.2.1.3 Outlier handling	50
4.2.2 Optimization of the initial splitting	50
4.2.3 Evaluation of the different algorithms	53
4.2.3.1 Results from the initial splitting	57
4.2.3.2 Results from the optimization	61

4.3	Generating piecewise linear classifiers	66
4.3.1	The nearest submean classifier	66
4.3.2	Piecewise linear classifier based on the mean squared error approach	67
4.3.3	Determination of a piecewise linear classifier using a weighed MSE-approach	69
4.3.4	Generating a piecewise linear classifier from a set of hyperplanes	72
4.3.4.1	Determining the pairs of subclasses to be separated	73
4.3.4.2	Generating a suitable linear classifier	79
4.3.4.3	Computing the piecewise linear discriminant function	84
4.3.5	Comparison of the performance of the different classifiers	85
4.4	Determining the final classifier	89
4.4.1	Termination of the splitting	89
4.4.2	Weighing of $\hat{P}_e$ and the number of subclasses	89
4.4.3	Determining a sufficiently good classifier	91
5	EVALUATION OF THE CLASSIFIERS	91
6	SUMMARY AND CONCLUSION	107
References		110
A	A ROBUST OUTLIER DETECTOR	113
B	THE CONNECTION BETWEEN $(\mu_1 - \mu)^t(\mu_2 - \mu)$ AND $\text{tr}(W)$	116
C	THE SINGULAR VALUE DECOMPOSITION METHOD USED FOR DETERMINING THE PIECEWISE LINEAR CLASSIFIER FROM A SET OF HYPERPLANES	118

ON ALGORITHMS FOR GENERATING COMPUTATIONALLY SIMPLE PIECEWISE LINEAR CLASSIFIERS

1 INTRODUCTION

In many applications one has to assign a given object to one out of several possible categories (classes). Theories for making such decisions, also called classification- or pattern recognition-theory, can be dated back to Fisher's classical work in 1936 [12]. However, it was not until the development of the electronic computer some 35 years ago that considerable advances were made. The range of pattern recognition applications is now very broad. A few examples are:

- Recognition of moving objects (vehicles, human beings etc) in TV-images, for automatic surveillance.
- Classification of remotely sensed data, i.e. assigning each picture element of a satellite image to one out of a given set of classes (acres, forests, roads, etc).
- Medical diagnostics, such as analysis of X-ray images (e.g. detection of cancer), predicting relapse in pulmonary tuberculosis sufferers.
- Waveform classification, e.g. speech recognition, seismic analysis (i.e. discrimination between earthquakes and nuclear explosions), target recognition from radar echoes and electroencephalogram analysis.

In statistical pattern recognition each object is being represented by a so called feature vector. Each component of the vector is a measured value of some feature of the object. The feature vector is input to the classifier which in turn assigns the object to one of the classes. Usually discriminant functions are used for classification purposes. These are a set of confidence functions, one function for each class. The function value increases with the confidence, and an object is assigned to the class with the maximum discriminant value.

Many of the problems we are concerned with, are to be used in a real time environment only. In real time applications fast decision making is a must. Hence only computationally simple discriminant functions such as those with linear, quadratic or piecewise linear discriminant functions may be used.

The linear classifiers are easy to implement and very fast to compute. Their error rate may, however, be quite high unless the classes are (almost) linearly separable. The quadratic classifiers are usually more reliable than the linear ones. Moreover they are almost as fast (assuming a "small" number of features to be used). Unfortunately the linear and the quadratic classifiers are not generally capable of handling data sets with multiple modes and/or concave data sets satisfactorily. Opposed to those classifiers, the piecewise linear classifiers manage to give reliable results in such situations as well. However, traditional piecewise linear discriminant functions may be quite complex and computationally heavy if classes are partly overlapping in feature space.

In this thesis new algorithms for generating a piecewise linear classifier are developed. The goal is to create a classifier which combines the linear classifier's speed with high reliability. Roughly speaking, the strategy is as follows: The sample space of each class is split into several disjoint subsample spaces. The samples included in the same subsample space are said to belong to the same subclass. Linear discriminant functions separating the subclasses are then constructed and employed to discriminate between the classes. To apply this method, several difficulties must be solved. Among the most important problems are

- how to split a given (sub)sample space,
- how to terminate the splitting,
- how to generate linear discriminant functions for separating the (sub)classes.

In the next chapter a brief introduction to pattern recognition is given.

Chapter 3 describes the new methods, while in chapter 4 the new algorithms are

developed and evaluated. In chapter 5 their performance are compared with that of other classifiers. Chapter 6 contains the summary and conclusion.

2 INTRODUCTION TO PATTERN RECOGNITION

2.1 Basic concepts

In statistical pattern recognition each object is represented by a feature vector $\mathbf{x}^t = [x_1, x_2, \dots, x_d]$ containing a measurement of predefined object features. Which features to use and how many depends on the application. We will, however, always search for features maximizing the separation between the classes. Hopefully, feature vectors from objects belonging to different classes will belong to separable regions in feature space.

The classification is usually done by use of so-called discriminant functions. This is a set of functions $g_i(\mathbf{x})$, one function for each class ω_i , $i = 1, 2, \dots, c$. An object with feature vector $\mathbf{x}$ is assigned to the class with maximum discriminant value. That is, the object is assigned to ω_k if

$$g_k(\mathbf{x}) = \max_i \{g_i(\mathbf{x})\} . \quad (2.1)$$

This decision rule corresponds to dividing the feature space into disjoint regions Ω_i , $i = 1, 2, \dots, c$ and assigning an object with feature vector $\mathbf{x}$ to ω_k if $\mathbf{x} \in \Omega_k$. A hypersurface which separates two regions, say Ω_i and Ω_j , is a hypersurface containing vectors $\mathbf{x}$ where

$$\begin{aligned} g_i(\mathbf{x}) &= g_j(\mathbf{x}) \quad \text{and} \\ g_i(\mathbf{x}) &> g_k(\mathbf{x}) \quad k \neq i, j . \end{aligned} \quad (2.2)$$

Figure 2.1 shows a 3-class, two dimensional feature space as an example.

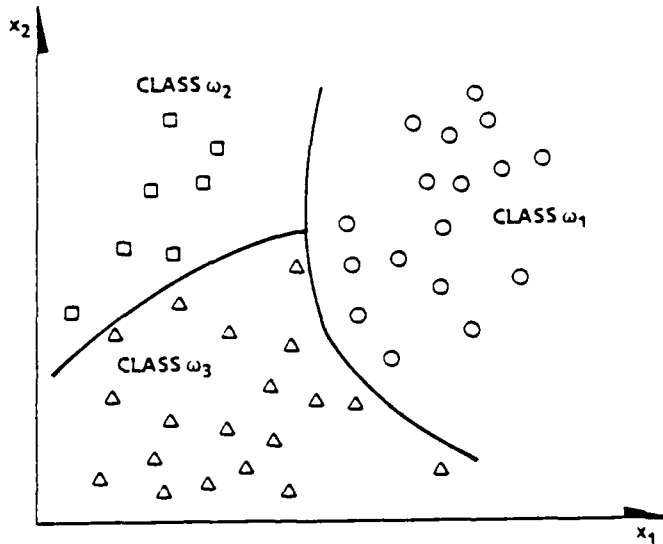


Figure 2.1: Two dimensional feature space, 3 classes.

A central problem in the classification theory is how to design efficient discriminant functions. Two main approaches are available; supervised and unsupervised learning. In this thesis we will consider the supervised learning approach only.

The most usual way of designing and evaluating discriminant functions is shown in figure 2.2 and is described in the following. In supervised learning we have available a data set where the true class of each feature vector (sample) is known in advance. Usually the data set is first divided into two subsets, one design set and one test set. The design set is then used for generating the classifier (discriminant functions). The test set is used for the evaluation. The reason for using an independent test set is that a too optimistic performance is achieved if design and testing are based on the same data. Other approaches for evaluating classifiers (error rate estimation) are also available (e.g. Hand [16, 17]).

The image processing and pattern recognition group at the Norwegian Defence Research Establishment (NDRE) is mostly concerned with real time applications of image analysis/pattern recognition. Typically, 25 TV-images are to be processed each second. Thus a very fast decision making (classification) algorithm is needed. However, the time required for designing (training) the

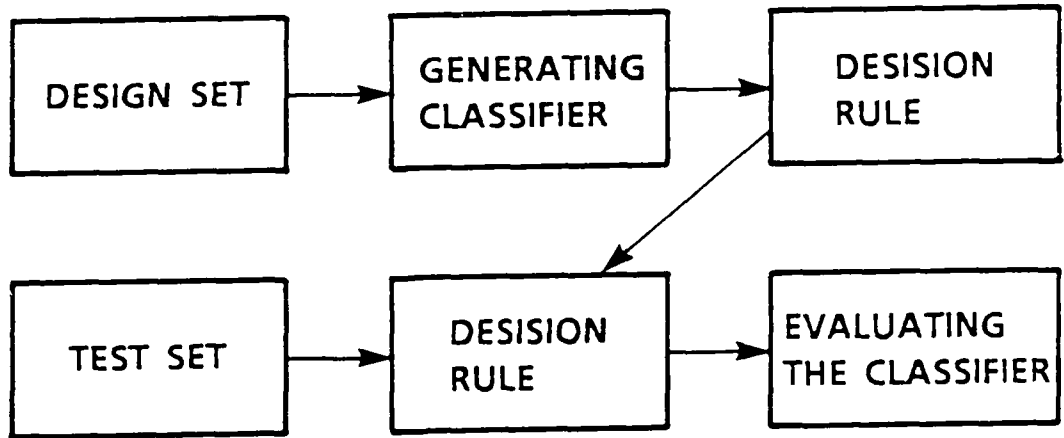


Figure 2.2: Illustration of design and evaluation of a classifier using the supervised learning approach.

classifier is of minor importance for us.

2.2 Linear classifiers

The discriminant functions of linear classifiers are of the form

$$g_i(\mathbf{x}) = \sum_{j=1}^d x_j a_{ij} + a_{i0} \quad (2.3)$$

where a_{ij} are real coefficients. By defining the augmented feature vector

$$\mathbf{y} = \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}$$

and the weight vector

$$\mathbf{a}_i = \begin{pmatrix} a_{i0} \\ a_{i1} \\ \vdots \\ a_{id} \end{pmatrix},$$

the discriminant function in 2.3 may be expressed by the inner product between $\mathbf{a}_i$ and the augmented feature vector $\mathbf{y}$:

$$g_i(\mathbf{x}) = \mathbf{a}_i^t \mathbf{y}.$$

It is easy to verify that the hypersurface separating two classes is a hyperplane perpendicular to the vector

$$\mathbf{a} = \mathbf{a}_i - \mathbf{a}_j. \quad (2.4)$$

Advantages of the linear classifier are the computational speed and ease of implementation. However, the error rate may be high if the classes are not (almost) linearly separable.

2.3 Quadratic classifiers

The discriminant functions of quadratic classifiers are given by

$$g_i(\mathbf{x}) = \mathbf{x}^t \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^t \mathbf{x} + w_i \quad (2.5)$$

where $\mathbf{W}_i$ is a $d \times d$ -matrix, $\mathbf{w}_i$ is a d -dimensional vector and w_i is a scalar. It may be shown that the decision surfaces between two classes are hyperquadratic, and can assume several forms: Pairs of hyperplanes, hyperspheres, hyperellipsoids and hyperparaboloids. Quadratic classifiers usually give a lower error rate than linear classifiers, and they are almost as fast provided that d is

“small” [28]. In fact, one may easily show that $\frac{d(d+3)}{2}$ multiplications (and additions) are required for computing the discriminant functions for each class.

In figure 2.3 a two class two-dimensional example is shown. 300 samples are generated from two bivariate Gaussian distributions. 150 of them are generated from a distribution with mean vector $[3, 0]^t$, unit variance and 150 samples from a $N\left(0, \begin{bmatrix} 1 & 0 \\ 0 & 9 \end{bmatrix}\right)$ distribution. The linear decision boundary is found by using the mean squared error method, and the quadratic surface is found by using the maximum likelihood method assuming Gaussian distributions. For details see Duda and Hart [8]. From the figure we can see that 6.0% of the samples in the

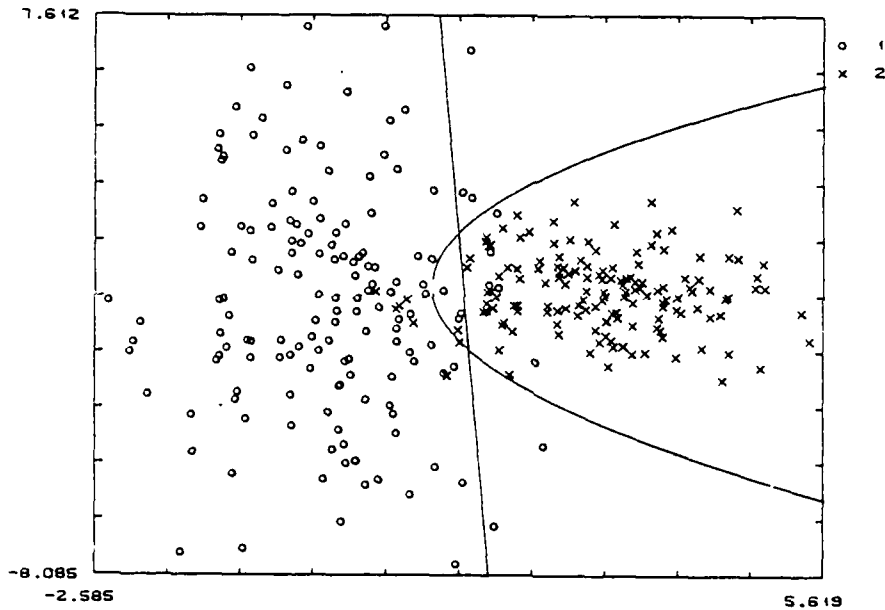


Figure 2.3: Examples of decisions boundaries when using linear and quadratic classifiers.

design set are misclassified if we're using the linear classifier and 4.7% if the quadratic one is used. Their error rates are 6.7% and 5.0% respectively.

Unfortunately, linear and quadratic classifiers will in general not be able to handle data sets containing multiple modes and/or concavities satisfactorily. An example is given in figure 2.4 where a linear classifier is used to discriminate between two “banana”-distributed classes. Although the classes form almost non-overlapping clusters, the error rate is quite high. The same will be true for the quadratic classifier.

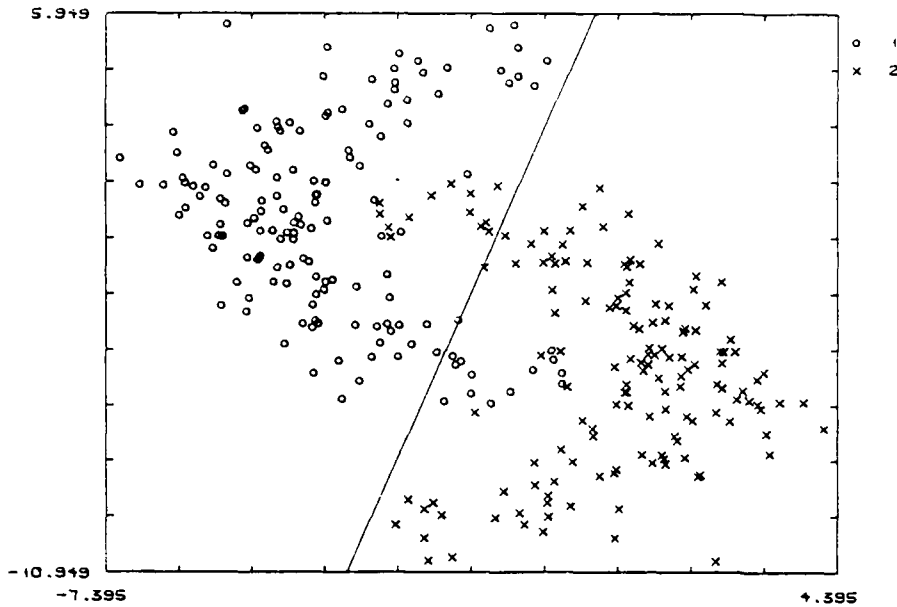


Figure 2.4: An example where both the linear and the quadratic classifiers fail.

2.4 Piecewise linear classifiers

A piecewise linear classifier, abbreviated PLC, may be defined as a classifier where the hypersurface separating two classes consists of hyperplanes. In other words, the hypersurface is piecewise linear. As an example, a piecewise linear classifier which discriminates well between the two banana distributions used in figure 2.4 is shown in figure 2.5.

Many algorithms for constructing piecewise linear classifiers have been suggested. The most important ones fall into one of the two following groups:

- a) Iterative methods based on a given (fixed) number of discriminant functions (number of weight vectors).
- b) Methods which increase the number of discriminant functions until the design set is correctly classified.

The group a) - algorithms require the number of weight vectors to be given in advance. This is generally a disadvantage because it often has to be chosen by trial and error. On the other hand, the group b) - classifiers may be very

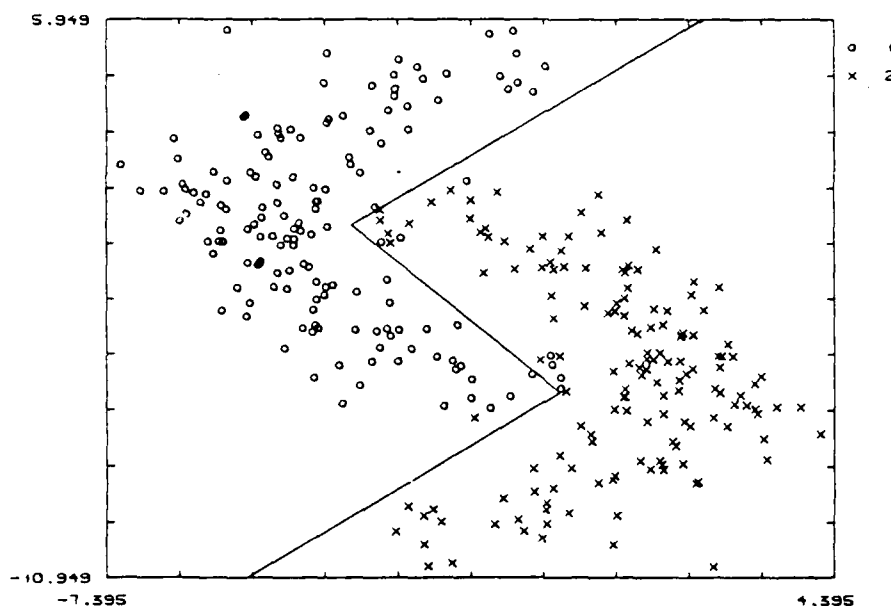


Figure 2.5: Example of a piecewise linear classifier with low error rate.

complicated. This is especially true if the different classes in the design set are not well separated. In this case the resulting classifier might turn out to be too complex and time consuming for many applications. We will now briefly describe the most important classifiers in the two groups.

Kesler constructed a c -class linear classifier [8] which in turn was modified to a c -class piecewise linear classifier by Duda and Fossum [7]. Instead of using only one weight vector for representing each class (as Kesler did), several weight vectors were applied. Assume we have determined n_i linear discriminant functions for ω_i (weight vectors $\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \dots, \mathbf{a}_{i_{n_i}}$). Then an object with feature vector $\mathbf{x}$ is assigned to ω_k if

$$\begin{aligned} g_k(\mathbf{x}) &= \max_i \{g_i(\mathbf{x})\} \\ &= \max_i \left\{ \max_j \{ \mathbf{a}_{i_j}^t \mathbf{y} \} \right\} \end{aligned} \quad (2.6)$$

where $\mathbf{y}$ denotes the corresponding augmented feature vector.

The problem is how to train the classifier. Kesler modified the fixed increment rule for linear classifiers. This training process is an iterative procedure, making the classifier more sensitive to the misclassified samples.

The classifier of Duda and Fossum is based on 2.6, and they modified Keslers rule further so several weight vectors could be trained for representing each class. Assume for a given augmented feature vector $\mathbf{y}$ from ω_i

$$\mathbf{a}_{i,j}^t \mathbf{y} = \max_k \{ \mathbf{a}_{i,k}^t \mathbf{y} \} \quad , \quad k = 1, 2, \dots, n_i .$$

Now, a correction is made if and only if $g_i(\mathbf{x}) - g_k(\mathbf{x})$, $k \neq i$, is less than a margin M . In other words

$$\mathbf{a}_{i,j}^t \mathbf{y} - M < \mathbf{a}_{k,l}^t \mathbf{y} \quad , \quad \begin{matrix} k \neq i \\ M > 0 \end{matrix} \quad (2.7)$$

and the following new weight vectors are generated in the m 'th correction, $m > 1$

$$\mathbf{a}_{p,q}(m) = \begin{cases} \mathbf{a}_{i,j}(m-1) + w_m \mathbf{y} & , \quad p = i, \quad q = j \\ \mathbf{a}_{k,l}(m-1) - w_m \mathbf{y} & , \quad p = k, \quad q = l \\ \mathbf{a}_{p,q}(m-1) & , \quad \text{otherwise} \end{cases} \quad (2.8)$$

where

$$0 < w_{\min} < w_m < w_{\max} .$$

The constants $w_{\min}$, $w_{\max}$, and the margin M have to be found by trial and error, and the way of generating w_m has to be known in advance.

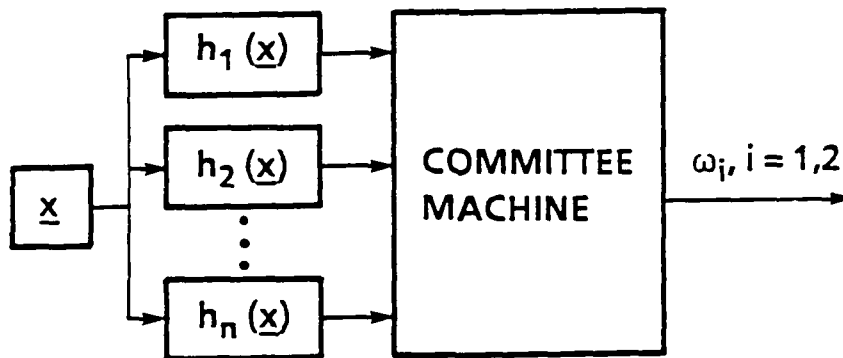
Chang [3] presented an algorithm for the two-class problem where the piecewise linear discriminant function $p(\mathbf{x}, \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$ is represented in terms of the so called maximum and minimum functions. For example,

$$p(\mathbf{x}, \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n) = \max \left\{ \min \{ \mathbf{a}_1^t \mathbf{y}, \dots, \mathbf{a}_{n_1}^t \mathbf{y} \}, \dots, \min \{ \mathbf{a}_{n_q+1}^t \mathbf{y}, \dots, \mathbf{a}_n^t \mathbf{y} \} \right\} \quad (2.9)$$

is a piecewise linear function, and $\mathbf{y}$ is still the augmented vector of $\mathbf{x}$. We also notice that 2.9 is a generalization of 2.6 (for the two-class problem). Moreover, a sample $\mathbf{x}$ is assigned to ω_1 if $p(\mathbf{x}, \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n) \geq 0$, otherwise it is assigned to ω_2 . The training procedure is iterative, and the weight vectors are trained one at a time. In iteration $i + 1$, the weight vector $\mathbf{a}_k(i + 1)$, $k = 1, 2, \dots, n$ is found by using a linear separation procedure on the data set $\mathcal{A}_k(i)$. This is a subset of the design set containing all samples for which $p(\mathbf{x}, \mathbf{a}_1, \dots, \mathbf{a}_n) = \mathbf{a}_k^t(i)\mathbf{y}$. This algorithm will hopefully find a suitable solution within a prespecified number of iterations.

Unfortunately the combination of max and min functions of the piecewise linear function has to be determined by trial and error. This may be a very difficult task.

Takiyama has made a committee machine for the two-class problem [34]. The committee machine is illustrated in figure 2.6. The decision logic and the



$$h_i(\underline{\mathbf{x}}) = \begin{cases} 1, & \underline{\mathbf{a}}_i^t \underline{\mathbf{y}} \geq 0 \\ 0, & \text{OTHERWISE} \end{cases}$$

Figure 2.6: Diagram of the committee machine

number of weight vectors has to be known in advance. The best known logics are probably the majority logic and the veto logic. The majority logic assigns a sample $\mathbf{x}$ to ω_1 if

$$\sum_{i=1}^n h_i(\mathbf{x}) \geq \frac{n+1}{2} \quad (2.10)$$

and the veto logic assigns a sample to ω_1 if

$$\sum_{i=1}^n h_i(\mathbf{x}) = n. \quad (2.11)$$

The problem is to determine the weight vectors $\mathbf{a}_1, \dots, \mathbf{a}_n$. Takiyama [34] has proposed an algorithm which try to optimize a modified perceptron criterion function with a gradient descent procedure.

A general disadvantage with the group a) – approaches is, as already mentioned, that the number of weight vectors has to be determined in advance. Furthermore, in Chang's and Takiyama's proposals, only the two-class problem has been considered.

Mangasarian [25], Mizouguchi et al [26] and Lee and Richards [24] have developed group b) – algorithms for the two-class problem. We will briefly describe their methods in the following.

Mangasarian [25] proposed an algorithm for using piecewise linear decision boundaries. The method is illustrated in figure 2.7 and figure 2.8. Linear programming minimizing the perceptron criterion function is used for dividing the feature space into three regions by two parallel hyperplanes. The region to the positive side of both hyperplanes, $\{\mathbf{y} : \mathbf{a}_1^t \mathbf{y} > \beta_1\}$, will contain only samples from ω_1 . The region to the negative side of both hyperplanes, $\{\mathbf{y} : \mathbf{a}_1^t \mathbf{y} < \alpha_1\}$, (we assume $\alpha_i < \beta_i \forall i$), contains samples from ω_2 only. The "confusion" region between the two hyperplanes may contain samples from both classes. The algorithm proceeds by applying the same procedure to the samples in the confusion region. This results in a confusion region within the confusion region, and so forth. The process terminates when the confusion region is empty. In other words, the termination takes place when all samples in the design set are correctly classified. Alternatively one may simply terminate the algorithm after a predetermined number of iterations, leaving a region with unclassified

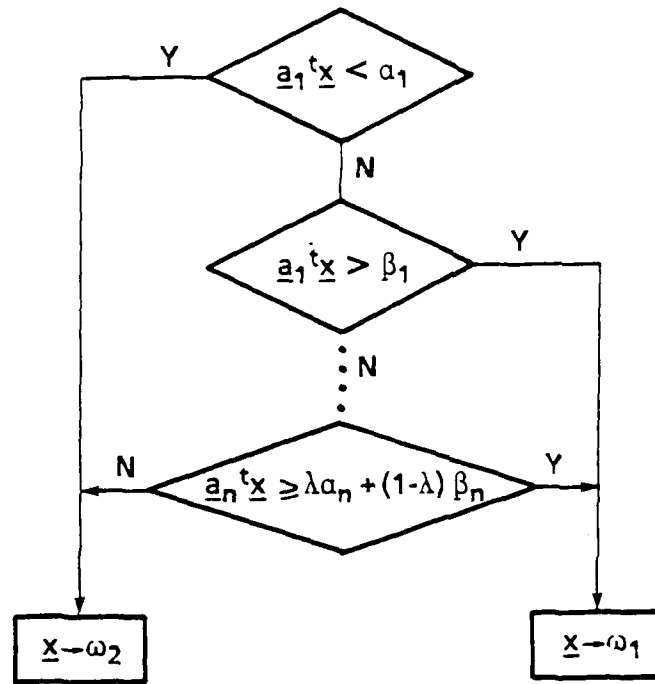


Figure 2.7: Mangasarian's algorithm, $0 \leq \lambda \leq 1$.

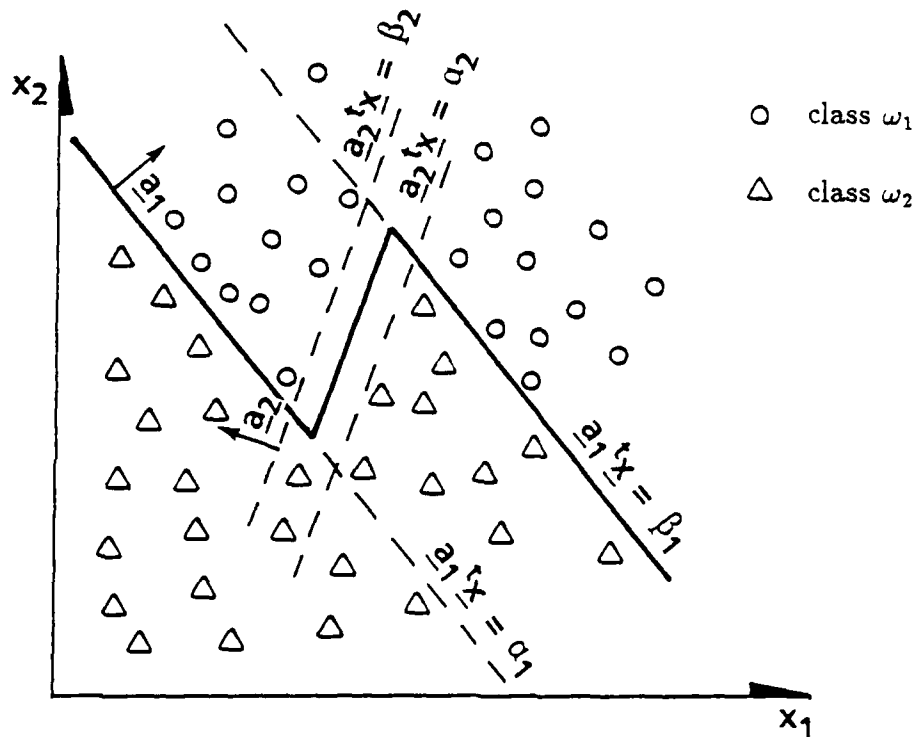


Figure 2.8: Illustration of decision boundary achieved by using Mangasarian's classifier.

samples. The decision boundary of the algorithm is illustrated in figure 2.8 for the situation where separation between the classes is obtained by using two weight vectors.

The piecewise linear classifier of Mizoguchi et al [26] is based on combining linear discriminant functions in a tree design. First the feature space is divided into two regions by using a linear discriminant function. Next, if the classes are not linearly separable, each of the two regions is divided into two subregions with linear discriminant functions. This process is repeated until all samples within a subregion belong to the same class. An example of a piecewise linear classifier constructed in this way is given in figure 2.9.

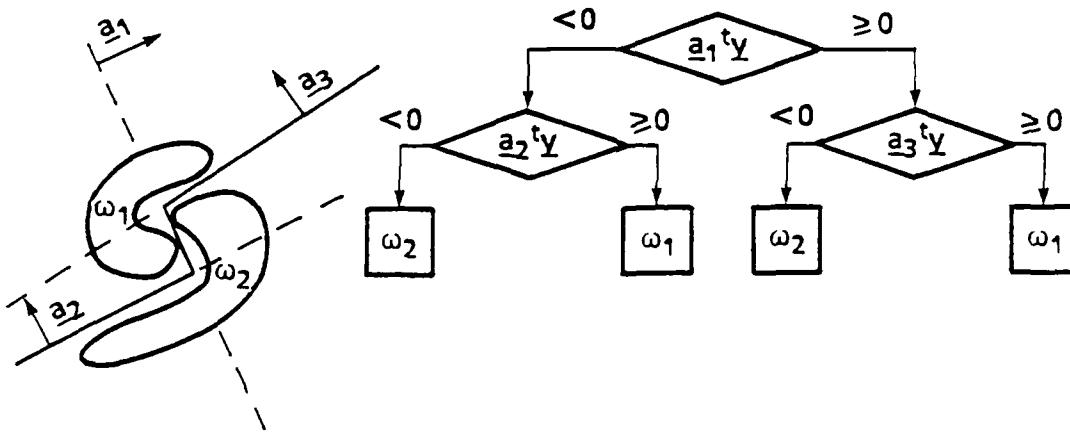


Figure 2.9: Example of a piecewise linear classifier constructed with the algorithm of Mizoguchi et al. The corresponding tree structure is also shown.

Lee and Richards [24] have developed a piecewise linear classifier based on the concept of single sided decision hyperplane. This is a hyperplane given by $\mathbf{a}^t \mathbf{y} = 0$ which has samples of one class only on the positive side of the plane ($\mathbf{a}^t \mathbf{y} > 0$). On the negative side, samples from several classes may be present. Initially, a single sided decision hyperplane perpendicular to $\mathbf{a}_1$ is computed. Then, the class of the samples on the positive side of the hyperplane is identified. Now, if samples from both classes are located on the negative side of

the hyperplane, then a new single sided decision hyperplane is created in this sub space. This process continues until samples from one class only are found on the negative side of the last constructed hyperplane. When the algorithm has terminated, we know that the samples in the design set is correctly classified.

This classifier may also be considered as a committee machine using seniority logic. In majority and veto logics, all the committee members have the same influence. This is not true for the seniority logic where some members are more influential than others. The influence and the number of committee members are determined during the training process.

The algorithm is illustrated in figure 2.10. $h_i(x)$ is defined in figure 2.6. We

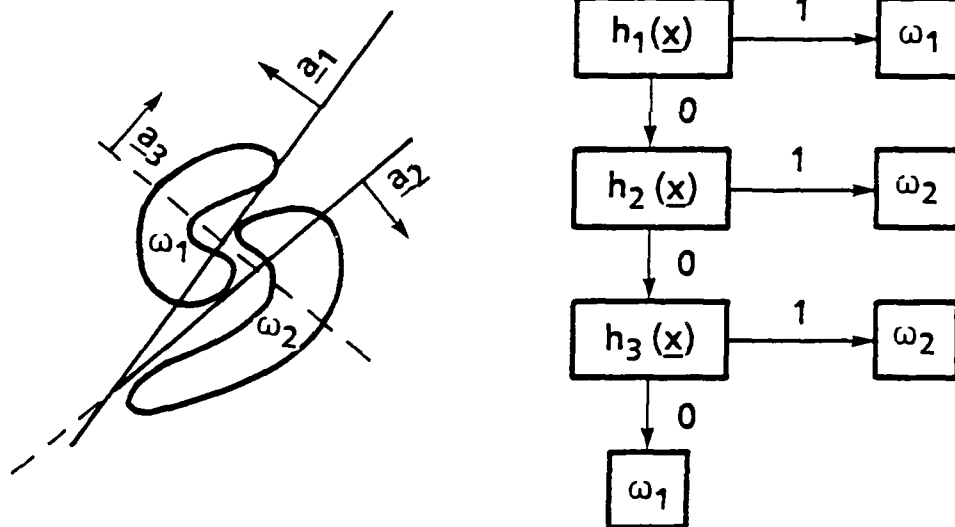


Figure 2.10: Examples of classification with Lee and Richards algorithm.

may also compute all $h_i(x)$ and then use a logical network for the classification. The network's truth table of the classifier in figure 2.10 is given in table 2.1. X denotes a "don't care" state of a committee member.

Hoffman and Moe [19] have developed an interesting two-class algorithm which

Committee member			Class
h_1	h_2	h_3	
1	X	X	ω_1
0	1	X	ω_2
0	0	1	ω_2
0	0	0	ω_1

Table 2.1: Truth table for the decision logic for the classifier used in figure 2.10.

does not fit into any of the two above mentioned groups of algorithms. They constructed a linear discriminant function and created a confusion region ($\alpha < \mathbf{a}'\mathbf{y} < \beta$) in the same way as Mangasarian [25]. But, opposed to him, they clustered the samples from each class in the confusion region into several clusters according to the following procedure. First the samples (from each class) are divided into two clusters. Next, the clusters, which are sufficiently "large" (with respect to the within-class scatter matrix), are divided further. This process continues until all clusters are "equal" and sufficiently "small" (also with respect to the within-class scatter matrix). Finally, they computed the sample mean of each cluster. The classifier assigns a sample $\mathbf{y}$ to ω_2 if $\mathbf{a}'\mathbf{y} < \alpha$, to ω_1 if $\mathbf{a}'\mathbf{y} > \beta$ and to the class with nearest cluster mean if $\alpha < \mathbf{a}'\mathbf{y} < \beta$.

As we have already mentioned, all algorithms we have described except the classifier of Duda and Fossum [7] are developed for the two-class problem only. If we want to classify an object to one out of c classes, $c > 2$, this may be done by creating two main class groups where several classes are merged together. While classifying, a sample is classified through different class groups until it is assigned to one of the c classes. Unfortunately this process makes the classifier much more complicated for the c -class problem than e.g. the quadratic classifier. An example where $c = 4$ is shown in figure 2.11. ω_1 and ω_2 is merged together to one class group, ω_3 and ω_4 to another. After first being assigned to either the class group $\omega_1 \cup \omega_2$ or the class group $\omega_3 \cup \omega_4$, a sample $\mathbf{y}$ can be assigned to one of the 4 classes.

If samples from the different classes are partly overlapping each other, no simple and reliable classifier is available. It is difficult to have any general knowledge

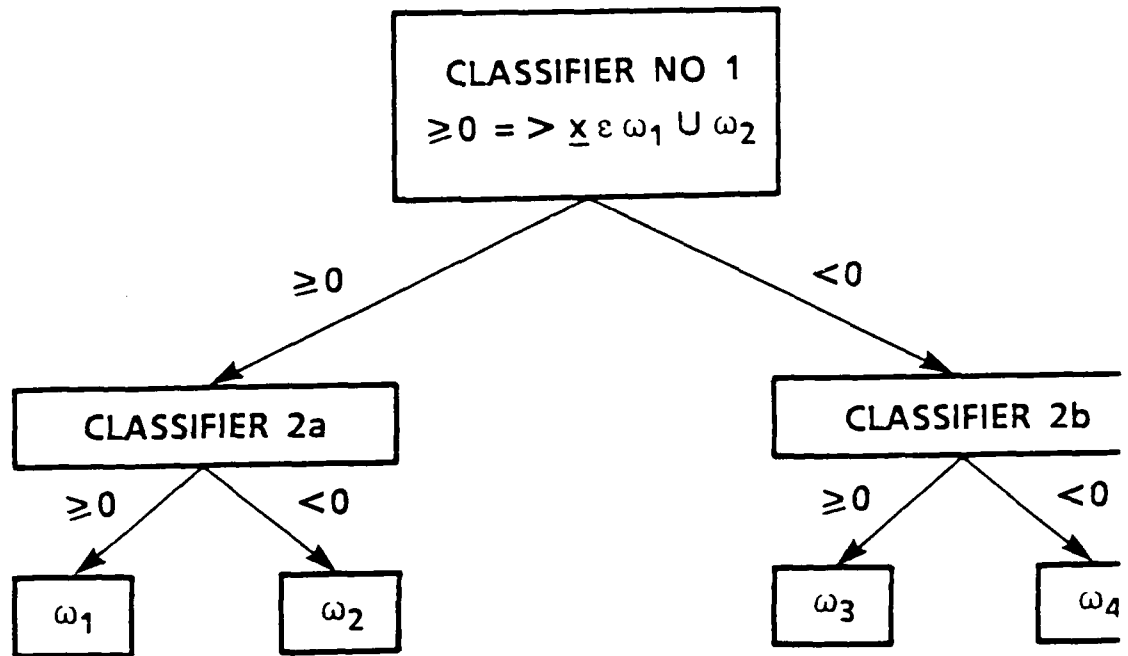


Figure 2.11: Illustration of how to use a two-class classifier for the c -class problem.

about the performance of the group a)-algorithms. Furthermore, one may easily be convinced that the group b)-algorithms tend to be quite complicated. This is of course inconvenient from a computational point of view. It is also important to remember that a correctly classified design set does not in general imply low error rate and hence a reliable classifier.

Therefore, one objective is to develop a classifier which attempts to fulfill the following requirements:

- It must be able to handle the c -class problem easily.
- A low error rate should be achieved with a relatively small number of (linear) discriminant functions rather than resulting in a correctly classified design set.

In the next chapter we will describe a method which attempts to accomplish these requirements.

3 THE NEW CONCEPT

In this chapter an introduction to a new concept for constructing a piecewise linear classifier is presented (for a brief presentation of the method, see also [29]). First the sample space of each class may be partitioned (split) into several (≥ 1) subsample spaces. The samples from a given class laying in the same subsample space are said to belong to the same *subclass*. The partitioning is done in order to make the classifier more adaptable to the classes. Next, the generated subclass information is used for constructing the piecewise linear discriminant function. Each subclass is represented by one linear discriminant function (one weight vector). However, since the distributions of the classes usually are unknown, estimation techniques have to be used. As an example the splitting process becomes a clustering process which divides the samples in a given (sub)class into clusters.

The concept is illustrated by the following "quasi-Pascal" statements:

```

n := c; finished:=FALSE;
<Initially, assign one subclass to each class, ( $\alpha_i := \omega_i$ ; ,  $i = 1, \dots, c$ ) >;
REPEAT
    <Construct a piecewise linear classifier using the subclasses  $\alpha_1, \dots, \alpha_n$  >;
    <Compute an evaluation criterion for the classifier>;
    IF <Sufficiently good performance> THEN finished:=TRUE
    ELSE BEGIN
        <Split the sample space of one of the subclasses  $\alpha_1, \dots, \alpha_n$  >;
    END;
UNTIL ((finished) OR ( $n \geq n_{max}$ ));
<Generate the final classifier>;

```

The algorithm is illustrated in figure 3.1. Here we have assumed the classes to be "banana" distributed. In figure 3.1a, a linear classifier is applied, in figure 3.1b the sample space of one of the classes is split into two subsample spaces and the resulting classifier is shown. Finally in figure 3.1c the sample

space of both classes are split in two subsample spaces each and the resulting classifier is shown.

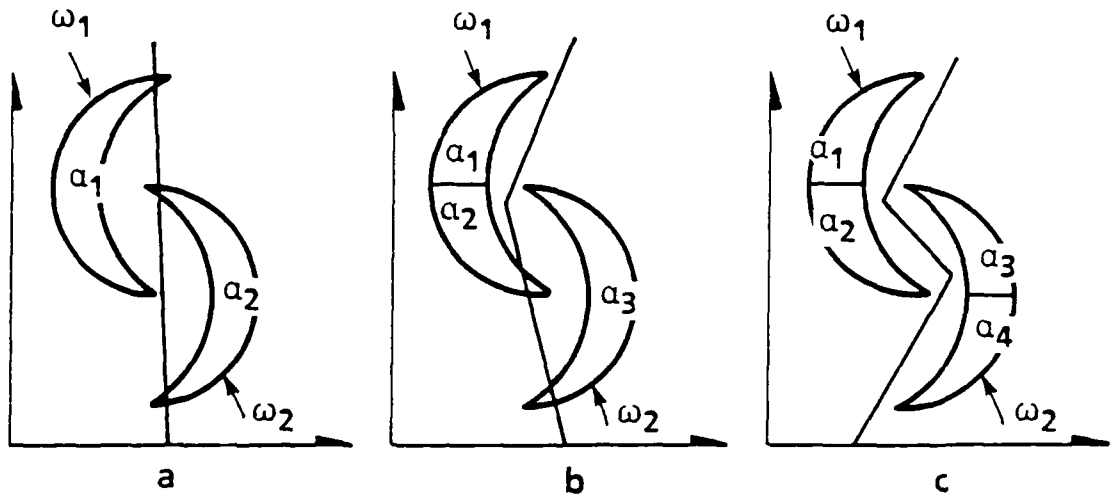


Figure 3.1: Illustration of how the algorithm works.

Although the basic idea is simple, there are several difficulties which have to be solved. The most important decisions to make are:

- which (sub)sample space to split,
- how to split a given (sub)sample space,
- how to use the subclass information for generating a piecewise linear discriminant function,
- how to terminate the splitting.

The algorithms making these decisions are developed in the next chapter. In the following, a brief description of the main ideas is given.

3.1 Finding a suitable (sub)sample space for splitting

The first problem we will study is how to find the best (sub)sample space to split. This is very important because we want to make the classifier as

adaptable as possible to the classes with only a limited number of linear discriminant functions. Two different splitting strategies have been developed.

The first strategy is a hillclimbing approach. This algorithm proceeds through a sequence of essentially equal steps. Assume that there are n subclasses. Now, we will in turn pick one of the n subclasses and

- split the sample space,
- generate a classifier for the particular set of subclasses,
- evaluate the classifier.

The subclass (when split) which result in a classifier having the best performance is chosen. We see that we will always split the "best" (sub)sample space. However, this approach requires a large amount of computation.

The procedure may also be illustrated with the following "quasi-Pascal" statements, where n is the number of subclasses and α_i denotes the i 'th subclass.

```

 $P_{e_{min}} := 1.0;$ 
FOR  $i := 1$  TO  $n$  DO
BEGIN
    <Split the sample space of  $\alpha_i$  >;
    <Generate a piecewise linear classifier using  $\alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_n$  >;
    <Compute the error rate  $P_e$  >;
    IF ( $P_e < P_{e_{min}}$ ) THEN
    BEGIN
         $k := i;$ 
         $P_{e_{min}} := P_e;$ 
    END;
    <Merge the sample spaces of  $\alpha_{i-1}$  and  $\alpha_{i+1}$  >;
END;
<Split the sample space of  $\alpha_k$  >;

```

As seen, this approach is very time consuming. Therefore, an other approach has been considered for situations where the design time is important.

It is reasonable to assume that the classifier need to be more sensitive with respect to the (sub)class having the greatest contribution to the (estimated) error rate. A performance improvement is hopefully achieved by splitting the sample space of this (sub)class. It is however based on the assumption that the error rate of a given (sub)class is caused by either multimodality or concavity and *not* by overlap with other (sub)classes. The procedure is illustrated in the following statements:

<Generate a piecewise linear classifier using the (sub)classes $\alpha_1, \dots, \alpha_n$ >
 <Find the (sub)class α_k contributing most to P_e >
 <Split the sample space of (sub)class α_k >

3.2 How to split a given subsample space

The purpose of the splitting module is to divide the sample space of a given (sub)class into two new disjoint sample spaces which are as "much separated" as possible. This process becomes a clustering procedure because no a priori information concerning the distributions and the sample spaces is assumed. The clustering is a quite difficult task because in a particular splitting there are far more possible partitions than can be evaluated. As an example, it is easily shown that there are approximately 10^{30} possible allocations of 100 samples into two clusters. Thus it is necessary to find a more efficient search method than the exhaustive search.

To achieve our goal we choose the method of limiting the search space. The algorithm consists of two parts.

First a coarse clustering is performed, followed by an optimization procedure based on the initial clustering.

Two strategies for the coarse clustering are developed. The simplest one divides

the data set along the direction of maximum variance. This is done by finding the principal axis of the data set. Hopefully the clusters are well separated.

The other strategy is somewhat more complicated. First a pair of widely separated samples is identified and used as starting samples (one for each cluster). Then each sample in the data set is assigned to one of the two clusters according to a given rule.

The results of these coarse clusterings are obviously not optimal with respect to any criterion. Therefore an optimization algorithm may be applied.

This optimization algorithm is based on evolutionary search, which unfortunately is a suboptimal procedure. This is because it is impossible to use an optimal branch and bound algorithm [23] together with the chosen criterion function. The branch and bound algorithm does also require a great amount of computer power if there are many samples in the data set to be split.

The evolutionary search uses the result from the previous step as an initial clustering. Next each sample is in turn considered as a candidate for reallocation. If reallocation results in an improved value of the criterion function, the sample is transferred. Otherwise it remains in its original cluster. When all the samples in the data set have been investigated, one iteration is finished. The algorithm terminates when no samples are transferred during an iteration.

3.3 Generation of piecewise linear classifiers

Given a set of n subclasses, a piecewise linear discriminant function has to be generated.

The simplest way of generating a piecewise linear discriminant function is to use the nearest sub-mean classifier. This classifier assigns a sample to the class with the nearest sub-mean. It is reasonable to assume that it has only limited ability to be adapted to the data set satisfactorily. Therefore other – and more sophisticated – methods are introduced.

Two of them are based on the mean squared error approach. In this approach weight vectors are produced which map the data set into a prespecified set of points with minimum squared error. It was initially assumed that classifiers based on the mean squared error approach would perform well because these classifiers are minimum mean squared error approximations to the Bayes classifier.

We will also introduce a new approach. The aim here is to generate a piecewise linear classifier based on separating certain pairs of (sub)classes by a hyperplane. First the algorithm determines the pairs of (sub)classes to be separated. Let us for a moment assume the density of each class to be known. Then two (sub)classes are assumed to need separation if the optimal hypersurface is intersecting the line going between the means of the (sub)classes. However, as long as these densities are unknown, these pairs are determined by using estimation techniques. Next, we have to design a linear classifier for each pair of (sub)classes. Finally, a piecewise linear classifier based on the linear classifiers is computed. This is done by using the generalized inverse, which is computed with the singular value decomposition method.

3.4 Termination of the algorithm and generation of the final classifier

The last operation to be done is to terminate the splitting process and generate a final classifier.

We first define the maximum number of subclasses of the classifier (maximum number of discriminant functions). Afterwards the splitting process continues until

- a) a maximum number of subclasses is reached,
- b) no subclass can be split further,
- c) the maximum contribution to the error rate (from one subclass) is sufficiently small.

The first criterion terminates the splitting when the maximum number of discriminant functions of the classifier is reached, and the second one is used if the number of samples in any subclass is too low. The third criterion terminates the splitting process if it is reasonable to believe that little reduction in the (design set based) error rate estimate is gained by further splitting.

Among a set of classifiers (based on different number of subclasses), we are to find a classifier with sufficiently good performance. Two ways to do this are suggested.

The first one is simply to find the classifier minimizing a given criterion function. The criterion function proposed in this work combines the error rate and the number of linear classifiers.

The second method selects the classifier having the smallest number of weight vectors among those which are sufficiently close to the classifier with the lowest design set based error rate estimate.

4 DEVELOPMENT OF THE ALGORITHMS

4.1 Determination of a suitable subsample space to split

In chapter 3 we stated that the sample space of each class (the space for which $p(\mathbf{x}|\omega_i) > 0$) was partitioned into disjoint subsample spaces in order to make the classifier more adaptable to the classes. This process is iterative. One (sub)sample space is being split during one iteration, and it is never later merged together with other (sub)sample spaces. Therefore it is of considerable importance to determine the right one.

In the previous chapter, we suggested two different strategies for selecting a (sub)sample space for splitting. The first one was a hillclimbing approach, which found the "best" (sub)sample space to split. The other one split the sample space of the (sub)class contributing most to the error rate. These strategies will be described in section 4.1.1 and 4.1.2 respectively. However, we will start with defining the (sub)sample spaces and deriving simple estimators for the error rate of a (sub)class.

At some point in the process, let us assume that we have split the sample spaces of the c classes into n subsample spaces and that the class ω_i consists of n_i disjoint subsample spaces. In other words

$$n = \sum_{i=1}^c n_i .$$

Furthermore, let S_i denote the sample space of ω_i and let S_{ij} denote the j 'th subsample space of ω_i . The subsample spaces of each class are disjoint which implies

$$S_{ij} \cap S_{ik} = \emptyset, \quad j \neq k$$

and

$$S_i = S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_{n_i}} .$$

We will in this section assume the n subsample spaces to be known a priori.

Finally, let β_i , denote the statement "The sample $\mathbf{x} \in S_i$,".

The error rate of a classifier, also called the probability of misclassifying samples, is given as

$$P(e) = \sum_{i=1}^c P(\omega_i) P(e|\omega_i) \quad (4.1)$$

where $P(e)$ is the overall probability of misclassifying samples (the error rate), $P(\omega_i)$ is the a priori probability of ω_i and $P(e|\omega_i)$ is the conditional probability of misclassifying samples from ω_i . However, when computing the error rate, we need to know the conditional probability density $p(\mathbf{x}|\omega_i) \forall i$. Usually these densities are not available. Therefore we have to use estimation techniques for these tasks. The estimation is based on a data set consisting of samples (realisations) drawn independently according to the class densities $p(\mathbf{x}|\omega_i)$, $i = 1, \dots, c$.

Now, assume that a classifier is available. Using

$$\hat{P}(e|\omega_i) = \frac{f_i}{N_i} , \quad (4.2)$$

we obtain

$$\hat{P}(e) = \sum_{i=1}^c P(\omega_i) \frac{f_i}{N_i} \quad (4.3)$$

where f_i is the number of misclassifications and N_i denotes the number of samples representing ω_i .

If desirable, we may also use the subclass information for obtaining the error rate. This is easily seen by using $\frac{N_{ij}}{N_i}$ and $\frac{f_{ij}}{N_i}$ as estimators for $P(\beta_{ij} | \omega_i)$ and

$P(e | \beta_{ij}, \omega_i)$ respectively. N_{ij} and f_{ij} denote respectively the number of samples and the number of misclassifications from the samples in the j 'th subclass of ω_i .

As a special case, we see that $\hat{P}(e) = \frac{f}{N}$ if $P(\omega_i) = \frac{N_i}{N}$ where f and N denote the total number of misclassifications and the total number of samples in the data set.

We will also in the following use a performance measure P_p , defined as $P_p = 1 - P_e$, where P_e denotes the design set based error rate estimate. This performance measure is useful for measuring how well a classifier is adapted to the data set.

Now, when the error rates and their estimates have been stated, we will study how this information may be used for determining which subsample space to split in the next iteration.

4.1.1 The hillclimbing approach

This algorithm proceeds through a sequence of essentially equal steps. Assume that n subsample spaces are available at a certain step. Then, pick one of the n subsample spaces in turn, split, generate a classifier and evaluate the result. The split, resulting in a classifier having the best performance is chosen, and the next step involving $n + 1$ subclasses is initiated.

As in chapter 3 we define:

$$\begin{aligned}
 \alpha_1 &= \text{x belongs to } \omega_1 \text{ and } \text{x} \in S_{1_1} \\
 &\vdots \\
 \alpha_{n_1} &= \text{x belongs to } \omega_1 \text{ and } \text{x} \in S_{1_{n_1}} \\
 \alpha_{n_1+1} &= \text{x belongs to } \omega_2 \text{ and } \text{x} \in S_{2_1} \\
 &\vdots \\
 \alpha_n &= \text{x belongs to } \omega_c \text{ and } \text{x} \in S_{c_{n_c}}
 \end{aligned} \tag{4.4}$$

The approach may be illustrated in the following statements:

```

Ppmas := 0.0;
FOR i := 1 TO n DO
BEGIN
    <Split the sample space of  $\alpha_i$  >;
    <Generate a PLC using the subclasses  $\alpha_1, \dots, \alpha_{i_1}, \alpha_{i_2}, \dots, \alpha_n$  >;
    <Compute the performance Pp >;
    IF (Pp > Ppmas) THEN
    BEGIN
        k := i;
        Ppmas := Pp;
    END;
    <Merge the sample spaces of  $\alpha_{i_1}$  and  $\alpha_{i_2}$  >;
END;
<Split the sample space of  $\alpha_k$  >;

```

The performance is computed according to eq 4.3. This algorithm is quite time consuming because in a given step each subsample space has to be split, a classifier generated and the result evaluated.

Even though the computation time required for the design phase is not important in most cases, it is of interest to investigate whether or not it is possible to reduce the design time without reducing reliability. Therefore another algorithm is described in the following.

4.1.2 A contribution based splitting

This algorithm splits and evaluates only one (sub)sample space in each iteration. We must therefore try to select the split which is most likely to improve the classification result. In order to obtain this, it is reasonable to split the sample space of the (sub)class with the largest contribution to the design set based error rate P_e . This is based on the assumption that contribution to P_e from a subclass is caused by concavity and/or multimodality of the (sub)class

and *not* by overlap with another subclass. The procedure may be illustrated with the following statements:

<Generate a PLC using the subclasses $\alpha_1, \dots, \alpha_n$ >;
 <Find the subclass α_k contributing most to P_e >
 <Split the sample space of α_k >;

We will now search for the subclass with the largest contribution to P_e . Using previously defined notation, this is the subclass α_k for which

$$\hat{P}(e, \alpha_k) = \max_j \left\{ P(\omega_{g(j)}) \frac{f_{g(j)h(j)}}{N_{g(j)}} \right\} \quad (4.5)$$

where α_j is defined according to 4.4. Moreover, $g(j)$ finds the class which the subclass α_j belongs to, and $h(j)$ returns the subclass number of α_j within $\omega_{g(j)}$.

In the two previous sections, we have developed two different splitting strategies. For being able to evaluate these different algorithms, several simulation experiments have been carried out.

4.1.3 Comparison of the two strategies

The only way of comparing these two strategies is through Monte Carlo simulations. In this context Monte Carlo simulations consists of first generating data sets for the different classes. Then for each subclass selection strategy, splitting algorithm and classifier algorithm, the performance is computed. This process is called a replication, and it is repeated a number of times in order to obtain reliable statistics on the performance. The mean of the performance and the corresponding standard deviation are used as statistics in this evaluation. The standard deviation of the performance may be viewed as a stability criterion. If the standard deviation is high, it may be assumed that the splitting performance depends very much on actual realizations, and hence the algorithm may be unreliable.

The disadvantage of using Monte Carlo simulations is that we cannot draw general conclusions. However, by using many different probability distributions

and number of samples in the data sets, the evaluation should fairly well represent the algorithms capability. In the simulations we used classes containing Gaussian distributed, double exponentially distributed, "banana" distributed (to be defined below) and "bimodal Gaussian" distributed samples. Moreover, in the simulations using Gaussian or double exponential distributed samples, different relative positions of the distributions were evaluated, see figure 4.1. Simulations were made for $\gamma = k \times 10^\circ, k = 0, 1, \dots, 9$. For each

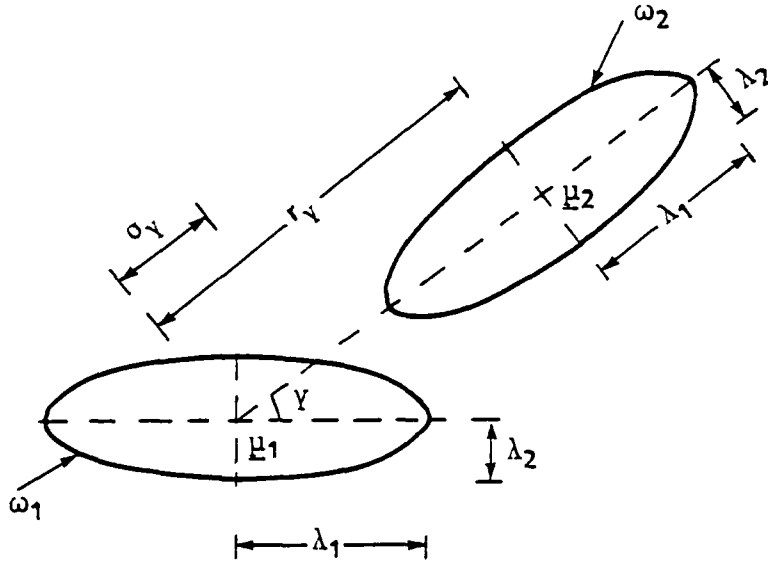


Figure 4.1: Distribution of Gaussian and double exponential samples.

angle γ , 4 different distances r_γ were used. In all simulations we have used

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} = \begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix}$$

In order to have a "rotation invariant" distance r_γ , we defined

$$r_\gamma = d(\sigma_\gamma + \lambda_1)$$

where

$$\sigma_\gamma = [\cos(\gamma), \sin(\gamma)] \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} \cos(\gamma) \\ \sin(\gamma) \end{pmatrix}.$$

This means that σ_γ denotes the standard deviation of the samples from ω_1 projected into the vector $[\cos(\gamma), \sin(\gamma)]^t$. When μ_1 and γ are known, μ_2 can be determined as

$$\mu_2 = \begin{pmatrix} \mu_1(1) + r_\gamma \cos(\gamma) \\ \mu_1(2) + r_\gamma \sin(\gamma) \end{pmatrix}.$$

By using this way of defining the distance between μ_1 and μ_2 , the overlap of samples between ω_1 and ω_2 is, for a given d , almost independent of the angle γ . Simulations are made using $d = 0.5, 1.0, 1.5$ and 2.0 .

Simulations have also been carried out with samples from ω_1 taken from a "banana" distribution, and the samples from ω_2 taken from a Gaussian distribution. This is shown in figure 4.2. In these simulations, the ω_2 samples

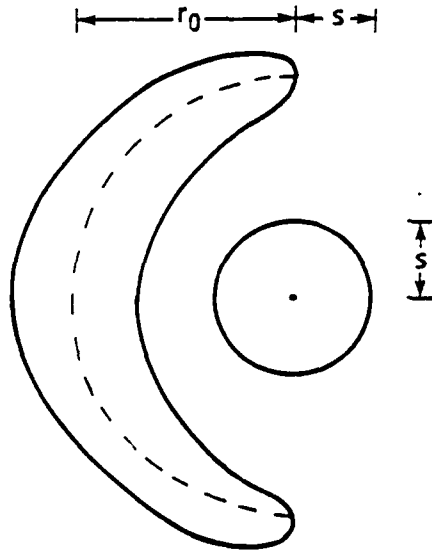


Figure 4.2: Distribution of banana and Gaussian distribution.

are taken from a $N(0, s \mathbf{I})$ distribution. The banana distribution is defined according to figure 4.3. In polar coordinates, using μ as origo, the banana distribution has a density

$$p(r, \theta) = p(\theta) p(r|\theta)$$

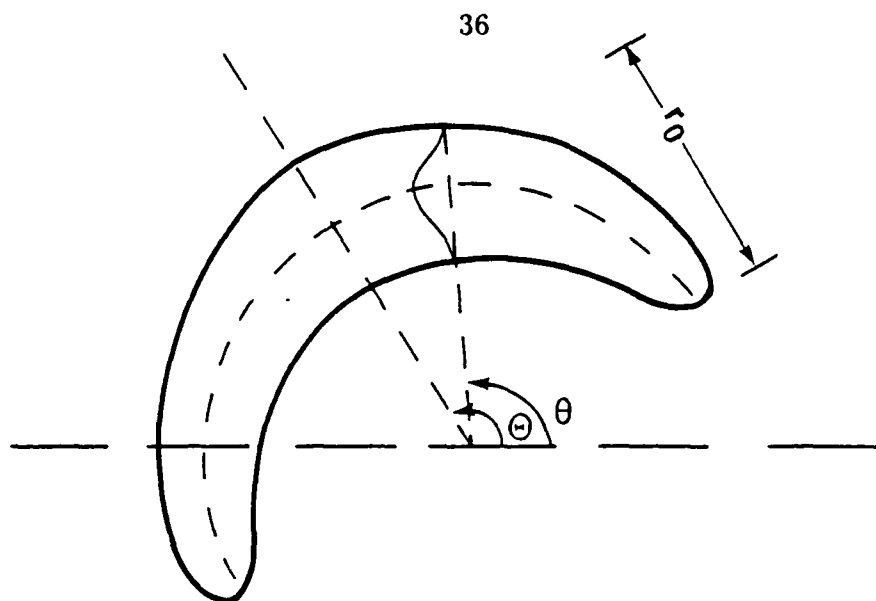


Figure 4.3: The banana distribution.

where

$$p(\theta) = \begin{cases} -\frac{2}{\pi^2}|\theta - \Theta| + \frac{3}{2\pi} & , |\theta - \Theta| \leq \frac{\pi}{2} \\ 0 & , \text{otherwise} \end{cases}$$

$$p(r|\theta) = N(5, \sigma(\theta)) \quad (r_0 = 5)$$

and

$$\sigma(\theta) = 1 - \frac{0.7}{\pi}|\theta - \Theta|.$$

Due to symmetry, the results are independent of Θ . Simulations are made for 6 different values of s ($s = \frac{k}{2}, k = 1, \dots, 6$) to investigate various degrees of overlap.

Finally, a 3 class problem was simulated in order to demonstrate the multi-class properties of the classifiers, as well as to see how the methods handle multimodal distributions. The distributions were chosen as follows:

$$p(\mathbf{x}|\omega_1) = N \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, s \begin{pmatrix} 1 & 0 \\ 0 & 9 \end{pmatrix} \right)$$

$$p(\mathbf{x}|\omega_2) = \frac{1}{2} \left[N \left(\begin{pmatrix} 3 \\ 0 \end{pmatrix}, s \mathbf{I} \right) + N \left(\begin{pmatrix} -2 \\ -2 \end{pmatrix}, s \mathbf{I} \right) \right]$$

$$p(\mathbf{x}|\omega_3) = N \left(\begin{pmatrix} 2 \\ 3 \end{pmatrix}, s \mathbf{I} \right)$$

Simulations using $s = 0.5, 1.0, 1.5$ and 2.0 have been made.

All classifiers to be described in section 4.3 were used in the simulations. Moreover, there were 25 samples in the data set of each class. In some of the simulations we have also used 150 samples in the data set of each class (Gaussian and double exponential distributions using $\gamma = 30^\circ$ and $\gamma = 60^\circ$, the banana/Gaussian distributions and the three class problem). The reason for mostly using small data sets is that the computation time increases considerably as the number of samples increases. Equal a priori probabilities have also been assumed everywhere in these simulations, and only two-dimensional samples have been used. These restrictions are caused by the limitations imposed by the high computationally costs of the simulations. Therefore, we prefer to study situations in reasonable detail rather than doing exploratory analysis of a larger but less well defined classes of situations. Moreover, we have no indications that the algorithms should behave essentially different in a higher dimensional space, or in situations with unequal a priori probabilities.

It is unfortunately impossible to present all results from the simulations. Therefore we will only illustrate the main results using examples and give some general remarks. However, all results are available on request.

Before discussing our observations, we present some examples of the simulation results. Figure 4.4 shows the mean performance of simulations using double exponential distributions and the classifier described in 4.3.4. In figure 4.5 we have plotted the results from the simulations where the classes are banana and Gaussian distributed. Furthermore, the results are also tabulated in tables 4.1–4.4. Both mean and standard deviation of P_e are given.

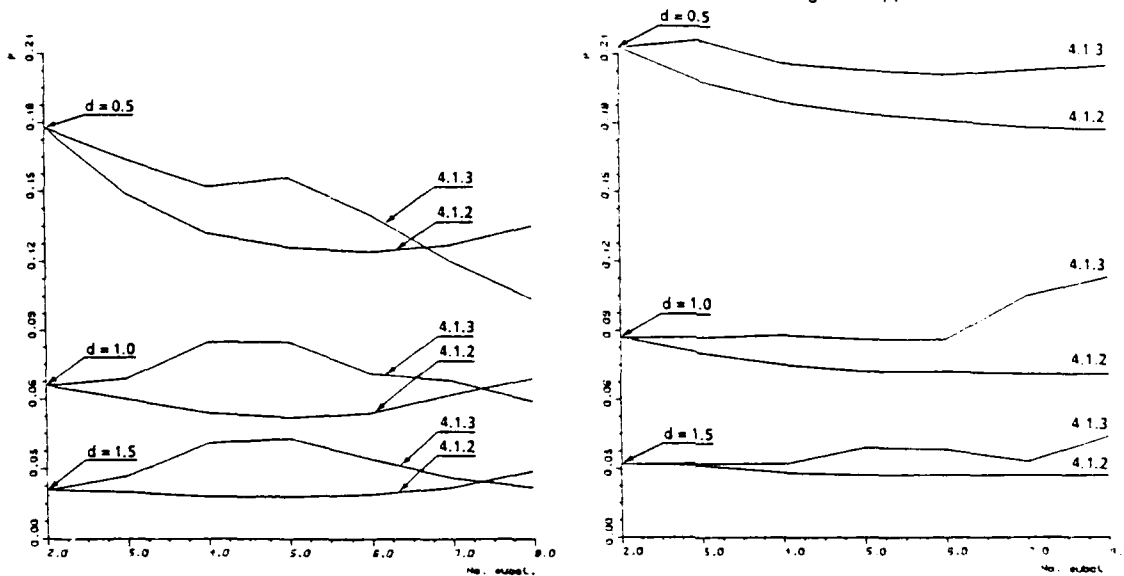


Figure 4.4: Design set based error rate estimates using 25 (a) and 150 (b) samples taken from double exponential distributions, $\gamma = 30^\circ$. Plots are made for both splitting strategies with $d=0.5$, 1.0 and 1.5.

As expected we found the hillclimbing approach to be clearly better than the contribution based splitting. At least 10% - 20% better performance is achieved by using the hillclimbing approach. When using the contribution based splitting, the standard deviation of the performance increases rapidly as the number of subclasses increases. These observations should indicate that this procedure is somewhat unstable and hence unreliable. However, this simplified approach may perform satisfactorily if the main reason for errors is caused by multiple mode and/or concavities. This fact is seen from the simulations using classes with Gaussian and banana distributed samples with small values of s . However, when there are moderate or more overlap between the classes, the contribution based splitting performs poorly. In both approaches, the performance do also tend to be poor and unreliable when the number of samples in the subclasses are small. This is for instance seen in the simulations using 25 samples in each class and having 6–8 subclasses. In these simulations,

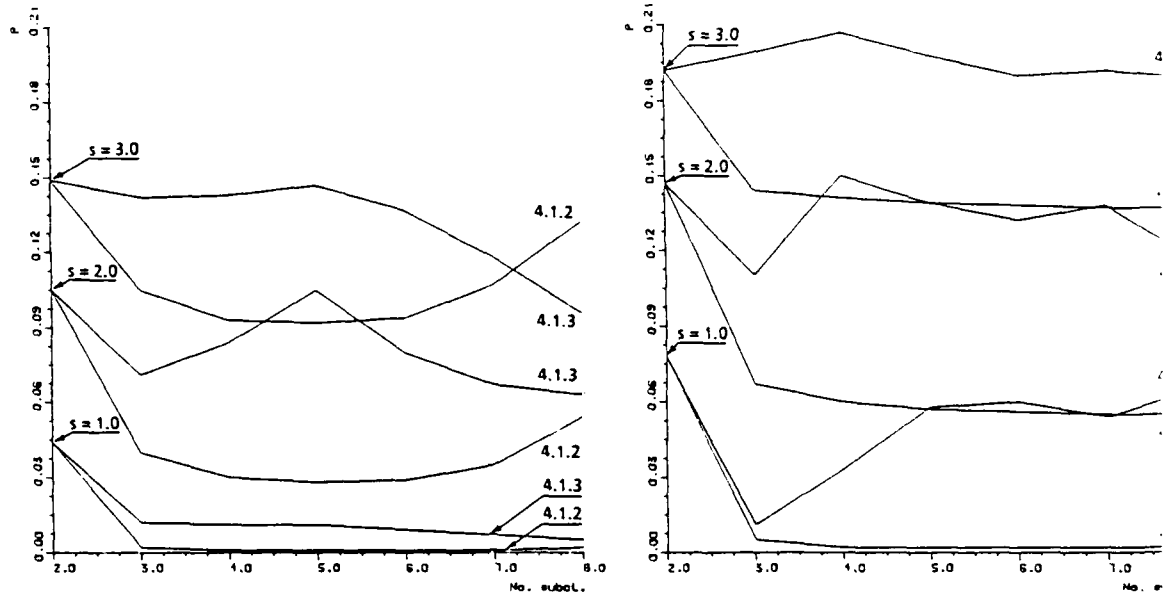


Figure 4.5: Design set based error rate estimates using 25 (a) and 150 (b) samples taken from banana and Gaussian distributions. Plots are made for both splitting strategies with $s=1.0$, 2.0 and 3.0 .

the standard deviation becomes quite high.

To conclude this discussion, we found the hillclimbing approach to be clearly superior to the contribution based splitting. Thus, it should be used in all situations for which the design time is of minor importance.

Spl.	d	No. of subclasses						
		2	3	4	5	6	7	8
4.1.2	0.5	0.178	0.149	0.132	0.126	0.124	0.127	0.135
		0.049	0.043	0.041	0.042	0.045	0.055	0.073
4.1.2	1.0	0.066	0.060	0.054	0.052	0.054	0.062	0.069
		0.034	0.031	0.028	0.028	0.033	0.052	0.066
4.1.2	1.5	0.021	0.020	0.018	0.018	0.019	0.022	0.029
		0.021	0.019	0.018	0.020	0.020	0.031	0.044
4.1.2	2.0	0.006	0.007	0.006	0.006	0.007	0.009	0.022
		0.011	0.014	0.014	0.015	0.018	0.026	0.034
4.1.3	0.5	0.178	0.164	0.152	0.156	0.140	0.120	0.104
		0.049	0.047	0.049	0.093	0.122	0.146	0.153
4.1.3	1.0	0.065	0.069	0.085	0.085	0.071	0.068	0.059
		0.033	0.043	0.091	0.113	0.135	0.149	0.149
4.1.3	1.5	0.021	0.027	0.041	0.043	0.034	0.026	0.022
		0.021	0.031	0.076	0.099	0.102	0.098	0.095
4.1.3	2.0	0.006	0.009	0.018	0.016	0.013	0.012	0.010
		0.011	0.024	0.066	0.069	0.067	0.067	0.066

Table 4.1: Results using 25 samples taken from double exponential distributions, $\gamma = 30^\circ$. Both mean and standard deviation of the design set based error rate estimate are tabulated for each method/distance.

Spl.	d	No. of subclasses						
		2	3	4	5	6	7	8
4.1.2	0.5	0.213	0.198	0.189	0.184	0.181	0.178	0.177
		0.020	0.023	0.024	0.022	0.024	0.025	0.026
4.1.2	1.0	0.087	0.080	0.075	0.072	0.072	0.071	0.071
		0.015	0.016	0.015	0.015	0.016	0.016	0.016
4.1.2	1.5	0.032	0.031	0.028	0.027	0.027	0.027	0.027
		0.010	0.009	0.009	0.009	0.009	0.009	0.009
4.1.2	2.0	0.012	0.011	0.011	0.010	0.010	0.010	0.010
		0.007	0.007	0.007	0.007	0.007	0.007	0.007
4.1.3	0.5	0.213	0.216	0.206	0.203	0.201	0.203	0.205
		0.021	0.025	0.026	0.028	0.034	0.052	0.046
4.1.3	1.0	0.087	0.087	0.088	0.086	0.086	0.105	0.113
		0.015	0.018	0.019	0.018	0.023	0.081	0.116
4.1.3	1.5	0.032	0.032	0.032	0.039	0.038	0.033	0.044
		0.010	0.009	0.013	0.049	0.049	0.051	0.098
4.1.3	2.0	0.012	0.013	0.015	0.019	0.022	0.033	0.035
		0.006	0.007	0.011	0.024	0.054	0.090	0.103

Table 4.2: Results using 150 samples taken from double exponential distributions, $\gamma = 30^\circ$. Both mean and standard deviation of the design set based error rate estimate are tabulated for each method/distance.

Spl.	d	No. of subclasses						
		2	3	4	5	6	7	8
4.1.2	0.5	0.016	0.001	0.000	0.000	0.000	0.000	0.000
		0.017	0.006	0.003	0.003	0.003	0.003	0.007
4.1.2	1.0	0.044	0.002	0.001	0.001	0.001	0.001	0.002
		0.026	0.009	0.007	0.007	0.007	0.012	0.015
4.1.2	1.5	0.077	0.014	0.009	0.008	0.008	0.010	0.018
		0.034	0.018	0.014	0.013	0.014	0.028	0.046
4.1.2	2.0	0.105	0.040	0.030	0.028	0.029	0.035	0.054
		0.040	0.029	0.025	0.024	0.025	0.043	0.068
4.1.2	2.5	0.129	0.074	0.061	0.059	0.060	0.072	0.096
		0.044	0.037	0.034	0.033	0.037	0.056	0.074
4.1.2	3.0	0.149	0.105	0.093	0.092	0.094	0.107	0.132
		0.047	0.042	0.041	0.043	0.046	0.064	0.079
4.1.3	0.5	0.015	0.004	0.002	0.003	0.001	0.001	0.001
		0.016	0.018	0.014	0.024	0.015	0.014	0.009
4.1.3	1.0	0.044	0.012	0.011	0.011	0.009	0.007	0.005
		0.026	0.029	0.031	0.038	0.041	0.042	0.032
4.1.3	1.5	0.076	0.038	0.051	0.058	0.044	0.034	0.026
		0.033	0.045	0.073	0.088	0.083	0.083	0.078
4.1.3	2.0	0.105	0.071	0.084	0.105	0.080	0.067	0.063
		0.040	0.052	0.076	0.098	0.095	0.108	0.116
4.1.3	2.5	0.129	0.108	0.122	0.130	0.123	0.107	0.091
		0.043	0.055	0.076	0.092	0.113	0.133	0.136
4.1.3	3.0	0.149	0.142	0.143	0.147	0.137	0.118	0.096
		0.046	0.060	0.066	0.083	0.117	0.133	0.141

Table 4.3: Results using 25 samples taken from Gaussian and banana distributions. Both mean and standard deviation of the design set based error rate estimate are tabulated for each method/distance.

Spl.	d	No. of subclasses						
		2	3	4	5	6	7	8
4.1.2	0.5	0.036	0.000	0.000	0.000	0.000	0.000	0.000
		0.009	0.000	0.000	0.000	0.000	0.000	0.000
4.1.2	1.0	0.078	0.005	0.002	0.002	0.002	0.002	0.002
		0.014	0.004	0.003	0.002	0.002	0.003	0.002
4.1.2	1.5	0.115	0.029	0.022	0.020	0.019	0.019	0.019
		0.018	0.009	0.008	0.008	0.008	0.008	0.008
4.1.2	2.0	0.147	0.067	0.060	0.057	0.056	0.055	0.055
		0.019	0.013	0.014	0.014	0.014	0.014	0.014
4.1.2	2.5	0.173	0.107	0.101	0.099	0.098	0.097	0.097
		0.020	0.015	0.015	0.016	0.016	0.016	0.016
4.1.2	3.0	0.192	0.144	0.141	0.139	0.138	0.137	0.137
		0.020	0.019	0.019	0.020	0.020	0.020	0.020
4.1.3	0.5	0.036	0.001	0.004	0.003	0.001	0.003	0.003
		0.009	0.007	0.023	0.018	0.005	0.016	0.017
4.1.3	1.0	0.078	0.011	0.033	0.058	0.060	0.054	0.065
		0.014	0.022	0.066	0.085	0.086	0.076	0.100
4.1.3	1.5	0.115	0.054	0.098	0.100	0.080	0.087	0.080
		0.018	0.045	0.087	0.094	0.075	0.075	0.081
4.1.3	2.0	0.146	0.110	0.150	0.139	0.132	0.138	0.116
		0.019	0.051	0.081	0.085	0.079	0.085	0.078
4.1.3	2.5	0.172	0.168	0.188	0.178	0.174	0.169	0.167
		0.020	0.039	0.058	0.053	0.053	0.061	0.062
4.1.3	3.0	0.192	0.199	0.207	0.198	0.190	0.192	0.189
		0.020	0.030	0.034	0.041	0.039	0.049	0.047

Table 4.4: Results using 150 samples taken from Gaussian and banana distributions. Both mean and standard deviation of the design set based error rate estimate are tabulated for each method/distance.

4.2 How to split a given subsample space

The purpose of the splitting process is to make the classifier more adaptive to the classes. If a (sub)class to split is known, the splitting algorithm attempts to divide the given (sub)sample space into two disjoint parts which is as much separated as possible. Thus, our problem is to divide the sample space S into two sample spaces S_1 and S_2 ($S = S_1 \cup S_2$, $S_1 \cap S_2 = \emptyset$) such that a given criterion function is optimized. As an example, we found that maximizing the distance from the mean of the original (sub)class to the mean of the nearest new subclass gives good results. Intuitively, this is also a reasonable criterion to use. Due to unknown probability densities, we have to estimate the criterion function using the data set $\mathcal{X}$, which consists of N samples representing the (sub)class to split. Assume the sample spaces S'_1 and S'_2 are to be evaluated, and moreover let $\mathcal{X}'_1 = \{\mathbf{x}_{1_1}, \dots, \mathbf{x}_{1_{N_1}}\}$ denote the data set of samples for which $\mathbf{x}_{1_i} \in S'_1$. Furthermore, $\mathcal{X}'_2 = \{\mathbf{x}_{2_1}, \dots, \mathbf{x}_{2_{N_2}}\}$ denotes the data set of samples for which $\mathbf{x}_{2_i} \in S'_2$. N_i denotes the number of samples in $\mathcal{X}_i$, $i = 1, 2$. Then an estimate of the above criterion is

$$J(\mathcal{X}'_1, \mathcal{X}'_2) = \min_i \left\{ \left\| \frac{1}{N_i} \sum_{k=1}^{N_i} \mathbf{x}_{i_k} - \frac{1}{N} \sum_{k=1}^N \mathbf{x}_k \right\| \right\}$$

and the best to do – and hence our aim – is to determine the data sets $\mathcal{X}_1$ and $\mathcal{X}_2$ for which J is optimized. Because S_1 and S_2 cannot be uniquely determined, we are choosing one of the possible solutions, for example the one for which the surface separating S_1 and S_2 is the nearest neighbour hypersurface separating $\mathcal{X}_1$ and $\mathcal{X}_2$. This surface may easily be found when $\mathcal{X}_1$ and $\mathcal{X}_2$ have been determined.

As we have seen, the splitting is actually a clustering problem. This problem is quite difficult to solve due to the astronomical number of possible partitions which have to be evaluated. Thus we are left with two choices: Either finding a much more efficient search method than the exhaustive search, or to limit the search to only parts of the space of partitions. Optimal splitting according to a given criterion is possible for a very limited set of criterion functions only [23],

and unfortunately it cannot be used with the criterion function we found to be the best. Therefore we have to use a suboptimal algorithm based on the principle of evolutionary search. Nevertheless it has been found to perform well. Our algorithm consists of two parts. First a coarse clustering is performed for creating an initial splitting. Secondly, an optimization procedure based on the initial splitting is performed. In 4.2.1 the initial splitting is studied, and in 4.2.2 the optimization procedure is considered.

4.2.1 Initial splitting procedures

There are two ways of performing a hierarchical clustering of data sets, either agglomerative or divisive methods [15]. In agglomerative methods we are starting with n clusters, one sample in each cluster. Then the two nearest (according to a given criterion function) clusters are merged, and there are now $n - 1$ clusters left. This process continues until the samples are clustered in a predefined number of clusters. Agglomerative methods may perform well if the samples actually fall into clusters which are fairly well separated. Otherwise, most of the samples usually tend to be included in one cluster only, and hence a very poor splitting will result. For our problems, the above requirement is generally not fulfilled since samples may have been taken from a unimodal distribution. Therefore agglomerative clustering algorithms are not appropriate.

In a divisive hierarchical clustering algorithm, a class represented by a set of samples is split into two clusters. Divisive algorithms are useful for splitting in situations where all samples are taken from a unimodal probability distribution. We have studied two different strategies. These will be described in the following paragraphs. In 4.2.1.3 problems concerning outliers are treated.

4.2.1.1 Splitting based on the scatter matrix

The first strategy described is quite simple to compute. It is based on dividing the data set along the direction of maximum variance. This is a reasonable strategy to use for several reasons. First of all the method results in new clusters with reduced variance in the direction of the maximum variance of the parent cluster. Therefore we may assume that the trace of the (sub)classes will be

substantially reduced. Moreover, if we assume that the samples in the data set are taken from a Gaussian distribution, then we are maximizing the projected distance from the mean of the data set to the mean of the nearest (new) cluster.

This direction is given by the vector $\mathbf{a}$, for which

$$\sum_{k=1}^N [\mathbf{a}^t(\mathbf{x}_k - \boldsymbol{\mu})]^2 = \max_{\boldsymbol{\gamma}} \left\{ \sum_{k=1}^N [\boldsymbol{\gamma}^t(\mathbf{x}_k - \boldsymbol{\mu})]^2 \right\}, \quad \|\mathbf{a}\| = \|\boldsymbol{\gamma}\| = 1 \quad (4.6)$$

where N is the number of samples in the given data set, $\boldsymbol{\mu}$ its mean, and $\boldsymbol{\gamma}$ is an arbitrary vector of unit length. The vector $\mathbf{a}$ is easily obtained by using the scatter matrix $\mathbf{W}$ of the data set which is defined as

$$\mathbf{W} = \sum_{i=1}^N (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^t \quad (4.7)$$

The vector $\mathbf{a}$ is parallel to the eigenvector corresponding to the greatest eigenvalue of $\mathbf{W}$. When the direction vector is found, a sample $\mathbf{x}_k$ is assigned to cluster 1 if $\mathbf{a}^t(\mathbf{x}_k - \boldsymbol{\mu}) \geq 0$. Otherwise it is assigned to cluster 2.

Thus, using this algorithm the data set is split by a hyperplane perpendicular to $\mathbf{a}$ which is passing through the mean of the data set.

4.2.1.2 Splitting based on starting samples

This algorithm starts with finding two samples, one "representative" for each (new) clusters. Each unclustered sample is then assigned to one of the clusters, i.e. to the cluster with the nearest mean, the nearest sample etc. The algorithm may be illustrated with the following statements:

```

<Find the starting samples  $\mathbf{x}_{1_1}$  and  $\mathbf{x}_{2_1}$  for  $\mathcal{X}_1$  and  $\mathcal{X}_2$  respectively>;
FOR  $i := 1$  TO  $(N-2)$  DO
BEGIN
    IF  $(f(\mathbf{x}_i, \mathcal{X}_1) < f(\mathbf{x}_i, \mathcal{X}_2))$  THEN  $k:=1$  ELSE  $k:=2$ ;
     $N_k := N_k + 1$ ;
     $\mathcal{X}_k := \mathcal{X}_k \cup \mathbf{x}_i$ ; (* Add the sample to the data set of cluster  $k$ . *)
END;

```

Here, N_k , $k = 1, 2$ is the number of samples in cluster k , and $\mathcal{X}_k = \{\mathbf{x}_{k_1}, \dots, \mathbf{x}_{k_{N_k}}\}$ is the data set of cluster k . $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ denotes the data set being split, and f is the criterion function. We see that the algorithm consists of two parts. First the starting samples have to be determined, and then the rest of the samples are to be assigned to one of the two clusters. We will start with the problems concerning the determination of the starting samples.

4.2.1.2.1 Determination of the starting samples

The performance of the splitting algorithms depends very much on properly chosen starting samples. We are interested in a splitting resulting in a maximum distance from the mean of the data set to the mean of the nearest cluster. Therefore it is reasonable to define two widely separated samples as starting samples and then let the (new) clusters "grow" towards each other. Two possible ways of determining these samples are studied. The first and simplest way of choosing the starting samples is to use the two most distant samples. Let $\mathbf{x}_{1_1}$ and $\mathbf{x}_{2_1}$ denote the starting samples. Then

$$\|\mathbf{x}_{1_1} - \mathbf{x}_{2_1}\| = \max_{i,j} \{\|\mathbf{x}_i - \mathbf{x}_j\|\} \quad , \quad i, j = 1, 2, \dots, N. \quad (4.8)$$

However this method is not robust. If for instance one of the starting samples is an outlier (to be defined in 4.2.1.3) a bad initial splitting (clustering) may occur. Therefore a more robust strategy is proposed. This is to choose the two most distant vectors projected into a vector $\mathbf{a}$. $\mathbf{a}$ is parallel to the direction of maximum variance and is computed according to 4.2.1.1. In other words we choose the starting samples such that

$$\|\mathbf{a}^t(\mathbf{x}_{11} - \mathbf{x}_{21})\| = \max_{i,j} \{\|\mathbf{a}^t(\mathbf{x}_i - \mathbf{x}_j)\|\} \quad , i, j = 1, 2, \dots, N. \quad (4.9)$$

This approach is more robust with respect to outliers because a few outliers have moderate influence on the vector $\mathbf{a}$.

4.2.1.2.2 Splitting of a subclass when the starting samples are given

Let us assume that the starting samples have been found. Now, the problem is to assign the rest of the samples to one of the two clusters. Let us assume that N_1 samples have been assigned to cluster 1 and N_2 to cluster 2 ($N_1 + N_2 < N$). Furthermore, $\mathcal{X}_1$ and $\mathcal{X}_2$ denote the data set of cluster 1 and cluster 2 respectively. As stated earlier, a non-assigned sample is assigned to the cluster producing the smallest value of a given criterion function. In this work six different criterion functions have been studied. With exception of the first one, they are all described by Hand [15] in connection with agglomerative clustering. The criterion functions are as follows:

i) The distance to the starting sample.

$$f_1(\mathbf{x}, \mathcal{X}_k) = \|\mathbf{x} - \mathbf{x}_{k1}\|$$

ii) The distance to the nearest sample.

$$f_2(\mathbf{x}, \mathcal{X}_k) = \min_i \{\|\mathbf{x} - \mathbf{x}_{ki}\|\} \quad , i = 1, \dots, N_k$$

iii) The distance to the furthest sample.

$$f_3(\mathbf{x}, \mathcal{X}_k) = \max_i \{\|\mathbf{x} - \mathbf{x}_{ki}\|\} \quad , i = 1, \dots, N_k$$

iv) The distance to the mean.

$$f_4(\mathbf{x}, \mathcal{X}_k) = \left\| \mathbf{x} - \frac{1}{N_k} \sum_{i=1}^{N_k} \mathbf{x}_{k_i} \right\|$$

v) The distance to the median.

$$f_5(\mathbf{x}, \mathcal{X}_k) = \|\mathbf{x} - \text{Med}(\mathcal{X}_k)\|$$

where the median of the data set is defined as

$$\text{Med}(\mathcal{X}_k) = \begin{pmatrix} \text{Med}(x_{k_1}(1), \dots, x_{k_{N_k}}(1)) \\ \vdots \\ \text{Med}(x_{k_1}(d), \dots, x_{k_{N_k}}(d)) \end{pmatrix}.$$

$\text{Med}(x_{k_1}(i), \dots, x_{k_{N_k}}(i))$ denotes the median of the i 'th element of the vectors in $\mathcal{X}_k$.

vi) The group average distance.

$$f_6(\mathbf{x}, \mathcal{X}_k) = \frac{1}{N_k} \sum_{i=1}^{N_k} \|\mathbf{x} - \mathbf{x}_{k_i}\|$$

The first criterion is equivalent to separate the data set using a hyperplane given by the equation

$$\mathbf{z}^t(\mathbf{x}_{1_1} - \mathbf{x}_{2_1}) - \frac{1}{2}(\mathbf{x}_{1_1} - \mathbf{x}_{2_1})^t(\mathbf{x}_{1_1} - \mathbf{x}_{2_1}) = 0.$$

Hence it should be very clear that its performance depends very much on properly chosen starting samples. When using this criterion it is of great importance to be able to handle problems caused by outliers.

The properties of the other criteria are discussed by Hand [15], and will not be recapitulated here.

4.2.1.3 Outlier handling

Outliers may be considered as “wild” samples, – samples which are far away from the rest of the samples representing the same class. A somewhat diffuse definition may be as follows: Assume a class has samples taken from a distribution with the density $p(\tau)$, and the sample $\mathbf{z}_0$ is to be tested. If $p(\mathbf{z})$ is very small in a neighbourhood around $\mathbf{z}_0$, then $\mathbf{z}_0$ is considered to be an outlier. We stated earlier that use of eq. 4.8 should be avoided because it is very sensitive with respect to outliers. Moreover, outliers do also have influence on all criterion functions mentioned in the last paragraph. By detecting the outliers, we may reduce their influence considerably. This may be achieved by using the following steps:

- a) Detect all outliers in $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$.
- b) Determine the starting samples by using all samples in $\mathcal{X}$ except the outliers.
- c) Assign all non-outliers to one of the two clusters.
- d) Assign all outliers.

A Mahalanobis distance based outlier detector will most likely work well in many practical situations. However, in appendix A an alternative and more robust outlier detector is presented. This detector is used in the following, but using a Mahalanobis distance based detector would probably in most cases give similar results.

4.2.2 Optimization of the initial splitting

The result of the initial, coarse splitting is obviously not optimized with respect to any criterion. Therefore an optimization algorithm should be applied.

Various optimization criteria may be found in Hand [15]. They are all based on the scatter matrices. The within-class scatter matrix is defined as

$$\mathbf{W} = \sum_{k=1}^2 \sum_{i=1}^{N_k} (\mathbf{x}_{k_i} - \boldsymbol{\mu}_k) (\mathbf{x}_{k_i} - \boldsymbol{\mu}_k)^t$$

and the between-class scatter matrix is defined as

$$\mathbf{B} = \sum_{k=1}^2 (\boldsymbol{\mu}_k - \boldsymbol{\mu}) (\boldsymbol{\mu}_k - \boldsymbol{\mu})^t$$

One popular criterion is the trace of the within-class scatter matrix, $\text{tr}(\mathbf{W})$, which is to be minimized. This is a reasonable criterion for several reasons. First of all, $\text{tr}(\mathbf{W})$ is the sum of variances (and the sum of eigenvalues) of $\mathbf{W}$. Minimizing this trace is indicating that a good split is found because the sum of variances is "small", and thus the subclasses are assumed "compact". Secondly, as shown in appendix B, minimizing $\text{tr}(\mathbf{W})$ is the same as maximizing $|(\boldsymbol{\mu}_1 - \boldsymbol{\mu})^t(\boldsymbol{\mu}_2 - \boldsymbol{\mu})|$. This inner product may be viewed as the distance between $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}$ when projected into the vector $(\boldsymbol{\mu}_2 - \boldsymbol{\mu})$ which is also a reasonable criterion to optimize. However, the $\text{tr}(\mathbf{W})$ is little robust. If the data set contains outliers or some of its samples are "sparsely" distributed, then the majority of the samples may end up in one of the clusters only. Furthermore, scaling will have influence on the clustering.

Another popular criterion is the determinant of $\mathbf{W}$. By minimizing $|\mathbf{W}|$ we are in fact minimizing the product of the eigenvalues. Contrary to the $\text{tr}(\mathbf{W})$, scaling have no influence on the $|\mathbf{W}|$. Except for this feature, the disadvantages are the same as for $\text{tr}(\mathbf{W})$. For a more detailed discussion, see Hand [15].

As an alternative to the scatter matrices, first order statistics may be used. A criterion performing well is the distance between the mean of the original data set and the nearest (new) cluster. This criterion, $\min\{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}\|\}$, is to be maximized. Its main advantage that it is robust to outliers. However, scaling will have influence on the criterion.

For some criterion functions (and small data sets) it is possible to split the data set by using a so-called branch and bound algorithm [23]. This algorithm is able to find the optimal splitting of the data set. However, the computation

requirement is great when the number of samples is large. Moreover, if J_k denotes a criterion function computed using k samples from the data set, only criterion functions for which $J_1 < J_2 < \dots < J_N$ can be used. Thus $\text{tr}(\mathbf{W})$ may be used, while $\min\{\|\mu_i - \mu\|\}$ is not applicable. Therefore we have to use an algorithm based on the suboptimal evolutionary search principle rather than the branch and bound algorithm.

The evolutionary search procedure uses the result from 4.2.1 as an initial clustering. Then each sample is considered as a candidate for reallocation. If reallocation results in an improved value of the criterion function, the sample is transferred to the other cluster. Otherwise it remains in its original cluster. One iteration is finished when all samples have been considered. The algorithm terminates if no samples are transferred during an iteration. The principle is illustrated by the following statements:

```

<Generate an initial clustering according to 4.2.1>;
<Compute the initial value of the criterion function>;
REPEAT
    finished:=TRUE;
    FOR i:=1 to N DO
        BEGIN
            <Change the assignment of sample  $x_i$  >;
            <Compute the value of the criterion function>;
            IF <Improved value of the criterion function> THEN
                BEGIN
                    finished:=FALSE;
                    <Update the optimal value of the criterion function>;
                END
            ELSE <Change the assignment of sample  $x_i$  >;
        END;
    UNTIL (finished);

```

4.2.3 Evaluation of the different algorithms

We have made simulations using samples taken from two, three and five dimensional distributions. For each distribution and dimension, we have made simulations using three different sample sizes ($N = 25$, $N = 75$ and $N = 150$). We have used distributions containing outliers, "sparse" distributions and non-symmetrical distributions as well as more "compact" distributions. Briefly speaking, a distribution is considered as "sparse" if the majority of the samples in a data set drawn independently from the given distribution are located near the mean, and the rest of the samples are located both far away each other and the mean. Contrary, in a "compact" distribution *all* samples are located near its mean.

We define

$$\Sigma_1 = \mathbf{I},$$

$$\Sigma_2 = \text{diag}(4, 1) = \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}$$

and

$$\Sigma_3 = \text{diag}(1, 16).$$

The following two-dimensional distributions have been used:

- Two Gaussian distributions with covariance matrices Σ_1 and Σ_2 respectively
- Two Gaussian distributions contaminated with an "outlier distribution".

We have chosen a distribution having the density

$$p(\mathbf{x}) = (1 - \rho)N(\boldsymbol{\mu}, \Sigma_2) + \rho N(\boldsymbol{\mu}, \Sigma_3), \text{ and simulations are carried out for } \rho = 0.10 \text{ and } \rho = 0.25.$$

- Two double exponential distributions with covariance matrices Σ_1 and Σ_2 .
- Two negative exponential distributions with covariance matrices Σ_1 and Σ_2 .
- Rectangular distribution $R(0 : 1, 0 : 1)$.
- Triangular distribution $T(0 : 1, 0 : 1)$.

The triangular distribution used has the density function

$$p(\mathbf{x}) = \begin{cases} 2 & , 0 \leq x_1 \leq 1 \text{ and } 0 \leq x_2 \leq x_1 \\ 0 & , \text{Otherwise.} \end{cases}$$

Furthermore, we define

$$\Sigma_4 = \mathbf{I},$$

$$\Sigma_5 = \text{diag}(4, 1, 1),$$

$$\Sigma_6 = \text{diag}(1, 16, 1),$$

and the following three-dimensional distributions are used:

- Two different Gaussian distributions having covariance matrices Σ_4 and Σ_5 respectively
- Two different Gaussian distributions contaminated with outliers according to the distribution with the density $p(\mathbf{x}) = (1 - \rho)N(\boldsymbol{\mu}, \Sigma_5) + \rho N(\boldsymbol{\mu}, \Sigma_6)$ where $\rho = 0.10$ and $\rho = 0.25$
- Two different double exponential distributions with covariance matrices Σ_4 and Σ_5

- Two different negative exponential distributions with covariance matrices Σ_4 and Σ_5

Finally, we define

$$\Sigma_7 = \mathbf{I},$$

$$\Sigma_8 = \text{diag}(4, 1, 1, 1, 1),$$

$$\Sigma_9 = \text{diag}(4, 4, 4, 1, 1),$$

$$\Sigma_{10} = \text{diag}(4, 4, 1, 1, 1),$$

$$\Sigma_{11} = \text{diag}(1, 1, 16, 1, 1),$$

$$\Sigma_{12} = \text{diag}(1, 1, 16, 16, 16),$$

and simulations have been carried out for the following five-dimensional distributions:

- Three different Gaussian distributions with covariance matrices Σ_7 , Σ_8 and Σ_9 .
- Four different Gaussian distributions contaminated by outliers, with the densities $p_1(\mathbf{x}) = (1 - \rho)N(\boldsymbol{\mu}, \Sigma_8) + \rho N(\boldsymbol{\mu}, \Sigma_{11})$ and $p_2(\mathbf{x}) = (1 - \rho)N(\boldsymbol{\mu}, \Sigma_{10}) + \rho N(\boldsymbol{\mu}, \Sigma_{12})$ where $\rho = 0.10$ and $\rho = 0.25$.
- Three different double exponential distributions having covariance matrices Σ_7 , Σ_8 and Σ_9 .

- Three different negative exponential distributions having covariance matrices Σ_7 , Σ_8 and Σ_9 .

The following three criteria are used for the evaluation:

- a) $\min\{\|\mu_i - \mu\|\}$
- b) $\text{tr}(\mathbf{W})$
- c) $|\mathbf{W}|$

The motivation for using these criteria is found in 4.2.2. Furthermore, the criterion

$$\frac{\min\{N_1, N_2\}}{N}$$

is used for evaluating if there is an unbalanced number of samples in the subclasses.

In the tests involving sample sizes of $N = 25$ and $N = 75$, 1000 replications are used. This is usually sufficient for obtaining reliable statistics. However, only 500 replications are used for $N = 150$, since in this case 1000 replications would require around 60 CPU-hours on our ND-570 computer in order to simulate one distribution. Thus, 2.6 CPU-*months* is needed for simulating all distributions. Even with the reduced number of replications, the CPU-requirement is great, but further reduction is not justified. The reason for this great need of CPU-time, is to be found in the sorting required by the outlier detector.

The amount of data is so large that it is impossible to present all results here. Instead general remarks illustrated with some results are given.

4.2.3.1 Results from the initial splitting

Let us first present some of the simulation results. In table 4.5 the results from the three-dimensional double exponential distribution ($\Sigma = I$ and $N = 75$) are presented. The results from the five dimensional distribution $p(\mathbf{x}) = 0.9N(\boldsymbol{\mu}, \Sigma_8) + 0.1N(\boldsymbol{\mu}, \Sigma_{12})$ are shown in table 4.6. Both these distributions are fairly difficult to split, and hence the performance of different strategies is well illustrated. The tables show the results using the five best splitting methods with and without outlier detection in order to illustrate the difference in performance when using outlier detection. Only the results using the eigenvector based approach for determining the starting samples are shown because it is much more robust than the approach given by eq 4.8.

We first note that the performance of the algorithms is more or less independent of both the number of samples and the feature space dimension. This was expected since there is no mathematical indication that the splitting algorithms depend on the feature space dimension or the number of samples in the data set. The ranking of the various splitting strategies is also almost independent of the sample distribution. Since simulations have been carried out using a wide variety of distributions, it is reasonable to assume these results to have a quite general validity.

Not surprisingly the best results are obtained by using the eigenvector based splitting (E-V-M) from 4.2.1.1. We know that in the Gaussian case the distance between the means of the parent cluster and the nearest (new) cluster is maximized when projected into the direction of maximum variance of the parent cluster. As expected, the trace of the scatter matrix is also substantially reduced. Moreover, the strategy does *not* depend on properly chosen starting samples. The simulation does also show that it is very robust with respect to outliers.

The algorithms outlined in 4.2.1.2 are much less robust with respect to outliers. Therefore, the eigenvector approach for the determination of the starting samples should be used. Furthermore, the results show the performance of the splitting criterions from 4.2.1.2 to be in the following order:

- 1) Nearest starting sample, N-S-S (f_1)
- 2) Furthest sample, F-S (f_3)
- 3) Group average distance, G-A-D (f_6)
- 4) Mean (f_4)
- 5) Median (f_5)
- 6) Nearest sample (f_2)

These results do correspond to the findings of Bayne et al [2] and Jain et al [20]. Bayne et al used Monte Carlo simulations to estimate the percent misclassification of 13 clustering methods for six types of parameterizations of two bivariate normal populations. The methods were compared using probability of misclassification and incidence matrices. Jain et al compared six different hierarchical methods on univariate random data (all samples were taken from the same distribution) with respect to their tendencies to discover false clusters.

We also found that outliers and sparsely distributed samples heavily affected the splitting. If one of these conditions occur, the number of samples in the clusters becomes very skewly distributed. Hence the $\min\{\|\mu_i - \mu\|\}$ criterion signals poor splittings. Examples illustrating these findings are shown in tables 4.5 and 4.6.

Crit func	Sta sam	Out det	$\frac{\min\{N_1, N_2\}}{N}$	$\min\{\ \mu_i - \mu\ \}$	$\frac{1}{N}\text{tr}(\mathbf{W})$	$\frac{1}{N} \mathbf{W} $
E-V-M	-	N	0.464	0.783	2.331	0.456
			0.027	0.076	0.368	0.224
E-V-M	-	Y	0.467	0.778	2.347	0.401
			0.024	0.080	0.373	0.193
N-S-S	(4.25)	N	0.253	0.466	2.407	0.509
			0.137	0.197	0.392	0.259
N-S-S	(4.25)	Y	0.394	0.671	2.356	0.456
			0.072	0.122	0.370	0.223
F-S	(4.25)	N	0.293	0.494	2.459	0.542
			0.129	0.172	0.402	0.272
F-S	(4.25)	Y	0.429	0.646	2.480	0.535
			0.052	0.122	0.420	0.288
G-A-D	(4.25)	N	0.153	0.314	2.470	0.541
			0.127	0.191	0.401	0.271
G-A-D	(4.25)	Y	0.347	0.600	2.377	0.441
			0.091	0.143	0.374	0.216
Mean	(4.25)	N	0.138	0.293	2.457	0.547
			0.123	0.193	0.404	0.278
Mean	(4.25)	Y	0.324	0.570	2.380	0.448
			0.104	0.162	0.376	0.222

Table 4.5: Initial splitting results using 75 samples taken from a three dimensional double exponential distribution ($\Sigma = \mathbf{I}$). Both mean and standard deviation of each criterion are tabulated for each splitting function.

Crit func	Sta sam	Out det	$\frac{\min\{N_1, N_2\}}{N}$	$\min\{\ \mu_i - \mu\ \}$	$\frac{1}{N}\text{tr}(\mathbf{W})$	$\frac{1}{N} \mathbf{W} $
E-V-M	-	N	0.431	1.586	11.897	68.142
			0.056	0.260	3.528	146.473
E-V-M	-	Y	0.453	1.652	11.903	42.490
			0.034	0.228	3.522	79.348
N-S-S	(4.25)	N	0.237	0.997	11.520	50.508
			0.158	0.474	3.088	133.274
N-S-S	(4.25)	Y	0.376	1.413	11.999	53.054
			0.087	0.326	3.491	105.082
F-S	(4.25)	N	0.259	0.996	11.975	60.208
			0.154	0.420	3.265	133.546
F-S	(4.25)	Y	0.416	1.427	12.398	63.225
			0.061	0.297	3.732	124.674
G-A-D	(4.25)	N	0.188	0.852	11.336	38.534
			0.153	0.480	2.873	74.997
G-A-D	(4.25)	Y	0.366	1.366	11.979	44.978
			0.094	0.355	3.497	84.484
Mean	(4.25)	N	0.163	0.783	11.336	37.550
			0.143	0.470	2.825	70.910
Mean	(4.25)	Y	0.321	1.256	12.018	47.999
			0.116	0.424	3.497	100.798

Table 4.6: Initial splitting results using 25 samples taken from a five dimensional Gaussian distribution contaminated with 10% outliers. Both mean and standard deviation of each criterion are tabulated for each splitting function.

4.2.3.2 Results from the optimization

Tables 4.7–4.12 show examples of simulation results obtained by using the various optimization criteria. In tables 4.7–4.9 the results using the three dimensional double exponential distribution is shown. In table 4.7, the $\min\{\|\mu_i - \mu\|\}$ criterion is used, $\text{tr}(\mathbf{W})$ is used in table 4.8, and finally the $|\mathbf{W}|$ criterion is used in table 4.9. In the same way, the results using the five dimensional Gaussian distribution contaminated with 10% outliers are shown in tables 4.10–4.12.

Simulations are made for all three optimization criterions $\text{tr}(\mathbf{W})$, $|\mathbf{W}|$ and $\min\{\mu_i - \mu\}$, and for all combinations of sample sizes, dimensionalities and distributions described previously in this section. Each criterion is tested using various initial splittings. The purpose has been to study:

- Which of the three criterions have the best overall performance,
- if optimization can compensate a bad initial splitting,
- if optimization significantly improves good initial splittings (splittings produced by the eigenvector based method).

For each optimization criterion, the following six different initial splitting strategies are tested.

- Eigenvector method without outlier detection
- Furthest sample method with and without outlier detection
- Group average method with and without outlier detection
- Mean method without outlier detection

The eigenvector method is only tested without using the outlier detector because its performance is almost unaffected by outliers. The mean method is

also used without the outlier detector due to its very poor initial splittings. The two other criteria were tested both with and without the outlier detector.

Also here, we observe that the performance of the algorithms is almost independent of the number of samples used in the data set as well as the sample space dimension.

Moreover, if the data set contains outliers and/or the samples are sparsely distributed, $\text{tr}(\mathbf{W})$ and $|\mathbf{W}|$ often give quite unbalanced subclasses with respect to the number of samples. The high standard deviation of the $\frac{\min\{N_1, N_2\}}{N}$ criterion when using $\text{tr}(\mathbf{W})$ or $|\mathbf{W}|$ also indicates that the splitting performance is very sensitive to the actual data set. Thus the reliability may be poor when using $\text{tr}(\mathbf{W})$ or $|\mathbf{W}|$ as optimization criterions. These effects are avoided by using the $\min\{\|\mu_i - \mu\|\}$ criterion, which seems to have the best overall performance.

Furthermore, the criterion based on first order statistics is found to be little affected by the quality of the initial splitting, and the evaluation criteria indicate good splittings in all simulations. The criteria based on the scatter matrix on the other hand seems to be very sensitive to the initial splitting. These optimization procedures are performing best when using the best initial splitting, and the worst initial splitting strategies produce very poor "optimized" performance. Thus, the $\min\{\|\mu_i - \mu\|\}$ criterion is the only (evaluated) criterion which is able to improve a bad initial splitting significantly.

Finally, we found that the difference between the initial splitting based on the eigenvector and the optimized splitting is only marginal. This should indicate that the eigenvector method produces good splittings, and thus the optimization procedure is hardly needed in a practical situation.

From the discussion and the illustrations, it should be clear that the eigenvector method is able to split a data set satisfactorily. Furthermore, if optimization of the splitting is required, the $\min\{\|\mu_i - \mu\|\}$ criterion should be used.

Ini spl	Out det	$\frac{\min\{N_1, N_2\}}{N}$	$\min\{\ \mu_i - \mu\ \}$	$\frac{1}{N}\text{tr}(\mathbf{W})$	$\frac{1}{N} \mathbf{W} $
E-V-M	N	0.493	0.826	2.337	0.448
		0.001	0.069	0.367	0.219
F-S	Y	0.493	0.822	2.344	0.423
		0.001	0.070	0.368	0.205
G-A-D	Y	0.493	0.824	2.340	0.421
		0.001	0.069	0.366	0.203
F-S	N	0.493	0.819	2.349	0.443
		0.001	0.070	0.371	0.216
G-A-D	N	0.493	0.821	2.345	0.436
		0.001	0.071	0.368	0.211
Mean	N	0.493	0.821	2.346	0.436
		0.001	0.070	0.372	0.211

Table 4.7: Optimized splitting using the optimization criterion based on first order statistics. The data set contains 75 samples taken from a three dimensional double exponential distribution ($\Sigma = \mathbf{I}$). Both mean and standard deviation are tabulated.

Ini spl	Out det	$\frac{\min\{N_1, N_2\}}{N}$	$\min\{\ \mu_i - \mu\ \}$	$\frac{1}{N}\text{tr}(\mathbf{W})$	$\frac{1}{N} \mathbf{W} $
E-V-M	N	0.391	0.696	2.290	0.402
		0.084	0.127	0.357	0.194
F-S	Y	0.389	0.689	2.298	0.394
		0.082	0.122	0.362	0.189
G-A-D	Y	0.374	0.668	2.299	0.392
		0.087	0.132	0.359	0.187
F-S	N	0.330	0.602	2.311	0.419
		0.127	0.189	0.369	0.212
G-A-D	N	0.234	0.453	2.369	0.464
		0.159	0.250	0.399	0.253
Mean	N	0.220	0.431	2.385	0.477
		0.161	0.255	0.404	0.261

Table 4.8: Optimized splitting using the $\text{tr}(\mathbf{W})$ criterion. The data set contains 75 samples taken from a three dimensional double exponential distribution ($\Sigma = \mathbf{I}$).

Ini spl	Out det	$\frac{\min\{N_1, N_2\}}{N}$	$\min\{\ \mu_i - \mu\ \}$	$\frac{1}{N}\text{tr}(\mathbf{W})$	$\frac{1}{N} \mathbf{W} $
E-V-M	N	0.410	0.706	2.326	0.369
		0.067	0.108	0.373	0.176
F-S	Y	0.410	0.689	2.361	0.360
		0.063	0.103	0.377	0.168
G-A-D	Y	0.399	0.677	2.355	0.362
		0.075	0.119	0.374	0.175
F-S	N	0.362	0.625	2.358	0.386
		0.110	0.158	0.381	0.197
G-A-D	N	0.292	0.523	2.382	0.406
		0.143	0.208	0.389	0.220
Mean	N	0.286	0.513	2.385	0.411
		0.145	0.209	0.389	0.223

Table 4.9: Optimized splitting using the $|\mathbf{W}|$ criterion. The data set contains 75 samples taken from a three dimensional double exponential distribution ($\Sigma = \mathbf{I}$).

Ini spl	Out det	$\frac{\min\{N_1, N_2\}}{N}$	$\min\{\ \mu_i - \mu\ \}$	$\frac{1}{N}\text{tr}(\mathbf{W})$	$\frac{1}{N} \mathbf{W} $
E-V-M	N	0.480	1.744	11.946	58.387
		0.003	0.202	3.458	130.897
F-S	Y	0.480	1.739	11.971	49.916
		0.003	0.199	3.455	95.886
G-A-D	Y	0.480	1.741	11.958	47.297
		0.003	0.202	3.445	81.275
F-S	N	0.480	1.728	12.010	54.085
		0.003	0.205	3.472	106.906
G-A-D	N	0.480	1.726	12.018	53.072
		0.003	0.207	3.472	107.789
Mean	N	0.480	1.726	12.017	52.881
		0.003	0.207	3.470	107.678

Table 4.10: Optimized splitting using the optimization criterion based on first order statistics. The data set contains 25 samples taken from a five dimensional Gaussian distribution contaminated with 10% outliers.

Ini spl	Out det	$\frac{\min\{N_1, N_2\}}{N}$	$\min\{\ \mu_i - \mu\ \}$	$\frac{1}{N}\text{tr}(\mathbf{W})$	$\frac{1}{N} \mathbf{W} $
E-V-M	N	0.340	1.382	11.318	38.424
		0.142	0.455	3.019	68.316
F-S	Y	0.355	1.410	11.516	39.444
		0.124	0.406	3.172	69.155
G-A-D	Y	0.347	1.385	11.537	39.322
		0.123	0.415	3.178	71.104
F-S	N	0.236	1.043	11.158	35.633
		0.126	0.551	2.862	63.479
G-A-D	N	0.193	0.903	11.190	35.994
		0.162	0.539	2.852	65.241
Mean	N	0.180	0.857	11.242	37.555
		0.159	0.535	2.876	67.252

Table 4.11: Optimized splitting using the $\text{tr}(\mathbf{W})$ criterion. The data set contains of 25 samples taken from a five dimensional Gaussian distribution contaminated with 10% outliers.

Ini spl	Out det	$\frac{\min\{N_1, N_2\}}{N}$	$\min\{\ \mu_i - \mu\ \}$	$\frac{1}{N}\text{tr}(\mathbf{W})$	$\frac{1}{N} \mathbf{W} $
E-V-M	N	0.397	1.427	12.137	32.396
		0.088	0.314	3.447	55.457
F-S	Y	0.406	1.407	12.404	32.264
		0.071	0.303	3.568	62.226
G-A-D	Y	0.390	1.402	12.217	31.978
		0.080	0.312	3.497	54.032
F-S	N	0.286	1.083	11.879	30.884
		0.164	0.451	3.166	60.341
G-A-D	N	0.226	0.950	11.664	31.273
		0.163	0.481	3.094	57.670
Mean	N	0.219	0.920	11.728	31.486
		0.162	0.481	3.098	57.729

Table 4.12: Optimized splitting using the $|\mathbf{W}|$ criterion. The data set contains of 25 samples taken from a five dimensional Gaussian distribution contaminated with 10% outliers.

4.3 Generating piecewise linear classifiers

Earlier we defined a piecewise linear classifier as a classifier where the hypersurface separating two classes is piecewise linear. In this section we will see how the subclass information may be used for generating a piecewise linear classifier. Each subclass is represented by a data set containing all samples $\mathbf{x}$ for which $\mathbf{x} \in S_i$, and belongs to ω_i . We will use n_i weight vectors for representing ω_i (one weight vector for each subclass). The discriminant function will be defined as

$$g_i(\mathbf{x}) = \max_j \{ \mathbf{a}_{ij}^t \mathbf{y} \} \quad , \quad \begin{array}{l} j = 1, 2, \dots, n_i \\ \mathbf{y}^t = [1, \mathbf{x}^t] \end{array} \quad (4.10)$$

where $\mathbf{a}_{ij}$ is the weight vector of the j 'th subclass in ω_i . Now, $\mathbf{x}$ is assigned to the class for which $g_i(\mathbf{x})$ is maximum. It may be shown that 4.10 is a piecewise linear classifier [5].

In the next four subsections we will study various approaches for generating piecewise linear classifiers based on the subclass information.

4.3.1 The nearest submean classifier

This classifier assigns a sample $\mathbf{x}$ to the class with the nearest subclass. Let the subclasses $\alpha_1, \alpha_2, \dots, \alpha_n$ be defined according to 4.4. Moreover, μ_i denotes the mean (in feature space) of α_i . Then $\mathbf{x}$ is assigned to $\omega(\alpha_k)$ if

$$\|\mathbf{x} - \mu_k\| = \min_i \{ \|\mathbf{x} - \mu_i\| \} \quad , \quad i = 1, 2, \dots, n \quad (4.11)$$

where $\omega(\alpha_k)$ returns the class of α_k . It is easy to verify that the nearest submean classifier (NSMC) is piecewise linear and may be defined in accordance with 4.10 [5]. The weight vectors are then defined as

$$\mathbf{a}_i^t = \left[-\frac{1}{2} \mu_i^t \mu_i, \mu_i^t \right]$$

when using an augmented feature vector representation. Initially, we did not expect this classifier to be able to adapt itself sufficiently to the data set. It is also affected by scaling the axes. Therefore other strategies have been developed.

4.3.2 Piecewise linear classifiers based on the mean squared approach

A more advanced approach is to generate a piecewise linear classifier which is based on mean squared error (MSE) techniques. Also in this case we are defining one weight vector for each subclass, and it is quite easy to generate a classifier using a standard MSE algorithm [39]. First the n subclasses are defined according to 4.4. Then the weight matrix

$$\mathbf{A} = [\mathbf{a}_1 : \mathbf{a}_2 : \mathbf{a}_3 : \cdots : \mathbf{a}_n]$$

is chosen so that the cost function

$$J = \text{tr} \{ (\mathbf{Y}\mathbf{A} - \mathbf{B})^t (\mathbf{Y}\mathbf{A} - \mathbf{B}) \} \quad (4.12)$$

is minimized. The matrix $\mathbf{Y} = [\mathbf{y}_1 : \mathbf{y}_2 : \cdots : \mathbf{y}_N]^t$, $\dim\{\mathbf{Y}\} = N \times d$, contains the samples in the data set. N_i samples represent α_i , and totally there are $N = N_1 + N_2 + \cdots + N_n$ samples. Moreover, the samples from the same subclass are assumed to be contiguously stored in $\mathbf{Y}$. That is; the first N_1 samples belongs to α_1 , the next N_2 samples to α_2 , and so on. $\mathbf{B}$ is a cost matrix which is being considered later. The solution minimizing 4.12 is given as [39]

$$\begin{aligned} \mathbf{A} &= \mathbf{Y}^+ \mathbf{B} \\ &= (\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y}^t \mathbf{B} \end{aligned} \quad (4.13)$$

where “+” denotes the pseudo inverse. The cost matrix $\mathbf{B}$ may be interpreted in two different ways [39]. First it may be associated with the cost function used

for calculating Bayes risk. Secondly it may be interpreted as a set of vertices in an Euclidean space, to which the samples in $\mathbf{Y}$ are mapped. However, in this thesis we will only consider the latter interpretation.

Let

$$\mathbf{B}^t = [\mathbf{B}_1^t : \mathbf{B}_2^t : \dots : \mathbf{B}_n^t]$$

where

$$\mathbf{B}_i = \begin{pmatrix} \lambda_i^t \\ \lambda_i^t \\ \vdots \\ \lambda_i^t \end{pmatrix} \begin{array}{c} \overline{\uparrow} \\ N_i \\ \downarrow \end{array}$$

and

$$\lambda_i = \begin{pmatrix} \lambda_{1i} \\ \lambda_{2i} \\ \vdots \\ \lambda_{ni} \end{pmatrix}.$$

Now, the samples belonging to α_i are attempted mapped into the point λ_{ji} using the weight vector $\mathbf{a}_j$, and the weight vectors in $\mathbf{A}$ are chosen so as to minimize the squared sum of mapping the samples $\mathbf{Y}$ into the vertices given in $\mathbf{B}$. The problem is how to determine the elements in $\mathbf{B}$. Several methods have been suggested [39]. One simple possibility is to select the following cost:

$$\lambda_{ij} = \begin{cases} 1 & , i = j \\ 0 & , \text{otherwise,} \end{cases} \quad (4.14)$$

which means that the samples representing α_i are tried mapped into the point 1 and the rest of the samples into the point 0.

We have also tried to use the distance between the means of the subclasses in order to make a more reliable classifier. One simple approach is to let the samples be mapped into points proporsional to the distance between the subclasses, i.e. we will propose the following risk function:

$$\lambda_{ij} = \begin{cases} -d_{ij} = -\|\mu_i - \mu_j\| & , \omega(\alpha_i) \neq \omega(\alpha_j) \\ 0 & , \text{otherwise.} \end{cases} \quad (4.15)$$

Other and more complex ways of using the distance information for generating the coefficients λ_{ij} have also been studied. However, in all tests the MSE-classifier using 4.14 for generating λ_{ij} have been superior to the distance based ones. Thus, these methods are not considered in the following.

4.3.3 Determination of a piecewise linear classifier using a weighed MSE-approach

When using the mean squared error approach (MSE), one tries to map *all* samples onto a set of points defined by elements in the "risk matrix" **B** for a given subclass α_i . We want to improve the MSE-classifier. Then, it might be useful to reduce the number of samples involved in the determination of each weight vector, or generally to weigh the samples from the different subclasses.

First, it is no reason to use the samples representing α_j , $j \neq i$, $\omega(\alpha_j) = \omega(\alpha_i)$ when determining $\mathbf{a}_i$ (the weight vector representing α_i). Secondly, we will also weight the contribution to the mean squared error from the various subclasses differently. This is done in order to make each weight vector sensitive to selected (sub)classes only.

In the following, we will use **A**, **B** and **Y** as defined in 4.3.2. Moreover, we will define **Z** as

$$\mathbf{Z} = (\mathbf{YA} - \mathbf{B})^t(\mathbf{YA} - \mathbf{B}) .$$

In the MSE-approach described previously, we wanted to minimize

$$J = \text{tr}\{\mathbf{Z}\} = \sum_{k=1}^n Z_{kk} . \quad (4.16)$$

It is easy to see that

$$\min \{\text{tr}\{\mathbf{Z}\}\} = \sum_{k=1}^n \min\{Z_{kk}\} . \quad (4.17)$$

Minimizing 4.16 is thus the same as minimizing each diagonal element in $\mathbf{Z}$ separately. Therefore, in the following we will study the determination of the k 'th weight vector only. Using the standard MSE-procedure, this corresponds to determine the weight vector $\mathbf{a}_k$ minimizing

$$Z_{kk} = \sum_{j=1}^n \sum_{l:\alpha(\mathbf{y}_l)=\alpha_j} \left[\underbrace{\left(\sum_{i=0}^d Y_{li} A_{ik} \right)}_{\mathbf{y}_l^t \mathbf{a}_k} - \lambda_{jk} \right]^2 \quad (4.18)$$

where $\mathbf{y}_l$ is the l 'th sample vector which is stored in row no. l in the sample matrix $\mathbf{Y}$. Furthermore, $\alpha(\mathbf{y}_l)$ returns the subclass which $\mathbf{y}_l$ belongs to. For details about the vector λ_k , see section 4.3.2. Moreover,

$$\mathbf{a}_k = \left[A_{0k} : \dots : A_{dk} \right]^t$$

is chosen in order to minimize the squared sum.

By not including the samples from α_j , $\omega(\alpha_k) = \omega(\alpha_j)$, we are reducing the number of samples which is mapped into the point 0. Thus, we achieve the cost function

$$J_k = \sum_{\substack{j = k \\ j : \omega(\alpha_j) \neq \omega(\alpha_k)}} \sum_{l: \alpha(y_l) = \alpha_j} [y_l^t a_k - \lambda_{jk}]^2. \quad (4.19)$$

This cost function may be generalized by weighting the contribution to the squared error from each subclass. Then we obtain the following cost function

$$J_k = \sum_{i=1}^n \beta_{jk} \sum_{l: \alpha(y_l) = \alpha_j} [y_l^t a_k - \lambda_{jk}]^2, \quad \beta_{jk} \geq 0. \quad (4.20)$$

We can easily see that by selecting $\beta_{jk} = 1 \quad \forall j$ we obtain 4.18 and by defining

$$\beta_{jk} = \begin{cases} 1 & , j = k \text{ or } \omega(\alpha_j) \neq \omega(\alpha_k) \\ 0 & , \text{otherwise} \end{cases}$$

we will obtain 4.19.

We wanted to test whether it was desirable to weigh the squared error contributions from the nearest subclasses more than the most distant ones. This may be obtained by letting β_{jk} be inversely proportional to d_{jk} , and therefore we have

$$\beta_{jk} \leq \beta_{kk} \quad (4.21)$$

which is also in accordance with 4.18 and 4.19. The actual choice of β_{jk} is hardly important. Intuitively, one possibility is to let $\beta_{kk} = 1$, and then in accordance with 4.21 require $0 \leq \beta_{jk} \leq 1$. This requirement can be fulfilled by defining

$$\beta_{jk} = \begin{cases} 1 & , j = k \\ \left(\frac{d'_{jk}}{d'_{kk}} \right)^p & , \omega(\alpha_j) \neq \omega(\alpha_k) \\ 0 & , \text{otherwise} \end{cases} \quad (4.22)$$

where $p \geq 0$ and d'_k is the distance from the mean of α_k to the mean of the nearest subclass representing another class. We see that $p = 0$ results in 4.19. Furthermore, when p is "great", then

$$\beta_{jk} = \begin{cases} 1 & , j = k \text{ or } d'_k = d_{jk} \\ \approx 0 & , \text{otherwise} . \end{cases}$$

We will now derive the weight vectors minimizing 4.20. The gradient of the cost function is

$$\nabla J_k = 2 \left[\sum_{j=1}^n \beta_{jk} \sum_{l: \alpha(y_l) = \alpha_j} (a_k^t y_l - \lambda_{jk}) y_l \right] . \quad (4.23)$$

Now, the weight vector minimizing 4.20, is the one for which $\nabla J_k = 0$. We obtain

$$\underbrace{\left[\sum_{j=1}^n \beta_{jk} \sum_{l: \alpha(y_l) = \alpha_j} y_l y_l^t \right]}_{Y'} a_k = \underbrace{\sum_{j=1}^n \beta_{jk} \lambda_{jk} \sum_{l: \alpha(y_l) = \alpha_j} y_l}_{y'} . \quad (4.24)$$

Thus 4.24 implies

$$a_k = Y'^{-1} y' \quad (4.25)$$

which is the solution to our minimization problem.

4.3.4 Generating a piecewise linear classifier from a set of hyperplanes

As mentioned, in this approach we first define hyperplanes separating some of the (sub)classes. Then the hyperplanes are used for generating a piecewise linear classifier. Three problems have to be solved:

- a) How to select the pairs of (sub)classes to be separated by a hyperplane,
- b) How to generate a linear classifier,
- c) How to use the linear classifiers for generating the piecewise linear classifier.

4.3.4.1 Determining the pairs of subclasses to be separated

Let us first assume the density of each class and the mean of each subclass to be known. Let us also for a while assume $p(x|\omega(\alpha_i))$ to be monotonic along a line going from the mean, μ_i , of the (sub)class (we are only using this assumption for illustration purposes). Now, two (sub)classes, say α_i and α_j , are assumed to need separation if the optimal hypersurface is intersecting the line ℓ_{ij} going from μ_i to μ_j ($\mu_i \neq \mu_j$ is assumed). This proposal may be illustrated with the following statements in "quasi-Pascal". We are also here assuming n subclasses, that ω_i is split into n_i subclasses. Hence $n - n_c$ is the last subclass number belonging to ω_{c-1} . Moreover, let $f(i)$ denote the last subclass(number) belonging to $\omega(\alpha_i)$.

```

FOR i := 1 TO n - nc DO
  BEGIN
    FOR j := f(i) + 1 TO n DO
      BEGIN
        <Compute xij of  $\ell_{ij}$  (if it exists) for which
           $P(\omega(\alpha_i))p(x_{ij}|\omega(\alpha_i)) = P(\omega(\alpha_j))p(x_{ij}|\omega(\alpha_j)) >;$ 
        IF <xij exists> THEN b:=TRUE ELSE b:=FALSE;
        k := 1;
        WHILE ((k ≤ n) AND (b)) DO
          BEGIN
            IF ((k <> i) AND (k <> j)) THEN
              BEGIN
                IF ( $P(\omega(\alpha_k))p(x_{ij}|\omega(\alpha_k)) \geq P(\omega(\alpha_j))p(x_{ij}|\omega(\alpha_j))$ ) THEN
                  b:=FALSE;
              END;
            k := k + 1;
          END;
        IF (b) THEN <Compute a hyperplane separating  $\alpha_i$  and  $\alpha_j$  >;
      END;
    END;
  END;
END;

```

Estimation of the "density" function of the subclass may lead to poor performance because in many cases only a few samples are contained in parts of the sample space.

Alternatively, the concept of the Mahalanobis distance may be used. This distance is defined as

$$D_{\Sigma}(x_1, x_2) = (x_1 - x_2)^t \Sigma^{-1} (x_1 - x_2)$$

where Σ is a positive definite matrix. The decision algorithm given previously has to be slightly modified. First we determine the point x_{ij} on the line ℓ_{ij} for which

$$D_{\Sigma_i}(x_{ij}, \mu_i) = D_{\Sigma_j}(x_{ij}, \mu_j)$$

where Σ_i denotes the covariance matrix of α_i . Then, a hyperplane should be generated if

$$\max_k \{D_{\Sigma_k}(x_{ij}, \mu_k)\} > D_{\Sigma_i}(x_{ij}, \mu_i) \quad , \quad k \neq i, j .$$

It should be noticed that no use of a priori information is shown above. However, this information is used ad hoc by simply weighing the Mahalanobis distance with the a priori probability. If the samples of the subclasses are drawn from Gaussian distributions, then using the Mahalanobis distance in the decision equals the previous strategy. Equal a priori probability for all classes is then assumed.

The Mahalanobis distance strategy does *not* detect that subclasses often have a rather bounded sample space. Therefore, non-interesting subclasses with large variances may cause trouble even when they are located far away from the interesting subclasses. This case is illustrated in figure 4.6 where the problem of deciding if a hyperplane should be generated for separating the subclasses α_2 and α_4 is shown.

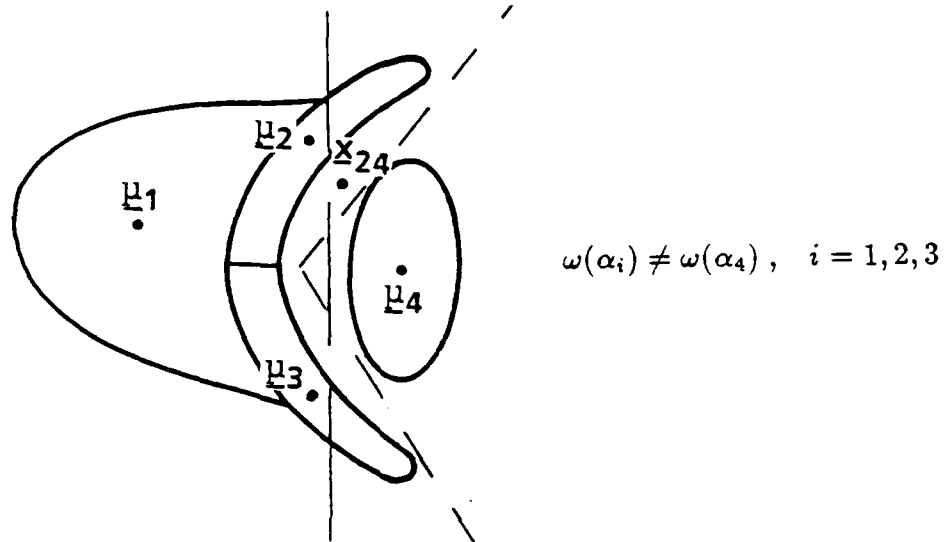


Figure 4.6: An illustration of a situation where the Mahalanobis distance strategy fails. The resulting classifier is based on discriminating α_1 and α_4 only (as illustrated with the solid line). However, a clearly more reliable classifier is obtained if it is based on discriminating both α_2 and α_4 as well as α_3 and α_4 (as shown with the dashed lines).

We see that $D_{\Sigma_1}(x_{24}, \mu_1)$ is less than $D_{\Sigma_2}(x_{24}, \mu_2)$. ($D_{\Sigma_1}(x_{34}, \mu_1)$ is also less than $D_{\Sigma_3}(x_{34}, \mu_3)$.) Hence a very poor decision about which subclasses to be separated will be made, and the resulting classifier will most likely provide a low performance. If we instead decide to combine two hyperplanes; one separating α_2 and α_4 , and one separating α_3 and α_4 , the resulting classifier may be clearly more reliable, as illustrated in the figure. Thus, we will now propose another approach which has been found to work better than the Mahalanobis distance in many examples.

This alternative approach to the Mahalanobis distance is based on the projected (empirical) cumulative distribution. Let σ_i denote the standard deviation of the samples projected into the vector $\mathbf{b}$ (in other words: $\sigma_i = \mathbf{b}^t \Sigma \mathbf{b}$). Assume for a while the samples within a subclass to be Gaussian distributed. The projected cumulative distribution ($y = \mathbf{b}^t \mathbf{x}$) is given as

$$F(\mathbf{b}^t \mathbf{x}) = \int_{-\infty}^{\mathbf{b}^t \mathbf{x}} \frac{1}{\sqrt{2\pi} \sigma_i} e^{-\frac{(z - \mathbf{b}^t \boldsymbol{\mu}_i)^2}{2\sigma_i^2}} dz. \quad (4.26)$$

Since the Mahalanobis distance also may be written as

$$D_{\Sigma}(\mathbf{x}, \boldsymbol{\mu}_i) = \left(\frac{\mathbf{b}^t (\mathbf{x} - \boldsymbol{\mu}_i)}{\sigma_i} \right)^2, \quad (4.27)$$

one may see that there is a connection between the Mahalanobis distance and the projected cumulative distribution. Hence it may be used in the same way as the Mahalanobis distance. However, using the projected cumulative distribution provides a different distance measure. Intuitively, it is more closely related to the actual distribution than the Mahalanobis distance. Thus, it may manage to handle "difficult" situations (e.g. situations where the variance(s) of the subclasses differs much from subclass to subclass) satisfactorily. Contrary to the Mahalanobis distance, it does also detect bounded sample spaces. Hence, it is more or less unaffected by situations similar to figure 4.6. However, one may expect the projected cumulative distribution method to produce somewhat unreliable decisions when only a small number of samples are representing a subclass. Thus, a relatively large design set is necessary if a large number of splittings is required.

The decision algorithm becomes very much similar to the Mahalanobis distance based one. Let us also here assume that α_i and α_j are to be tested. The point $\mathbf{x}_{ij}$ is now defined as the point of equal projected cumulative probability (projected into a vector parallel to the direction of ℓ_{ij}), and it is to be determined first. Next, for each α_k , $k \neq i, j$, we examine the projected cumulative probability (projected into a vector parallel to the line going between $\mathbf{x}_{ij}$ and $\boldsymbol{\mu}_k$) in order to make the decision if a hyperplane is to discriminate α_i and α_j . Even though the subclasses are not strictly defined in

general, we will in the following assume so (i.e. assuming the class densities and the (sub)sample spaces to be known) in order to improve the understanding of the strategy.

Let $p_b(y|\alpha_i)$ denote the "density" given the subclass when projected into the vector b . As mentioned, the point x_{ij} , which is the equal projected cumulative probability point (on ℓ_{ij}), has to be determined first (a solution is assumed to exist). In other words:

$$\int_{b^t x_{ij}}^{\infty} P(\alpha_i) p_b(y|\alpha_j) dy = \int_{-\infty}^{b^t x_{ij}} P(\alpha_j) p_b(y|\alpha_i) dy = P_{ij} \quad (4.28)$$

where $b = \frac{(\mu_i - \mu_j)}{\|\mu_i - \mu_j\|}$. This is illustrated in figure 4.7 for the situation where $P(\alpha_i) = P(\alpha_j)$.

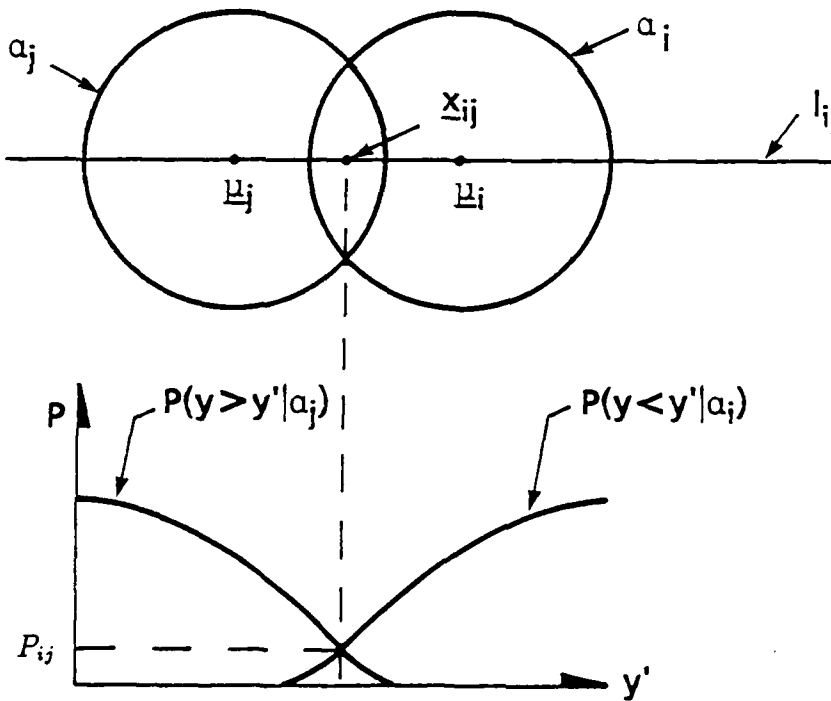


Figure 4.7: Determining x_{ij} using the projected cumulative distribution.

Now, let us define b_k as

$$b_k = \frac{(x_{ij} - \mu_k)}{\|x_{ij} - \mu_k\|}.$$

If

$$\max_k \left\{ \int_{b_k^T x_{ij}}^{\infty} P(\alpha_k) p_{b_k}(y|\alpha_k) dy \right\} \geq P_{ij} \quad k \neq i, j \quad (4.29)$$

then α_i and α_j is assumed not to need separation. In the Mahalanobis distance approach, this corresponds to the situation for which the Mahalanobis distance between μ_k and x_{ij} is less than the Mahalanobis distance between μ_i (or μ_j) and x_{ij} .

The strategy is implemented in the following algorithm:

```

FOR i := 1 TO n - nc DO
BEGIN
  FOR j := f(i) + 1 TO n DO
  BEGIN
    <Compute the point xij of lij (if it exists) according to eq. 4.28>;
    IF <xij exists> THEN b:=TRUE ELSE b:=FALSE;
    k := 1;
    WHILE ((k ≤ n) AND (b)) DO
    BEGIN
      IF ((k <> i) AND (k <> j)) THEN
      BEGIN
        IF <eq. 4.29 holds> THEN b:=FALSE;
      END;
      k := k + 1;
    END;
    IF (b) THEN <Compute a hyperplane separating αi and αj>;
  END;
END;

```

The probabilities used are of course estimated. Let $\mathcal{Y}_i^b = \{y_{i_1} : \dots : y_{i_{N_i}}\}$ denote the data set representing α_i (i.e. $\mathcal{X}_i$) projected into the vector b , and also assume $y_{i_1} < y_{i_2} < \dots < y_{i_{N_i}}$. Then a reasonable estimator for the projected cumulative distribution is

$$\hat{P}_b(y > y' | \alpha_i) = \frac{N_i - l}{N_i} \quad (4.30)$$

where l is the largest integer for which $y_{i_l} < y'$.

This algorithm is quite robust and it has been found to work pretty well. However, it is one situation where it will not work. Assume the subclasses to be well separated. This corresponds to situations for which $y_{i_{N_i}} < y_{j_1}$, where $y_{i_{N_i}} \in \mathcal{Y}_i^b$, $y_{j_1} \in \mathcal{Y}_j^b$, and $b = \frac{\mu_i - \mu_j}{\|\mu_i - \mu_j\|}$. Then we will find that $P_{ij} = 0$, and we only know that x_{ij} should lie between $x_{i_{N_i}}$ and x_{j_1} . The solution to this problem is to use estimates of the density of the subclasses based on the k nearest neighbour method in stead of the projected cumulative distribution. This can be done because for any point x on ℓ_{ij} for which $b^t x < \min\{y_{i_k}\}$ (or $b^t x > \max\{y_{i_k}\}$), the estimate is changing monotonically with $b^t x$.

4.3.4.2 Generating a suitable linear classifier

When we have decided which pairs of subclasses are to be discriminated by a hyperplane, these hyperplanes have to be constructed. During the years, several algorithms have been developed for this purpose [8].

Clark and Gonzalez [4] have recently presented an interesting linear classifier for the two-class problem. Their approach minimizes the number of

misclassifications. Thus, it minimizes the estimate of the error rate (based on the training samples) if $P(\omega_i) = \frac{N_i}{N_1+N_2}$, $i = 1, 2$. However, it would have been preferable to obtain a classifier minimizing the error rate regardless of the a priori probability.

Gallant [14] has proposed an algorithm called the pocket algorithm (PA). This is based on an error correction procedure, and it is minimizing the number of misclassifications when the number of samples reaches infinity. The pocket algorithm derived its name from the process of saving "in your pocket" the weight vector with the longest consecutive run of correct classification trials in the error correction procedure. The classes ω_i and ω_j are to be separated, and the algorithm works as follows:

```

a := 0; (* a is the current weight vector. *)
run_of_corr_class := 0; run_of_corr_class_p := 0; it := 0;
REPEAT
  <Randomly pick a training sample  $x_k$ >; it := it + 1;
  IF <correctly classified> THEN
    BEGIN
      run_of_corr_class := run_of_corr_class + 1;
      IF (run_of_corr_class > run_of_corr_class_p) THEN
        BEGIN
           $a_p := a$ ;
          run_of_corr_class_p := run_of_corr_class;
        END;
      END
    ELSE BEGIN
      IF ( $\omega(x_k) = \omega_i$ ) THEN  $a := a + x_k$ 
      ELSE  $a := a - x_k$ ;
    END;
  UNTIL (it > itmax);

```

Unfortunately the (design set based) error rate estimate is generally not decreasing monotonically as the number of consecutive correctly classified

samples is increasing. Moreover, it does not exist any known bound on the number of iterations needed for producing sufficiently good weight vectors. Furthermore, the error rate estimate is more preferable as minimizing criterion than consecutive the number of correct classification trials. Therefore we modify the pocket algorithm slightly.

It is known that only a finite number of different weight vectors can be reached using the error correction learning [14]. Therefore, by investigating the error rate estimate of the design set (or generally a cost function) instead of the number of consecutive correctly classified samples we are able to minimize the design set based error rate estimate. Moreover, it is easy to see from the algorithm to be presented that the design set based error rate estimate is decreasing monotonically with increasing number of iterations. Thus, the algorithm will always return the best weight vector (the weight vector giving the lowest design set based error rate estimate) of those which have been evaluated. This modified pocket algorithm (MPA) is as follows:

```

a := 0; (* a is the current weight vector. *)
 $\hat{P}_{e_p} := \hat{P}_e := 1$ ; it := 0;
REPEAT
  <Randomly pick a training sample  $\mathbf{x}_k$ >; it := it + 1;
  IF <misclassified> THEN
    BEGIN
      IF ( $\alpha(\mathbf{x}_k) = \alpha_i$ ) THEN a := a +  $\mathbf{x}_k$ 
      ELSE a := a -  $\mathbf{x}_k$ ;
      IF ( $\hat{P}_e < \hat{P}_{e_p}$ ) THEN
        BEGIN
           $\hat{P}_{e_p} := \hat{P}_e$ ;
           $\mathbf{a}_p := \mathbf{a}$ ;
        END;
      END;
    END;
  UNTIL ((it > itmax) OR ( $\hat{P}_{e_p} = 0$ ));

```

The price to pay for this improvement is increased computation time.

As an example, let us compare the PA and MPA using the data set shown in figure 4.8. In this example ω_1 is represented by 150 samples taken from a $N\left(0, \begin{bmatrix} 1 & 0 \\ 0 & 9 \end{bmatrix}\right)$ distribution, and ω_2 is represented by 150 samples taken from a $N\left(\begin{bmatrix} 3 \\ 0 \end{bmatrix}, I\right)$ distribution. In figure 4.9 the error rates (based on the training samples) are plotted as a function of number of iterations in order to show how the classifiers converge.

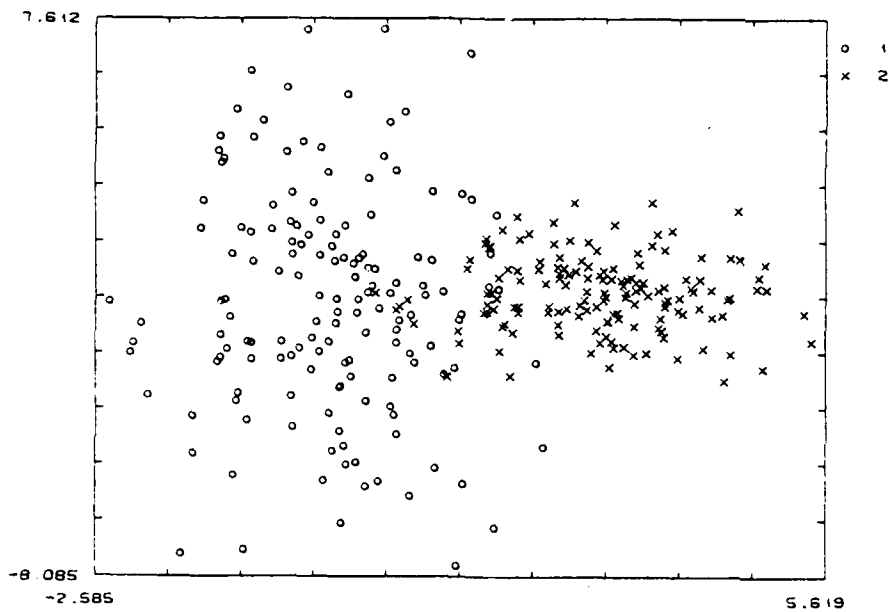


Figure 4.8: The samples representing the classes.

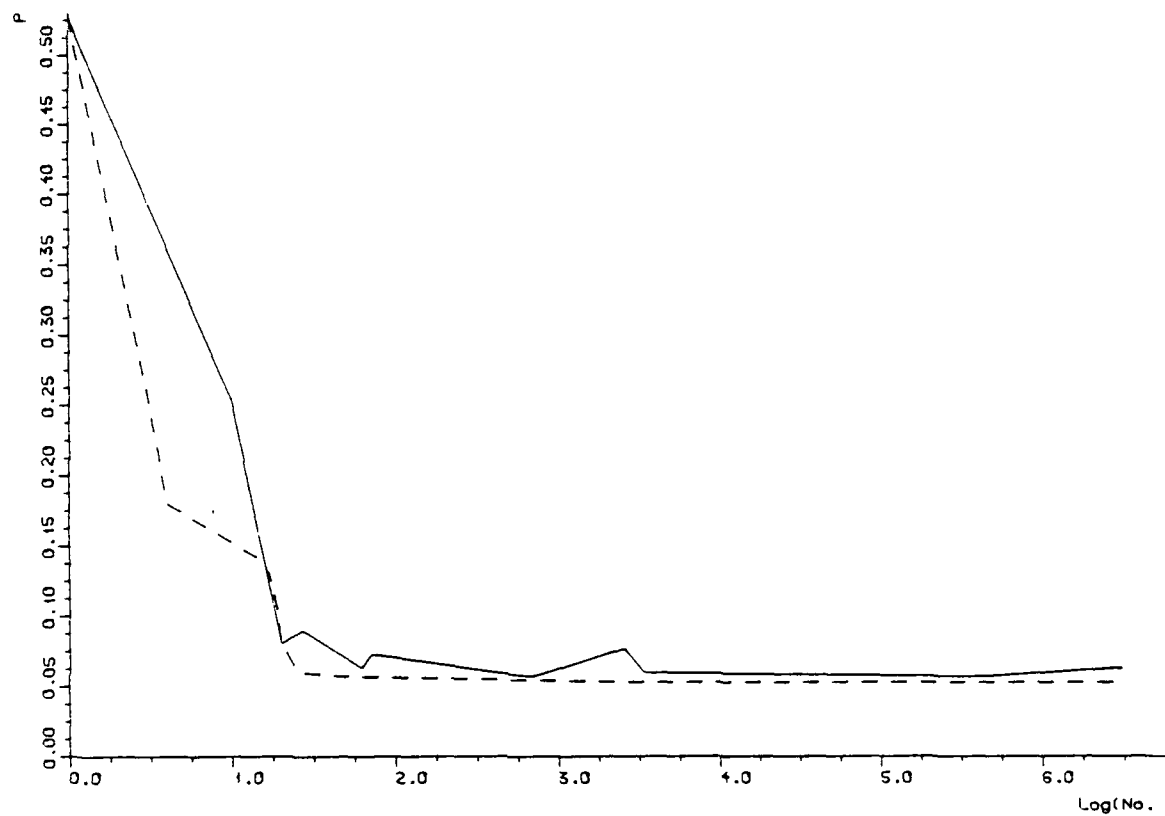


Figure 4.9: The design set based error rate estimate for the PA (solid line) and MPA (dashed line) based on the data set in figure 4.8 as a function of the number of iterations.

As we see from the figure, the error rate of the MPA is decreasing monotonically, and moreover the MPA performs better than the PA.

4.3.4.3 Computing the piecewise linear discriminant function

In 4.3.4.1 and 4.3.4.2 we have described algorithms for finding the pairs of subclasses which have to be separated by hyperplanes, and for generating the actual hyperplanes. Having obtained this knowledge, a piecewise linear classifier is to be generated. Let us assume that m pairs of subclasses should be separated, and that the m hyperplanes have been generated. Furthermore, let $\mathbf{a}'_{i_k j_k}$ define the hyperplane separating the subclasses α_{i_k} and α_{j_k} ($1 \leq i_k \leq n$ and $1 \leq j_k \leq n$). Since $\mathbf{a}'_{i_k j_k}$ is separating these two subclasses, it must be a solution of the equation $\mathbf{a}_{i_k} - \mathbf{a}_{j_k} = \mathbf{a}'_{i_k j_k}$. Now, we want to find the weight vectors $\mathbf{a}_1, \dots, \mathbf{a}_n$ (one for each subclass) by solving the following set of equations

$$\begin{array}{rcl} \mathbf{a}_{i_1} & - & \mathbf{a}_{j_1} = \mathbf{a}'_{i_1 j_1} \\ \mathbf{a}_{i_2} & - & \mathbf{a}_{j_2} = \mathbf{a}'_{i_2 j_2} \\ \vdots & & \vdots \\ \mathbf{a}_{i_m} & - & \mathbf{a}_{j_m} = \mathbf{a}'_{i_m j_m} \end{array} \quad (4.31)$$

Unfortunately, we are not guaranteed a unique solution of 4.31. In fact, we may also either have an infinite number of solutions, or a solution does not have to exist at all. In the first situation, we are interested in finding *one* of the possible solutions, and in the latter one, the best thing to do is to use the solution minimizing the mean squared error. It may be shown that an "optimal" solution (with respect to the least square minimum norm) can be obtained in all these situations by using the singular value decomposition method (SVD) [27]. For details, see appendix C where we have given a brief presentation of the method applied to our problem.

However, difficulties will occur if at least one subclass is *not* involved in the hyperplane separation. The solution to this problem is simply to detect all these subclasses, mark them as passive, and *not* include them in 4.10.

4.3.5 Comparison of the performances of the different classifiers

In 4.1.4 it was argued that the only way of comparing the two suggested splitting strategies is through Monte Carlo simulations. The same argument is valid for the evaluation of the performance of the different classifiers. We have made simulations using the same distributions as in 4.1.4. Moreover, all classifiers derived earlier in 4.3 have been tested, in other words

- The nearest submean classifier (NSMC),
- The mean squared error classifier (MSE),
- 4 classifiers based on the weighed mean squared error approach (W-MSE),
- The hyperplane based classifier (HPC).

Unfortunately it is impossible to present all results from the simulations. Therefore examples will be used to illustrate the main results.

We found the ranking of the classifiers (based on the performance) to be more or less independent of the number of samples in the design set.

First, the W-MSE classifier was evaluated. We have tested 4 different values of the power (p - see 4.22) in order to find how much the contribution to the squared error from each subclass should be weighed. We have made simulations using $p = 0, \frac{1}{2}, 1, 3$ and we found that the best classifier is produced for $p = 0$. Moreover, we found the performance to decrease monotonically with p . This fact indicates that the contribution from the different subclasses α_i for which $\omega(\alpha_i) \neq \omega(\alpha_k)$ should be almost equally weighted when determining $\mathbf{a}_k$. In figure 4.10 these results are shown. The samples representing ω_1 and ω_2 are drawn from Gaussian distributions ($d = 0.5, \gamma = 60^\circ$) in accordance to 4.1.4. Moreover, 25 samples are representing each class.

Next, the best W-MSE classifier ($p = 0$) is tested against the MSE, the NSMC and the HPC. As expected, we found the W-MSE classifier to perform better

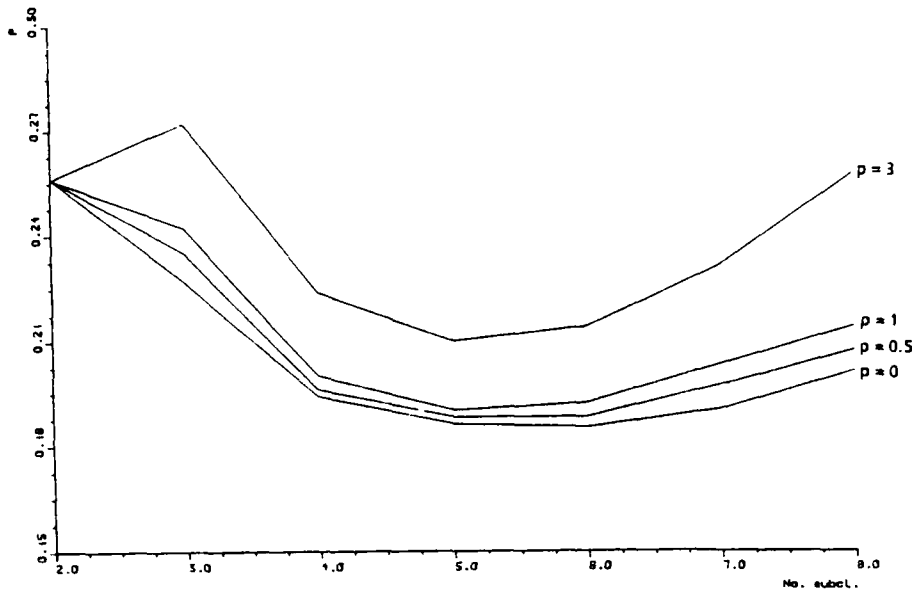


Figure 4.10: Mean performance of the W-MSE classifier as a function of number of subclasses. 25 samples are representing each class and they are drawn from Gaussian distributions ($d = 0.5$ and $\gamma = 60^\circ$).

than the MSE classifier. This is reasonable because all unnecessary contributions to the squared error are removed. However, the mean squared error based approaches do not perform as good as the two others. The simulations show the NSMC (section 4.3.1) to perform quite well, and it adapts quite fast to the data set. The disadvantage is that the classifier is based only on the the mean of the subclasses. Hence, the classifier *may* not be well adjusted to the data set for a given number of subclasses. This fact is easily illustrated by considering the situation where the direction of the vector perpendicular of a hyperplane separating two (sub)classes (e.g. α_i and α_j) satisfactorily, differs significantly from the direction of the vector $\mu_i - \mu_j$.

The HPC (section 4.3.4) is found to do a good job. It adapts nicely to the data set, often using only a few subclasses. However, in situations where the weight vectors cannot be found exactly, we are not guaranteed a well performing classifier (for details see section 4.3.4.3). This effect is seen in the three class tests where the NSMC adapts somewhat faster to the data set than the HPC. However, the HPC is superior to the MSE based classifiers in this case too.

In figures 4.11–4.13 these findings are illustrated. In figure 4.11 the classes are Gaussian distributed ($d = 1.0$, $\gamma = 60^\circ$), and in figure 4.12 the classes are Gaussian/banana distributed using $d = 1.5$. Finally in figure 4.13 the results of the three-class problem using $d = 1.0$ is shown. All plots are based on using 150 samples for representing each class in a given replication.

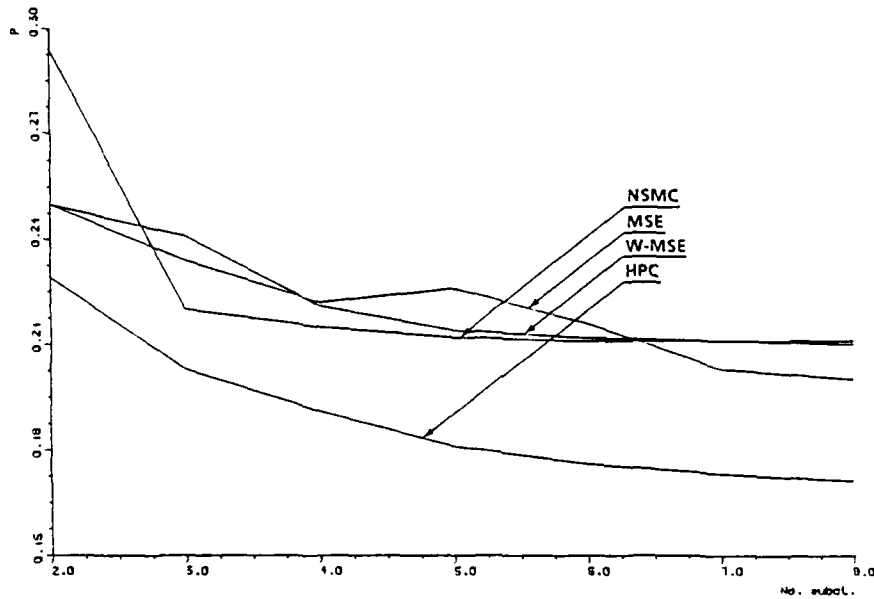


Figure 4.11: Mean performance as a function of number of subclasses using Gaussian distributed samples. 150 samples are representing each class in a given replication, $d = 1.0$ and $\gamma = 60^\circ$.

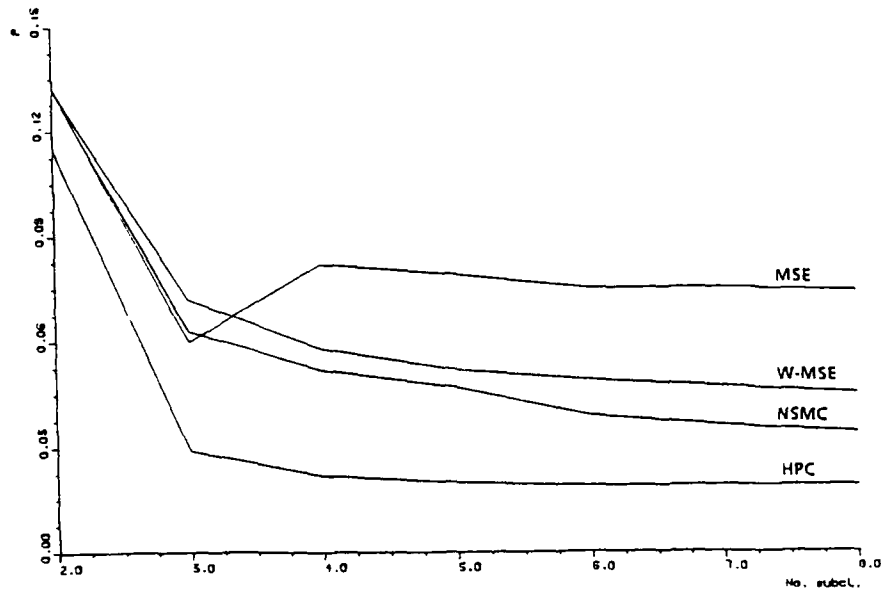


Figure 4.12: Mean performance as a function of number of subclasses using Gaussian and banana distributed samples. 150 samples are representing each class in a given replication and $d = 1.5$.

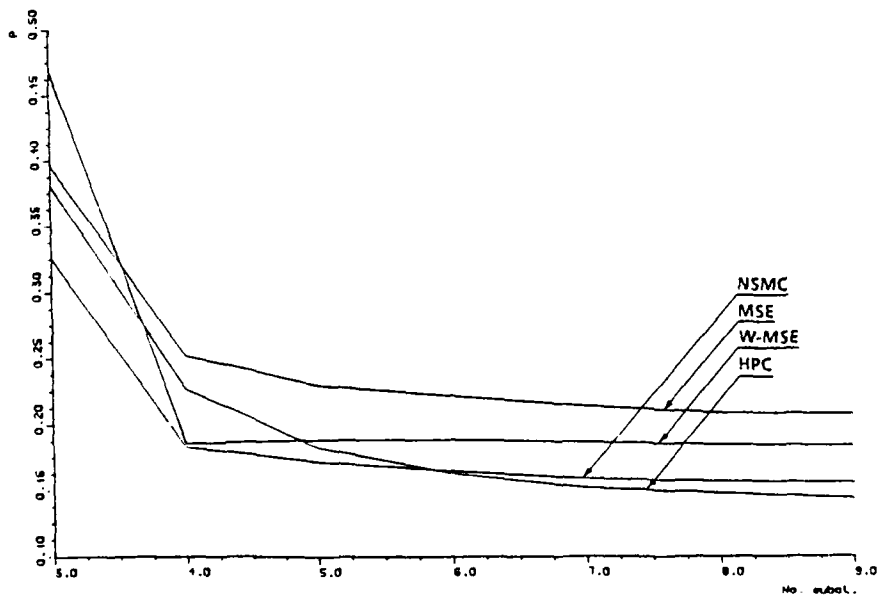


Figure 4.13: Mean performance as a function of number of subclasses in the three-class problem. 150 samples are representing each class in a given replication.

4.4 Determining the final classifier

According to chapter 3, the splitting process is at some time to be terminated. For example, one may stop the splitting when the performance of the classifier is sufficiently good. However, there is one great problem: The performance does not generally increase monotonically as the number of subclasses is increasing. Therefore we have to continue the splitting until either the maximum permitted number of subclasses is reached, or we are convinced that the performance will not be significantly improved by further splitting.

4.4.1 Termination of the splitting

First, the maximum number of subclasses, and hence the maximum number of discrimination functions of the classifier is to be determined. Afterwards, the splitting proceeds until

- a) the maximum number of subclasses is reached,
- b) no subclasses can be split according to a certain criterion,
- c) the greatest contribution from a subclass to the performance is sufficiently small.

When the number of samples in a given subclass is small, it is difficult to do any reliable splitting. Therefore, the second criterion is defined. The third criterion terminates the splitting when it is reasonable to believe that little reduction in the design set based error rate estimate is gained by further splitting.

4.4.2 Weighing of $\hat{P}_e$ and the number of subclasses

In this proposal, we will repeat the splitting until we have at most $n \leq n_{max}$ subclasses. In each iteration a criterion combining the design set based error rate and the number of subclasses, $f(\hat{P}_e(i), i)$ is computed. $\hat{P}_e(i)$ denotes the design set based error rate of the classifier used in iteration $i - c$. The classifier using k subclasses,

$$f(\hat{P}_e(k), k) = \min_i \{f(\hat{P}_e(i), i)\} \quad i = c, \dots, n \quad (4.32)$$

is assumed to be the best classifier to use. Now, the problem is how to choose f . One reasonable strategy is obtained by adding $\hat{P}_e$ and a contribution involving the number of subclasses together. A possible criterion may be as follows:

$$f(\hat{P}_e(i), i) = (\hat{P}_e(i))^q + \left(\frac{i - c + 1}{n_{max} - c}\right)^{1-q}, \quad 0 \leq q \leq 1 \quad (4.33)$$

The disadvantage of 4.33 is its sensitivity of n_{max} . Thus, one is to be careful when using it.

Another strategy is to use

$$f(\hat{P}_e(i), i) = \sqrt[q]{(i - c + 1)} \hat{P}_e(i) \quad , \quad q \geq 1. \quad (4.34)$$

It is also easy to see from 4.34 that the criterion becomes more independent of the number of subclasses when q is increasing. However, q should not be too small. Then this strategy will obviously weight $\hat{P}_e$ too little and the number of subclasses too much. For example, $\hat{P}_e(c) = 1.0$ will give almost the same criterion value as $\hat{P}_e(c + 9) = 0.1$. Eq 4.34 seems to be a reasonable criterion to use. We also see, as a special case, that we will choose the classifier having the smallest design set based error rate if q is large. However it is important to notice that $\hat{P}_e$ is *not* a good error rate estimate. Therefore, it is reasonable to assume that two classifiers, for which $\hat{P}_e(i) \approx \hat{P}_e(j)$ and $\hat{P}_e(i) > \hat{P}_e(j), i < j$, do have almost the same error rate. Hence $\hat{P}_e(i)$ is preferable since $\hat{P}_e(i) \approx \hat{P}_e(j)$ and less subclasses (and thus weight vectors) are involved. This leads us over to another strategy.

4.4.3 Determining a sufficiently good classifier

We are using the performance in the different iterations, in other words $\hat{P}_e(c), \dots, \hat{P}_e(n)$, $n \leq n_{max}$. n is the number of subclasses when the splitting process terminates. As argued previously, it is reasonable to assume a classifier with design set based error rate close to $\min\{\hat{P}_e(i)\}$ as sufficiently adapted to the data set. Therefore we will choose the classifier involving k subclasses where k is the lowest number of subclasses for which

$$\hat{P}_e(k) < \max \left\{ \gamma \min_i \{ \hat{P}_e(i) \}, P_{min} \right\} \quad (4.35)$$

where $i = c, \dots, n$, $P_{min} \geq 0$ and $\gamma \geq 1$. As a special case we see that 4.35 equals 4.34 if $\gamma = 1$, $P_{min} = 0$ and $q = \infty$.

5 EVALUATION OF THE CLASSIFIERS

In the previous chapter a new strategy for generating a multiclass piecewise linear classifier was developed. The evaluation of the classifiers was only based on the adaptability to the data set for a given number of discriminant functions (subclasses). The error rate, computational speed, memory requirement, etc was not considered. However, all these topics are necessary to study, and we also have to compare the classifiers derived in chapter 4 with other classifiers in order to evaluate their performance. Therefore, in this chapter we will compare the classifiers using both real and synthetic data sets. In the tests involving synthetic data, Monte Carlo simulation is used for estimating the various evaluation criteria. The following 10 classifiers are evaluated:

- a) The classifiers derived in 4.31 – 4.34 (NSMC, MSE, W-MSE, HPC)
- b) Bayes classifier with known distribution (wherever it is known!!) (B)
- c) Bayes classifier assuming Gaussian distributions (B-G)
- d) Bayes classifier with probability densities estimated with the k-N-N method (B-kNN)
- e) The nearest neighbour rule (NNR)
- f) The tree classifier of Mizoguchi et al [26] (TC)
- g) The seniority logic committee machine of Lee and Richard [24] (SLCM)

As seen, a wide range of classifiers are chosen including a fast, easy computable and often used classifier (B-G), reliable and complex classifiers (B-kNN and NNR) (fast algorithms for finding the k nearest neighbours are also available, e.g. [22]) as well as “competing” piecewise linear classifiers (TC and SLCM). Thus other interesting classifiers such as various tree classifiers [33, 37, 38], classifiers assuming mixed Gaussian distributions [18, 35] or classifiers based on vector quantization [36] are excluded for several reasons. First of all, the number of classifiers used in the evaluation has to be restricted. Furthermore, some of the interesting classifiers have also been actualized after the initiation of our study.

The evaluation criteria have been chosen as

- a) P_e – the error rate of a classifier using the design set (P_{e_D}) or the test set (P_{e_T}).
- b) $P(e|\omega)$ – the conditional error rates.
- c) The number of discriminant functions (used for NSMC, MSE, W-MSE, HPC, TC, SLCM)
- d) The average number of computed discriminant functions (used for NSMC, MSE, W-MSE, HPC, TC, SLCM)

The reason for using the error rate is obvious. Moreover, it is also of interest to study the conditional error rate. Then we are able to see how well each class is being classified (compared to the optimal classifier wherever it may be found). The two last criterions are for evaluating the classifiers with respect to computational speed and memory requirement of the different piecewise linear classifiers.

In the Monte Carlo simulations we have computed – for each criterion – both the mean criterion value and its standard deviation.

The piecewise linear classifiers derived in [24] and [26] are designed for two class problems only. Therefore, the tests used in this chapter are mainly based on two class problems. Three synthetic data sets (two class problems) used in the tests, consist of samples drawn from Gaussian distributions, Gaussian/banana distributions and banana/banana distributions. Two data sets – both two class problems – involving real data are also used. The first data set is derived from geophysical events [9] and the second data set is derived from the silhouettes of two different cars. Finally, the classifiers derived in this thesis, the Bayes classifier(s) and the nearest neighbour rule are tested on the synthetic three class problem described in 4.1.4 in order to demonstrate the classifiers multiclass properties.

We use the algorithm described in 4.4.3 ($\gamma = 1.2$, $P_{min} = 0.02$) for determining the (final) classifier. As we remember, this strategy is only based on the (design set based) error rate estimate. Thus, we are able to evaluate both the number of discriminant functions needed to obtain a well adapted classifier as well as the error rate.

For each Monte Carlo experiment (each set of distributions), we have made tests using 25 and 150 samples for representing each class (in a replication). Therefore we are able to evaluate the classifiers both for small data sets and for moderate to large data sets. Equal a priori probabilities have been assumed in all experiments. Moreover, 500 replications are used in the simulations involving 150 samples from each class, and 1000 replications when only 25 samples are used in each class.

In the experiment using Gaussian distributions, the samples representing ω_1 is taken from a $N\left(0, \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}\right)$ distribution. The samples from ω_2 is drawn from a $N\left(\begin{bmatrix} 0 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 9 \end{bmatrix}\right)$ distribution. Thus we will have some overlap between the classes. This is an interesting situation for investigating how well a classifier is able to discriminate the classes without being too adaptable to the design set. The results (containing the means and standard deviations of the criteria) from the 150-samples test are shown in table 5.1.

The next experiment consists of Gaussian and banana distributed samples. The samples representing ω_1 are drawn from a banana distribution (parameters: $\mu = 0$, $r_0 = 5$ and $\Theta = 180^\circ$), and the samples from ω_2 are taken from a standard Gaussian distribution (see 4.1.4). This is an interesting situation both because the overlap between the classes is small and because only non-linear surfaces are able to discriminate the classes well. The results when using 150 samples for representing each class in each replication are shown in table 5.2.

In the experiment presented in table 5.3, all samples are taken from banana distributions in order to see how the classifiers handle concave data sets. Here, only 25 samples are representing each class in each replication. The parameters of the distribution of ω_1 are: $\mu = 0$, $r_0 = 5$ and $\Theta = 180^\circ$. The parameters of the distribution of ω_2 are: $\mu = \begin{pmatrix} -3 \\ -5 \end{pmatrix}$, $r_0 = 5$ and $\Theta = 0^\circ$. In this experiment, only classifiers resulting in non-quadratic discrimination surfaces are able to discriminate the classes.

The last synthetic test to be evaluated is the three class problem. The results from the 150 samples experiment are to be found in table 5.4.

Classifier	$P(e \omega_1)$	$P(e \omega_2)$	P_{e_D}	P_{e_T}	No discr func	No comp
NSMC	0.0420	0.2227	0.1253	0.1323	2.92	2.92
	0.0386	0.0447	0.0205	0.0152	1.23	1.23
MSE	0.0465	0.2119	0.1231	0.1292	2.53	2.53
	0.0256	0.0457	0.0210	0.0136	0.72	0.72
W-MSE	0.0365	0.2242	0.1232	0.1303	2.77	2.77
	0.0257	0.0364	0.0195	0.0115	0.86	0.86
HPC	0.0768	0.1985	0.1137	0.1376	2.49	2.49
	0.0418	0.0393	0.0207	0.0118	0.68	0.68
B	0.0566	0.1636	—	0.1101	—	—
	0.0089	0.0137	—	0.0080	—	—
B-G	0.0583	0.1637	0.1100	0.1110	—	—
	0.0133	0.0168	0.0178	0.0080	—	—
B-kNN	0.0572	0.1839	—	0.1206	—	—
	0.0214	0.0222	—	0.0085	—	—
NNR	0.1566	0.1594	—	0.1580	—	—
	0.0258	0.0196	—	0.0131	—	—
TC	0.1623	0.1938	—	0.1781	37.25	4.17
	0.0407	0.1621	—	0.0702	8.99	2.72
SLCM	0.1627	0.1609	—	0.1618	51.75	12.60
	0.0323	0.0231	—	0.0152	7.34	15.75

Table 5.1: Results using Gaussian distributed samples. The samples from ω_1 are taken from a $N(\mathbf{0}, \text{diag}[9,1])$ distribution, and the samples from ω_2 are drawn from a $N([0,4]^t, \text{diag}[1,9])$ distribution. There are 150 samples in each class, and 500 replications are used in the simulations. Both the mean and the standard deviation of each criterion and classifier are shown.

Classifier	$P(e \omega_1)$	$P(e \omega_2)$	P_{eD}	P_{eT}	No discr func	No comp
NSMC	0.0047	0.0225	0.0121	0.0136	3.05	3.05
	0.0040	0.0065	0.0059	0.0033	0.24	0.24
MSE	0.0439	0.0134	0.0254	0.0287	3.12	3.12
	0.0131	0.0095	0.0087	0.0056	0.55	0.55
W-MSE	0.0379	0.0230	0.0268	0.0305	3.12	3.12
	0.0064	0.0127	0.0074	0.0065	0.54	0.54
HPC	0.0133	0.0162	0.0041	0.0148	3.00	3.00
	0.0096	0.0102	0.0051	0.0037	0.00	0.00
B	0.0037	0.0123	—	0.0080	—	—
	0.0022	0.0039	—	0.0023	—	—
B-G	0.0009	0.0382	0.0175	0.0195	—	—
	0.0012	0.0096	0.0061	0.0047	—	—
B-kNN	0.0095	0.0071	—	0.0083	—	—
	0.0042	0.0033	—	0.0024	—	—
NNR	0.0125	0.0117	—	0.0121	—	—
	0.0067	0.0058	—	0.0038	—	—
TC	0.0148	0.0279	—	0.0213	7.15	2.63
	0.0109	0.0675	—	0.0334	5.27	0.74
SLCM	0.0117	0.0257	—	0.0187	6.13	3.15
	0.0083	0.0131	—	0.0066	2.38	1.42

Table 5.2: Results using Gaussian/banana distributed samples. The samples in ω_1 are taken from a banana distribution with parameters $\mu = 0$, $r_0 = 5$, $\Theta = 180^\circ$, and the samples from ω_2 are drawn from a standard Gaussian distribution. There are 150 samples in each class, and 500 replications are used in the simulations.

Classifier	$P(e \omega_1)$	$P(e \omega_2)$	P_{eD}	P_{eT}	No discr func	No comp
NSMC	0.0494	0.0533	0.0227	0.0514	4.86	4.86
	0.0406	0.0400	0.0141	0.0275	1.35	1.35
MSE	0.0933	0.0876	0.0503	0.0904	3.97	3.97
	0.0473	0.0500	0.0255	0.0174	1.88	1.88
W-MSE	0.0844	0.0943	0.0506	0.0894	4.37	4.37
	0.0349	0.0373	0.0244	0.0157	2.33	2.33
HPC	0.0685	0.0886	0.0145	0.0786	3.22	3.22
	0.0529	0.0532	0.0152	0.0320	1.20	1.20
B	0.0119	0.0119	—	0.0119	—	—
	0.0039	0.0039	—	0.0028	—	—
B-G	0.0954	0.0953	0.0813	0.0953	—	—
	0.0301	0.0306	0.0381	0.0113	—	—
B-kNN	0.0460	0.0456	—	0.0458	—	—
	0.0277	0.0272	—	0.0181	—	—
NNR	0.0296	0.0296	—	0.0296	—	—
	0.0194	0.0192	—	0.0101	—	—
TC	0.0902	0.1012	—	0.0957	4.24	1.97
	0.0520	0.0527	—	0.0303	2.55	0.86
SLCM	0.0738	0.0738	—	0.0738	5.04	2.61
	0.0459	0.0451	—	0.0262	1.12	1.36

Table 5.3: Results using banana distributed samples. The samples in ω_1 and ω_2 are drawn from banana distributions with parameters $\mu = 0$, $r_0 = 5$, $\Theta = 180^\circ$ and $\mu = [-3, -5]^t$, $r_0 = 5$, $\Theta = 0^\circ$. There are 25 samples in each class, and 1000 replications are used in the simulations.

Classifier	$P(e \omega_1)$	$P(e \omega_2)$	$P(e \omega_3)$	P_{e_D}	P_{e_T}	No dicr func
NSMC	0.1982	0.1444	0.1870	0.1635	0.1765	6.05
	0.0516	0.0254	0.0390	0.0203	0.0121	1.07
MSE	0.4680	0.2316	0.0937	0.2546	0.2644	4.93
	0.1125	0.1296	0.1045	0.0286	0.0269	1.12
W-MSE	0.1894	0.2235	0.1722	0.1878	0.1984	5.22
	0.0799	0.0466	0.0493	0.0225	0.0199	1.09
HPC	0.2562	0.1625	0.1107	0.1442	0.1765	5.96
	0.0545	0.0610	0.0689	0.0238	0.0194	0.97
B	0.1989	0.1421	0.0993	—	0.1476	—
	0.0144	0.0129	0.0107	—	0.0073	—
B-G	0.3348	0.1248	0.0923	0.1794	0.1840	—
	0.0333	0.0187	0.0156	0.0187	0.0106	—
B-kNN	0.2465	0.1513	0.0851	—	0.1610	—
	0.0288	0.0214	0.0179	—	0.0094	—
NNR	0.2721	0.2027	0.1691	—	0.2145	—
	0.0268	0.0254	0.0240	—	0.0127	—

Table 5.4: Results from the three class problem. For details about the distributions, please see the text. There are 150 samples in each class, and 500 replications are used in the simulations. The two-class classifiers TC and the SLCM are, of course, not included in this test.

The first thing to notice is that the mean squared error and the weighed mean squared error based classifiers generally perform non-satisfactorily. The only exception is the first experiment (concerning Gaussian distributions) where they work well due to the linear classifier's capability of discriminating convex distributions.

However, the NSMC and the HPC perform satisfactorily, and they work equally well as the B-kNN and the NNR. The NNR perform better than the NSMC and the HPC in the experiments where there are little overlap between the classes, and the B-kNN perform slightly better than the NSMC and the HPC in all experiments. These results show that it seems possible to generate reliable classifiers which are computationally simple.

Even though the Gaussian assumption in some experiments is far from fulfilled, the Bayes classifier with Gaussian assumptions works well in all experiments except for the one concerning banana/banana distributions. Furthermore, according to section 2.3, it is almost as fast to compute as our classifiers.

Compared with the TC and the SLCM, our classifiers perform better. When it is some overlap between the classes, such as in the experiments considering samples taken from Gaussian distributions, the TC and the SLCM require relatively large amount of memory and computation time (relative to the NSMC and the HPC) too.

The first data set involving real data, contains 311 seismic events (113 nuclear detonations and 198 earthquakes) recorded at the Large Apparture Seismic Array (LASA) near Billings, Montana. The duration of each signal is 60 seconds. The signals are sampled at 20 Hz. Most of the signal energy which has been found useful for teleseismic discrimination is within the frequency band 0.3 Hz to 5.0 Hz. Therefore, the signals are resampled with 10 Hz after a median filtering of the original time series. Moreover, the signals are also peak to peak scaled and normalized to zero mean.

The total set of events is randomly divided into a design set and a test set. The distribution of events with respect to class and subset is given in table 5.5.

Class	Design set	Test set
Explosion	56	57
Earthquake	93	105

Table 5.5: The number of geoseismic events (samples) in the design set and test set for each class.

For each time series, the autoregressive (AR) coefficients are computed. In other words, we use the coefficients a_j of an m 'th order AR model

$$x_{k+1} = \sum_{j=0}^{m-1} a_j x_{k-j} + e_k$$

where e_k is stochastic and may be viewed as the prediction error. These coefficients are determined by Burgs algorithm which uses the available time serie for minimizing the error power P_m given as:

$$P_m = \sum_{k=1}^{n-m} \hat{e}_k^2$$

The maximum entropy power spectral estimate for real input data is

$$S(f) = \frac{P_m \Delta t}{|1 - \sum_{k=1}^m a_k e^{-j2\pi f k \Delta t}|^2}$$

where Δt denotes the sampling interval. Furthermore, the spectral ratio is defined as

$$SR = \frac{\int_{1.45}^{1.95} \sqrt{S(f)} df}{\int_{0.35}^{0.85} \sqrt{S(f)} df}.$$

In order to use dynamic information, the AR coefficients and the spectral ratio are computed from windows taken at N regular intervals. These spectral ratios and the bodywave magnitude M_b are then combined in the following $N + 1$ dimensional feature vector

$$\mathbf{x} = [\text{SR}_1, \text{SR}_2, \dots, \text{SR}_N, M_b]^t.$$

We have used 3 spectral ratios in the feature vector. Moreover, a two dimensional version of the data set is also classified after applying the Foley-Sammon transform for dimensionality reduction. More details about the signal processing and the computation of the feature vector may be found in [9]. The Foley-Sammon transformed data set is plotted in figure 5.1.

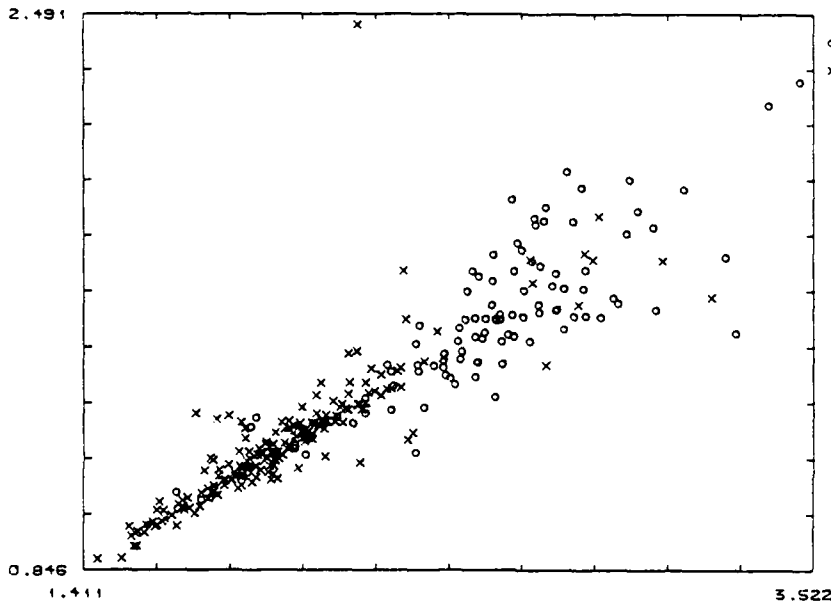


Figure 5.1: Scatter plot of the geoseismic events. Class 1 (o) is explosions and class 2 (x) is earthquakes.

All the different classifiers have been tested on these data sets, and the results using the four dimensional data set are shown in table 5.6. The results using the two dimensional data set are given in table 5.7. In the experiments, the a priori probability $P(\omega_1) = \frac{56}{149}$ is assumed.

These data sets are useful for the evaluation of the classifiers. As can be seen from the plot in figure 5.1, they seem to be rather difficult data sets to classify due to the overlap between the classes. Application of the strategies developed in this thesis shows (with one exception) that a linear classifier is sufficient for the discrimination. Intuitively, from the figure, this is a reasonable decision to make.

Classifier	$P(e \omega_1)$	$P(e \omega_2)$	P_{e_D}	P_{e_T}	No discr func	No comp
NSMC	0.3333	0.0952	0.1611	0.1847	2	2.00
MSE	0.3509	0.0952	0.1208	0.1913	2	2.00
W-MSE	0.2982	0.1238	0.1342	0.1894	2	2.00
HPC	0.2281	0.1905	0.1678	0.2046	2	2.00
B-G	0.3333	0.1238	0.1912	0.2026	-	-
B-kNN	0.2982	0.1238	—	0.1894	-	-
NNR	0.3158	0.1619	—	0.2197	-	-
TC	0.2982	0.2476	—	0.2666	21	4.15
SLCM	0.4035	0.2857	—	0.3300	37	19.85

Table 5.6: Results using data set containing seismic events. The data set with four dimensional feature vectors are used.

Classifier	$P(e \omega_1)$	$P(e \omega_2)$	P_{e_D}	P_{e_T}	No discr func	No comp
NSMC	0.2807	0.1429	0.1342	0.1947	2	2.00
MSE	0.2982	0.0571	0.1275	0.1478	4	4.00
W-MSE	0.2456	0.1333	0.1409	0.1755	2	2.00
HPC	0.3158	0.0606	0.1208	0.1544	2	2.00
B-G	0.2807	0.1048	0.1544	0.1709	-	-
B-kNN	0.2632	0.1905	—	0.2178	-	-
NNR	0.2982	0.1810	—	0.2250	-	-
TC	0.2456	0.2286	—	0.2350	35	4.60
SLCM	0.2456	0.2190	—	0.2290	38	19.26

Table 5.7: Results using data set containing seismic events. The data set with two dimensional (Foley Sammon transformed) feature vectors are used.

We also see that the results in general are better by classifying the two dimensional data set than by classifying the four dimensional data set. This effect probably results from the increased number of parameters which have to be determined. However, there is no (evaluated) classifier that performs significantly better in the four dimensional case than those developed in the last chapter (NSMC, MSE, W-MSE, HPC). Furthermore, compared to the TC and the SLCM, our classifier are much better.

In the two dimensional experiment, all our classifiers except the NSMC give very low error rates. In fact, only the the B-G has comparable results (relative to our classifiers) in this experiment. Due to the heavy overlap between the classes, the NNR and the B-kNN provide a surprisingly high error rate. We also notice that our classifiers requires much less memory than the "competing" piecewise linear classifiers, and that they are significantly faster than the SLCM.

The next and last example to be given is the discrimination of two different cars (Nissan Sunny and Nissan Prairie). An image sequence of these cars has been recorded with a CCD-TV camera. The sequence is digitized using the Teragon 4000 Image Processing computer of NDREs image processing and pattern recognition group. 200 frames are recorded for each car. Then each frame is segmented in a very simple way. First the absolute difference image of the frame and a reference frame is thresholded. A global entropy based thresholding procedure is used for this purpose [21]. Next, noise is removed from the binary image by a 3×3 median filter. Finally the segment(s) are filled by use of a logical filter. This procedure gives a relatively good vehicle silhouette and only a few small noise segments. Therefore, since each frame contains one vehicle only, the largest connected segment in each frame is treated as a vehicle silhouette. In figure 5.2, the segmentation result for frame no 1 is shown. Figure 5.2a shows the the original TV-image, figure 5.2b the difference image, figure 5.2c the thresholded image, and finally in figure 5.2d the median and logical filtered image is shown.

For each segment several moments are computed. The $(p + q)$ 'th central moment , M_{pq} , is defined as

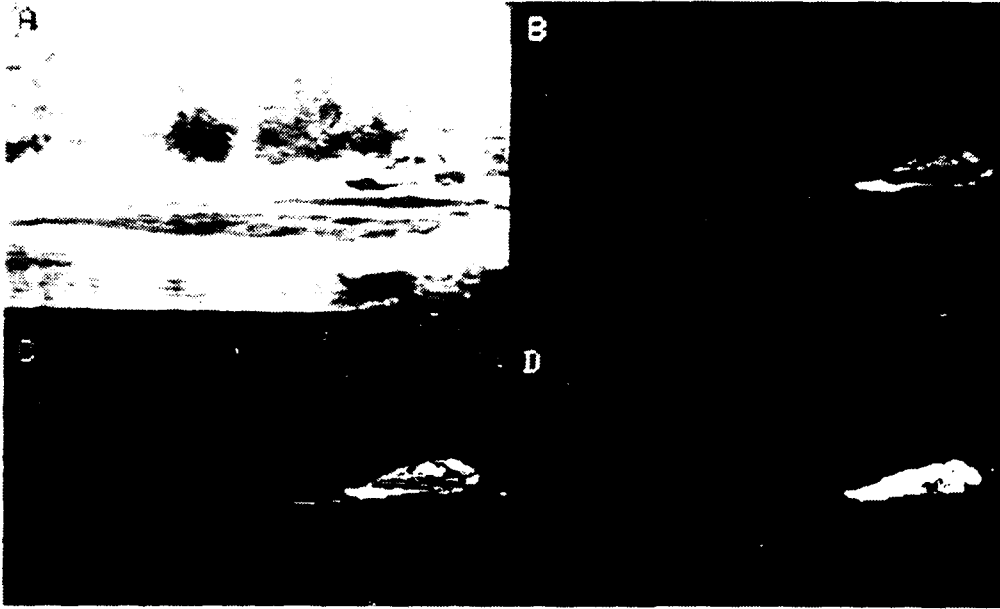


Figure 5.2: Segmentation result for frame no 1 in the Sunny/Prairie example.

$$M_{pq} = \sum_{(x,y) \in S} (x - \bar{x})^p (y - \bar{y})^q$$

where $p > 0$, $q > 0$ and S denotes the set of pixels which defines the segment. Moreover, $(\bar{x}, \bar{y})$ denotes the centroid of the segment. We have computed the moments for which $p + q \leq 3$ only. It is easily seen that M_{pq} is neither rotation invariant nor scale invariant. Scale invariance is obtained by simply dividing M_{pq} by $(M_{00})^{\sqrt{p+q+2}}$. Furthermore, rotation invariance is obtained by rotating the coordinate system an angle α . Here, α denotes the angle between the inertial axis of the segment and the x-axis. For details, see [31]. Now, let S_{pq} denote a scale, rotation and translation invariant moment. The moments S_{20} , S_{02} , S_{21} , S_{12} , S_{30} and S_{03} may now be used in the classification.

Also in this example, the total data set is randomly divided into a design set and a test set. The distribution of samples with respect to class and subset is given in table 5.8

Class	Design set	Test set
Sunny	95	105
Prairie	91	109

Table 5.8: The number of cars (samples) in the design set and test set for each class.

All feature combinations are tested using the nearest neighbour rule in order to find well performing feature candidates. The classification system IPACS [10] is used for this task. The moments S_{20} , S_{21} and S_{12} in conjunction with the Folley-Sammon transform [13] have been found to be the best feature combination (the combination with the smallest error rate estimate using the N-N-R). In figure 5.3 the data set is shown, and in table 5.9 the results using the different classifiers are given.

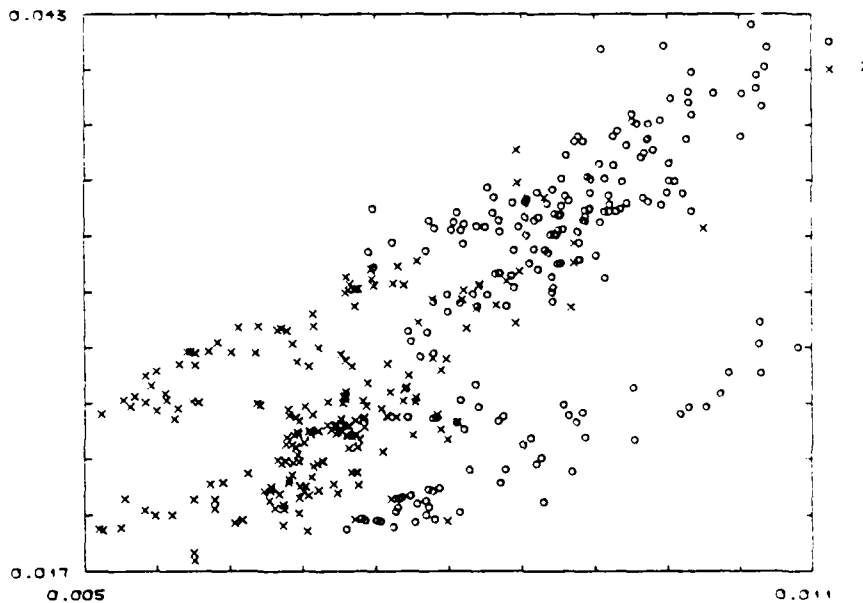


Figure 5.3: Scatter plot of the car discrimination problem. Class 1 (o) is the Sunny and class 2 (x) is Prairie.

All classifiers, except the NSMC and the SLCM, managed to discriminate the two cars satisfactorily. The results obtained by using the HPC and the W-MSE are encouraging. In fact, these classifiers produce the best classification result for this particular data set.

Classifier	$P(e \omega_1)$	$P(e \omega_2)$	P_{e_D}	P_{e_T}	No discr func	No comp
NSMC	0.3238	0.0734	0.1708	0.1986	5	5.00
MSE	0.1619	0.1193	0.1342	0.1406	2	2.00
W-MSE	0.0762	0.1284	0.0980	0.1023	4	4.00
HPC	0.1619	0.0275	0.1389	0.0947	8	8.00
B-G	0.1238	0.1009	0.1129	0.1124	—	—
B-kNN	0.2095	0.0367	—	0.1231	—	—
NNR	0.1429	0.0826	—	0.1128	—	—
TC	0.1048	0.1835	—	0.1442	39	5.08
SLCM	0.2381	0.1376	—	0.1879	52	26.04

Table 5.9: Classification results using the data set containing features derived from the silhouettes of two different cars.

To summarize, the majority of the strategies proposed in the previous chapter produce good classification results in all experiments. Thus, we have succeeded in designing fast and reliable classifiers with performance comparable with more complex ones.

However, we notice that the MSE-based classifiers show an inconvenient behaviour. They both work well only in situations where a single hyperplane is sufficient to discriminate between the classes. In other situations, at least one of them is not useful.

Moreover, the NSMC and the HPC have much better performance than the TC and the SLCM, and usually only a small number of discriminant functions is required for obtaining a reliable classifier. Furthermore, the HPC seems to be slightly better than the NSMC, but in most cases there is no significant difference. However, in a few situations the classifiers do not perform well. The NSMC may become unreliable if the vector perpendicular to a suitable hyperplane (separating α_i and α_j) differs significantly from the direction of $(\mu_i - \mu_j)$. It is also affected by scaling of the axes. Furthermore, the linear classifier used in the HPC occasionally adapts too well to the data set. This may produce an unreliable classifier when only a few training samples are available.

Even though the Gaussian assumption often is far from fulfilled, the Bayes classifier with Gaussian assumptions (B-G) performs well in all situations where hyperquadratic surfaces are feasible for class separation. Thus, in many cases this classifier is an alternative to the NSMC and the HPC. Furthermore, this result indicates that classifiers based on mixtures of Gaussian distributions also will show a good adaptability. However, they are most likely more demanding concerning computational power than linear, quadratic or piecewise linear classifiers.

The NNR and the B-kNN perform somewhat better than our classifiers in all examples except in situations with large class overlap. Unfortunately, they are computationally "heavy", and since they need the data set stored, their memory requirements may cause problems in some applications.

6 SUMMARY AND CONCLUSION

In this thesis we have developed a new strategy for generating a piecewise linear classifier. Our goal have been to find classifier(s) which combines high reliability with fast computational speed. Generally speaking, the algorithm is as follows: First the algorithm attempts to discriminate the classes by using a linear classifier. If this classifier shows poor performance, then the sample space of one of the classes is split into two disjoint subsample spaces. All samples in a design set from a given class lying in the same (sub)sample space are said to belong to the same (sub)class. In a practical situation the splitting is a clustering process in which the set of samples representing a given (sub)class are separated into two clusters. Next, the (sub)class information is used for generating a piecewise linear classifier, and its performance evaluated. If the performance is unsatisfactorily, the sample space of one of the (sub)classes is split and a new classifier generated and evaluated. This process continues until either a maximum number of subclasses is reached, further splitting is impossible, or the performance is expected *not* to be significantly improved by further splitting.

Several problems had to be solved. First we had to develop an algorithm for determining a suitable subsample space to split. Two different strategies have been developed and evaluated. The first (and best!) one is a hillclimbing approach which first evaluates all possible (sub)sample spaces to split, and then selects the best one for splitting. In the other approach, the sample space of the (sub)class contributing most to the design set based estimate of the error rate is split. Not surprisingly, the hillclimbing approach perform much better than this contribution based splitting. The reason is that the hillclimbing approach evaluates all possible splits in a given iteration.

Several splitting algorithms have been developed and tested. We found that the best strategy is to first split the data set representing a given (sub)class in the direction of maximum variance and then maximize the distance from the mean of the data set of the original (sub)class to the mean of the nearest new subclass by using an algorithm based on the principle of evolutionary search.

Given a set of (sub)classes, four different ways of generating a piecewise linear classifier have been developed and studied. The simplest one is the nearest sub-mean classifier (NSMC). This classifier assigns a sample to the class with the nearest (sub)mean. Two other strategies, based on the mean squared error approach, have in many situations been discarded due to poor performance of the corresponding classifiers. The last strategy (the HPC) is more complicated to design. In this approach, we first have to find the pairs of (sub)classes which are to be discriminated by a hyperplane. Next, the hyperplanes are generated, and finally used for computation of the piecewise linear classifier. This classifier as well as the NSMC have been found to perform well in most situations.

The evaluation of the classifiers is based on both synthetic and real data. Monte Carlo simulation techniques have been used in the experiments involving synthetic data.

Our classifiers have also been compared with other classifiers such as Bayes classifier (wherever the probabilities are known), Bayes classifier with estimated densities (based on the k-NN approach), Bayes classifier with assumed Gaussian distributions, the nearest neighbour rule, as well as with two piecewise linear classifiers designed for the two-class problem.

The HPC performs well in all tests. It adapts itself to the data set using only a small number of discriminant functions. However, it seems that the linear classifier used, adapts "too well" to the design set especially when only a small number of samples are available for the training. This may cause a high error rate. Thus it seems that the error rate estimate (based on the design set) is not a sufficiently good optimizing criterion. However, to this authors knowledge, no linear classifier is available which performs well in all situations, but this problem will be a topic for a future study. Also the NSMC shows a good performance in almost all tests. Except for the car discrimination example, it manages to adapt itself to the data set using only a small number of discriminant functions. However, we have seen that this classifier may have problems with the adaptability if the direction of the vector perpendicular to the hyperplane separating two subclasses (e.g. α_i and α_j) differs significantly from the direction of $\mu_i - \mu_j$. These findings are contrary of the results obtained for the other two-class piecewise linear classifiers. These classifiers seem to be quite complicated when there are some overlap between the classes. In some situations our classifiers even show a better performance than the computationally heavy nearest neighbour rule. Moreover, Bayes classifier with estimated densities is better in most situations, but the difference is not large. These results encourage further studies of the HPC and the NSMC classifiers. Topics for such investigation may be comparisons with neural net classifiers, as well as with classifiers based on mixtures of Gaussian distributions.

Acknowledgments

The work presented in this report is my dissertation for the doctor scientiarum degree at the University of Oslo, and several persons involved in this project are

to be thanked. First of all, I would like to express my utmost gratitude to my advisors, senior scientist Idar Dyrdal and Professor Dr Philos Knut Liestøl. Their judicious guidance and suggestions have been invaluable. Furthermore, I wish to extend thanks to my colleagues Jan Petter Fjellanger, Stein Grinaker and Eilert Heyerdahl for fruitful comments and discussions. Last but not least, appreciation is also extended to the rest of the image processing and pattern recognition group at the NDRE for their patience during my one CPU-year simulation study.

References

- [1] Andrews D F, Bickel P J, Hampel F R, Huber P J, Rogers W H, Tukey J W (1972): Robust estimates of location, Survey and advances, Princeton University Press
- [2] Bayne C K, Beauchamp J J, Begovich C L, Kane V E (1980): Monte Carlo studies in clustering procedures, Pattern Recognition, **12**, pp 51-62
- [3] Chang C L (1973): Pattern recognition by piecewise linear discriminant functions, IEEE Trans Comp, **C-22**, pp 859-862
- [4] Clark D C, Gonzalez R C (1984): Optimal solution of linear inequalities with applications to pattern recognition, IEEE Trans Patt Anal and Mach Int, **PAMI-3**, pp 643-655
- [5] Devijver P A, Kittler J (1982): Pattern Recognition: A statistical approach, Prentice/Hall International.
- [6] Dubes R, Jain A K (1979): Validity studies in clustering methodologies, Pattern Recognition, **11**, pp 235-254
- [7] Duda R O, Fossum H (1966): Pattern classification by iteratively determined linear and piecewise linear discriminant functions, IEEE Trans Electron Comp, **EC-15**, no 2, pp 220-232
- [8] Duda R O, Hart P E (1973): Pattern classification and scene analysis, John Wiley and Sons

- [9] Dyrdal I (1987): Teleseismic discrimination of earthquakes and nuclear detonations with features derived from maximum entropy power spectral estimates, *Int Journal of Pattern Recognition and Artificial Intelligence*, 1, no 3-4, pp 323-333
- [10] Dyrdal I (1988): Introduction to IPACS - Interactive Pattern Analysis and Classification System, FFI/RAPPORT-88/4019, Norwegian Defence Research Establishment.
- [11] Fenstad G U, Kjærnes M, Walløe L (1980): Robust estimation of standard deviation, *J Scand Comput Simul*, 10, pp 113-132
- [12] Fischer R A (1936): The use of multiple measurements in taxonomic problems, *Annal of Eugenics*, 7, pp 179-188
- [13] Foley D H, Sammon J W (1975): An optimal set of discriminant vectors, *IEEE Trans Comp C-24*, pp 281-289
- [14] Gallant S I (1986): Optimal Linear Discriminants, *Proc Eight Int Conf on Pattern Recognition*, pp 849-852
- [15] Hand D J (1981): *Discrimination and Classification*, John Wiley and Sons
- [16] Hand D J (1986): Recent advances in error rate estimation, *Pattern Recognition Letters*, 4, pp 335-346
- [17] Hand D J (1986): An optimal error rate estimator based on average conditional error rate: Asymptotic results, *Pattern Recognition Letters*, 4, pp 347-350
- [18] Hjort N L (1986): Notes on the theory of statistical symbol recognition, Research report no 778, Norwegian Computing Centre
- [19] Hoffman R H, Moe L M (1969): Sequential algorithm for the design of piecewise linear classifiers, *IEEE Trans Sys Sciences and Cybernetics*, 5, pp 166-168
- [20] Jain N C, Indrayan A, Goel L R (1986): Monte Carlo comparison of six hierarchical clustering methods of random data, *Pattern Recognition*, 19, pp 95-99

- [21] Kapur J N, Sahoo P K, Wong A K C (1985): A new method for gray-level picture thresholding using the entropy of the histogram, *Comp Graph Image Proc*, **29**, pp 273-285
- [22] Kim B S, Park S B (1986): A fast k nearest neighbour finding algorithm based on the ordered partition, *IEEE Trans Patt Anal and Mach Int*, **PAMI-8**, pp 761-766
- [23] Koontz W L G, Patrenahalli M N, Fukunaga K (1975): A branch and bound clustering algorithm, *IEEE Trans Comp*, **C-24**, pp 908-915
- [24] Lee T, Richards J A (1984): Picewise linear classification using seniority logic committee methods, with application to remote sensing, *Pattern Recognition*, **17**, pp 453-464
- [25] Mangasarian O L (1968): Multisurface method of pattern separation, *IEEE Trans Inf The*, **IT-14**, pp 801-807
- [26] Mizoguchi R, Shimura M, Kakusho O (1980): A new algorithm for construcing piecewise linear discriminant functions, *Proc Fifth Int Conf on Pattern Recognition*, pp 666-670
- [27] Mæland E (1986): Seismic data processing, University of Bergen (in norwegian)
- [28] Palm H C, Grinaker S (1988): Parralisation of algorithms for image processing and pattern recognition FFI/NOTAT-88/4022, The Norwegian Defence Resarch Establishment (in norwegian)
- [29] Palm H C (1988): A new piecewise linear classifier, *Proc conf on image analysis and pattern recognition*, Report no 818, Norwegian Computing Center (in norwegian)
- [30] Rahbar R, Mix D F (1980): Pattern recognition based on piecewise linear or quadratic discriminant functions, *Proc Fifth Int Conf on Pattern Recognition*, pp 674-676
- [31] Rosenfeld A, Kak A C (1982): Digital picture processing, Academic Press

- [32] Selim S Z, Ismail M A (1984): K-means-type algorithms: A generalized convergence theorem and characterization of local optimality, IEEE Trans Patt Anal and Mach Int, **PAMI-6**, pp 81-87
- [33] Shi Q Y, Fu K S (1983): A method for the design of binary tree classifiers, Pattern Recognition, **16**, pp 593-603
- [34] Takiyama R (1978): A general method for training the committee machine, Pattern Recognition, **10**, pp 255-259
- [35] Taxt T, Eikvil L, Hjort N L (1989): Statistical classification using mixtures of multinormal densities, Proc Conf on Pattern Recognition, Paris
- [36] Therrien C W (1989): Decision estimation and classification John Wiley and Sons
- [37] Wang Q R, Suen S Y (1984): Analysis and design of decision trees based on entropy reduction and its application to large character set recognition, IEEE Trans Patt Anal and Mach Int, **PAMI-6**, pp 406-417
- [38] Wang Q R (1987): A flexible tree design in an edit-partition scheme, Pattern Recognition Letters, **5**, pp 261-265
- [39] Wee W G (1968): Generalized inverse approach to adaptive multiclass pattern classification, IEEE Trans Comp, **C-17**, pp 1157-1164

A: A ROBUST OUTLIER DETECTOR

Commonly used outlier detectors is based on the Mahalanobis distance between a sample $\mathbf{z}$ and the centroid $\boldsymbol{\mu}$ (the expected value) of a class ω . The Mahalanobis distance $D_{\Sigma}(\mathbf{z}, \boldsymbol{\mu})$ is given as

$$D_{\Sigma}^2(\mathbf{z}, \boldsymbol{\mu}) = (\mathbf{z} - \boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1} (\mathbf{z} - \boldsymbol{\mu}). \quad (\text{A:.1})$$

Now, $\mathbf{z}$ is considered as an outlier if

$$D_{\Sigma}(\mathbf{z}, \boldsymbol{\mu}) > d. \quad (\text{A:.2})$$

In other words; if $\mathbf{z}$ is outside the $d\sigma$ ellipsoid, i.e. if there are more than d standard deviations between $\mathbf{z}$ and $\boldsymbol{\mu}$, then $\mathbf{z}$ is assumed to be an outlier.

Usually, $\boldsymbol{\mu}$ and Σ are unknown. Therefore, they need to be estimated. Given a data set $\mathcal{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_n\}$, they can be estimated the usual way, i.e.

$$\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^n \mathbf{z}_i \quad (\text{A:.3})$$

$$\hat{\Sigma} = \frac{1}{n-1} \sum_{i=1}^n (\mathbf{z}_i - \boldsymbol{\mu})(\mathbf{z}_i - \boldsymbol{\mu})^t \quad (\text{A:.4})$$

However, A:.3 and A:.4 are known to be little robust. Therefore, we will try to find a more robust way of detecting outliers.

Our outlier detector is based on projecting of a sample into a one dimensional space. First, let us as an alternative to A:.1, project the samples in $\mathcal{Z}$ into the vector $\mathbf{a} = (\mathbf{z} - \boldsymbol{\mu})$. Moreover, let σ denote the standard deviation of the distribution of the projected sample(s). Furthermore, y is defined as $y = \mathbf{a}^t \mathbf{z}$. Then $\mathbf{z}$ is considered as an outlier if

$$D_{\Sigma}^2(\mathbf{z}, \boldsymbol{\mu}) = \left(\frac{y}{\sigma}\right)^2 = \frac{(\mathbf{a}^t \mathbf{z})^2}{\mathbf{a}^t \Sigma \mathbf{a}} > d^2 \quad (\text{A:.5})$$

A robust estimate of σ may be easily obtained. Fenstad et al [11] have studied several standard deviation estimators, and they concluded that a quartile based estimator is very robust. This result leads us to a method for one dimensional outlier detection (and rejection) given in [1] which has been found to work out well. This algorithm is based on the interquartile difference, and we may easily adapt it for our d dimensional outlier detection.

Let $\mathbf{z}$ denote the sample to be tested using the data set $\mathcal{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_n\}$.

Moreover, let us define $\mathbf{a} = (\mathbf{z} - \boldsymbol{\mu})$, $y_i = \mathbf{a}^t \mathbf{z}_i$ and the data set $\mathcal{Y} = \{y_1, \dots, y_n\}$.

Furthermore, let H_1 and H_2 denote the lower and upper quartile of $\mathcal{Y}$ respectively. Now, if

$$H_1 - c(H_2 - H_1) \leq \mathbf{a}^t \mathbf{z} \leq H_2 + c(H_2 - H_1) \quad (\text{A:.6})$$

then $\mathbf{z}$ is *not* considered as an outlier.

In the simulations presented in 4.2 we used $c = 1.5$, and it is shown that A:.6 manage to handle the outlier problem.

Finally, let us illustrate our algorithm using 100 samples taken from bivariate Gaussian distribution. The outlier rejection boundary is computed and the result is shown in figure A:.1 ($c = 1.5$).

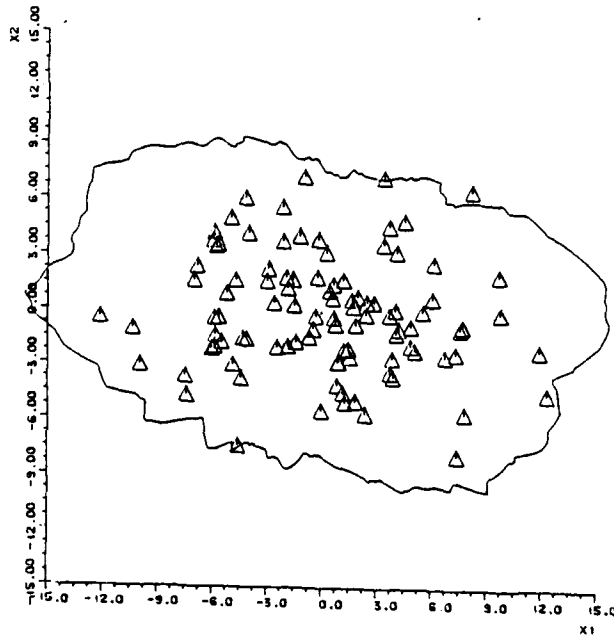


Figure A:.1: The outlier rejection boundary computed from 100 samples taken from a bivariate Gaussian distribution.

B: THE CONNECTION BETWEEN $(\mu_1 - \mu)^t(\mu_2 - \mu)$ AND $\text{tr}\{\mathbf{W}\}$

Let us first define the within-class scatter matrix ($\mathbf{W}$), the between-class scatter matrix ($\mathbf{B}$) and the (total) scatter matrix ($\mathbf{T}$) the usual way

$$\mathbf{W} = \sum_{i=1}^2 \sum_{j=1}^{N_i} (\mathbf{z}_{ij} - \mu_i)(\mathbf{z}_{ij} - \mu_i)^t,$$

$$\mathbf{B} = \sum_{i=1}^2 N_i (\mu_i - \mu)(\mu_i - \mu)^t$$

and

$$\mathbf{T} = \sum_{j=1}^N (\mathbf{z}_j - \mu)(\mathbf{z}_j - \mu)^t = \mathbf{W} + \mathbf{B}$$

where

$$\mu_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{z}_{ij},$$

$$\mu = \frac{1}{N} (N_1 \mu_1 + N_2 \mu_2),$$

$\mathcal{Z}_i = \{\mathbf{z}_{i1}, \dots, \mathbf{z}_{iN_i}\}$, $i = 1, 2$, $\mathcal{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_N\} = \mathcal{Z}_1 \cup \mathcal{Z}_2$, N_i denotes the number of samples in a subclass and N is the total number of samples ($N = N_1 + N_2$).

Now we may state the following theorem:

Theorem: Minimizing the cost function $J_1 = (\mu_1 - \mu)^t(\mu_2 - \mu)$ is equivalent to minimizing the cost function $J_2 = \text{tr}\{\mathbf{W}\}$.

Proof:

It is easy to be convinced that

$$\min\{\text{tr}\{\mathbf{W}\}\} = \max\{\text{tr}\{\mathbf{B}\}\} = \max\{\text{tr}\{\mathbf{T} - \mathbf{W}\}\}$$

Let us first find an expression for $\text{tr}\{\mathbf{B}\}$.

$$\begin{aligned} \text{tr}\{\mathbf{B}\} &= \sum_{i=1}^d \left\{ \left[N_1 (\mu_1(i) - \mu(i))^2 \right] + \left[N_2 (\mu_2(i) - \mu(i))^2 \right] \right\} \\ &= \sum_{i=1}^d \left[N_1 \mu_1(i)^2 + N_2 \mu_2(i)^2 + N \mu(i)^2 - 2 (N_1 \mu_1(i) + N_2 \mu_2(i)) \mu(i) \right] \\ &= \sum_{i=1}^d \left[N_1 \mu_1(i)^2 + N_2 \mu_2(i)^2 - N \mu(i)^2 \right]. \end{aligned}$$

Then, we find an expression for the inner product.

$$\begin{aligned} (\boldsymbol{\mu}_1 - \boldsymbol{\mu})^t (\boldsymbol{\mu}_2 - \boldsymbol{\mu}) &= \sum_{i=1}^d \left[\mu_1(i) \mu_2(i) - (\mu_1(i) + \mu_2(i)) \mu(i) + \mu(i)^2 \right] \\ &= \sum_{i=1}^d \left[\mu_1(i) \mu_2(i) - \frac{1}{N} (\mu_1(i) + \mu_2(i)) (N_1 \mu_1(i) + N_2 \mu_2(i)) + \mu(i)^2 \right] \\ &= \sum_{i=1}^d \left[-\frac{N_1}{N} \mu_1(i)^2 - \frac{N_2}{N} \mu_2(i)^2 + \mu(i)^2 \right] \\ &= -\frac{1}{N} \text{tr}\{\mathbf{B}\}. \end{aligned}$$

Now, we find that $\min\{(\boldsymbol{\mu}_1 - \boldsymbol{\mu})^t (\boldsymbol{\mu}_2 - \boldsymbol{\mu})\}$ implies $\max\{\text{tr}\mathbf{B}\}$ which implies $\min\{\text{tr}\mathbf{W}\}$ which in turn implies the theorem.

C: THE SINGULAR VALUE DECOMPOSITION METHOD USED FOR DETERMINING THE PIECEWISE LINEAR CLASSIFIER FROM A SET OF HYPERPLANES

Let us assume that m pairs of subclasses should be separated, and that the m hyperplanes have been generated. Furthermore, let $\mathbf{a}'_{i_k j_k}$ define the hyperplane separating the subclasses α_{i_k} and α_{j_k} ($1 \leq i_k \leq n$ and $1 \leq j_k \leq n$). Since $\mathbf{a}'_{i_k j_k}$ is separating these two subclasses, it has to be a solution of the equation $\mathbf{a}_{i_k} - \mathbf{a}_{j_k} = \mathbf{a}'_{i_k j_k}$. Now, we want to find the weight vectors $\mathbf{a}_1, \dots, \mathbf{a}_n$ (one for each subclass) by solving the following set of equations

$$\begin{aligned} \mathbf{a}_{i_1} - \mathbf{a}_{j_1} &= \mathbf{a}'_{i_1 j_1} \\ \mathbf{a}_{i_2} - \mathbf{a}_{j_2} &= \mathbf{a}'_{i_2 j_2} \\ \vdots &\quad \quad \quad \vdots \\ \mathbf{a}_{i_m} - \mathbf{a}_{j_m} &= \mathbf{a}'_{i_m j_m} \end{aligned} \tag{C:.1}$$

As stated in 4.3.4.3, a solution of C:.1 can be obtained by using the singular value decomposition method (SVD) [27]. Transferring C:.1 to matrix form yields

$$\mathbf{V}'\mathbf{a} = \mathbf{a}' \tag{C:.2}$$

where

$$\mathbf{a} = \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n \end{pmatrix}$$

and

$$\mathbf{a}' = \begin{pmatrix} \mathbf{a}'_{i_1 j_1} \\ \vdots \\ \mathbf{a}'_{i_m j_m} \end{pmatrix}.$$

Moreover, we see that $\dim\{\mathbf{a}\} = nd \times 1$ and $\dim\{\mathbf{a}'\} = md \times 1$. $\mathbf{V}'$ denotes a $md \times nd$ matrix where 2 elements differs from 0 in a given row. In fact, in row k we have

$$v_{kl} = \begin{cases} 1 & , l = (i_h - 1)d + \text{mod}(k - 1, d) + 1 \\ -1 & , l = (j_h - 1)d + \text{mod}(k - 1, d) + 1 \\ 0 & , \text{otherwise} . \end{cases}$$

Here we are assuming $(h - 1)d + 1 \leq k \leq hd$, $h = 1, \dots, m$.

However, due to redundancy, we observe that $m(d - 1)$ of the rows and $n(d - 1)$ of the columns of $\mathbf{V}'$ may be removed because of redundancy by defining

$$\mathbf{V} = \begin{pmatrix} \xrightarrow{i_1} & \xrightarrow{j_1} \\ 0 & \dots & 1 & \dots & 0 & \dots & -1 & \dots & 0 & \dots & 0 \\ & & & & & & \vdots & & & & \\ 0 & \dots & 0 & \dots & 1 & \dots & 0 & \dots & -1 & \dots & 0 \\ \xrightarrow{i_m} & \xrightarrow{j_m} \end{pmatrix}$$

and it is easy to verify that

$$(\mathbf{V}'\mathbf{a}) = \left[\left[\sum_{k=1}^n v_{1k} \mathbf{a}_k \right]^t, \dots, \left[\sum_{k=1}^n v_{mk} \mathbf{a}_k \right]^t \right]^t .$$

Now, let $\mathbf{Q}$ be an $m \times k$ orthogonal matrix, and let $\mathbf{P}$ be an $n \times k$ orthogonal matrix where $k = \text{Rank}\{\mathbf{V}\} \leq \min\{m, n\}$. Furthermore, the columns in $\mathbf{Q}$ and $\mathbf{P}$ are solutions to the eigen value problem

$$\mathbf{V}\mathbf{V}^t\mathbf{q} = \lambda^2\mathbf{q}$$

and

$$\mathbf{V}^t\mathbf{V}\mathbf{p} = \lambda^2\mathbf{p}$$

(only vectors associated with an eigen value $\neq 0$ are considered). $\mathbf{Q}$ and $\mathbf{P}$ are said to consist of the left and right hand singular vectors respectively. It may now be shown that $\mathbf{V}$ can be decomposed as

$$\mathbf{V} = \mathbf{Q}\mathbf{\Lambda}\mathbf{P}^t$$

where $\mathbf{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_k)$ is a diagonal matrix whose elements are the singular values of $\mathbf{V}$. The generalized inverse $\mathbf{V}^+$ is now defined as

$$\mathbf{V}^+ = \mathbf{P}\mathbf{\Lambda}^{-1}\mathbf{Q}^t, \quad \dim\{\mathbf{V}^+\} = n \times m,$$

and moreover, the weight vectors are computed as

$$\hat{\mathbf{a}}_i = \sum_{l=1}^m v_{il}^+ \mathbf{a}'_l. \quad (\text{C}::3)$$