Stabilizing Weak Classifiers

Regularization and Combining Techniques in Discriminant Analysis

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft
op gezag van de Rector Magnificus prof. ir. K.F. Wakker,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 15 oktober 2001 te 16:00 uur
door

Marina SKURICHINA

matematikė , Vilnius State University
geboren te Vilnius, Litouwen

Dit proefschrift is goedgekeurd door de promotor
Prof. dr. I.T. Young

Toegevoegd promotor: Dr. ir. R.P.W. Duin

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. I.T. Young,	Technische Universiteit Delft, promotor
Dr. ir. R.P.W. Duin,	Technische Universiteit Delft, toegevoegd promotor
Prof. dr. Š. Raudys,	Institute of Mathematics and Informatics, Vilnius, Lithuania
Prof. dr. ir. E. Backer,	Technische Universiteit Delft
Prof. dr. ir. L.C. van der Gaag,	Universiteit Utrecht
Dr. ir. M.J.T. Reinders,	Technische Universiteit Delft
Dr. L.I. Kuncheva,	University of Wales, Bangor, UK, adviseur
Prof. dr. ir. L.J. van Vliet,	Technische Universiteit Delft, reservelid

This research has been sponsored by the dutch foundation for scientific research in the Netherlands (NWO) and by the Dutch Technology Foundation (STW).

Skurichina, Marina

Proefschrift Technische Universiteit Delft
ISBN 90-75691-07-6
Keywords: Regularization, Noise Injection, Combining Classifiers

Contents

Abbreviations and Notations

DT - the Decision Tree

FLD - the Fisher Linear Discriminant

GNI - Gaussian Noise Injection

KLLC - the Karhunen-Loeve's Linear Classifier

LDA - Linear Discriminant Analysis

MGCC - Model of Gaussian data with Common Covariance matrix

MLP - Multilayer Perceptron

NI - Noise Injection

NMC - the Nearest Mean Classifier

NNC - the Nearest Neighbour Classifier

PFLD - the Pseudo Fisher Linear Discriminant

QC - the Quadratic Classifier

RDA - Regularized Discriminant Analysis

RFLD - the Regularized Fisher Linear Discriminant

RQC - the Regularized Quadratic Classifier

RSM - the Random Subspace Method

SLP - Single Layer Perceptron

SSSC - the Small Sample Size Classifier

STD - a STandard Deviation

SVC - the Support Vector Classifier

WD - Weight Decay

k-NN - k Nearest Neighbours

B - the number of classifiers in the ensemble

N - the number of training objects per class

R - the number of noise injections

k - the number of the nearest neighbours in the k-NN directed noise injection

n - the training sample size (the total number of training objects)

p - the data dimensionality (the number of features)

r - the intrinsic data dimensionality

κ - the number of data classes

ρ - the number of redundant noise features

π_i - the ith data class (i=1,..., κ)

μ_i - a mean vector of the class π_i , i=1,..., κ

Σ - a common covariance matrix of pattern classes

δ^2 - a Mahalanobis distance between pattern classes π_1 and π_2

λ - a regularization parameter

λ_{opt} - the optimal value of the regularization parameter

λ_R - the regularization parameter in the ridge estimate of the covariance matrix

λ_{WD} - the weight decay regularization parameter

λ_{NOISE} - the noise variance

λ_{PW} - the smoothing parameter in the Parzen window classifier

$\boldsymbol{I}$ - an identity matrix of the order $p \times p$

$\boldsymbol{D}$ - a diagonal matrix of the order $p \times p$

$\boldsymbol{S}$ - a sample estimate of the covariance matrix

$\boldsymbol{S}_R$ - a ridge estimate of the covariance matrix

$\boldsymbol{T}$ - an orthonormal matrix of the order $p \times p$

tr$\boldsymbol{A}$ - a trace of the matrix $\boldsymbol{A}$ (a sum of diagonal elements)

$\boldsymbol{W}$ - a vector of weights (coefficients) of the perceptron

$\boldsymbol{w}$ - a vector of weights (coefficients) of the discriminant function

w_0 - a free weight or a threshold value of the perceptron or of the discriminant function

$N(\mu, \Sigma)$ - a normal distribution with the mean μ and the covariance matrix Σ

$U(-a, a)$ - an uniform distribution in the interval $[-a, a]$

$\Phi\{ \cdot \}$ - a standard cumulative Gaussian distribution $N(0, 1)$

$E\{ \cdot \}$ - an expectation

$V\{ \cdot \}$ - a variance

$Cost$ - a cost function

P_B - a Bayes error

P_N - a classification (generalization) error

EP_N - an expected classification error

$EP_N^{(min)}$ - a minimal value of the expected classification error

$\boldsymbol{x} = (x_1, \ldots, x_p)$ - a p-variate vector to be classified

$\boldsymbol{X}_j^{(i)} = (x_{j1}^{(i)}, \ldots, x_{jp}^{(i)})$ - the jth p-variate training vector from the class π_i ($i = 1, \ldots, \kappa$; $j = 1, \ldots, N$)

$\boldsymbol{X} = (\boldsymbol{X}_1^{(1)}, \ldots, \boldsymbol{X}_N^{(1)}, \ldots, \boldsymbol{X}_1^{(\kappa)}, \ldots, \boldsymbol{X}_N^{(\kappa)}) = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$ - a training data set

$\tilde{\boldsymbol{X}} = (\tilde{\boldsymbol{X}}_1, \tilde{\boldsymbol{X}}_2, \ldots, \tilde{\boldsymbol{X}}_m)$ - a validation data set

$\boldsymbol{X}^b = (\boldsymbol{X}_1^b, \boldsymbol{X}_2^b, \ldots, \boldsymbol{X}_n^b)$ - a bootstrap sample (replicate) of the training set $\boldsymbol{X}$

$\boldsymbol{Z}_{jr}^{(i)}$ - noise vectors ($i = 1, \ldots, \kappa$; $j = 1, \ldots, N$; $r = 1, \ldots, R$)

$\boldsymbol{U}_{jr}^{(i)} = \boldsymbol{X}_j^{(i)} + \boldsymbol{Z}_{jr}^{(i)}$ - "noisy" training vectors ($i = 1, \ldots, \kappa$; $j = 1, \ldots, N$; $r = 1, \ldots, R$)

$\bar{\boldsymbol{X}}^{(i)}$ - a sample mean vector of the class π_i , i=1,..., κ

$\boldsymbol{C}(\boldsymbol{X})$ - a covariance matrix

$\boldsymbol{G}(\boldsymbol{X})$ - a data dependency matrix

$g_F(\boldsymbol{x})$ - the Fisher Linear Discriminant function (FLD)

$g_{NMC}(\boldsymbol{x})$ - the Nearest Mean Classifier (NMC)

$g_{PFLD}(\boldsymbol{x})$ - the Pseudo Fisher Linear Discriminant function (PFLD)

$g_R(\boldsymbol{x})$ - the Regularized Fisher Linear Discriminant function (RFLD)

$C_b(\boldsymbol{x})$ - the bth base classifier in the ensemble (b=1,...,B)

$\beta(\boldsymbol{x})$ - a final classifier obtained by combining the base classifiers $C_b(\boldsymbol{x})$ (b=1,...,B)

$f(s)$ - an activation function in the perceptron

$f_{GH}(\boldsymbol{x}|\pi_i)$ - a density estimate of the generalized histogram classifier with Gaussian noise injection

$f_{PW}(\boldsymbol{x}|\pi_i)$ - a Parzen window density estimate

$P(\pi_i|\boldsymbol{x})$ - a posteriori probability of an object $\boldsymbol{x} \in \boldsymbol{X}$ to belong to the class π_i , $i = 1, \ldots, \kappa$

$H(\boldsymbol{x}, \boldsymbol{X}_j^{(i)})$ - a distance between vectors $\boldsymbol{x}$ and $\boldsymbol{X}_j^{(i)}$

$K(H)$ - a kernel function (a window function) in the Parzen window classifier

the **learning curve** - the dependency of the generalization error as a function on the training sample size.

Chapter 1

The Problem of Weak Classifiers

1.1 Introduction

The purpose of pattern recognition is to recognize a new pattern (object) on the basis of the available information. The available information is usually some set of measurements obtained from earlier observed objects. For instance, if objects are fruits such as apples, pears or peaches, the obtained measurements (features of an object) could be size, colour, shape, weight etc. When having several objects of the same kind (e.g. pears) described by the same set of features, one may get a generalized description of this kind (class) of objects either applying statistics or using some prior knowledge. In statistical discriminant analysis one uses a statistical description of data classes obtained from a set of training objects (examples) in order to construct a classification rule (discriminant function), that classifies the learning objects correctly. Then one expects that this classification rule is capable of correctly recognizing (classifying) a newly observed object. The accuracy of the classification rule often depends on the number of training objects. The more training objects, the more accurately a generalized description of data classes can be obtained. Consequently, a more robust classifier may be constructed.

When the number of training objects (the training sample size) is small compared to the number of features (the data dimensionality), one may face the *small sample size problem*. The statistical parameters estimated on such small training sample sets are usually inaccurate. It means, that one may obtain quite dissimilar discriminant functions when constructing the classifier on different training sample sets of the same size. By this, classifiers obtained on small training sample sizes are unstable and are weak, i.e. having a high classification error. Additionally, when the number of training objects is smaller than the data dimensionality, some classifiers cannot be constructed at all, as the number of parameters in the discriminant function needed to be estimated is larger than the number of available training objects.

In order to overcome the small sample size problem and to improve the performance of a weak classifier constructed on small training sample sizes, one may use different approaches. First, one may artificially increase the number of training objects (e.g. by noise injection). By this, the small sample size problem will be solved. More robust estimates of parameters will be obtained that will result in a more stable discriminant function usually having a better performance. Second, it is possible to stabilize the classifier using more stable, e.g. regularized parameter estimators. Third, instead of stabilizing a single weak classifier, one may use a combined decision of an ensemble of weak classifiers [17, 108, 47] in order to improve the performance beyond that of a single weak classifier.

In this thesis we intend to study different techniques that allow us to improve the performance of a weak classifier constructed on small training sample sizes. We study regularization techniques applied to parameters in classification rules, such as a ridge estimate of the covariance matrix [30] in *Linear Discriminant Analysis* (LDA) and weight decay [77] in single layer and multilayer perceptron training (see Chapter 2). We also investigate a more global approach when regularization is performed directly on data (noise injection). One may generate additional objects injecting noise to the training data [42] (see Chapters 2, 3 and 8) or generate additional redundant features [119] by noise injection to the feature space (see Chapter 4). Another approach (distinct from regularization), that may improve the performance of a single classification rule, is to construct multiple classifiers on modified training data and combine them into a powerful final decision rule. The most popular combining techniques are bagging [15] (see Chapter 5), boosting [107] (see Chapter 6) and the random subspace method [47] (see Chapter 7), and in this thesis we study their application to linear classifiers. Both, regularization and combining techniques, may improve the performance of the classifier. Regularization, either applied to parameters of the classification rule, or performed on data itself, stabilizes a discriminant function and also reduces the standard deviation of its classification error. On the other hand, some combining techniques are not stabilizing in spite of the fact that they may reasonably improve the performance of a weak single classification rule. Regularization and combining techniques are studied by us mainly for linear classifiers. In general, the same data sets are used in our simulation study in order to perform a proper comparison of studied techniques.

This chapter summarizes some common material, in order to avoid multiple repetitions in each chapter of the thesis. First, in Section 1.2, the data used in our simulation study are described. In Section 1.3, we discuss linear classifiers. The peculiarities of the training of single layer and multilayer perceptrons are considered in Section 1.4. In Section 1.5 we discuss the small sample size problem and the instability of classifiers. In this section we also introduce the instability measure for classifiers. In Section 1.6 we introduces three stabilizing techniques (ridge estimate, weight decay and noise injection) studied in this thesis. Finally, in Section 1.7 the contents of the thesis are summarized.

1.2 Data Used in Experimental Investigations

To illustrate and compare the performance of regularization and combining techniques in different situations and also to show by example whether they are stabilizing or not, a number of artificial and real data sets are used in our experimental investigations. Highly dimensional data sets are usually considered, because we are interested in unstable situations when the data dimensionality is smaller or comparable to the training sample size. As the intrinsic data dimensionality affects the small sample size properties of classification rules, it may be of

interest to consider the effectiveness of studied techniques on the data sets having a different intrinsic dimensionality. Therefore, in our study, we use several artificial data sets (Gaussian and non-Gaussian) having different intrinsic dimensionalities, because this helps us to understand better which factors affect the effectiveness of the regularization and combining techniques. Real data sets are needed to show that these techniques may be effective when solving the real world problems. Some of the real data sets we will use (ionosphere and diabetes data sets from UCI Repository [12]) are used by other researchers [17] in a comparative study of bagging and boosting in LDA. Therefore, we decided to involve them in our study too. Only in the last chapter we are interested in data by itself, when we study the real world problem in machine diagnostics. This data set is described in details in Chapter 8. The description of all other data sets used in our simulation studies follows here.

200-dimensional Gaussian correlated data

The first set is a 200-dimensional correlated Gaussian data set consisting of two classes with equal covariance matrices. Each class consists of 500 vectors. The mean of the first class is zero for all features. The mean of the second class is equal to 3 for the first two features, and equal to 0 for all other features. The common covariance matrix is a diagonal matrix with a variance of 40 for the second feature and a unit variance for all other features. The intrinsic class overlap (Bayes error) is 0.0717. This data set is rotated in the subspace spanned by the first two features using a rotation matrix $\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$. We will call these data "*200-dimensional Gaussian correlated data*". Its first two features are presented in Fig. 1.1.

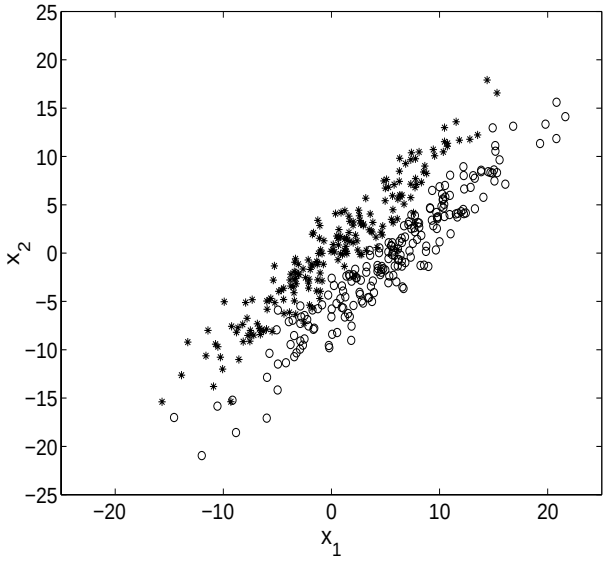


Fig. 1.1. Scatter plot of a two-dimensional projection of the 30-dimensional Gaussian correlated data.

30-dimensional Gaussian correlated data

This data set consists of the first 30 features of the previous data set. The intrinsic class overlap is 0.0640. We will call these data "*30-dimensional Gaussian correlated data*".

200-dimensional Gaussian spherical data

This artificial data set consists of two 200-dimensional Gaussian spherical distributed data classes with equal covariance matrices. Each data class contains 500 vectors. The first data class is Gaussian distributed with unit covariance matrix and with zero mean. The covariance matrix of the second data set is also unit. The mean of the second class is equal to 0.25 for all features. The intrinsic class overlap (Bayes error) is 0.0364. We will call these data "*Gaussian spherical data*".

30-dimensional Gaussian spherical data with unequal covariance matrices

The second data set consists of two 30-dimensional Gaussian distributed data classes with unequal covariance matrices. Each data class contains 500 vectors. The first data class is distributed spherically with unit covariance matrix and with zero mean. The mean of the second class is equal to 4.5 for the first feature and equal to 0 for all other features. The covariance matrix of the second class is a diagonal matrix with a variance of 3 for the first two features and a unit variance for all other features. The Bayes error of this data set is 0.03. We call these data "*Gaussian spherical data with unequal covariance matrices*". Its first two features are presented in Fig. 1.2.

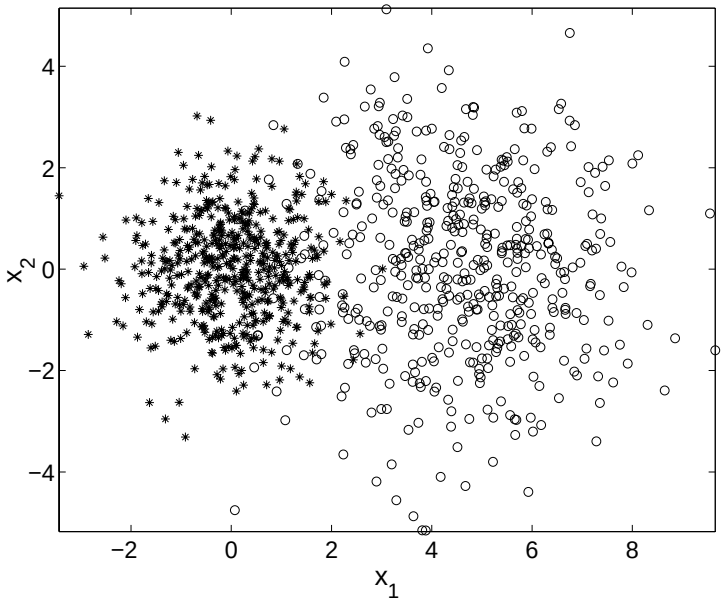


Fig. 1.2. Scatter plot of a two-dimensional projection of the 30-dimensional Gaussian spherical data with unequal covariance matrices.

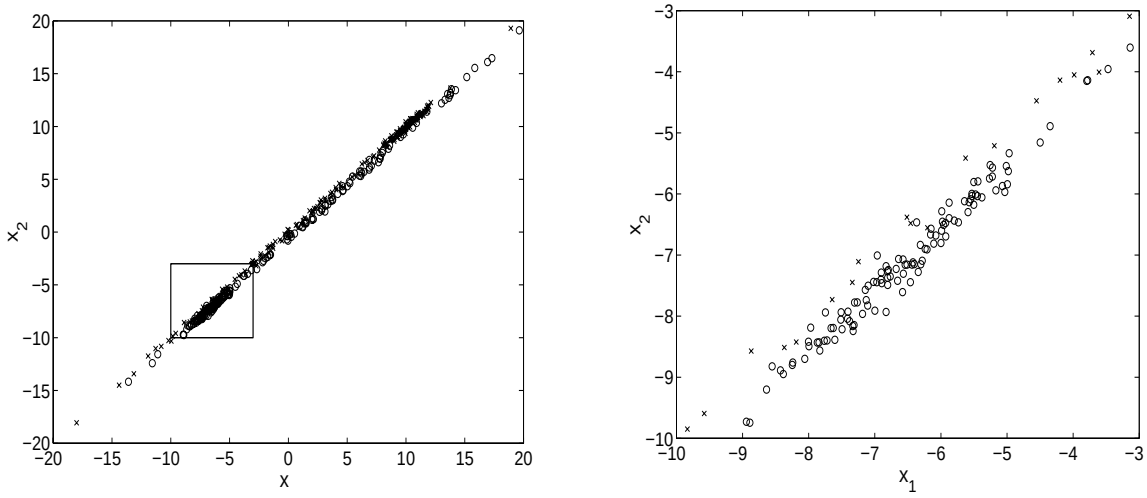


Fig. 1.3. Scatter plot of a two-dimensional projection of the 30-dimensional Non-Gaussian data. Left: entire data set, right: partially enlarged.

30-dimensional Non-Gaussian data

In order to have a non-Gaussian distributed problem we choose for the third set a mixture of two 30-dimensional Gaussian correlated data sets. The first one consists of two strongly correlated Gaussian classes (200 vectors in each class) with equal covariance matrices. The common covariance matrix is a diagonal matrix with a variance of 0.01 for the first feature, a variance of 40 for the second feature and a unit variance for all other features. The mean of the first class is zero for all features. The mean of the second class is 0.3 for the first feature, 3 for the second feature and zero for all other features.

As our aim is to construct non-Gaussian data, we mix the above described data set with another Gaussian data set consisting of two classes with equal covariance matrices. Each class also consists of 200 vectors. The common covariance matrix is a diagonal matrix with a variance of 0.01 for the first feature and a unit variance for all other features. The means of the first two features are [10,10] and [-9.7,-7] respectively for the first and second classes. The means of all other features are zero.

As a result of the union of the first class of the first data set with the first class of the second data set and the second class of the first data set with the second class of the second data set, we have two strongly correlated classes (400 vectors in each class) with shifted weight centers of classes to the points [10,10] and [-9.7,-7] for the first two features. The intrinsic class overlap (Bayes error) is 0.060. This data set was also rotated like the first data set over the first two features. Further these data are called "*Non-Gaussian data*". Its first two features are presented in Fig. 1.3.

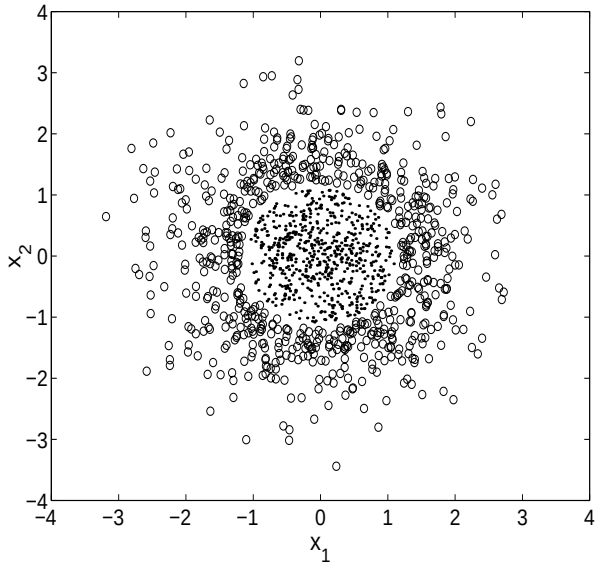


Fig. 1.4. Scatter plot of a two-dimensional projection of the 8-dimensional circular data.

8-dimensional circular data

This data set is composed of two separable data classes having 615 and 698 objects respectively in the first and in the second data class. The first data class consists of 8-dimensional vectors $\boldsymbol{X} = (x_1, \ldots, x_8)'$, that have the Gaussian distribution $N(\mathbf{0}, \boldsymbol{I})$ and lie within the circle $x_1^2 + x_2^2 \le 1.1$. Vectors of the second class also have the Gaussian distribution $N(\mathbf{0}, \boldsymbol{I})$ but lie outside the circle $x_1^2 + x_2^2 \ge 1.45$. As a result, in the first two features, one data class is located inside the other data class. We will call these data "*circular data*". Its first two features are presented in Fig. 1.4.

8-dimensional banana-shaped data

This data set consists of two 8-dimensional classes. The first two features of the data classes are uniformly distributed with unit variance spherical Gaussian noise along two $2\pi/3$ concentric arcs with radii 6.2 and 10.0 for the first and the second class respectively.

$$\vec{\boldsymbol{X}}^{(1)} = \begin{pmatrix} X_1^{(1)} \\ X_2^{(1)} \end{pmatrix} = \begin{pmatrix} \rho_1 \cos(\gamma_1) + \xi_1 \\ \rho_1 \cos(\gamma_1) + \xi_2 \end{pmatrix}, \rho_1 = 6.2,$$

$$\vec{\boldsymbol{X}}^{(2)} = \begin{pmatrix} X_1^{(2)} \\ X_2^{(2)} \end{pmatrix} = \begin{pmatrix} \rho_2 \cos(\gamma_2) + \xi_3 \\ \rho_2 \cos(\gamma_2) + \xi_4 \end{pmatrix}, \rho_2 = 10,$$

where $\xi_i \sim N(0, 1)$, $i = \overline{1, 4}$, $\gamma_k \sim U\left(-\frac{\pi}{3}, \frac{\pi}{3}\right)$, $k = 1, 2$.

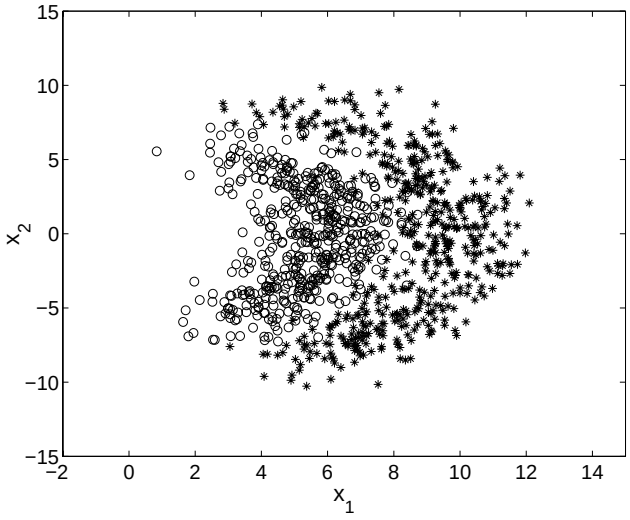


Fig. 1.5. Scatter plot of a 2-dimensional projection of the 8-dimensional banana-shaped data.

The other six features have the same spherical Gaussian distribution with zero mean and variance 0.1 for both classes. The Bayes error of this data set is 0.053. The scatter plot of a two dimensional projection of the data is presented in Fig. 1.5. We will call these data "*banana-shaped data*".

8-dimensional banana-shaped data with the intrinsic dimensionality of two

This data set consists also of two 8-dimensional banana-shaped classes. The first two features of the data classes are the same as in the data described above. But the other six features have been generated as $x_j = x_2^2 + 0{,}001 \cdot \xi_j$, $\xi_j \sim N(0, 1)$, $j = \overline{3, 8}$ in order to make the local intrinsic dimensionality of this data set equal to two. The intrinsic class overlap is 0.032. This data set will be called "*banana-shaped data with the intrinsic dimensionality of two*".

3-dimensional sinusoidal data

This data set is composed of two separable 3-dimensional pattern classes. In the first two features both data classes have the same Gaussian distribution with zero mean and with covariance matrix $\Sigma = \begin{bmatrix} 40 & -5 \\ 30 & 1 \end{bmatrix}$. The third data feature has been generated as a function of the first feature $x_3 = \sin(x_1 / \pi)$ for the first data class and $x_3 = \sin(x_1 / \pi) + 0{,}1$ for the second data class. The two data classes are identical with the exception of the third data feature, for which the values of the second data class is 0.1 larger than the values of the first data class. This data set is presented in Fig. 1.6 and it will be called "*sinusoidal data*".

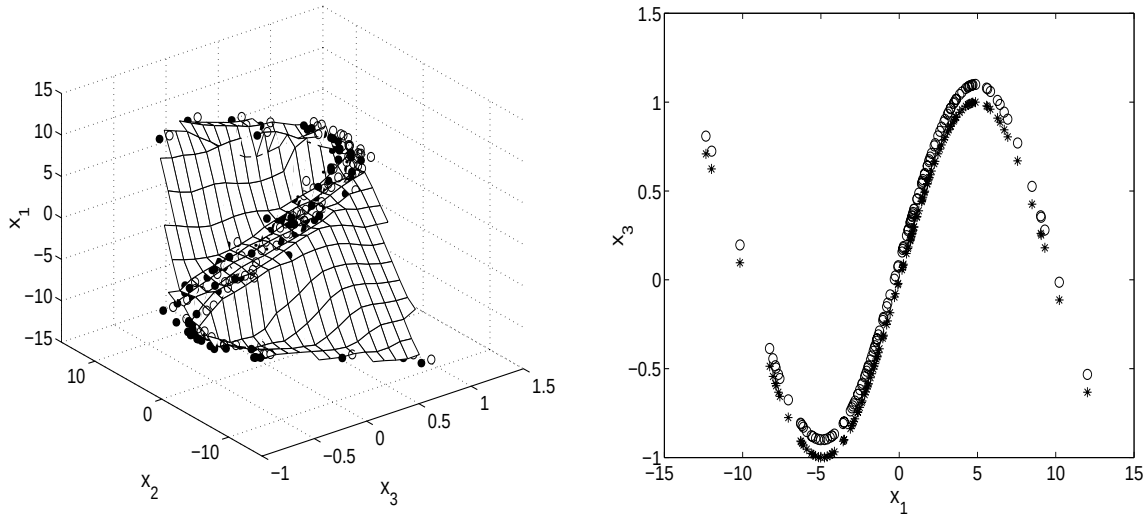


Fig. 1.6. The scatter plot of the 3-dimensional sinusoidal data and the scatter plot of its two-dimensional projection in the first and the third data features space.

12-dimensional uniformly distributed data

This data set consists of two separable 12-dimensional classes. Vectors in both pattern classes are uniformly distributed in the first two features space across the line $x_1 = x_2 / \sqrt{2}$ as it is presented in Fig. 1.7. Another ten data features are generated as $x_j = x_2^2 + 0{,}001 \cdot \xi_j$, $\xi_j \sim N(0, 1)$, $j = \overline{3, 12}$. This way, the local intrinsic dimensionality of this data set is equal to two. We will call these data "*uniformly distributed data*".

34-dimensional ionosphere data

This data set is taken from UCI Repository [12]. These radar data were collected by a system consisted of 16 high-frequency antennas with a total transmitted power of about 6.4 kW in Goose Bay, Labrador. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of structure in the ionosphere. "Bad" returns are those that do not return anything: their signals pass through the ionosphere. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this data set are described by two attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal. Thus, data are described by 34 features. This data set consists of 351 objects in total, belonging to two classes. 225 objects belong to "good" class and 126 objects belong to "bad" class. We will call this data set "*ionosphere*" data set.

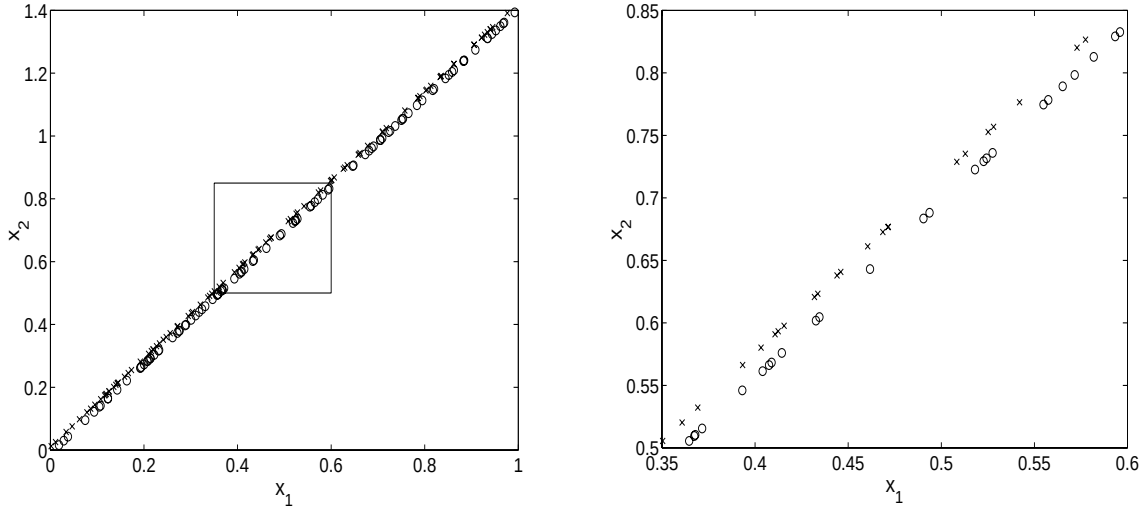


Fig. 1.7. The scatter plot of a two dimensional projection of the 12-dimensional uniformly distributed data. Left: entire dataset, right: partially enlarged.

8-dimensional diabetes data

This is the Pima Indians Diabetes data set taken from UCI Repository [12]. The observed population lives near Phoenix, Arizona, USA. The data are collected by the National Institute of Diabetes and Digestive and Kidney Diseases. All patients tested on diabetes are females at least 21 years old of Pima Indian heritage. Each tested person is described by 8 attributes (features), including age, body mass index, medical tests etc. The data set consists of two classes. In the first data class 500 healthy patients are described. The second data class represents 268 patients that show signs of diabetes according to World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). We will call this data "*diabetes*" data set.

60-dimensional sonar data

This is the sonar data set taken from UCI Repository [12] and used by Gorman and Sejnowski in their study of the classification of sonar signals using a neural network [39]. The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. Thus the data set consists of two data classes. The first data class contains 111 objects obtained by bouncing sonar signals off a metal cylinder at various angles and under various conditions. The second class contains 97 objects obtained from rocks under similar conditions. The transmitted sonar signal is a frequency-modulated chirp, rising in frequency. The data set contains signals obtained from a variety of different aspect angles, spanning 90 degrees for the cylinder and 180 degrees for the rock. Each object is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular

frequency band, integrated over a certain period of time. The integration aperture for higher frequencies occur later in time, since these frequencies are transmitted later during the chirp. We will call this 60-dimensional data set "*sonar*" data set.

30-dimensional wdbc data

This is the Wisconsin Diagnostic Breast Cancer (WDBC) data set taken from UCI Repository [12] representing a two-class problem. The aim is to discriminate between two types of breast cancer: benign and malignant. The data set consists of 569 objects: 357 objects belong to the first data class (benign) and 212 objects are from the second data class (malignant). Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Ten real-valued features are computed for each cell nucleus: radius (mean of distances from center to points on the perimeter), texture (standard deviation of gray-scale values), perimeter, area, smoothness (local variation in radius lengths), compactness (squared perimeter / area - 1.0), concavity (severity of concave portions of the contour), concave points (number of concave portions of the contour), symmetry, fractal dimension ("coastline approximation" - 1). The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image resulting in 30 features. Therefore, in total, each object of the data set is described by 30 features. For instance, feature 3 is Mean Radius, feature 13 is Radius Standard Error, feature 23 is Worst Radius. The data classes are linearly separable using all 30 input features. We will call this data set "*wdbc*" data set.

24-dimensional german data

This is the German Credit data set taken from UCI Repository [12]. The data set describes good and bad bank customers by 24 features which represent different aspects of the customer (e.g., age, sex, personal status, property, housing, job etc.) and of the bank account (e.g., status of existing account, duration in month, credit history and amount etc.). The data set consists of 1000 objects. The first data class (good customers) contains 700 objects. The second class (bad customers) consists of 300 objects. We will call this data set "*german*" data set.

256-dimensional cell data

This data set consists of real data collected through spot counting in interphase cell nuclei (see, for instance, Netten *et al* [71] and Hoekstra *et al* [50]). Spot counting is a technique to detect numerical chromosome abnormalities. By counting the number of colored chromosomes ('spots'), it is possible to detect whether the cell has an aberration that indicates a serious disease. A FISH (Fluorescence In Situ Hybridization) specimen of cell nuclei was scanned using a fluorescence microscope system, resulting in computer images of the single

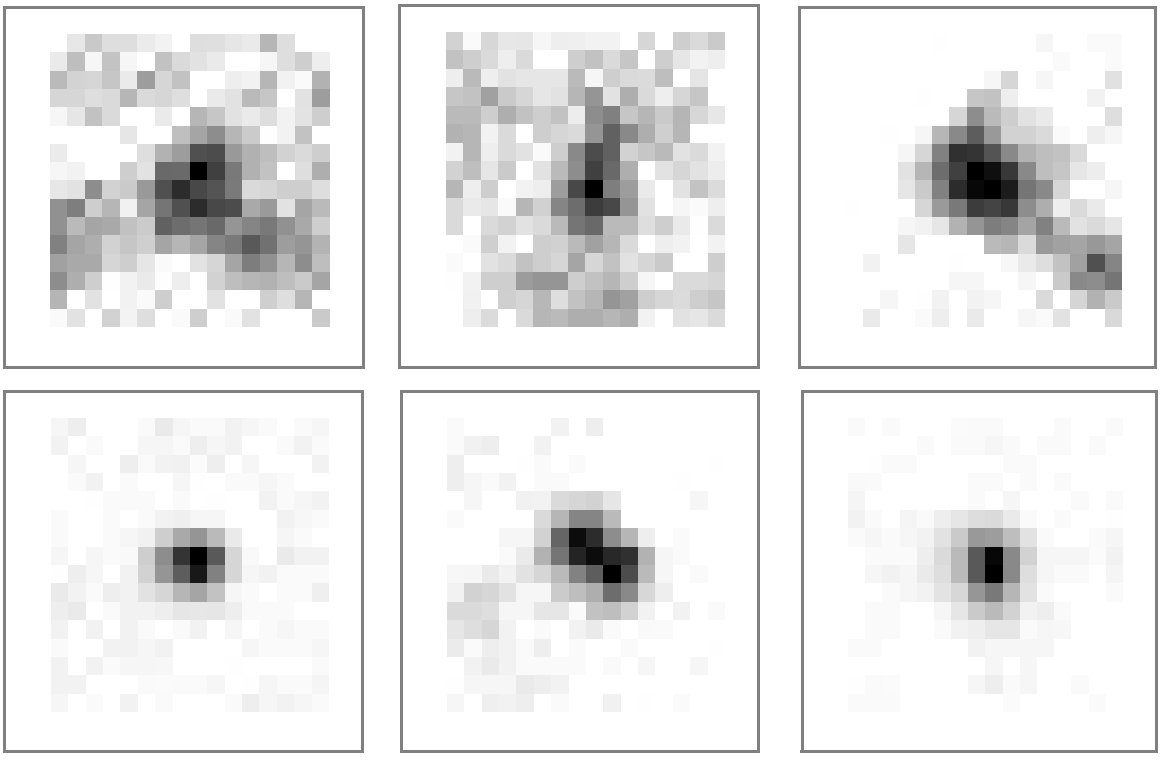


Fig. 1.8. The examples of cell images: noisy backgrounds (the upper row of images) and single spots (the lower row of images).

cell nuclei. From these single cell images 16×16 pixel regions of interest were selected. These regions contain either background spots (noise), single spots or touching spots. From these regions, we constructed two classes of data: the noisy background and single spots, omitting the regions with touching spots. The samples of size 16×16 were considered as a feature vector of size 256. The first class of data (the noisy background) consists of 575 256-dimensional vectors and the second class (single spots) - of 571 256-dimensional vectors. We call these data "*cell data*" in the experiments. Examples are presented in Fig. 1.8.

47-dimensional Zernike data

This data set is a subset of the multifeature digit data set described in UCI Repository [12]. It consists of 47 features, which represent Zernike moments [58], calculated for the images of handwritten digits "3" and "8" (see examples in Fig. 1.9) for the first and the second data class, respectively. Each data class consists of 200 objects. Some features of these data are strongly correlated. Therefore we can expect that the intrinsic dimensionality of this data set is smaller than 47. The peculiarity of this data set is that features of this data set have large differences in scaling. Further these data will be called "*Zernike data*".

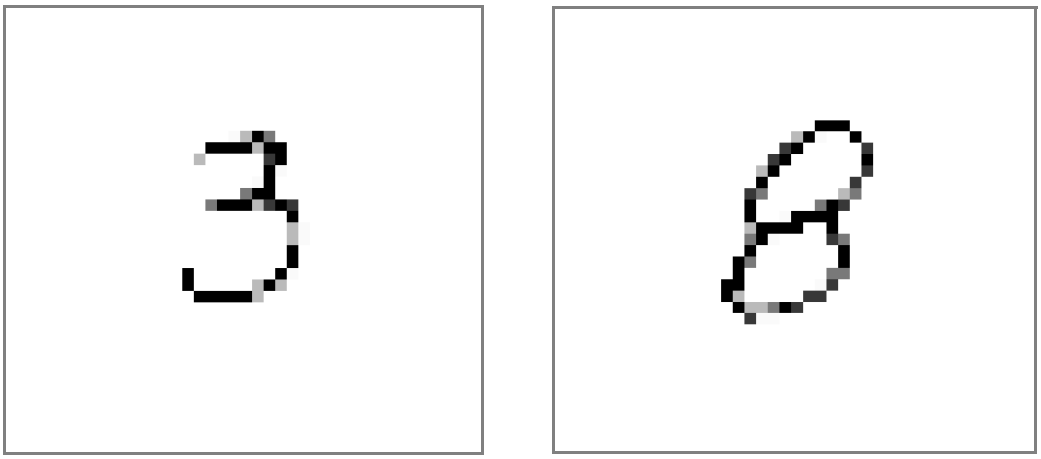


Fig. 1.9. The example of the restored images of handwritten digits "3" and "8".

1.3 Linear Classifiers and Their General Peculiarities

When studying different regularization and combining techniques, sometimes it is not clear what factors may affect their performance: the data peculiarities (data distribution, the dimensionality, the training sample size etc.) or the peculiarities of the classification rule to which these techniques are applied. Linear classifiers are the most simple and well studied classification rules. Therefore, in order to get a good understanding of the regularization and combining techniques, it is wise to apply these techniques to linear classifiers first. And then, when their behaviour is well understood, one may apply them to other more complex classification rules.

The most popular and commonly used linear classifiers are the *Fisher Linear Discriminant* (FLD) [28, 29] and the *Nearest Mean Classifier* (NMC) also called the *Euclidean Distance Classifier* [35]. The standard FLD is defined as

$$g_F(\boldsymbol{x}) = \boldsymbol{x}'\hat{\boldsymbol{w}}^F + \hat{w}_0^F = \left[\boldsymbol{x} - \frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})\right]' \boldsymbol{S}^{-1}(\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)}) \tag{1.1}$$

with coefficients $\hat{\boldsymbol{w}}^F = \boldsymbol{S}^{-1}(\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)})$ and $\hat{w}_0^F = -\frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})'\hat{\boldsymbol{w}}^F$, where $\boldsymbol{S}$ is the standard maximum likelihood estimation of the $p \times p$ covariance matrix Σ, $\boldsymbol{x}$ is a p-variate vector to be classified, $\bar{\boldsymbol{X}}^{(i)}$ is the sample mean vector of the class π_i, $i = 1, 2$.

The Nearest Mean Classifier (NMC) can be written as

$$g_{NMC}(\boldsymbol{x}) = \left[\boldsymbol{x} - \frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})\right]' (\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)}) \quad . \tag{1.2}$$

It minimizes the distance between the vector $\boldsymbol{x}$ and the class mean $\bar{\boldsymbol{X}}^{(i)}$, i=1, 2.

Notice that (1.1) is the mean squared error solution for the linear coefficients $(\boldsymbol{w}, w_0)$ in

$$g_F(\boldsymbol{x}) = \boldsymbol{w} \bullet \boldsymbol{x} + w_0 = L \tag{1.3}$$

with $\boldsymbol{x} \in \boldsymbol{X}$ and with L being the corresponding desired outcomes, 1 for class π_1 and -1 for

class π_2. When the number of features p exceeds the number of training vectors N, the sample estimate $\boldsymbol{S}$ of the covariance matrix will be a singular matrix, that cannot be inverted [82]. For feature sizes p increasing to N, the expected probability of misclassification rises dramatically [54] (see Fig. 1.10).

The NMC generates the perpendicular bisector of the class means and thereby yields the optimal linear classifier for classes having the spherical Gaussian distribution of features with the same variance. The advantage of this classifier is that it is relatively insensitive to the number of training examples [89]. The NMC, however, does not take into account differences in the variances and the covariances.

The modification of the FLD, which allows us to avoid the inverse of an ill-conditioned covariance matrix, is the so-called *Pseudo Fisher Linear Discriminant* (PFLD) [35]. In the PFLD a direct solution of (1.3) is obtained by (using augmented vectors):

$$g_{PFLD}(\boldsymbol{x}) = (\boldsymbol{w}, w_0) \bullet (\boldsymbol{x}, 1) = (\boldsymbol{x}, 1)(\boldsymbol{X}, \boldsymbol{I})^{-1} L \quad , \tag{1.4}$$

where $(\boldsymbol{x}, 1)$ is the augmented vector to be classified and $(\boldsymbol{X}, \boldsymbol{I})$ is the augmented training set. The inverse $(\boldsymbol{X}, \boldsymbol{I})^{-1}$ is the Moore-Penrose Pseudo Inverse which gives the minimum norm solution. Before the inversion the data are shifted such that they have zero mean. This method is closely related to singular value decomposition [35].

The behaviour of the PFLD as a function of the sample size is studied elsewhere [25, 117]. For one sample per class this method is equivalent to the Nearest Mean and to the Nearest Neighbor methods (see Fig. 1.10). For values $N > p$ the PFLD, maximizing the total distance to all given samples, is equivalent to the FLD (1.1). For values $N \leq p$, however, the Pseudo Fisher rule finds a linear subspace, which covers all the data samples. The PFLD builds a linear discriminant perpendicular to this subspace in all other directions for which no samples

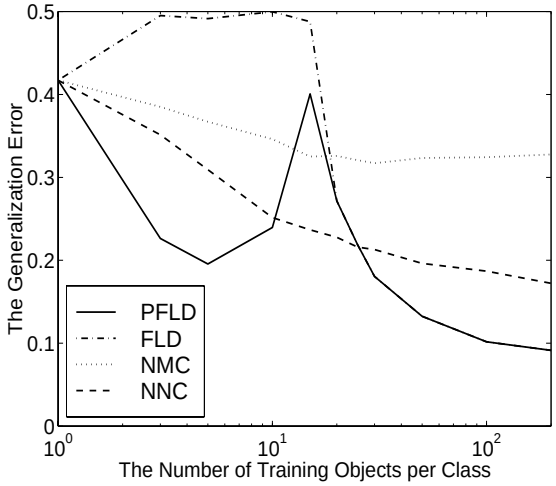


Fig. 1.10. Learning curves of the Pseudo Fisher linear discriminant (PFLD), the nearest mean classifier (NMC) and the Nearest neighbour classifier (NNC) for 30-dimensional Gaussian correlated data.

are given. In between, the generalization error of the PFLD shows a maximum at the point $N=p$. This can be understood from the observation that the PFLD succeeds in finding hyperplanes with equal distances to all training samples until $N=p$. One obtains the exact solution, but all noise presented in the data is covered. In Raudys and Duin [96], an asymptotic expression for the generalization error of the PFLD is derived, which explains theoretically the behaviour of the PFLD.

However, there is no need to use all training samples (objects) if they are already classified correctly by a subset of the training set. Therefore, it is possible to modify the PFLD with the following editing procedure in which misclassified objects are iteratively added until all objects in the training set are correctly classified:

1. Put all objects of the training set in set $\boldsymbol{U}$. Create an empty set $\boldsymbol{L}$.
2. Find in $\boldsymbol{U}$ those two objects, one from each class, that are the closest.
   Move them from $\boldsymbol{U}$ to $\boldsymbol{L}$.
3. Compute the PFLD D for the set $\boldsymbol{L}$.
4. Compute the distance of all objects in the set $\boldsymbol{U}$ to D.
5. If all objects are correctly classified, stop.
6. Move the object in $\boldsymbol{U}$ with the largest distance to D from $\boldsymbol{U}$ to $\boldsymbol{L}$.
7. If the number of objects in $\boldsymbol{L}$ is smaller than p (the dimensionality), go to 3.
8. Compute FLD for the entire set of objects, $\boldsymbol{L} \cup \boldsymbol{U}$ and stop.

This procedure is called the *Small Sample Size Classifier* (SSSC) [25]. It heuristically minimizes the number of training objects by using only those objects for the PFLD, which determine the boundary region between the classes (the so-called margin) and which are absolutely needed for constructing a linear discriminant that classifies all objects correctly. If this appears to be impossible, due to a too large training set relative to the class overlap, the FLD is used. The Small Sample Size Classifier is closely related to the Vapnik's support vector classifier [19], which systematically minimizes the set of objects needed for finding a zero error on the training set.

The *Support Vector Classifier* (SVC) [19] is the generalization for overlapping data classes of the *maximum margin classifier* designed for two separable data classes [128]. The main idea of the maximum margin classifier is to find a linear decision boundary which separates two data classes and leaves the largest margin between the correctly classified training objects belonging to these two classes. It was noticed that the optimal hyperplane is determined by a small fraction of the training objects, which Vapnik called the support vectors. In order to find the support vectors one should solve a quadratic programming problem, which can be very time consuming for large training sample sizes. In the SVC the global minimum (the set of support vectors) is found, while in the SSSC one may find a local minimum. A subset of training objects found by the SSSC is not necessary a set of support vectors. Thus, in the SVC one uses less training objects for constructing the classifier in a feature subspace. By

this, a more stable classifier is obtained.

The *Karhunen-Loeve's Linear Classifier* (KLLC) based on Karhunen-Loeve's feature selection [94] is a popular linear classifier in the field of image classification. The KLLC with the joint covariance matrix of data classes builds the Fisher linear classifier in the subspace of principal components corresponding to the, say q, largest eigenvalues of the joint data distribution. This classifier performs nicely when the most informative features have the largest variances. In our simulations the KLLC uses the 4 largest eigenvalues of the data distribution.

In Fig. 1.11 the behaviour is shown of the Fisher Linear Discriminant, the Nearest Mean Classifier, the Pseudo Fisher Linear Discriminant, the Small Sample Size and Karhunen-Loeve's Linear Classifiers as functions of the training sample size for 30-dimensional Gaussian correlated data (plot a) and for 30-dimensional Non-Gaussian data (plot b) (all results are averaged over 50 independent repetitions). For both data sets the PFLD shows a critical behaviour with a high maximum of the generalization error around $N=p$. For Gaussian correlated data the NMC has a rather bad performance for all sizes of the training data set because this classifier does not take into account the covariances. Therefore it is outperformed by the other linear classifiers. The Non-Gaussian data set is constructed such that it is difficult to separate by linear classifiers: the data classes are strongly correlated, their means are shifted and the largest eigenvalues of the data distribution do not correspond to the most informative features. Thus all linear classifiers have a bad performance for small sample sizes. By increasing training sample sizes, however, the PFLD (which is equivalent to the FLD for $N \geq p$) and the SSSC manage to exclude the influence of non-informative features with large variances and give three times smaller generalization errors than other classifiers.

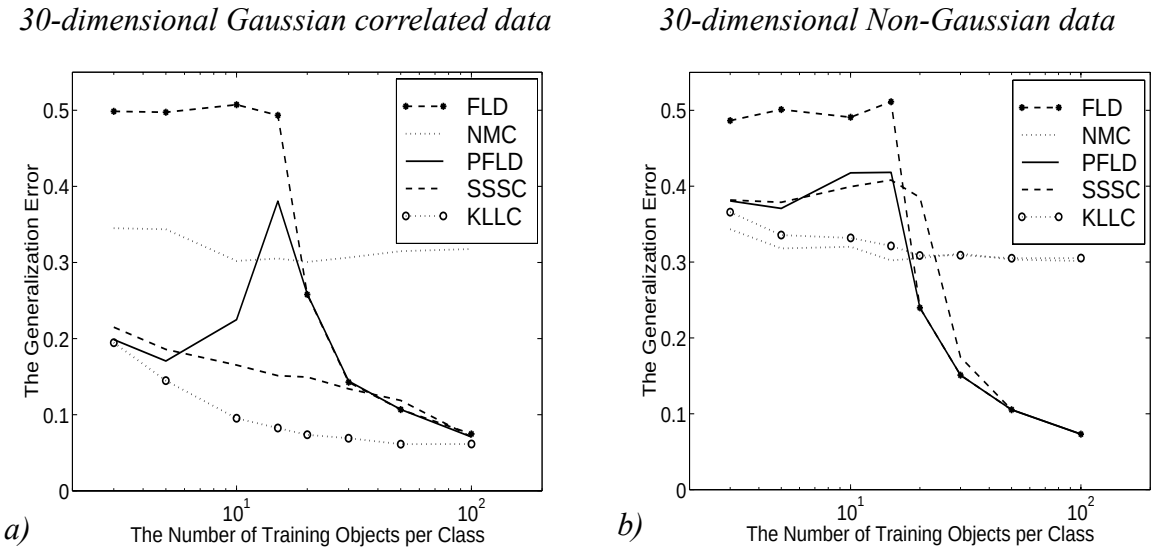


Fig. 1.11. Learning curves of the FLD, the NMC, the PFLD, the SSSC and the KLLC for 30-dimensional Gaussian correlated data (plot a) and Non-Gaussian data (plot b).

1.4 Feed-Forward Neural Classifiers and Their General Peculiarities

Due to the fast developments in the computer science that allow us to perform calculations faster and faster using powerful computers, multilayer feed-forward neural networks became very popular during the last two decades. The large interest to neural networks is based on the fact that these classifiers are generally capable of successfully solving difficult problems as character recognition [64], sonar target recognition [39], car navigation [78] etc. One can design rather complex neural networks and train them on huge data sets using modern computers (parallel machines). The neural network, once trained, may be very fast when recognizing a new incoming object. However, in spite of these nice features, the training of neural networks encounters certain difficulties. Often, the training is slow, has a bad convergence and is very unstable due to its high sensitivity to initial conditions. For this reason, applying neural networks to a specific problem is not an easy task. As a rule, one needs a lot of trials in order to choose a proper architecture for the neural network, proper values for parameters, the starting values for weights in the neural network etc. When studying regularization techniques in Chapters 2 and 3, we apply them also to feed-forward neural networks. In this section we therefore present a short description of a single layer and a multilayer perceptron, and discuss some of peculiarities of their training.

1.4.1 Single Layer and Multilayer Perceptrons

The basic unit of each neural network is a neuron (see Fig. 1.12). Some neural networks may consist of thousands of neurons and some may have only a few neurons. The feed-forward neural network is a network, in which each neuron is connected only to the neurons of the next layer (see Fig. 1.14). Further we will talk only about the feed-forward neural networks. The neural network consisting of one layer of neurons is called a *perceptron* or a *single layer perceptron*. In each neuron, incoming signals $x_1, \ldots, x_p$ (features or measurements of an object

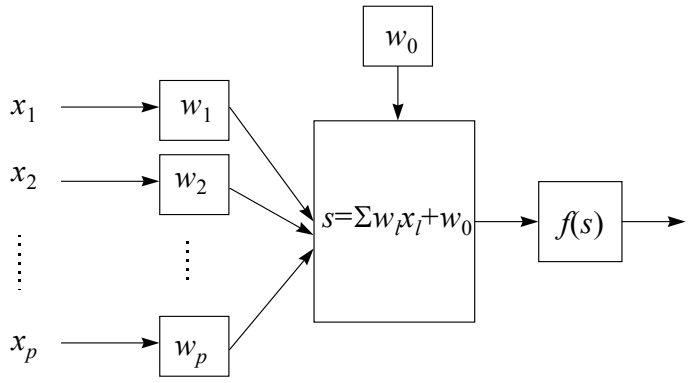


Fig. 1.12. Structure of the neuron ($\mathbf{X}=(x_1,\ldots,x_p)$ is an incoming signal, $w_1,\ldots,w_p$ are weights, s is a weighted sum of incoming signal elements, f(s) is a nonlinear operator).

$\boldsymbol{X} = (x_1, \ldots, x_p)$) are weighted and summed producing an activation signal $s = \sum_{l=1}^{p} w_l x_l + w_0$, which is then transformed by some activation function $f = f(s)$ into an output belonging to a final interval (usually [0, 1] or [-1, 1]). Usually the activation function f is some threshold function. It may be identity output function $f(s) = s$, hard-limiting (see Fig. 1.13a)

$$f(s) = \begin{cases} 1, s \geq 0; \\ 0, s < 0, \end{cases}$$

or smooth-limiting as the sigmoid function (see Fig. 1.13b)

$$f(s) = \frac{1}{1 + e^{-\theta s}},$$

where θ is the sharpness of the sigmoid function (usually θ=1), or as the hyperbolic tangent [64] (see Fig. 1.13c)

$$f(s) = A \tanh(Bs),$$

where A is the amplitude and B is the displacement from the coordinate origin. If the activation function is the identity output function $f(s) = s$, then the output is linear and one obtains a linear perceptron. The classes separating surface produced by the linear perceptron, which consists of a single neuron, will be a linear hyperplane. In our simulations we have used the sigmoid function for non-linear perceptrons and the identity output function for linear ones.

As a rule, neural networks encounter many neurons organized in layers. Such perceptrons are called multilayer perceptrons. The more layers in the neural network, the more complex decision boundary can be achieved. For instance, if the neural network consists of two layers of neurons with linear outputs (with the hard-limited threshold function), its dividing surface will be piecewice linear. The neural network organized by three layers of neurons having linear outputs is capable of separating classes which lie inside one other [73]. However, using a more complex output function (i.e., the sigmoid threshold) allows us to obtain complex decision boundaries with a smaller number of layers. For instance, classes contained inside one other can be separated by the neural network with one hidden layer (two-layered network) with sigmoid outputs [53].

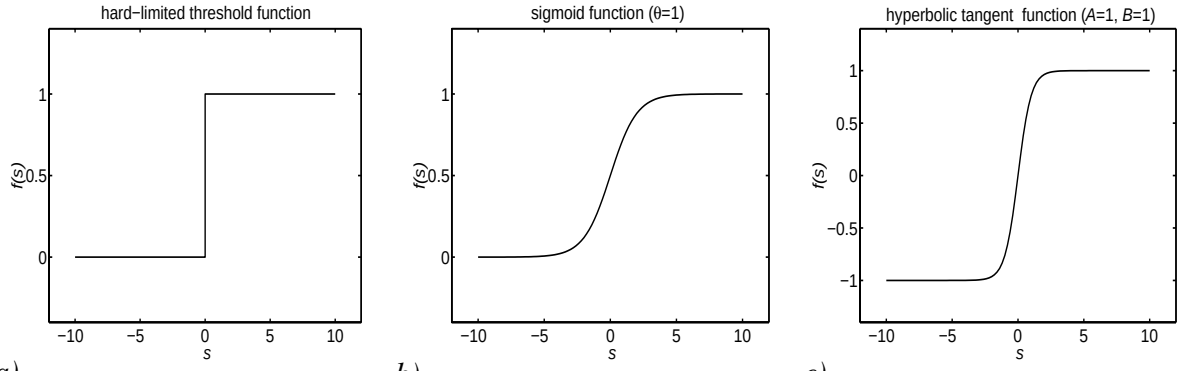


Fig. 1.13. The examples of the threshold function: hard-limited (a), sigmoid (b) and hyperbolic tangent (c) functions.

1.4.2 Training of Perceptrons

Many algorithms exist to train neural networks. The most popular one is *back-propagation* [105], which is a generalization of the DELTA-rule [134]. Let us now describe the back-propagation algorithm for a neural network consisting of two layers (see Fig. 1.14) with p inputs $x_{j1}, ..., x_{jp}$, H neurons in the hidden layer and κ neurons in the output layer. Let v_{lh} and v_{0h} ($l = \overline{1,p}$, $h = \overline{1,H}$) be weights of inputs x_{lh} to the hidden layer neurons, let w_{hi} and w_{0i} ($h = \overline{1,H}$, $i = \overline{1,\kappa}$) be weights from the hidden layer neurons to the output layer neurons, let $\tilde{o}_{jh} = f(\tilde{s}_{jh}) = f\left(\sum_{l=1}^{p} v_{lh} x_{jl} + v_{0h} \right)$ be the outputs of the hidden layer neurons and let $o_{ji} = f(s_{ji}) = f\left(\sum_{h=1}^{H} w_{hi} \tilde{o}_{jh} + w_{0i} \right)$ be the outputs of the output layer neurons.

Back-propagation is an iterative algorithm based on the gradient search optimization technique, which minimizes the cost function (e.g., the sum of squared errors - the mean squared error)

$$Cost = \frac{1}{2} \frac{1}{\kappa N} \sum_{i=1}^{\kappa} \sum_{j=1}^{N} (t_j^{(i)} - o_{ji})^2 \quad ,$$

where $t_j^{(i)}$ are the desired outputs (the targets) of the neural network and o_{ji} are the actual outputs. At each iteration τ of this procedure, one calculates the error signal $Cost$ and then propagates this error back to each neuron correcting all weights of the neural network

$$w_{hi}^{\tau+1} = w_{hi}^{\tau} + \Delta w_{hi}^{\tau+1} \quad , \; (h = \overline{0,H} ; \; i = \overline{1,\kappa}),$$

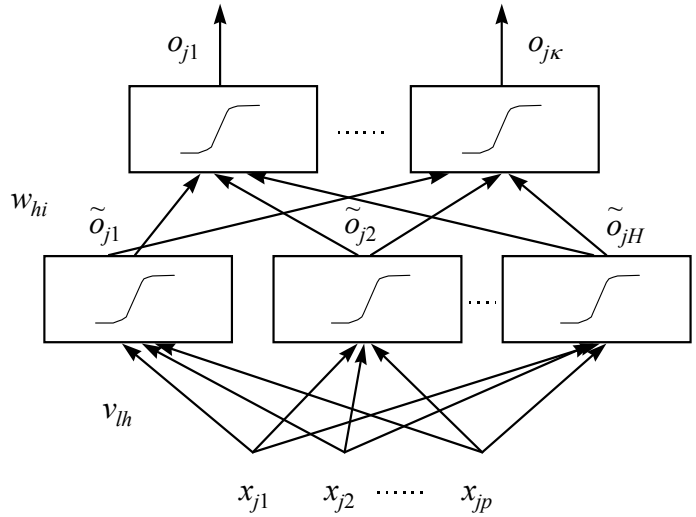


Fig. 1.14. Two-layer neural network with weights v_{0h}, v_{lh}, w_{0i}, w_{hi} and outputs $\tilde{o}_{jh}, o_{jk}$ ($l = \overline{1,p}$; $h = \overline{1,H}$; $i = \overline{1,\kappa}$; $j = \overline{1,N}$) in the hidden and output layer, respectively. Here N is the number of training objects, p is the number of features, κ is the number of classes (the number of neurons in the output layer) and H is the number of neurons in the hidden layer.

where $\Delta w_{hi}^{\tau+1} = -\eta \frac{\partial Cost}{\partial w_{hi}} = \eta f'(s_{ji})(t_j^{(i)} - o_{ji})\tilde{o}_{jh}$,

$$v_{lh}^{\tau+1} = v_{lh}^{\tau} + \Delta v_{lh}^{\tau+1}, \ (\ l = \overline{0,p}\ ;\ h = \overline{1,H}\ ),$$

where $\Delta v_{lh}^{\tau+1} = -\eta \frac{\partial Cost}{\partial v_{lh}} = \eta f'(\tilde{s}_{jh}) x_{jl} \sum_{i=1}^{\kappa} f'(s_{ji})(t_j^{(i)} - o_{ji}) w_{hi}^{\tau}$.

The parameter η is called the learning step, which determines the stepsize and influences the speed of the convergence of the learning procedure.

The back-propagation training algorithm has been studied by many researchers. It was noticed that often the training by back-propagation is slow, may have a bad convergence and may be heavily dependent on initial conditions - weight initialization - and learning parameters [61, 132, 93, 20]. When the training sample set is small, the surface of the cost function $Cost$ is very complex, and contains a large number of deep narrow gorges and wide plateaus [40, 130]. In these conditions, training may often end in a bad local minimum, especially when the training sample size is small [93]. Additionally, the neural network may overfit if the distribution of the training set differs from the real data distribution [113]. Recently, it has been found that perceptrons trained by back-propagation and statistical classifiers are related. Raudys [97] has shown that by training the non-linear single layer perceptron one may obtain seven statistical classifiers of different complexity: the NMC, the RFLD, the FLD, the PFLD, the generalized FLD [81], the minimum empirical error classifier [3, 97] and the maximum margin classifier [13, 19]. It was also shown that the complexity of the obtained classifier strongly depends on the chosen values of training parameters (targets, initial conditions, learning step and the number of iterations) and on the complexity of the problem being solved. Moreover, many parameters act simultaneously, and sometimes the influence of one parameter is compensated by an other one. It is not necessary that the obtained classifier will be optimal for the problem being solved. This also explains why training of neural networks is so unstable and may have sometimes a bad performance.

In order to stabilize the training and improve the generalization of neural networks, one may perform many tricks [69, 10, 72]: regularization by adding a penalty term to the cost function (e.g., weight decay) [100], noise injection (to inputs, weights, outputs etc.) [101, 4], the target values control [98], the learning step control [68], non-random weight initialization [133], sigmoid scaling [101], early stopping [113], node pruning [100] etc. In this thesis, we will consider two of these, weight decay and noise injection to the training objects (inputs).

When studying weight decay and noise injection for neural networks, we use the back-propagation training algorithm. However, sometimes (when we do not study directly the effect of regularization techniques), in order to speed up the training of the neural network we use the *Levenberg-Marquardt learning rule*, which involves second order information, instead of back-propagation, which uses only the first order derivatives. The Levenberg-Marquardt learning rule (sometimes called the damped Newton method) is a modification of Newton

method which avoids some of its disadvantages. The Levenberg-Marquardt algorithm [66, 10] approximate the Hessian matrix used in the Newton method by the matrix $[\nabla^2 Cost + \nu \boldsymbol{I}]$, where ν is a positive factor, $\boldsymbol{I}$ is the identity matrix and $Cost$ is an error function. The Levenberg-Marquardt update rule is

$$\Delta \boldsymbol{W} = [\nabla^2 Cost + \nu \boldsymbol{I}]^{-1} \nabla Cost,$$

where $\Delta \boldsymbol{W}$ is the vector of weight changes at each step, $\nabla^2 Cost$ is the Hessian matrix of the second order derivatives of the error function $Cost$ with respect to the weight vector $\boldsymbol{W}$, $\nabla Cost$ is the Jacobian matrix of derivatives of the error function $Cost$ to each weight.

This optimization technique is more powerful than gradient descent used in the back-propagation learning algorithm, but requires more memory. If the scalar ν is very large, the above expression approximates gradient descent method, while if it is small the above expression becomes the Gauss-Newton method. The Gauss-Newton method is faster and more accurate near an error minimum. The Levenberg-Marquardt algorithm possesses quadratic convergence close to the minimum where it approximates the Newton method. Moreover, it provides convergence if the weight initialization is relatively poor (the starting point is far away from a minimum) similar to the method of steepest descent.

1.5 Small Sample Size Problem

In statistical discriminant analysis, one may face the *small sample size problem*. It happens, when the training sample size is small as compared to the data dimensionality. In this case, it may be difficult to construct a good discriminant function. The sample estimates of the classes means and of the covariance matrix may be biased. Constructing the classifier by using such estimates may result in a discriminant function, which has a poor performance [54]. Additionally, when the number of training objects is less than the data dimensionality, some classifiers can not be constructed in general, if they involve the inverse of the covariance matrix. Under these conditions, the sample estimate of the covariance matrix will be a singular matrix, which cannot be inverted. In any case, classifiers constructed on small training sets are usually biased and may have a large variance. By this, they are unstable.

In order to study the instability of classifiers, one needs to measure it in some way. Intuitively the instability may be considered in relation with the standard deviation of its classification error obtained on the validation set (if it is available) or on the training data set. In this way one can see how much the classification error changes when considering different training sample sets of the same size or different initial values of the parameters (e.g., weight initialization in perceptrons) used in classification rules. However, the standard deviation of the classification error does not directly take into account the training sample size or certain changes in the composition of the training set. In fact, the standard deviation of the classification error does not show the instability of the discriminant function. It only shows the

instability of its performance (the classification error) on the available data set (training or validation data set). The classification error does not necessarily reflect the instability of the discriminant function. When both, the training sample size and the data dimensionality, are small, quite dissimilar discriminant functions can be obtained on the training data sets of the same size while the performance (the classification error) of these discriminant functions can be the same. As well, when both the training sample size and the data dimensionality are large, the conditional classification error can be the same, however, discriminant functions will be different [99]. Therefore, in order to measure the instability of the discriminant function and to see how the discriminant function changes itself, one should use another way and should take into account the size and the composition of the training set. This becomes especially important when one needs to measure the instability of classifiers constructed, for instance, on bootstrap replicates of the training set.

Let us introduce one possible measure of the instability of a classifier [115]. Providing this *instability measure*, we want to take into account the influence of the changes in the composition of the training data set on the instability of the discriminant function. To make changes in the composition of the training data set we use the well known data sampling technique of bootstrapping [26]. In bootstrapping the data set $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$, a random selection with replacement $\boldsymbol{X}^b = (\boldsymbol{X}_1^b, \boldsymbol{X}_2^b, \ldots, \boldsymbol{X}_n^b)$ from the original set of n objects $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$ is made. This random selection $\boldsymbol{X}^b$ is called a bootstrap sample. So some objects could be represented in a new set $\boldsymbol{X}^b$ once, twice or even more times and some objects may not be represented at all. We suggest measuring the instability of a classifier in the following way.

Let $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$ be the training data set and $\tilde{\boldsymbol{X}} = (\tilde{\boldsymbol{X}}_1, \tilde{\boldsymbol{X}}_2, \ldots, \tilde{\boldsymbol{X}}_m)$ be the validation data set. Then

1. Construct the classifier C on the training data set $\boldsymbol{X}$ and classify the validation data set $\tilde{\boldsymbol{X}} \to \rho$, where $\rho = (\rho_1, \rho_2, \ldots, \rho_m)$, $\rho_j = \pi_i$ ($j = 1, \ldots, m$; $i = 1, \ldots, \kappa$), if $\tilde{\boldsymbol{X}}_j$ is classified by C to the class π_i;
2. Repeat for $b = 1, \ldots, B$:

a) Take the bootstrap sample $\boldsymbol{X}^b$ of size n of the training data set $\boldsymbol{X}$, construct the base classifier C_b on this bootstrap sample and classify the validation data set $\tilde{\boldsymbol{X}} \to \rho^b$, where $\rho^b = (\rho_1^b, \rho_2^b, \ldots, \rho_m^b)$, $\rho_j^b = \pi_i$ ($j = 1, \ldots, m$; $i = 1, \ldots, \kappa$), if $\tilde{\boldsymbol{X}}_j$ is classified by C_b to the class π_i;

b) Calculate the difference $Prob(\rho^b \neq \rho)$, when classifying the validation data set $\tilde{\boldsymbol{X}}$ by C and C_b;

3. Calculate the instability $Instability = \frac{1}{B} \sum_{b=1}^{B} Prob(\rho^b \neq \rho)$.

To compute the instability, one may use the validation data set $\tilde{\boldsymbol{X}}$ or the training data set $\boldsymbol{X}$ instead of the validation set $\tilde{\boldsymbol{X}}$, if it is unavailable. In our simulations, we always use B=25, because calculating the instability by using bootstrap samples of the validation (or training) set

is computationally expensive.

In order to demonstrate the advantages of the introduced instability measure compared to the standard deviation of the classification error, let us consider the example of the 30-dimensional Gaussian correlated data set. In Fig. 1.15 the standard deviation of the mean classification error and the instability measure described above are presented. The results presented in figures are averaged over 50 independent repetitions. In Fig. 1.15a,b one can see that judging the instability of the classifier by the standard deviation of the mean classification error (obtained on the test set - generalization error, and obtained on the training set - apparent error) is not good for small training sample sizes. The standard deviation of the mean generalization error fails when classifiers obtained on different training sets of a small size have a similar bad performance on the test set (e.g. as for the FLD, see Fig. 1.11), however

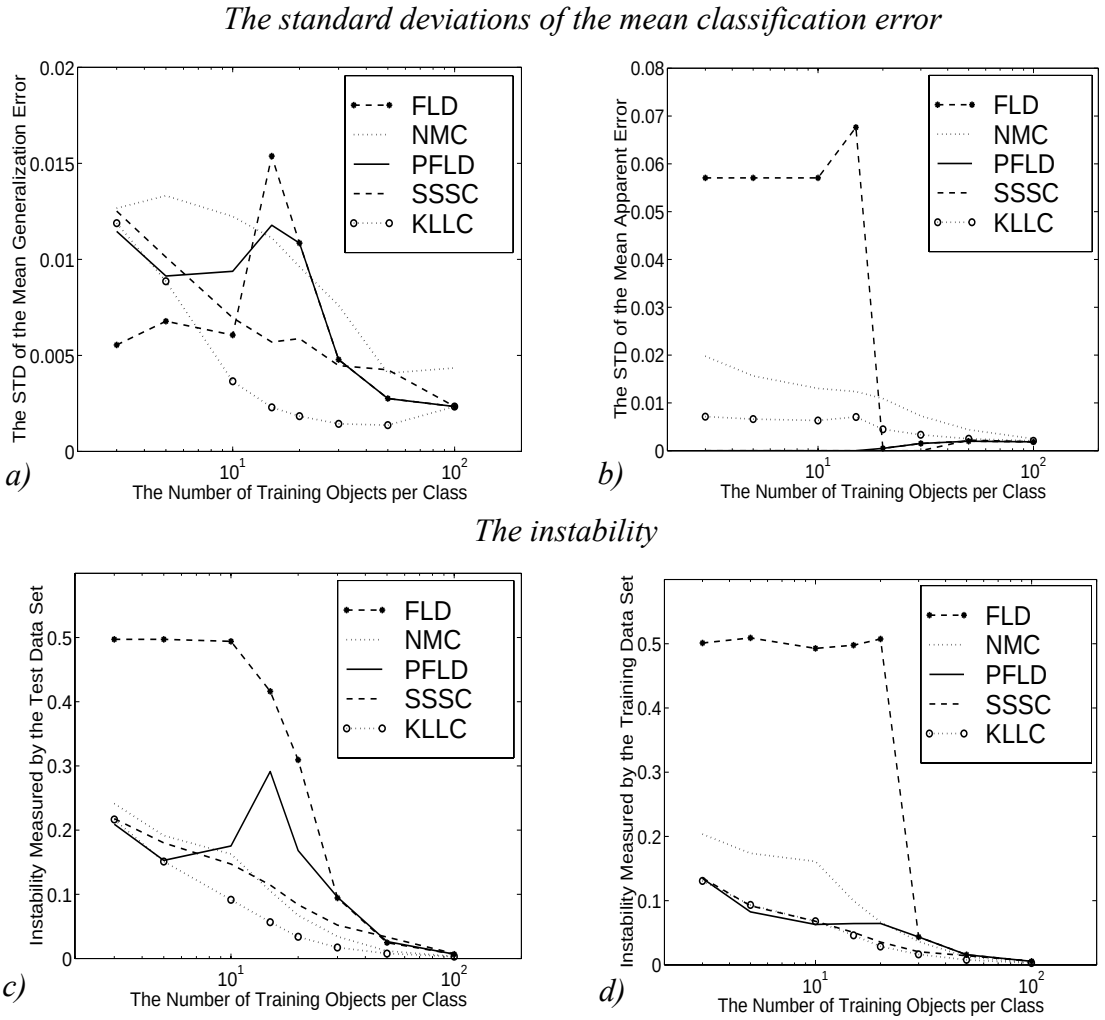


Fig. 1.15. The standard deviation (STD) of the mean generalization error (plot a) and of the mean apparent error (plot b), and the instability measured by the test data set (plot c) and by the training data set (plot d) of the FLD, the NMC, the PFLD, the SSSC and the KLLC versus the training sample size for 30-dimensional Gaussian correlated data.

they may represent very dissimilar discriminant functions. In this case, the standard deviation of the mean generalization error will be relatively small, although the classifier is very unstable. The standard deviation of the mean apparent error fails in the case when one has different classifiers performing very well (with zero apparent error). In spite of the unstable classifier, the standard deviation of the apparent error will be equal to zero claiming a high stability of the classifier. Figures 1.15c,d show that the instability measure introduced above does not have these shortcomings. It is more objective when measuring the instability of the classifier. This instability measure is studied more explicitly in Chapter 5. However, one can see that the performance of classifiers (see Fig. 1.11) and the instability (see Fig. 1.15c,d) are correlated. As a rule, more unstable classifiers have a worse performance. More stable ones perform better. Therefore, it is reasonable to conclude that in order to improve the performance of an unstable classifier, it should be stabilized.

1.6 Stabilizing Techniques

There exist a number of stabilizing techniques which allow us to improve the performance of the classifier. Stabilizing techniques may be divided into two groups: 1) techniques which regularize (stabilize) parameters of the classification rule (e.g., a ridge estimate of the covariance matrix in statistical classifiers, adding penalty terms to the cost function of neural networks or controlling the learning rate, target values etc. during neural networks training), and 2) techniques where regularization is performed directly on the data (e.g., noise injection, data scaling). In this thesis, only some of them will be studied (see Chapters 2 and 3). They are the ridge estimate of the covariance matrix, weight decay and noise injection to the training objects. A short discussion on these regularization techniques follows below. The relationship between these regularization techniques will be explicitly studied in Chapter 2.

1.6.1 Regularizing Parameters

1.6.1.1 Ridge Estimate of the Covariance Matrix in Linear Discriminant Analysis

A well-known technique to avoid the inverse of an ill-conditioned covariance matrix and to stabilize a discriminant function consists in adding small positive constants to diagonal elements of the sample estimate of the covariance matrix,

$$\boldsymbol{S}_R = \boldsymbol{S} + \lambda_R^2 \boldsymbol{I} \ , \tag{1.5}$$

where $\boldsymbol{I}$ is the identity matrix of the order $p \times p$ and a small positive constant λ_R is called the regularization parameter. The new estimate $\boldsymbol{S}_R$ is called the ridge estimate of the covariance matrix. The notion of the ridge estimate is taken from regression analysis [1, 51, 102], where ridge estimates are quite popular. Using this approach in discriminant analysis leads to

Regularized Discriminant Analysis (RDA) [30]. Applying the ridge estimate (1.5) to the linear discriminant function (1.1) one obtains a ridge, or *Regularized Fisher Linear Discriminant* function (RFLD)

$$g_R(\boldsymbol{x}) = \left[\boldsymbol{x} - \frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})\right]' (\boldsymbol{S} + \lambda_R^2 \boldsymbol{I})^{-1} (\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)}) \quad . \tag{1.6}$$

Obviously, when $\lambda_R \to \infty$, defining the classification rule, we lose information concerning covariances between features. Then, the classifier approaches the Nearest Mean Classifier (NMC) [35], and the probability of misclassification may appreciably increase (see Fig. 1.16a,b). It may be useful to employ small values of the regularization parameter because such values stabilize the decision (see Fig. 1.16c,d). However, for very small λ_R, the effect of regularization might be worthless. Very small values of λ_R almost do not transform the

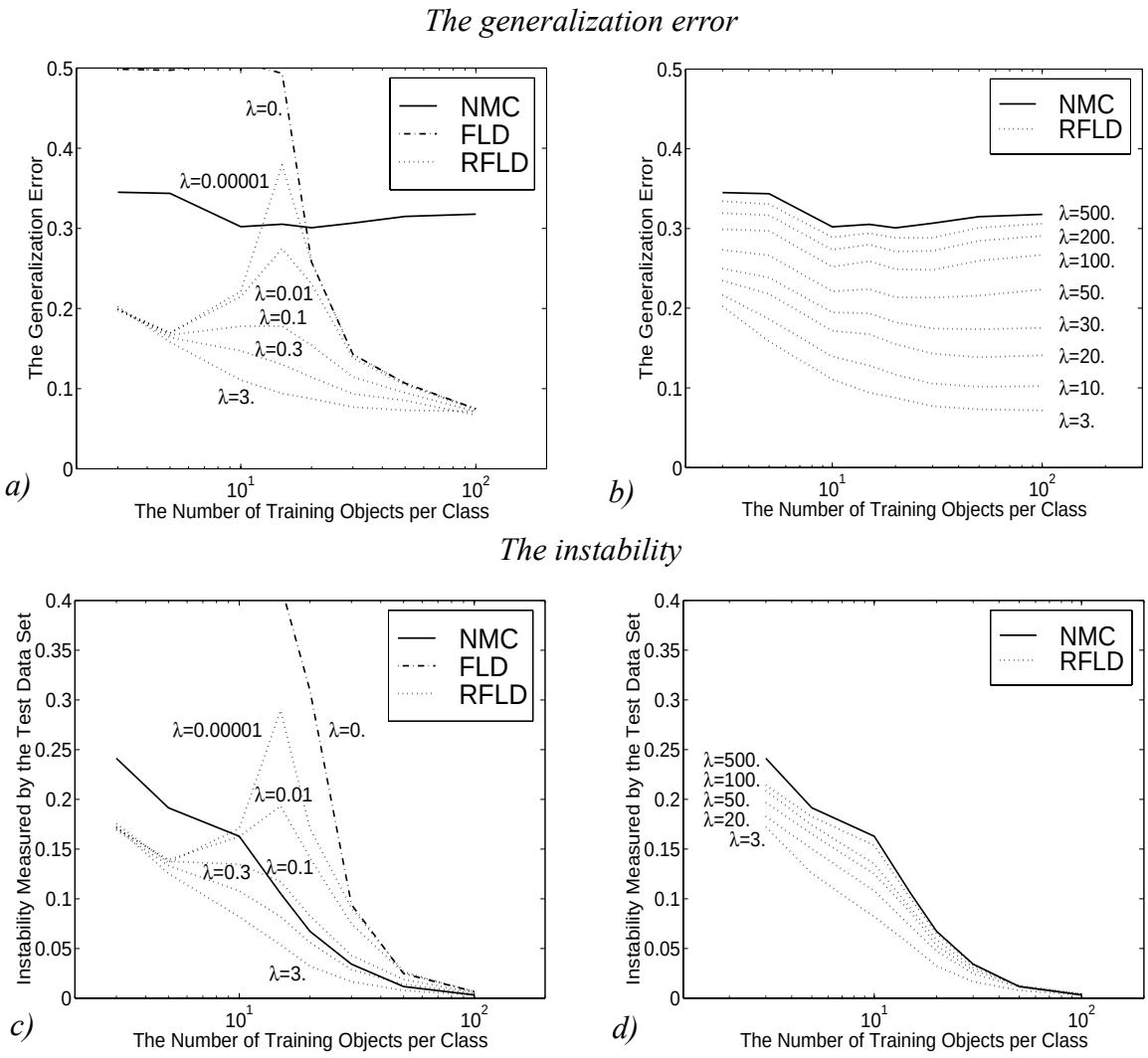


Fig. 1.16. The generalization error (plots a,b) and the instability measured by the test set (plots c,d) of the RFLD with different values of the regularization parameter $\lambda = \lambda_R^2$ *for 30-dimensional Gaussian correlated data.*

training objects. However, slight noise is added in the directions where no training objects are located. By this, one actually constructs the discriminant function in the subspace formed by the training objects and perpendicularly (due to a slight noise) to all other directions where no training objects are presented. One obtains the classifier similar to the PFLD, having a high classification error around the critical training sample sizes. Figure 1.16 shows that using the ridge estimate of the covariance matrix in the FLD stabilizes the discriminant function improving the performance of the FLD. However, the performance of the RFLD strongly depends on the value of the regularization parameter. Therefore, the choice of the value of the regularization parameter is very important.

The literature on pattern recognition provides various recommendations concerning the method of choosing the parameter $\lambda = \lambda_R^2$ [30, 76, 67, 7, 111, 95, 36]. Friedman [30] indicated that regression analysis makes it possible to determine an optimal value of the regularization parameter. To determine this optimal value, he proposed using the leave-one-out cross-validation technique. Furthermore, to reduce the computation time, he improved formulas for the covariance matrix. However, these studies do not provide analytical formulas for calculating the optimal value of the regularization parameter. Other researchers [76, 26] used the bootstrap approach to establish the optimal value of the regularization parameter. Barsov [7] and Serdobolskij [111] proved several theorems concerning the asymptotic classification error. Barsov analyzed the RFLD (1.6) and demonstrated that, if the mean vectors μ_1 and μ_2 are known, in the asymptotic case when $n = N_1 + N_2 \to p$ and $p \to \infty$, the expected classification error tends to

$$EP_N \to \Phi\left\{-\frac{1}{2}\frac{\Upsilon' \boldsymbol{A}(\lambda)\Upsilon}{\sqrt{\Upsilon' \boldsymbol{A}^2(\lambda)\Upsilon}}\sqrt{1-\frac{m^2(\lambda)}{n\lambda^2}\mathrm{tr}\boldsymbol{A}^2(\lambda)}\right\},$$

where $\Upsilon = \Sigma^{-1}(\mu_1 - \mu_2)$, $\boldsymbol{A}(\lambda) = \left(\Sigma^{-1} + \frac{m(\lambda)}{\lambda}\boldsymbol{I}\right)^{-1}$, and the function $m(\lambda)$ satisfies the equation $\lambda\left(\frac{1}{m(\lambda)} - 1\right) \approx \frac{1}{n}\mathrm{tr}\boldsymbol{A}(\lambda)$. In practice, the covariance matrix Σ is unknown. Therefore, to determine the optimal value of λ, Barsov proposed to minimize the expression

$$R_{ERROR} = \Phi\left\{-\frac{1}{2}\frac{\Upsilon' \boldsymbol{S}_R^{-1}\Upsilon}{\sqrt{\Upsilon' \boldsymbol{S}_R^{-1}\boldsymbol{S}\boldsymbol{S}_R^{-1}\Upsilon}}\sqrt{1-\frac{p}{n}+\frac{\mathrm{tr}\boldsymbol{S}_R^{-1}}{n}}\right\}.$$

Serdobolskij replaced (1.5) by the generalized regularized error

$$\boldsymbol{S}_{GR} = \int_{\Omega}(\boldsymbol{S} + \lambda\boldsymbol{S})d\eta(\lambda),$$

and developed the technique for determining the optimal weighted function $\eta(\lambda)$. As a rule, the formulas obtained for computing the optimal value of the regularization parameter are asymptotic or they are based on another criterion (e.g., the mean squared error) than the generalization error. By this, they are not very useful in practice when having finite training

sample sizes and the generalization error should be minimized. The cross-validation technique is sufficiently precise only for large training sample size. However, it is computationally very expensive.

1.6.1.2 Weight Decay

The training of the perceptron is a rather unstable procedure, especially when the training sample size is small compared with the data dimensionality [61, 20]. In this case, the surface of the cost function is complex, consisting of long deep narrow gorges and large slowly lowering plateaus [92]. Moreover, the perceptron training is an iterative procedure, which depends on a number of simultaneously acting parameters used in the training algorithm. The most important of them are the learning step, the number of iterations and targets [101, 97]. In these circumstances, the training of the perceptron is unstable and strongly affected by initial conditions. There exists a number of methods to improve perceptron training. For instance, the learning step control during perceptron training, non-random weights initialization, early stopping and pruning nodes, sigmoid scaling and target smoothing, adding penalty terms to the cost function and noise injection. Overviews and explicit discussions of these techniques can be found in [10, 100, 101, 69].

The magnitude of weights in perceptron training plays a crucial role. Usually, training starts with small values of weights. During training weight values change and the total magnitude of weights gradually increases. When it becomes too large, weights are insensitive to subsequent changes. The relative weight values do not change anymore, just their magnitude increases linearly [97]. It means that training is actually stopped, although the values of

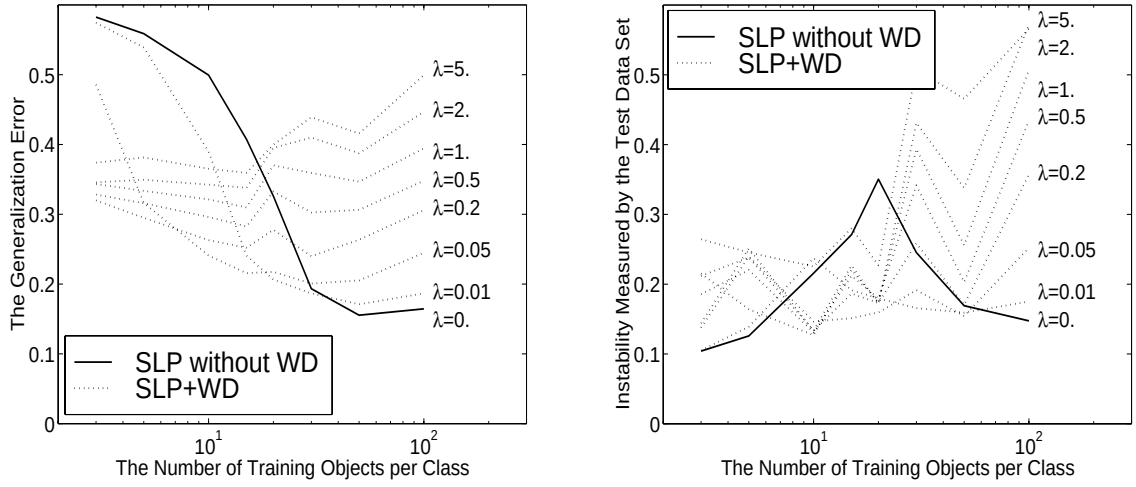


Fig. 1.17. The generalization error (left plot) and the instability measured by the test set (right plot) of the linear single layer perceptron (SLP) with weight decay (WD) having different values of the regularization parameter $\lambda = \lambda_{WD}^2$ *for 30-dimensional Gaussian correlated data.*

weights are still changing. Therefore, the main problem is to protect weights against growing. One way is to penalize large weights by adding a penalty term to the cost function. The most popular one is the *weight decay* regularization term $\lambda_{WD}^2 \boldsymbol{W}'\boldsymbol{W}$ [77] added to the mean square error $Cost$

$$Cost_{WD} = \frac{1}{2}\frac{1}{\kappa N}\sum_{k=1}^{\kappa}\sum_{n=1}^{N}[t_{nk} - f(\boldsymbol{W}'\boldsymbol{X}_n^{(k)} + w_0)]^2 + \lambda_{WD}^2 \boldsymbol{W}'\boldsymbol{W} \quad .$$

Here $\boldsymbol{W}$ is a vector consisting of all weights of the neural network, λ_{WD} is called the regularization parameter, which controls the relative contribution of the traditional mean square error $Cost$ and of the magnitude of the weights.

Weight decay has been studied by many researchers [46, 55, 43, 95, 100]. It was noticed that weight decay may reasonably improve the performance of neural networks [63, 43]. It was shown that weight decay smooths the cost function [10], helps against overtraining [100, 10, 113] and accelerates perceptron training. One of the major drawbacks of weight decay is that it tends to favour small weights even in cases when large weights (some of them) are more preferable [97]. Another problem is how to choose the regularization parameter in weight decay. Similar to the regularized discriminant analysis, too small and too large values of the regularization parameter λ_{WD} may deteriorate the performance of the perceptron (see left plot in Fig. 1.17). Cross-validation techniques are usually used in order to find the optimal value of the regularization parameter in weight decay. Figure 1.17 shows the effect of weight decay on the performance and on the instability of a linear single layer perceptron for 30-dimensional Gaussian correlated data set. One can see that the choice of the regularization parameter λ_{WD} strongly affects the performance and the instability of the linear single layer perceptron.

1.6.2 Regularizing Data by Noise Injection

In order to stabilize the classification rule one may perform regularization on the data. The most popular approach is *noise injection* to the training data (or jittering the data). By noise injection one enlarges the training set and, by this, the small sample size problem is solved. Usually training data are enriched by Gaussian noise $N(\mathbf{0}, \lambda_{NOISE}^2 \boldsymbol{I})$. One obtains $2NR$ "noisy" training vectors $\boldsymbol{U}_{jr}^{(i)} = \boldsymbol{X}_j^{(i)} + \boldsymbol{Z}_{jr}^{(i)}$, where $\boldsymbol{Z}_{jr}^{(i)} \sim N(\mathbf{0}, \lambda_{NOISE}^2 \boldsymbol{I})$ ($i = 1, \ldots, \kappa$; $j = 1, \ldots, N$; $r = 1, \ldots, R$) is noise and $\boldsymbol{X}_j^{(i)}$ are original training vectors belonging to classes $\pi_1, \pi_2, \ldots \pi_\kappa$. Here N is the number of training vectors from each class and R is the number of noise injections. When training neural networks, one performs noise injection to the training objects at each iteration (sweep) τ ($\tau = \overline{1, T}$) instead of adding noise only once before starting the training [52]. By this, one actually increases the training sample size by a factor T, where T is the number of iterations used in the back-propagation learning algorithm. Enlarging the training set by adding noise allows us to get more robust sample estimates of the means and of the covariance matrix. Thus, more stable classifiers can be obtained with a better performance (see Fig. 1.18) [52]. Adding noise to the training objects

during the perceptron training smooths the cost function [10, 101, 4]. This provides a more stable training (see Fig. 1.19c,d) [93]. Similar to weight decay, noise injection may improve the performance of perceptrons (see Fig. 1.19a,b) accelerating the training and preventing against the overtraining [4, 101]. However, the value of the variance of the noise is very important. There exist many ways to obtain the optimal value of the noise variance λ_{NOISE} [52, 36]: analytically, by maximizing the cross-validation likelihood function, by cross-validation on the data, etc. However, similar to optimizing regularization parameters λ_R and λ_{WD}, the formulas obtained for the optimal value of λ_{NOISE} are asymptotic and thus not very useful in practice. Cross-validation technique is effective only on large training sample sizes or/and when one has a validation data set.

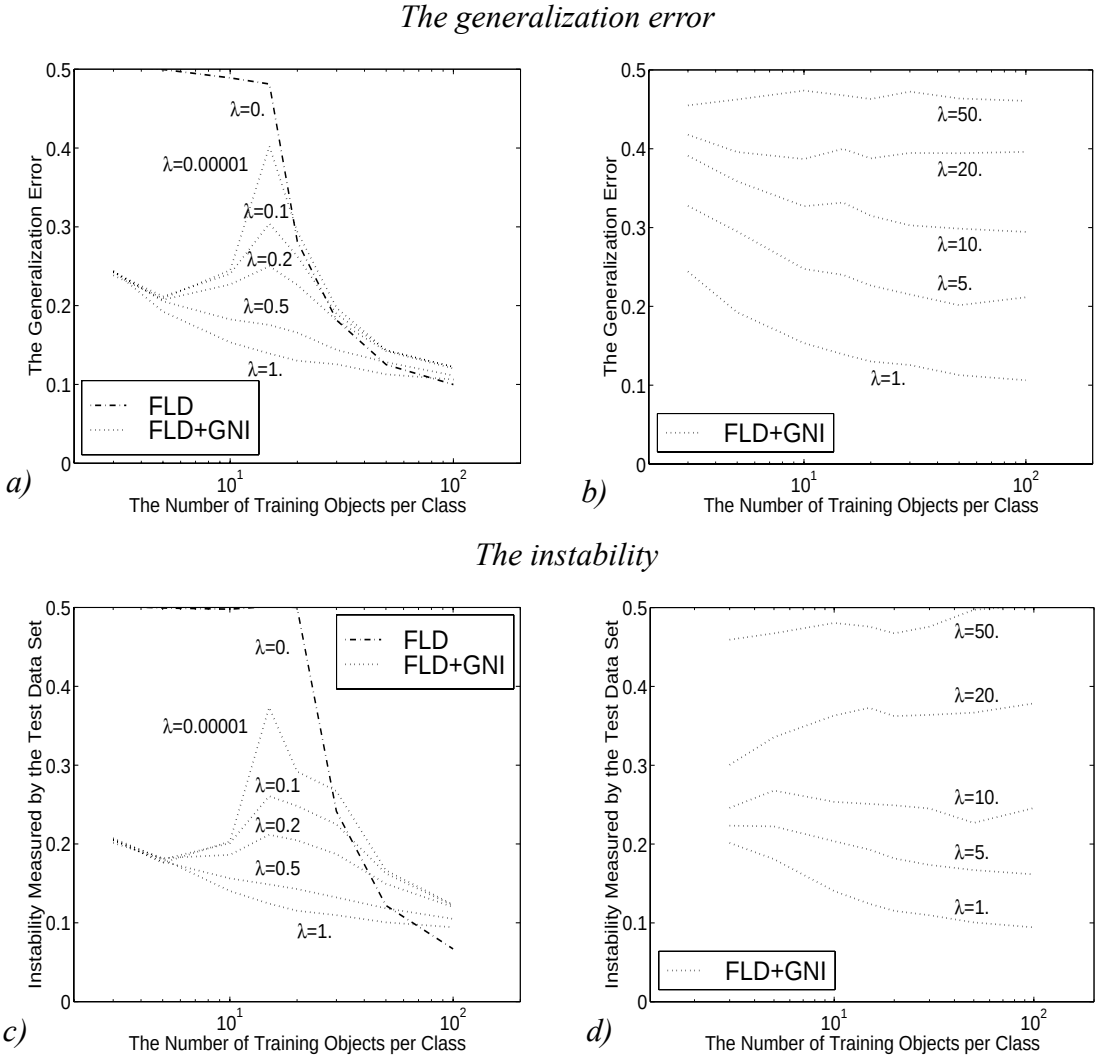


Fig. 1.18. The generalization error (plots a,b) and the instability measured by the test set (plots c,d) of the FLD with Gaussian noise injection (GNI) having different values of the regularization parameter $\lambda = \lambda^2_{NOISE}$ for 30-dimensional Gaussian correlated data.

1.7 Contents

The goal of this thesis is to study techniques which allow us to improve the performance of weak classifiers constructed on small training sample sizes. In order to improve the performance of the classifier one may either stabilize it by using regularization techniques, or construct an ensemble of weak classifiers (instead of a single classifier) combining them into a powerful final decision. In this thesis we study three regularization techniques and three combining techniques. In particular, we study them for linear classifiers.

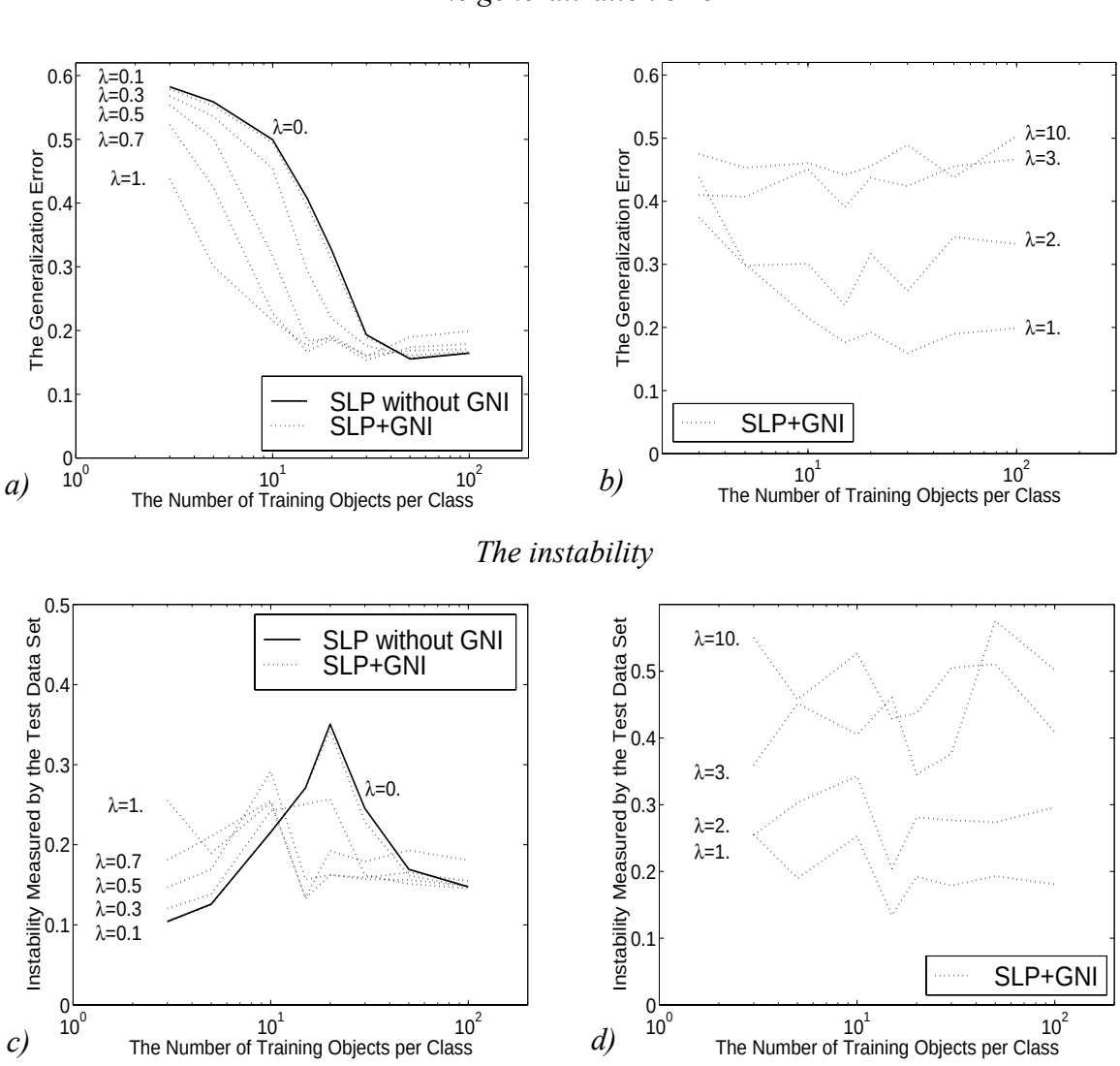


Fig. 1.19. The generalization error (plots a,b) and the instability measured by the test set (plots c,d) of the linear single layer perceptron (SLP) with Gaussian noise injection (GNI) having different values of the regularization parameter $\lambda = \lambda^2_{NOISE}$ *for 30-dimensional Gaussian correlated data.*

In **Chapter 2** we study three regularization techniques: ridge estimate of the covariance matrix in the Fisher linear discriminant function, weight decay for linear perceptrons and Gaussian noise injection to the training objects in the Fisher linear discriminant and in the linear perceptron training. We show analytically and by simulation study that under certain conditions these three regularization techniques are equivalent.

The role of the data intrinsic dimensionality, of the number of noise injections and of the shape of noise on the effectiveness of noise injection is studied in **Chapter 3**. This study reveals that both the intrinsic data dimensionality and the number of noise injections are very important factors when injecting noise. Based on the performed analytical study, we introduce the *k* nearest neighbours directed noise injection. We show that for the data having a small intrinsic dimensionality (as compared to the feature space where the data are described) *k* nearest neighbours directed noise injection is more effective than the Gaussian noise injection.

When the training sample size is critical, that is the data dimensionality is equal to the number of training objects, the generalization error of some classifiers may show a peaking behaviour: with an increase in the training sample size, the generalization error first decreases, reaching a minimum, then increases reaching a maximum at the point of the critical training sample size, and afterwards begins again to decrease. This problem can be solved by removing features, by adding objects or by removing objects. **Chapter 4** introduces the fourth possibility - adding redundant "noisy" features. It is demonstrated that this approach allows us to dramatically reduce the generalization error of the PFLD in the region of the critical training sample size. Mathematical analysis and simulation studies show that adding noise by redundant features is similar to other regularization techniques.

Combining techniques, such as bagging, boosting and the random subspace method have shown themselves beneficial when combining weak classifiers like decision trees. However, the behaviour of these techniques is still not completely understood. Additionally, attempts performed by other researchers to apply them to linear classifiers have failed. In **Chapters 5**, **6** and **7** we study bagging, boosting and the random subspace method for linear classifiers. We show that these techniques may be beneficial in linear discriminant analysis. They may improve the performance of the classifier, but not all of them are stabilizing.

In **Chapter 8**, we study the role of subclasses in machine diagnostics on the example of the water-pump vibration patterns. It is shown that subclasses representing running speeds are much more important than subclasses representing machine loads. In order to regenerate missing gaps between the data, *k* nearest neighbours directed noise injection may be used.

Finally the thesis ends with **Chapter 9**, where conclusions are summarized and the "User Guide" for possible usefulness of the regularization and combining techniques is presented.

Chapter 2

Regularization of Linear Discriminants

2.1 Introduction

The standard *Fisher Linear Discriminant* function (FLD) [28, 29] is defined as

$$g_F(\boldsymbol{x}) = \boldsymbol{x}'\hat{\boldsymbol{w}}^F + \hat{w}_0^F = \left[\boldsymbol{x} - \frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})\right]' \boldsymbol{S}^{-1}(\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)}) \quad . \tag{2.1}$$

It requires one to know the sample estimates $\bar{\boldsymbol{X}}^{(1)}$, $\bar{\boldsymbol{X}}^{(2)}$ of the means μ_1, μ_2 of pattern classes π_1 and π_2, and the sample estimate $\boldsymbol{S}$ of the covariance matrix Σ, which is assumed to be common for all pattern classes. From the training sample set $\boldsymbol{X}_1^{(1)}, \boldsymbol{X}_2^{(1)}, \ldots, \boldsymbol{X}_N^{(1)}, \boldsymbol{X}_1^{(2)}, \boldsymbol{X}_2^{(2)}, \ldots, \boldsymbol{X}_N^{(2)}$ ($n = 2N$ training objects in total), p+1 coefficients $\hat{w}_l^F$, $l = 0, 1, \ldots, p$, of the discriminant function (2.1) should be estimated. This means that, if the number of training vectors n is less than the data dimensionality p, the sample estimate of the covariance matrix will be a singular matrix [82], that, generally, cannot be inverted. One of possibilities to overcome this problem is to correct the sample estimate of the covariance matrix by adding certain constants to the diagonal elements

$$\boldsymbol{S}_R = \boldsymbol{S} + \lambda_R^2 \boldsymbol{I} \ , \tag{2.2}$$

where $\boldsymbol{I}$ is the identity matrix of the order $p \times p$ and λ_R is a small constant, which is called the regularization parameter. The new estimate $\boldsymbol{S}_R$ is called the *ridge estimate* of the covariance matrix. The notion of the ridge estimate was borrowed from regression analysis [1, 51]. This approach is referred to as regularized discriminant analysis [30] and has been discussed in Section 1.6.1.1. The effect of using the ridge estimate of the covariance matrix on the data distribution is illustrated in Fig. 2.1. One may see that the data distribution changes when increasing the value of the regularization parameter λ_R. Very small values of λ_R almost do not change the data distribution. However, for very large values $\lambda_R^2 \to \infty$, information concerning covariances between features is completely lost. Applying the ridge estimate (2.2) to the linear discriminant function (2.1) leads to a ridge, or *Regularized Fisher Linear Discriminant* function (RFLD)

$$g_R(\boldsymbol{x}) = \left[\boldsymbol{x} - \frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})\right]' (\boldsymbol{S} + \lambda_R^2 \boldsymbol{I})^{-1}(\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)}) \ , \tag{2.3}$$

which is discussed in Section 1.6.1.1. When $\lambda_R^2 = 0$, the RFLD is equivalent to the standard FLD. For $\lambda_R^2 \to \infty$, the classifier approaches the Nearest Mean Classifier (NMC) [35]. It may be useful to employ small values of the regularization parameter because such values stabilize the decision (see Section 1.6.1.1) without much loss of information. The literature on pattern

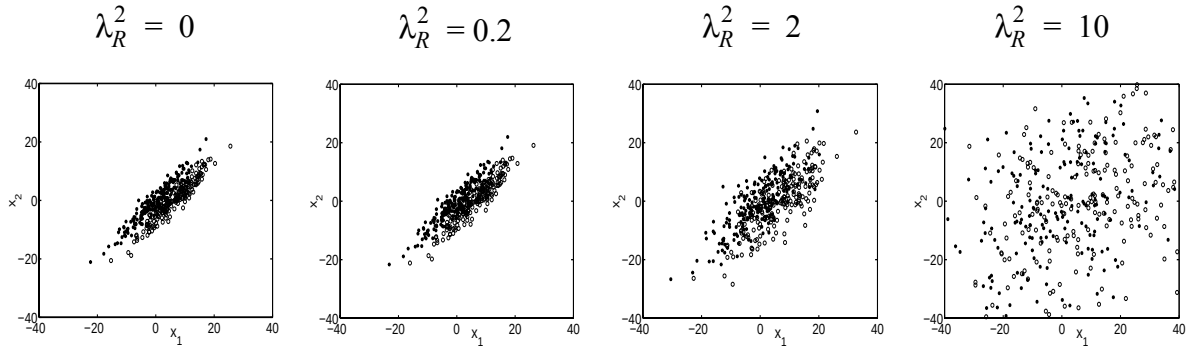


Fig. 2.1. The effect of using the ridge estimate of the covariance matrix on the data distribution.

recognition provides many recommendations how to choose the parameter λ_R [51, 36, 67]. Unfortunately a fast universal method, which successfully allows us to find the optimal value of the regularization parameter for any application, does not exist. This problem is discussed more explicitly in Chapter 1.

Another popular classification rule widely used in pattern recognition is the perceptron (see Section 1.4). Usually, many weights and training parameters should be specified in the process of the perceptron training. When the training sample size $n = 2N$ is small, perceptron training encounters the problem of overtraining [113, 100], when perceptron weights become too much adapted to the training data set, and consequently the performance of the perceptron might become worse on the test data set. In addition, perceptron training is known as a very unstable procedure, which strongly depends on the initial values of perceptron weights [61, 93, 132].

One of the methods that can help to overcome the overtraining and can stabilize the training procedure is *weight decay* [77, 55, 131, 63, 43], when an additional term $\lambda_{WD}^2 \boldsymbol{W}'\boldsymbol{W}$ is added to the cost function

$$Cost_{WD} = \frac{1}{2}\frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N}[t_j^{(i)} - f(\boldsymbol{W}'\boldsymbol{X}_j^{(i)} + w_0)]^2 + \lambda_{WD}^2 \boldsymbol{W}'\boldsymbol{W} \quad .$$

This regularization technique is discussed in Section 1.4.2.

Both, the ridge estimate of the covariance matrix in the FLD and weight decay for perceptrons, are built-in regularization techniques, which regularize the parameters used in the learning rule. An alternative is *noise injection* to the training objects, which is also a stabilizing technique (see Section 1.6.2) helping to improve the performance of the classification rule [77, 112, 52, 70, 44, 42, 101, 4]. However, the latter one regularizes by adjusting the data set.

In perceptron training, noise injection can be performed in such a way, that Gaussian noise $N(0, \lambda^2_{NOISE})$ is added to each component of the training vector in each training sweep. Matsuoka [70] has demonstrated that this noise injection is equivalent to weight decay in the linear *Single Layer Perceptron* (SLP) training. Reed *et al*. [101], Bishop [11], An [4] and Leen [65] have generalized his result. They have shown that, in general, noise injection to the training objects (not necessarily Gaussian noise) is closely related to regularization of the cost function of the perceptron. The development of weight decay terms as a result of training the SLP with noise injection to the training objects is shown by Hertz and Krogh [46].

Patterson and Mattson [75] investigated an ADALIN, linear adaptive classifier, which was proposed by Widrow and Hoff [134]. These studies have demonstrated that this classifier coincides with the FLD. Gallinari *et al*. [37] have proved theoretically that training the linear SLP is equivalent to using the FLD. As demonstrated by Sjöberg and Ljung [113], using the ridge estimate (2.2) in regression analysis is analogous to regularizing weights in a linear SLP.

In this chapter, which is based on our previously published works [94, 95], we consider three regularization techniques: the ridge estimate of the covariance matrix in the FLD, weight decay in the linear SLP training, and Gaussian noise injection to the training objects in the FLD and in the linear SLP training. First, in Section 2.2, we show that asymptotically the FLD with the ridge estimate of the covariance matrix is equivalent to the FLD with Gaussian noise injection to the training objects. Then, in Section 2.3, using analogy with regression analysis and generalizing the results obtained by Koford and Groner [60] and by Gallinari *et al*. [37], it is demonstrated that asymptotically regularizing weights in a linear SLP is equivalent to using, in discriminant analysis, the FLD with the ridge estimate of the covariance matrix. In Section 2.4, in some other way than Matsuoka [70] and other researches [11, 101, 65, 4] have done, we show that Gaussian noise injection to the training objects is equivalent to the weight decay regularization in the linear SLP training. Combining all obtained results leads, consequently, to the asymptotic equivalence of all three mentioned regularization techniques. In Raudys *et al.* [95] the asymptotic formula for the expected classification error of the RFLD (2.3) has been derived. In Sections 2.5 and 2.6 we study the analytical dependence of the expected classification error on the regularization parameter λ^2_R, the training sample size *n* and the data dimensionality *p*. It is demonstrated that, generally, an optimal value of λ^2_R (according to the criterion that requires the minimization of the expected classification error) exists. It is shown that this optimal value depends on the training sample size *n*. It tends to zero for an increasing number of training vectors *n*. It is also shown that there exists a relation between the optimal values of the different regularization parameters used in the considered regularization techniques. The results of our simulation study, provided in order to confirm the theoretical conclusions, are presented in Sections 2.7 and 2.8 for linear and nonlinear cases, respectively. The conclusions are summarized in Section 2.9.

2.2 Noise Injection and the Ridge Estimate of the Covariance Matrix in Linear Discriminant Analysis

In this section, we demonstrate that noise injection to the training objects is equivalent to the ridge estimate of the covariance matrix in the Fisher linear discriminant function.

Let us consider a noise injection model, in which R independent p-variate random Gaussian noise vectors $\boldsymbol{Z}_{jr}^{(i)} \sim N(\boldsymbol{0}, \lambda_{NOISE}^2 \boldsymbol{I})$ ($i = 1, 2$; $j = 1, \ldots, N$; $r = 1, \ldots, R$) are added to each of N p-variate training vectors $\boldsymbol{X}_j^{(i)}$ from two pattern classes π_1 and π_2. As a result, $2NR$ "noisy" training vectors $\boldsymbol{U}_{jr}^{(i)} = \boldsymbol{X}_j^{(i)} + \boldsymbol{Z}_{jr}^{(i)}$ are obtained.

The standard FLD [28, 35] is defined as

$$g_F(\boldsymbol{x}) = \boldsymbol{x}' \hat{\boldsymbol{w}}^F + \hat{w}_0^F = \left[\boldsymbol{x} - \frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})\right]' \boldsymbol{S}^{-1}(\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)}) \tag{2.4}$$

with $\hat{\boldsymbol{w}}^F = \boldsymbol{S}^{-1}(\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)})$ and $\hat{w}_0^F = -\frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})' \hat{\boldsymbol{w}}^F$, where $\boldsymbol{x}$ is p-variate vector to be classified. $\bar{\boldsymbol{X}}^{(i)} = \frac{1}{N}\sum_{j=1}^{N} \boldsymbol{X}_j^{(i)}$ (i=1, 2) and $\boldsymbol{S} = \frac{1}{2N-2}\sum_{i=1}^{2}\sum_{j=1}^{N}(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)})(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)})'$ are the sample estimates of the ith class mean and the common covariance matrix, respectively.

Considering the noise injection model described above, we can design the FLD from $2NR$ "noisy" training vectors $\boldsymbol{U}_{jr}^{(i)}$ with the sample estimate of the ith class mean $\bar{\boldsymbol{U}}^{(i)} = \frac{1}{NR}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{X}_j^{(i)} + \boldsymbol{Z}_{jr}^{(i)}) = \frac{1}{N}\sum_{j=1}^{N}\boldsymbol{X}_j^{(i)} + \frac{1}{NR}\sum_{j=1}^{N}\sum_{r=1}^{R}\boldsymbol{Z}_{jr}^{(i)} = \bar{\boldsymbol{X}}^{(i)} + \bar{\boldsymbol{Z}}^{(i)}$ ($i = 1, 2$) and obtain the following sample estimate of the covariance matrix

$$\boldsymbol{S}^* = \frac{1}{2N-2}\frac{1}{R}\sum_{i=1}^{2}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{U}_{jr}^{(i)} - \bar{\boldsymbol{U}}^{(i)})(\boldsymbol{U}_{jr}^{(i)} - \bar{\boldsymbol{U}}^{(i)})' =$$

$$\frac{1}{2N-2}\frac{1}{R}\sum_{i=1}^{2}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)} + \boldsymbol{Z}_{jr}^{(i)} - \bar{\boldsymbol{Z}}^{(i)})(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)} + \boldsymbol{Z}_{jr}^{(i)} - \bar{\boldsymbol{Z}}^{(i)})' =$$

$$\frac{1}{2N-2}\sum_{i=1}^{2}\sum_{j=1}^{N}(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)})(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)})' + \frac{1}{N-1}\frac{1}{R}\sum_{i=1}^{2}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)})(\boldsymbol{Z}_{jr}^{(i)} - \bar{\boldsymbol{Z}}^{(i)})' +$$

$$\frac{1}{2N-2}\frac{1}{R}\sum_{i=1}^{2}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{Z}_{jr}^{(i)} - \bar{\boldsymbol{Z}}^{(i)})(\boldsymbol{Z}_{jr}^{(i)} - \bar{\boldsymbol{Z}}^{(i)})' = \boldsymbol{S} + \boldsymbol{S}_0 + \boldsymbol{S}_N \quad .$$

Here additional random terms $\boldsymbol{S}_0 = \frac{1}{N-1}\frac{1}{R}\sum_{i=1}^{2}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)})(\boldsymbol{Z}_{jr}^{(i)} - \bar{\boldsymbol{Z}}^{(i)})'$ and $\boldsymbol{S}_N = \frac{1}{2N-2}\frac{1}{R}\sum_{i=1}^{2}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{Z}_{jr}^{(i)} - \bar{\boldsymbol{Z}}^{(i)})(\boldsymbol{Z}_{jr}^{(i)} - \bar{\boldsymbol{Z}}^{(i)})'$ arise due to the random nature of noise vectors $\boldsymbol{Z}_{jr}^{(i)}$. In expectation over noise vectors $\boldsymbol{Z}_{jr}^{(i)}$, one can obtain

$$E\{\boldsymbol{S}_0\} = \frac{1}{N-1}\frac{1}{R}\sum_{i=1}^{2}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)})E\{(\boldsymbol{Z}_{jr}^{(i)} - \bar{\boldsymbol{Z}}^{(i)})'\} = 0 \ ,$$

because $E\{\boldsymbol{Z}_{jr}^{(i)}\} = 0$ and $E\{\bar{\boldsymbol{Z}}^{(i)}\} = 0$. As well, in expectation over noise vectors $\boldsymbol{Z}_{jr}^{(i)}$,

one can write

$$E\{\boldsymbol{S}_N\} = E\left\{\frac{1}{2N-2}\frac{1}{R}\sum_{i=1}^{2}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{Z}_{jr}^{(i)}-\bar{\boldsymbol{Z}}^{(i)})(\boldsymbol{Z}_{jr}^{(i)}-\bar{\boldsymbol{Z}}^{(i)})'\right\} =$$

$$E\left\{\frac{1}{2N-2}\frac{1}{R}\sum_{i=1}^{2}\sum_{j=1}^{N}\sum_{r=1}^{R}(\boldsymbol{Z}_{jr}^{(i)}\boldsymbol{Z}_{jr}^{(i)'}-2\boldsymbol{Z}_{jr}^{(i)}\bar{\boldsymbol{Z}}^{(i)'}+\bar{\boldsymbol{Z}}^{(i)}\bar{\boldsymbol{Z}}^{(i)'})\right\} =$$

$$E\left\{\frac{1}{2N-2}\frac{1}{R}\sum_{i=1}^{2}\left(\sum_{j=1}^{N}\sum_{r=1}^{R}\boldsymbol{Z}_{jr}^{(i)}\boldsymbol{Z}_{jr}^{(i)'}-2NR\bar{\boldsymbol{Z}}^{(i)}\bar{\boldsymbol{Z}}^{(i)'}+NR\bar{\boldsymbol{Z}}^{(i)}\bar{\boldsymbol{Z}}^{(i)'}\right)\right\} =$$

$$\frac{1}{2N-2}\frac{1}{R}\sum_{i=1}^{2}\left(\sum_{j=1}^{N}\sum_{r=1}^{R}E\{\boldsymbol{Z}_{jr}^{(i)}\boldsymbol{Z}_{jr}^{(i)'}\}-NR\,E\{\bar{\boldsymbol{Z}}^{(i)}\bar{\boldsymbol{Z}}^{(i)'}\}\right) =$$

$$\frac{1}{2N-2}\frac{1}{R}\sum_{i=1}^{2}\left(\sum_{j=1}^{N}\sum_{r=1}^{R}\lambda_{NOISE}^{2}\boldsymbol{I}-NR\frac{\lambda_{NOISE}^{2}\boldsymbol{I}}{NR}\right) = \frac{N-\frac{1}{R}}{N-1}\lambda_{NOISE}^{2}\boldsymbol{I}\ .$$

Asymptotically, as the number of noise vectors tends to infinity, $R \to \infty$, the expected sample estimate $\boldsymbol{S}_N$ of the covariance matrix of noise vectors $\boldsymbol{Z}_{jr}^{(i)}$ tends to the diagonal matrix, $E\{\boldsymbol{S}_N\} \to \frac{N}{N-1}\lambda_{NOISE}^{2}\boldsymbol{I}$. Therefore, we have asymptotically and in expectation

$$\boldsymbol{S}^* \to \boldsymbol{S}+\frac{N}{N-1}\lambda_{NOISE}^{2}\boldsymbol{I} = \boldsymbol{S}+\lambda_R^2\boldsymbol{I}\ , \tag{2.5}$$

which is used in the RFLD [30, 10]. Thus, when R is infinite, $\boldsymbol{S}_0 = 0$ and $\boldsymbol{S}_N = \lambda_R^2\boldsymbol{I}$. However, $\boldsymbol{S}_0 \neq 0$ and $\boldsymbol{S}_N \neq \lambda_R^2\boldsymbol{I}$ for finite R, and we have another classification rule, different from the RFLD. Obviously, for small R a random nature of matrices $\boldsymbol{S}_0$ and $\boldsymbol{S}_N$ may deteriorate the estimate $\boldsymbol{S}^*$ and increase the generalization error.

One can see that *asymptotically Gaussian noise injection to the training objects in the FLD is equivalent to the RFLD* with the regularization parameter

$$\lambda_R^2 = \frac{N}{N-1}\lambda_{NOISE}^{2}\ . \tag{2.6}$$

Note that, for large training sample sizes, regularization parameters λ_R and λ_{NOISE} have practically an equal effect.

2.3 The Relationship between Weight Decay in Linear SLP and Regularized Linear Discriminant Analysis

We will now demonstrate that adding a supplementary weight decay term $\lambda_{WD}^2\boldsymbol{W}'\boldsymbol{W}$ [77] to the cost function of the linear SLP is equivalent to the RFLD [94, 95] (in regression similar results were obtained by Hoerl and Kennard [51], Sjöberg and Ljung [113]).

For the case of two classes, without loss of generality, we can assume targets to be $t_j^{(1)} = 1$ and $t_j^{(2)} = -1$ ($j = 1, \ldots, N$) for the pattern classes π_1 and π_2, respectively.

Then, the loss function of the linear SLP (which has the identity output function $f(s) = s$) with a supplementary weight decay term could be written as

$$Cost_{WD} = \frac{1}{2}\frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N}[t_j^{(i)} - (\boldsymbol{W}'\boldsymbol{X}_j^{(i)} + w_0)]^2 + \frac{1}{2}\lambda_{WD}^2\boldsymbol{W}'\boldsymbol{W} \ , \tag{2.7}$$

where $t_j^{(i)}$ is the desired output (target) of the perceptron corresponding to the presentation of the *j*th object $\boldsymbol{X}_j^{(i)}$ ($j = 1, \ldots, N$) from the class π_i ($i = 1, 2$); $\boldsymbol{W}$ is the vector of the weights (coefficients) of the perceptron; w_0 is the free weight or the threshold value of the perceptron. Note, that in (2.7) the regularization of w_0 is omitted in order to simplify calculations. It could be done without loss of generality, as a free weight w_0 could be kept constant during perceptron training when all other weights $\boldsymbol{W}$ are changing.

The partial derivative of the function (2.7) with respect to w_0 is

$$\frac{\partial Cost_{WD}}{\partial w_0} = \frac{1}{2}\boldsymbol{W}'(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)}) + w_0 \ . \tag{2.8}$$

Equating (2.8) to zero, we find the optimal value of w_0,

$$w_0 = -\frac{1}{2}\boldsymbol{W}'(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)}) \ . \tag{2.9}$$

Substituting (2.9) into (2.7), we can find the gradient of function (2.7) with respect to $\boldsymbol{W}$,

$$\frac{\partial Cost_{WD}}{\partial \boldsymbol{W}} = -\frac{1}{2}\Delta\bar{\boldsymbol{X}} + \left[\left(1 - \frac{1}{N}\right)\boldsymbol{S} + \frac{1}{4}\Delta\bar{\boldsymbol{X}}\Delta\bar{\boldsymbol{X}}' + \lambda_{WD}^2\boldsymbol{I}\right]\boldsymbol{W} \ , \tag{2.10}$$

where $\Delta\bar{\boldsymbol{X}} = \bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)}$, and $\boldsymbol{S} = \frac{1}{2N-2}\sum_{i=1}^{2}\sum_{j=1}^{N}(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)})(\boldsymbol{X}_j^{(i)} - \bar{\boldsymbol{X}}^{(i)})'$.

Equating (2.10) to zero and solving the resulting matrix equation, we derive

$$\boldsymbol{W} = \frac{1}{2}m\left[(\boldsymbol{S} + \lambda_R^2\boldsymbol{I}) + \frac{1}{4}m\Delta\bar{\boldsymbol{X}}\Delta\bar{\boldsymbol{X}}'\right]^{-1}\Delta\bar{\boldsymbol{X}} \ , \tag{2.11}$$

where $m = \frac{N}{N-1}$ and $\lambda_R^2 = m\lambda_{WD}^2$.

Next we can use the Bartlett formula [8] for the inversion of symmetric matrices,

$$(\boldsymbol{S} + \boldsymbol{V}\boldsymbol{U}')^{-1} = \boldsymbol{S}^{-1} - \frac{\boldsymbol{S}^{-1}\boldsymbol{V}\boldsymbol{U}'\boldsymbol{S}^{-1}}{1 + \boldsymbol{V}'\boldsymbol{S}^{-1}\boldsymbol{U}} \ ,$$

where **S** is the matrix of the order $p \times p$ and $\boldsymbol{U}$ and $\boldsymbol{V}$ are *p*-dimensional vectors. Applying this formula to expression (2.11), and taking $\Delta\bar{\boldsymbol{X}}$ out of the brackets to the left, we derive

$$\boldsymbol{W} = \frac{1}{2}m(\boldsymbol{S} + \lambda_R^2\boldsymbol{I})^{-1}\Delta\bar{\boldsymbol{X}}\left[1 - \frac{\frac{1}{4}m\Delta\bar{\boldsymbol{X}}'(\boldsymbol{S} + \lambda_R^2\boldsymbol{I})^{-1}\Delta\bar{\boldsymbol{X}}}{1 + \frac{1}{4}m\Delta\bar{\boldsymbol{X}}'(\boldsymbol{S} + \lambda_R^2\boldsymbol{I})^{-1}\Delta\bar{\boldsymbol{X}}}\right] = (\boldsymbol{S} + \lambda_R^2\boldsymbol{I})^{-1}(\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)})\alpha \ , \tag{2.12}$$

where $\alpha = \dfrac{\frac{1}{2}m}{1 + \frac{1}{4}m\Delta\bar{\boldsymbol{X}}'(\boldsymbol{S} + \lambda_R^2\boldsymbol{I})^{-1}\Delta\bar{\boldsymbol{X}}}$ is a scalar quantity.

Thus, we obtain a linear discriminant function

$$g(\boldsymbol{x}) = \boldsymbol{W}'\boldsymbol{x} + w_0 = \left[\boldsymbol{x} - \frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})\right]'(\boldsymbol{S} + \lambda_R^2 \boldsymbol{I})^{-1}(\bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)})\alpha \quad , \tag{2.13}$$

which differs from (2.3) only by the scalar quantity α .

Consequently, *adding an auxiliary regularization term* $\lambda_{WD}^2 \boldsymbol{W}'\boldsymbol{W}$ *to the loss function in the linear SLP training* with equal numbers of vectors in each pattern class *is equivalent to using the ridge estimate for the covariance matrix in the FLD* with the regularization parameter

$$\lambda_R^2 = \frac{N}{N-1}\lambda_{WD}^2 . \tag{2.14}$$

Note that, for large training sample sizes, regularization parameters λ_R and λ_{WD} virtually coincide.

As the regularization parameter λ_R^2 increases, the relative effect of the sample estimate for the covariance matrix Σ in (2.12) decreases. In the limiting case, the influence of this estimate vanishes and the classifier approaches the NMC

$$g_{NMC}(\boldsymbol{x}) = \boldsymbol{x}'\hat{\boldsymbol{W}} + \hat{w}_0 ,$$

where $\hat{\boldsymbol{W}} = \bar{\boldsymbol{X}}^{(1)} - \bar{\boldsymbol{X}}^{(2)}$ and $\hat{w}_0 = -\frac{1}{2}(\bar{\boldsymbol{X}}^{(1)} + \bar{\boldsymbol{X}}^{(2)})'\hat{\boldsymbol{W}}$. The vector of weights in this expression is proportional to the distance between mean vectors of pattern classes. Thus, the regularization parameter controls the similarity of a classifier with the FLD and the NMC.

2.4 The Relationship between Weight Decay and Noise Injection in Linear SLP

Matsuoka [70] has shown that Gaussian noise injection to the training objects is equivalent to adding a supplementary weight decay term $\lambda_{WD}^2 \boldsymbol{W}'\boldsymbol{W}$ to the cost function of the linear SLP

$$Cost = \frac{1}{2}\frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N}[t_j^{(i)} - (\boldsymbol{W}'\boldsymbol{X}_j^{(i)} + w_0)]^2 \quad . \tag{2.15}$$

He has suggested using an additional term of the perceptron sensitivity in the cost function *Cost*, which was defined as the ratio of the mean square values of the perceptron disturbance caused by the noise injection to the training objects and the corresponding disturbance in the training objects. Applying a first order Taylor expansion to the sensitivity term [41] and using other approximations, he has demonstrated that minimizing the cost function *Cost* with Gaussian noise injection to the training objects is equivalent to minimizing the cost function with additional sensitivity term, which coinsides with the weight decay term for the linear SLP. This result was obtained for regression models. However, it is also valid for discriminant analysis.

Bishop [11] has generalized Matsuoka's result. In his theoretical study he has also used the Taylor expansion of the cost function. To get rid of high order terms in the Taylor expansion, he had to assume that the noise variance was small. He has shown that, under this condition, noise injection to the training objects in the perceptron training is equivalent to Tikhonov regularization of the cost function if the number of training objects is infinite. An [4] has proved that, for a finite number of training objects, adding noise to the training objects leads to a cost function with two penalty terms. One of them corresponds to a smoothing term as used in regularization theory, while the other depends on the fitting residuals. However, both results [11, 4] lead to the equivalence of Gaussian noise injection to the training objects and regularization by weight decay in the case of the linear SLP.

We will show now that, for the linear SLP, the same result can be obtained in an easier direct way without involving the Taylor expansion and without calculating derivatives for the back-propagation gradient descent method. Noise injection in perceptron training is usually performed [77, 112, 52] in such a way that, during each training sweep (epoch) *r*, the original training vectors $\boldsymbol{X}_j^{(i)}$ ($i = 1, 2$; $j = 1, \ldots, N$) are distorted by adding some noise $Z \sim N(0, \lambda_{NOISE}^2)$ to each component of the training vectors $\boldsymbol{X}_j^{(i)}$, i.e. $\boldsymbol{U}_{jr}^{(i)} = \boldsymbol{X}_j^{(i)} + \boldsymbol{Z}_{jr}^{(i)}$, where $\boldsymbol{Z}_{jr}^{(i)} \sim N(\boldsymbol{0}, \lambda_{NOISE}^2 \boldsymbol{I})$, ($r = 1, \ldots, R$), and *R* is the number of training epochs. Let us note, that this noise injection model is actually the same as the noise injection model presented in Section 2.2, in which the number of noise injections *R* is equal to the number of training epochs.

Let us now apply the noise injection model described above to the cost function of the linear SLP. Substituting the training vectors $\boldsymbol{X}_j^{(i)}$ in the cost function (2.15) by the "noisy" training vectors $\boldsymbol{U}_{jr}^{(i)}$ and considering the expectation of this function over noise *Z*, we obtain

$$Cost_{NOISE} = \frac{1}{2}\frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N} E_Z\left\{ [t_j^{(i)} - (\boldsymbol{W}'\boldsymbol{U}_{jr}^{(i)} + w_0)]^2 \right\} =$$

$$\frac{1}{2}\frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N} E_Z\left\{ [t_j^{(i)} - (\boldsymbol{W}'(\boldsymbol{X}_j^{(i)} + \boldsymbol{Z}_{jr}^{(i)}) + w_0)]^2 \right\} =$$

$$\frac{1}{2}\frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N} E_Z\left\{ [t_j^{(i)} - (\boldsymbol{W}'\boldsymbol{X}_j^{(i)} + w_0) - \boldsymbol{W}'\boldsymbol{Z}_{jr}^{(i)}]^2 \right\} =$$

$$\frac{1}{2}\frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N} [t_j^{(i)} - (\boldsymbol{W}'\boldsymbol{X}_j^{(i)} + w_0)]^2 - \frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N} E_Z\{ (t_j^{(i)} - (\boldsymbol{W}'\boldsymbol{X}_j^{(i)} + w_0))\boldsymbol{W}'\boldsymbol{Z}_{jr}^{(i)} \} +$$

$$\frac{1}{2}\frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N} E_Z\left\{ \boldsymbol{W}'\boldsymbol{Z}_{jr}^{(i)}\boldsymbol{Z}_{jr}^{(i)\prime}\boldsymbol{W} \right\} =$$

$$Cost + \frac{1}{2}\frac{1}{2N}\sum_{i=1}^{2}\sum_{j=1}^{N} \boldsymbol{W}' E_Z\{ \boldsymbol{Z}_{jr}^{(i)\prime}\boldsymbol{Z}_{jr}^{(i)} \}\boldsymbol{W} = Cost + \frac{1}{2}\lambda_{NOISE}^2 \boldsymbol{W}'\boldsymbol{W} = Cost_{WD} \ .$$

Thus, in expectation over noise *Z*, *Gaussian noise injection to the training objects is equivalent to weight decay* in the linear SLP training, and

$$\lambda_{WD}^2 = \lambda_{NOISE}^2 . \quad (2.16)$$

Combining the results, presented in the previous section and this one leads to the *asymptotic equivalence of Gaussian noise injection in the linear SLP design to weight decay regularization and/or to the RFLD*. This equivalence holds only for the linear SLP case. Note, that for the finite number of noise injections or the training epochs, *R*, the covariance matrix of noise vectors $\boldsymbol{Z}_{jr}^{(i)}$ is random. Therefore, when *R* is small, a worse classifier may be obtained. It means, that noise injection may perform worse than the built-in regularization techniques. In addition, according to (2.6), (2.14) and (2.16), the regularization parameters of considered regularization techniques relate each to other in the following way

$$\lambda_{NOISE}^2 = \lambda_{WD}^2 = \frac{N-1}{N}\lambda_R^2 . \quad (2.17)$$

2.5 The Expected Classification Error

As it has been shown above, all three techniques employed for stabilizing the classification rule are, in fact, equivalent. To estimate the quality of classification, usually some criterion is used. One of the most objective criteria for estimating the classification quality is the classification error P_N, which can be obtained after the learning of the decision rule and the classification of the test data sample according to this rule. The classification error P_N strongly depends on the training data set $\boldsymbol{X}_1^{(1)}, \boldsymbol{X}_2^{(1)}, \ldots, \boldsymbol{X}_N^{(1)}, \boldsymbol{X}_1^{(2)}, \boldsymbol{X}_2^{(2)}, \ldots, \boldsymbol{X}_N^{(2)}$. Due to the random nature of the training data set, the classification error P_N has also a random nature. By this reason, in many cases, it is useful to consider the expected classification error EP_N.

For the FLD the expected classification error, derived in [86], is equal to

$$\mathrm{EP}_N = \Phi\left\{-\frac{\delta}{2}\left[\frac{(n-3)(n-p-3)}{(n-p-2)(n-p-5)}\left[1+\frac{2}{n}\left(1+\frac{2p}{\delta^2}\right)+\frac{4}{n^2}\frac{p}{\delta^2}\right]+\frac{\delta^2}{2(n-p-5)}\right]^{-\frac{1}{2}}\right\} , \quad (2.18)$$

where $n=2N$ is the training sample size, $\delta^2 = (\mu_1-\mu_2)'\Sigma^{-1}(\mu_1-\mu_2)$ is the Mahalanobis distance between pattern classes π_1 and π_2, $\Phi\{u\} = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{u} e^{-\frac{t^2}{2}}dt$ is the standard cumulative Gaussian distribution $N(0,1)$. This formula shows the dependence of the expected classification error on the training sample size *n*, the data dimensionality *p* and the Mahalanobis distance δ^2 between pattern classes.

Formula (2.18) is quite complicated. To simplify analysis, we will consider the asymptotics of (2.18) for $n \to \infty$ and $p \to \infty$, that increase with the same rate. Furthermore, we will assume that δ^2 is constant, and it is not too small. Let us write (2.18) as

$$\mathrm{EP}_N = \Phi\left\{-\frac{\delta}{2}[A_1(1+B_1+B_2+B_3)+A_2]^{-\frac{1}{2}}\right\} \quad , \tag{2.19}$$

where $A_1 = \frac{(n-3)(n-p-3)}{(n-p-2)(n-p-5)}$, $A_2 = \frac{\delta^2}{2(n-p-5)}$, $B_1 = \frac{2}{n}$, $B_2 = \frac{4p}{n\delta^2}$, $B_3 = \frac{4}{n^2}\frac{p}{\delta^2}$. Obviously, terms B_1 and B_3 tend to zero when $n \to \infty$ and $p \to \infty$. Denoting $p = const \cdot n$, one obtains that $A_2 = \frac{\delta^2}{2n(1-const-5)}$ also tends to zero for $n \to \infty$. The term A_1 can be expressed in the following way

$$A_1 = \frac{(n-3)}{(n-p-2)}\frac{(n-p-3)}{(n-p-5)} = \left(\frac{n}{n-p-2} - \frac{3}{n-p-2}\right)\left(\frac{n-p-5}{n-p-5} + \frac{2}{n-p-5}\right) =$$

$$\left(\frac{1}{\frac{n-p}{n} - \frac{2}{n}} - \frac{3}{n-p-2}\right)\left(1 + \frac{2}{n-p-5}\right) \quad .$$

As terms $\frac{2}{n}$, $\frac{3}{n-p-2}$ and $\frac{2}{n-p-5}$ in A_1 tend to zero for $n \to \infty$ and $p \to \infty$, then $A_1 \to \frac{n}{n-p}$. Consequently, only two terms A_1 and B_2 are kept in (2.19). Thus, instead of (2.18) we can use the following asymptotic formula

$$\mathrm{EP}_N \to \Phi\left\{-\frac{\delta}{2}\frac{1}{\sqrt{\frac{n}{n-p}\left(1+\frac{4p}{n\delta^2}\right)}}\right\} \quad . \tag{2.20}$$

In [136] it has been shown that the accuracy of this expression is surprisingly high. Therefore, it seems to be reasonable to use a similar approach [86] and derive an approximate formula for the expected classification error that corresponds to the RFLD (2.3) [95]

$$\mathrm{EP}_N \to \Phi\left\{-\frac{\delta}{2}\frac{1}{\sqrt{\frac{n}{n-p}\left(1+\frac{4p}{n\delta^2}\right)}}\frac{\sqrt{1+\lambda_R^2 B}}{1+\lambda_R^2 C}\right\} \tag{2.21}$$

with $B = \frac{2}{1-y}\beta_2 + \frac{\mathrm{tr}\boldsymbol{D}^{-1}}{n}$, $C = \frac{1}{1-y}\beta_1$, $y = \frac{p}{n}$, $\beta_i = \frac{\boldsymbol{M}'\boldsymbol{D}^{-1}\boldsymbol{M}}{\delta^2}\alpha_i + \frac{\mathrm{tr}\boldsymbol{D}^{-1}}{n-p}$ $(i = 1, 2)$.

Here $\alpha_1 = 1$, $\alpha_2 = \frac{1+\frac{4\mathrm{tr}\boldsymbol{D}^{-1}}{n\boldsymbol{M}'\boldsymbol{D}^{-1}\boldsymbol{M}}}{1+\frac{4p}{n\delta^2}}$. $\boldsymbol{D} = \boldsymbol{T}\Sigma\boldsymbol{T}'$ is a diagonal matrix of the order $p \times p$, $\boldsymbol{T}$ is a orthonormal matrix of the order $p \times p$ and Σ is the common covariance matrix of pattern classes. $\boldsymbol{M}$ is a p-dimensional vector such that $\delta^2 = \boldsymbol{M}'\boldsymbol{M} = (\mu_1 - \mu_2)'\Sigma^{-1}(\mu_1 - \mu_2)$.

The details of deriving this formula can be found in [95].

It can be easily seen that, for $\lambda_R^2 \to 0$, expression (2.21) coincides with asymptotic formula (2.20) for the expected classification error of the standard FLD without ridge estimate of the covariance matrix. In such situation, if the covariance matrices of both classes are equal to each other, the expected classification error depends only on the number of features *p*, the training sample size n and the Mahalanobis distance δ^2 between the classes.

2.6 The Influence of the Regularization Parameter on the Expected Classification Error

Let us consider the RFLD (2.3) with the ridge estimate of the covariance matrix $\boldsymbol{S}_R$ and analyze the dependence of the expected classification error EP_N (2.21) upon the value of the regularization parameter λ_R^2 for the given training sample size *n*, data feature space dimensionality *p*, mean vectors μ_1 and μ_2, and covariance matrix Σ of the training data set.

Let us denote $\lambda = \lambda_R^2$ and write approximate formula (2.21) for the expected classification error EP_N in the form

$$\mathrm{EP}_N = \Phi\{U(\lambda)\} = \Phi\left\{A\frac{\sqrt{1+\lambda B}}{1+\lambda C}\right\}, \tag{2.22}$$

where $A = -\frac{\delta^2}{2}\sqrt{\frac{1-y}{\delta^2+\frac{4p}{n}}}$ and parameters *B*, *C* and *y* are specified in the previous section.

We will investigate the optimal value of the regularization parameter λ_{opt} and show that it has its unique minimum for the expected classification error EP_N.

In accordance with the definition of the Gaussian distribution function $\Phi\{\cdot\}$, we can write

$$\mathrm{EP}_N = \Phi\{U(\lambda)\} = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{U(\lambda)} e^{-\frac{t^2}{2}}dt \quad . \tag{2.23}$$

To find the minimum of the expected classification error EP_N as a function of the parameter λ, we should differentiate function (2.23) with respect to λ and equate the resulting expression to zero. Applying the Leibniz rule [62] for differentiating an integral in a parameter, we derive

$$\frac{\partial\Phi\{U(\lambda)\}}{\partial\lambda} = \frac{\partial}{\partial\lambda}\left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{U(\lambda)} e^{-\frac{t^2}{2}}dt\right) = \frac{1}{\sqrt{2\pi}}e^{-\frac{U(\lambda)^2}{2}}\frac{\partial U(\lambda)}{\partial\lambda} = 0 \quad . \tag{2.24}$$

For any λ, $e^{-\frac{U(\lambda)^2}{2}} > 0$ and $\left|\frac{U(\lambda)^2}{2}\right| < \gamma$, where $\gamma = const > 0$. Consequently, the above expression is equal to zero only when $\frac{\partial U(\lambda)}{\partial\lambda} = 0$. Then, differentiating the function $U(\lambda)$ with respect to λ,

$$\frac{\partial U(\lambda)}{\partial \lambda} = \frac{\partial}{\partial \lambda}\left(A\frac{\sqrt{1+\lambda B}}{1+\lambda C}\right) = A\frac{B-2C-\lambda BC}{2\sqrt{1+\lambda B}(1+\lambda C)^2} \tag{2.25}$$

and equating the resulting expression to zero, we determine the optimal value of the regularization parameter λ ,

$$\lambda_{opt} = \frac{B-2C}{BC} = \frac{1}{C}-\frac{2}{B} \ . \tag{2.26}$$

Calculating the second derivative of the function $\Phi\{U(\lambda)\}$ with respect to λ ,

$$\frac{\partial^2}{\partial \lambda^2}\Phi\{U(\lambda)\} = \frac{1}{\sqrt{2\pi}}e^{-\frac{U(\lambda)^2}{2}}\left[\frac{\partial^2 U(\lambda)}{\partial \lambda^2} - U(\lambda)\left[\frac{\partial U(\lambda)}{\partial \lambda}\right]^2\right] =$$

$$-\frac{A\left[2BC(1+\lambda B)(1+\lambda C)+(B-2C-\lambda BC)\left(B+4C+5\lambda BC+\frac{A^2(1+\lambda B)}{1+\lambda C}\right)\right]}{4\sqrt{1+\lambda B}(1+\lambda B)(1+\lambda C)^3}\frac{1}{\sqrt{2\pi}}e^{-\frac{A^2(1+\lambda B)}{(1+\lambda C)^2}} \ ,$$

verifying the sign of the second derivative at the optimum point λ_{opt} and assuming that $C>0$ and $B>C$, we find that

$$\frac{\partial^2}{\partial \lambda^2}\Phi\{U(\lambda_{opt})\} = -\frac{AB^3C\sqrt{C}}{8\sqrt{B-C}(B-C)^2}\frac{1}{\sqrt{2\pi}}e^{-\frac{A^2B^2}{4C(B-C)}} \geq 0 \ ,$$

because $A \leq 0$, $B>0$, $C>0$ and $e^{-\frac{A^2B^2}{4C(B-C)}} > 0$. Consequently, the derived optimal parameter λ_{opt} (2.26) corresponds to the minimum of the expected classification error EP_N . Thus, for given training sample size *n*, dimensionality *p* of the feature space, mean vectors μ_1 and μ_2 , and covariance matrix Σ , an optimal value of the regularization parameter

$$\lambda_{opt} = \frac{1}{C}-\frac{2}{B} = \frac{1-y}{\beta_1}-\frac{1}{\frac{1}{1-y}\beta_2+\frac{\mathrm{tr}\boldsymbol{D}^{-1}}{2n}} \tag{2.27}$$

exists and at this point the expected classification error EP_N reaches its unique minimum,

$$\mathrm{EP}_N^{(\min)} = \Phi\left\{\frac{AB}{2\sqrt{C(B-C)}}\right\} = \Phi\left\{-\frac{\delta^2}{4}\sqrt{\frac{1-y}{\delta^2+\frac{4p}{n}}}\frac{\frac{2}{1-y}\beta_2+\frac{\mathrm{tr}\boldsymbol{D}^{-1}}{n}}{\sqrt{\frac{1}{1-y}\beta_1\left[\frac{1}{1-y}(2\beta_2-\beta_1)+\frac{\mathrm{tr}\boldsymbol{D}^{-1}}{n}\right]}}\right\} , \tag{2.28}$$

where parameters *A*, *B*, *C*, *y*, β_1 and β_2 are specified above.

Both, the optimal value of the regularization parameter λ_{opt} and the minimal value of the expected classification error $\mathrm{EP}_N^{(\min)}$, depend on the number of training vectors *n* and on parameters δ^2 , *p*, $\boldsymbol{M}'\boldsymbol{D}^{-1}\boldsymbol{M}$ and $\mathrm{tr}\boldsymbol{D}^{-1}$, which describe true distributions of data classes. One can see, that when $n \to \infty$, then $y \to 0$, $\alpha_1 = \alpha_2$, $\beta_1 = \beta_2$ and, consequently, the optimal value of the regularization parameter tends to zero, $\lambda_{opt} \to 0$.

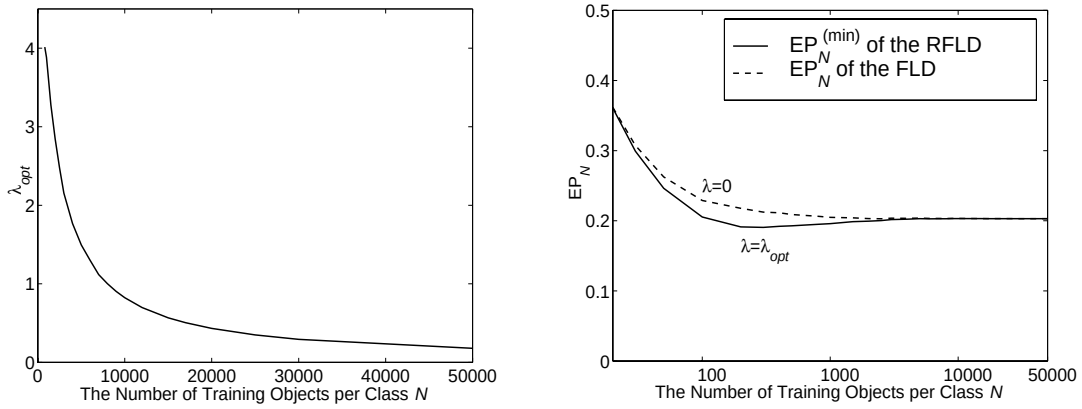


Fig. 2.2. Dependence of the optimal value of the regularization parameter λ_{opt} *and the minimal value of the expected classification error* $\mathrm{EP}_N^{(\min)}$ *on the training sample size N.*

Figure 2.2 shows the dependence of the optimal value of the regularization parameter λ_{opt} and the dependence of the minimal value of the expected classification error $\mathrm{EP}_N^{(\min)}$ on the number N of training vectors in each class calculated for p=30, δ^2 =2.7684, $\boldsymbol{M}'\boldsymbol{D}^{-1}\boldsymbol{M}$ = 0.1481 and $\mathrm{tr}\boldsymbol{D}^{-1}$= 28.17. These parameters are obtained on the 30-dimensional Gaussian correlated data set described in Chapter 1. One can see that in agreement with theoretical predictions, when increasing the number of training vectors, the optimal value of regularization parameter λ_{opt} tends to zero, and the minimal value of the expected classification error $\mathrm{EP}_N^{(\min)}$ tends to some constant value. It means that the larger the training data set, the closer its distribution to the true distribution of the data. When the training set is sufficiently large to represent quite precisely the true distribution of the data, regularization is not necessary any more. Let us note that, in the example considered, very large N is needed in order to make regularization useless. We should also remember, that formulas (2.27) and (2.28) are valid only for large training samples. Therefore, the analytical values obtained for λ_{opt} and $\mathrm{EP}_N^{(\min)}$ might be untrue, when the training sample size N is not sufficiently large.

2.7 The Experimental Investigation of the Fisher Linear Discriminant and a Linear Single Layer Perceptron

When deriving the analytical formula (2.21) for the expected classification error EP_N, some approximations [95] were used, and it was assumed that

- the dimensionality p of the feature space is large,
- the number n of the training objects is large,
- the distribution of the RFLD (2.3) is close to the Gaussian distribution,
- the regularization parameter λ_R^2 is small.

The derived formula (2.21) for the expected classification error EP_N is an asymptotic expression. Therefore, this formula is only valid for large dimensionalities *p* of the feature space and large training sample sizes *n*. Moreover, due to restrictions [95] imposed on the regularization parameter λ_R^2, it is difficult to apply this formula in practice. However, this expression is of considerable interest from the viewpoint of studying and analyzing the main features of the behaviour of the expected classification error EP_N and the optimal value of the regularization parameter λ. In addition, it has been theoretically demonstrated that the RFLD and the FLD with noise injection to the training objects are equivalent to a linear SLP with weight decay and/or to a linear SLP with noise injection to the training objects. Therefore, it would be of interest to check applicability of these theoretical predictions in practice, for the finite training sets. To test these predictions and to analyze theoretical conclusions, in our experimental investigations we have used artificially generated data described by the Gaussian distribution. The employed data consisted of thirty features that strongly depended on each other. They are 30-dimensional Gaussian correlated data, described in Chapter 1. Experiments were carried out for the RFLD (2.3) and the linear SLP (2.7) with an additive regularization term $\frac{1}{2}\lambda_{WD}^2 \boldsymbol{W}'\boldsymbol{W}$. Furthermore, we investigated the FLD (2.1) and a linear SLP with Gaussian noise injection to the training vectors. To get a linear SLP, we have used a non-linear SLP with the target values 0.48 and 0.52, that is actually very close to using the linear SLP. In the case of the FLD, normally distributed noise with zero mean and the covariance matrix $\lambda_{NOISE}^2 \boldsymbol{I}$ was injected to the training objects in such a way, that each data class consisted of the 100 noisy vectors (*NR*=100). As shown by Section 2.3, the long-duration training of a linear SLP with noise injection also yields a regularized FLD. In the latter case, during perceptron training in each sweep, normally distributed independent random quantities with zero means and variances λ_{NOISE} are added to each component of the training vectors. The duration of the perceptron training was 100 sweeps (epochs) for the both cases: the linear SLP with weight decay and the linear SLP with noise injection.

In all experiments, we used training sample sets with the number of the training objects *n*=5+5, *n*=10+10, *n*=20+20 and *n*=30+30. In each case, the remaining data set was used for testing. For each size of the training set, we repeated the experiments 50 times. Each time, we used a new independent random training set of the correspondent size, which remained unchanged for each of four classification rules. Figures 2.3-2.6 display experimental results averaged over 50 independent repetitions. These results illustrate the dependence of the expected classification error EP_N (we call it the generalization error) upon the regularization parameter for the RFLD, for the FLD with noise injection to the training objects, for a linear SLP with weight decay, and for a linear SLP with noise injection to the training objects. Note that, for the linear perceptron, we repeated the experiments for 3 different initializations of perceptron weights. We did not choose the best weight initialization from them (with respect to the minimum value of the cost function). Therefore, all presented results for the linear

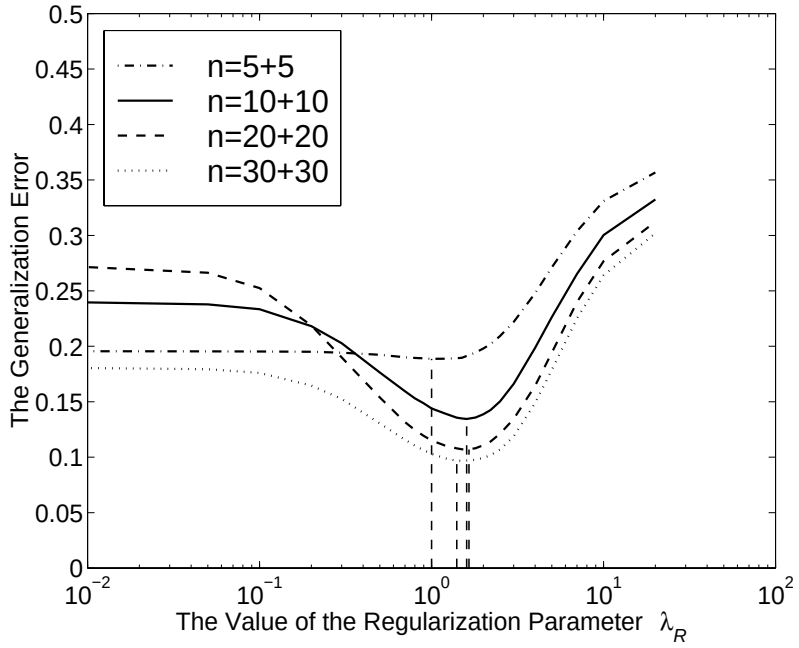


Fig. 2.3. Dependence of the expected classification error EP_N *on the regularization parameter* λ_R *for the Fisher linear discriminant function with the ridge estimate of the covariance matrix.*

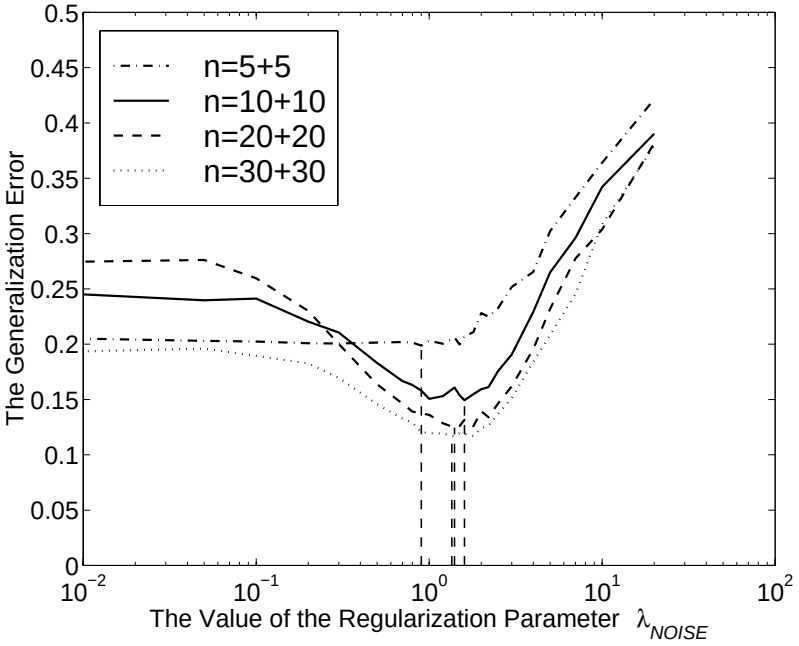


Fig. 2.4. Dependence of the expected classification error EP_N *on the regularization parameter* λ_{NOISE} *for the Fisher linear discriminant function with noise injection to the training objects.*

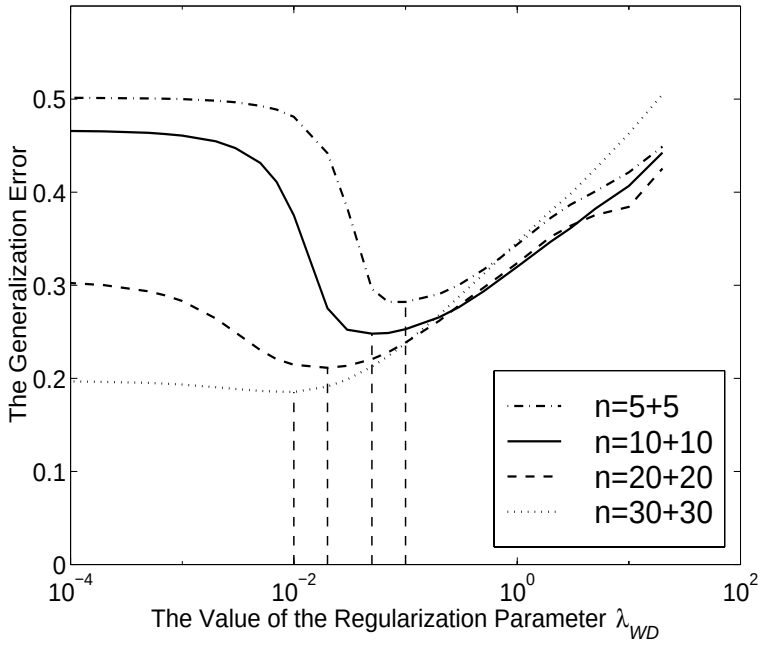


Fig. 2.5. Dependence of the expected classification error EP_N *on the regularization parameter* λ_{WD} *for a linear single layer perceptron with weight decay.*

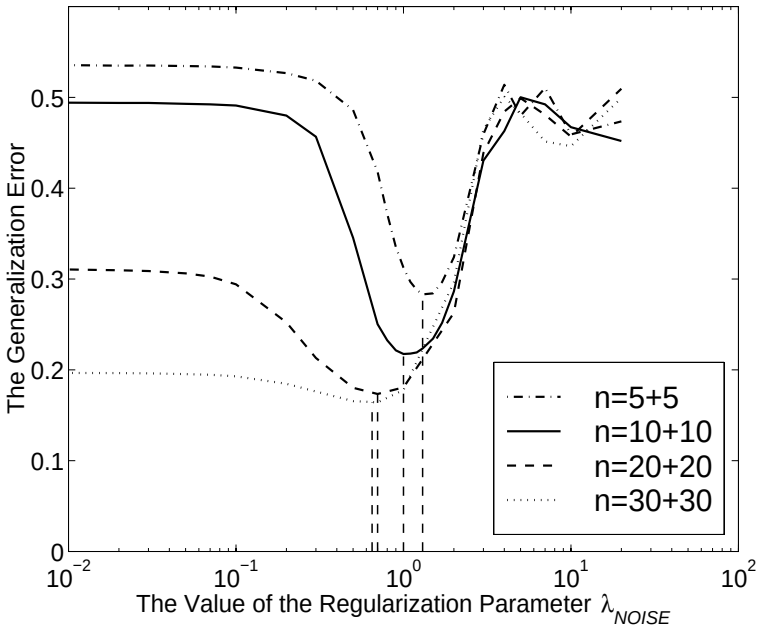


Fig. 2.6. Dependence of the expected classification error EP_N *on the regularization parameter* λ_{NOISE} *for a linear single layer perceptron with noise injection to the training vectors.*

perceptrons were additionally averaged over these 3 repetitions (in total, the averaging is performed over 3×50 independent experiments). The vertical dashed lines show the optimal values of the regularization parameter which give the minimum of the classification error obtained on the test set.

It can be easily seen that all the plots obtained for all above-discussed regularization techniques have much in common. All these plots feature a clearly pronounced minimum of the generalization error, which is shifted to the left with an increase in the training sample size. The exception is the FLD with the training sample size n=5+5. In this case, the generalization error does not have a clear pronounced minimum, having similar values for a large range of values of the regularization parameter. One can also notice, that, in spite of theoretically proved equivalence, plots obtained for different regularization techniques are not completely equivalent. This occurs because the finite training sets and a finite number of noise injections are used in our simulation study, while the formulas are asymptotic. For the same reason, noise injection performs slightly worse than the built-in regularization techniques, as a finite number of noise injections is used. Concerning the linear SLP case, we should note that, the training of a perceptron is an iterative procedure, which strongly depends on the weight initialization, on the learning step η and on the chosen number of training sweeps (epochs). It is very unstable and may sometimes result in a bad local minimum. We averaged results over three different initializations. Some of them may be disadvantageous. Therefore, plots obtained for the linear SLP are somewhat different from those obtained for the FLD. Note also that the optimal values of the parameter λ_{WD} differ considerably from those of the parameters λ_R and λ_{NOISE}, because λ_{WD} is the parameter that regularizes weights at each step of the iterative procedure of perceptron training. However, the similarity in the behaviour of all plots confirms that

- all above discussed regularization techniques perform similar to each other,
- the function that describes the dependence of the classification error on the regularization parameter exhibits a minimum, and
- the optimal value of the regularization parameter decreases with the growth in the training sample size.

2.8 The Experimental Investigation of Multilayer Nonlinear Perceptron

To provide a qualitative validation of theoretical predictions concerning the existence of the optimal value of the regularization parameter for nonlinear *multilayer perceptrons* (MLPs), we investigated a perceptron with a single hidden layer.

We studied an 8-8-2 perceptron with eight inputs, eight neurons in the hidden layer, and two outputs with targets values 0.1 and 0.9. We carried out experiments using artificially generated data that had eight features: the “circular” data set, described in Chapter 1. Three

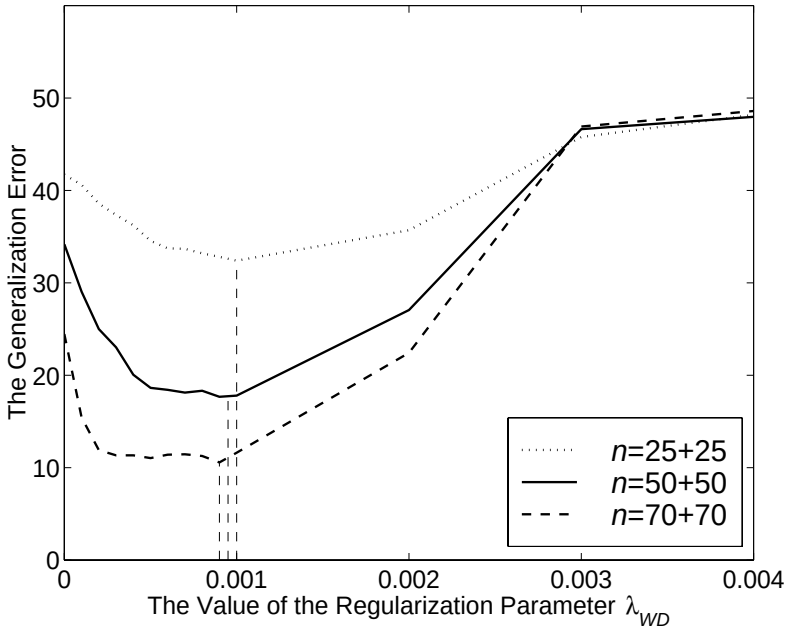


Fig. 2.7. Dependence of the expected classification error EP_N *on the value of the regularization parameter* λ_{WD} *for a nonlinear 8-8-2 perceptron with weight decay.*

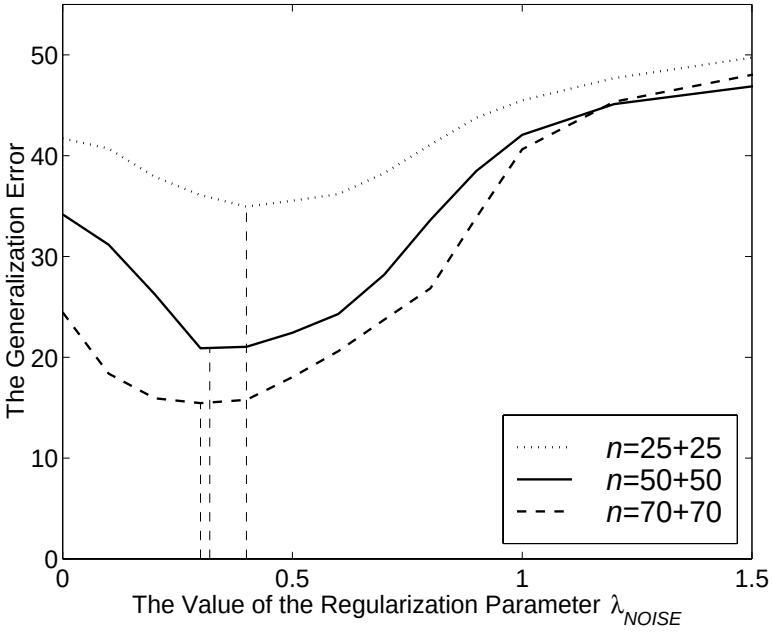


Fig. 2.8. Dependence of the expected classification error EP_N *on the value of the regularization parameter* λ_{NOISE} *for a nonlinear 8-8-2 perceptron with noise injection to the training vectors.*

series of experiments were performed. The number of the training vectors in these series of experiments were n=25+25, n=50+50 and n=70+70. In all these experiments the test set was fixed, with the size n_{TEST}=1000=500+500. Figures 2.7 and 2.8 present the average results obtained for seven independent random training sample sets and two different weight initializations. These investigations employed a nonlinear 8-8-2 perceptron with an additive weight decay term and a nonlinear 8-8-2 perceptron with Gaussian noise injection to the training objects, both trained during 220 sweeps. Similarly to a linear SLP, in the case under consideration, we can observe a strong dependence of the expected classification error EP_N upon the regularization parameter. One can also notice that the optimal value of the regularization parameter decreases with an increase in the training sample size n. The nonlinear MLP with weight decay performs similarly to the nonlinear MLP with noise injection to the training objects. This also supports theoretical results obtained by Bishop [11] and Leen [65].

2.9 Conclusions and Discussion

In certain cases, when the training sample size is small and the data dimensionality is large, the number of training vectors n may become comparable to the number of features p. In such a situation, the sample estimate of the covariance matrix $\boldsymbol{S}$ in the FLD becomes close to a degenerate matrix, and the inversion of this matrix encounters certain difficulties. Under these conditions, the expected classification error drastically increases. One possible method to overcome this difficulty involves using the ridge estimate of the covariance matrix. We demonstrated that adding an auxiliary regularization term $\frac{1}{2}\lambda_{WD}^2 \boldsymbol{W}'\boldsymbol{W}$ to the loss function in training a linear SLP with equal numbers of training vectors in each class, as well as Gaussian distributed noise injection to the training objects, is equivalent to applying the ridge estimate for the covariance matrix in the FLD.

We have analyzed the analytical approximate asymptotic formula derived by Raudys *et al*. [95] for the expected classification error EP_N of the RFLD, which depends on the regularization parameter λ, on the training sample size n, on the data dimensionality p, on the Mahalanobis distance δ^2, and on some other parameters describing true distributions of data classes. It has been demonstrated theoretically that, in general, there exists a unique optimal value of the regularization parameter. Note that, because of assumptions employed in deriving the approximate formula (2.21), this expression is valid only for feature spaces with large dimensions p, large training sample sizes n, and small values of the regularization parameter λ. Therefore, the corresponding quantitative estimates are applicable only to a limited range of parameters. However, this formula qualitatively illustrates general properties of the behaviour of the expected classification error EP_N and the optimal value of the regularization parameter λ_{opt}.

The simulation study performed for the RFLD, for the linear SLP with weight decay, for the FLD with Gaussian noise injection and for the linear SLP with Gaussian noise injection confirmed the equivalency of these regularizing techniques; supported the consistency of the derived formula; and demonstrated the existence of a unique optimal value of the regularization parameter, which tends to zero with an increase in the training sample size n. We also experimentally demonstrated that the optimal value of the regularization parameter exists for nonlinear MLPs with weight decay and for nonlinear MLPs with Gaussian noise injection to the training objects. Note that, in these cases, the optimal value of the regularization parameter also tends to zero with an increase in the training sample size n.

Chapter 3

Noise Injection to the Training Data

3.1 Introduction

In discriminant analysis one often has to face the *small training sample size problem*. This arises when the data feature space dimensionality is large compared with the number of available training objects. Sometimes large difficulties appear in constructing a discriminant function on small training sample sets, resulting in discriminant functions having a bad performance [54, 91]. In order to make a good choice for the classification rule or to judge the training sample size it is important to be familiar with the small sample properties of the sample-based classification rules. These properties can be characterized by the difference and/ or the ratio of the generalization error and the asymptotic probability of misclassification.

Small sample properties of statistical classifiers depend on their complexity and on the data dimensionality [23, 57, 85, 88]. For a data model with multivariate Gaussian distributed pattern classes having a common covariance matrix with r significant nonzero eigenvalues, it was shown that the generalization errors of the nearest mean classifier [84], the Parzen window classifier with a spherical Gaussian kernel [87, 90], and the zero empirical error classifier [92] depend on the true (intrinsic) data dimensionality r. Simulation experiments [92] have also confirmed that for this data model the small sample properties of the nonlinear single layer perceptron are not determined by the dimensionality of the feature space p but by the intrinsic dimensionality r.

A well-known technique to solve the small training sample size problem involves the generation of more training objects by *noise injection* (NI) to the training data [70, 10]. Usually, spherical Gaussian distributed noise is generated around each training object. In case of high dimensional data, however, it may happen that the intrinsic data dimensionality is smaller than the dimensionality of the feature space, in which the data are represented. The data are thereby located in a subspace of the feature space. Moreover, different measurements (features) may have large differences in scale due to their different nature. In this case, besides the variance of the added noise, also the directions of the noise become important. When spherical noise is injected to the data with low intrinsic dimensionality, noise is also added in the directions where no data are located. Thus spherical noise injection can distort the distribution of the training sample set and destroy the low intrinsic dimensionality of the data. Consequently, the small sample properties of the classifier will be not improved but will be degraded. The injection of Gaussian noise in only the direction of the k-nearest neighbours of an object may be more effective, as the local distribution of the training set is taken into

account (see Fig. 3.1). This noise will be called the *k-Nearest Neighbours* (*k*-NN) directed noise, whose basic idea is given in [24]. The *k*-NN abbreviation should not be confused with the *k*-NN classifier which is not studied or used in this thesis.

In this chapter we study the effectiveness of Gaussian noise and *k*-NN directed noise injection on the classifier performance. We also study the effect of the intrinsic dimensionality of data on the small sample size properties of classifiers and on the effectiveness of noise injection. Additionally, we consider the effect of the shape of the noise density function on the performance of noise injection and the role of the number of noise injections. Our theoretical study and the simulations will show that the variance of noise, the amount of noise injection and the directions of noise injection are important factors which determine the effectiveness for classifier training. When the training sample size is small, the shape of the noise density function may sometimes affect the performance of noise injection.

This chapter is based on our previously published works [114, 116, 120] and is organized as follows. At first, in Section 3.2 the Parzen window classifier and its small sample size properties are discussed. This is of interest as, by Gaussian noise injection to the training objects, we actually generate data according to he Parzen window density estimate of the data set, as is demonstrated in Section 3.3. Then, in Sections 3.3 and 3.4, we show that both, the number of noise injections and the intrinsic data dimensionality, affect the performance of noise injection. It is shown theoretically, for the case of statistical classification rules, why the *k*-NN directed noise injection is to be preferred above Gaussian noise injection in the case of low intrinsic dimensionalities. For the case of multilayer perceptrons a theoretical analysis involves nonlinearities and leads to very complicated algebra. Therefore, in Section 3.5, a simulation study is performed on the effectiveness of noise injection in multilayer perceptron training. In this section, the injection of *k*-NN directed noise and of the Gaussian distributed

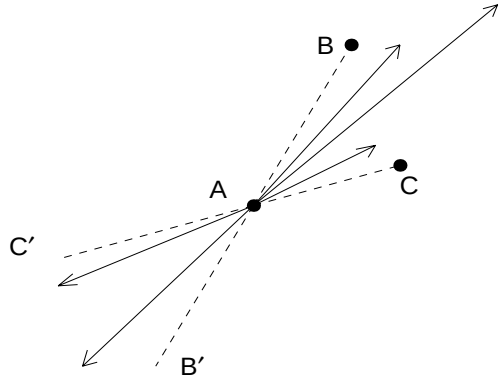


Fig. 3.1. The k-NN directed noise injection. Here, A is the object under consideration, B and C are two nearest neighbours to A. Gaussian distributed noise is added only inside of the angles $\angle BAC$ *and* $\angle B'AC'$.

noise is investigated and compared for a number of different data sets with high and low intrinsic dimensionality. In our simulations we have mainly used a perceptron with 3 hidden neurons trained by the Levenberg-Marquardt learning rule. We have chosen k=2 for the k-NN directed noise. This choice is well grounded in Section 3.5.1. The noise variances are optimized with respect to the smallest apparent error. Afterwards, in Section 3.6, with the example of the Parzen window classifier, we study the importance of the noise shape when injecting noise in small training sample sets. The effect of data scaling on the performance of different models of noise injection is considered in Section 3.7. Finally, in Section 3.8, the conclusions are given.

3.2 Parzen Window Classifier

The *Parzen Window Classifier* (PWC) is one of the most popular classification rules in discriminant analysis [74, 35]. It is known, that the PWC often performs better than parametric classifiers [93], because, when solving a real problem in statistical pattern recognition, the probability density function of the real data distribution is usually unknown and often not normal. Therefore, a discriminant function has to be constructed only on the basis of available observations $\boldsymbol{X}_j^{(i)}$, $j = 1, \ldots, N_i$, $i = 1, \ldots, \kappa$. Here N_i is the number of available training objects in the data class π_i, κ is the number of data classes. Let us consider the case of two classes π_1 and π_2 with an equal number of training objects in each data class N_1=N_2=N. Then the Parzen Window estimate of the conditional density function $f(\boldsymbol{x}|\pi_i)$ is

$$f_{PW}(\boldsymbol{x}|\pi_i) = \frac{1}{N}\sum_{j=1}^{N} K[H(\boldsymbol{x}, \boldsymbol{X}_j^{(i)})] \quad , \tag{3.1}$$

where $K(H)$ is called the kernel function or the window function, $H(\boldsymbol{x}, \boldsymbol{X}_j^{(i)})$ is a distance between vectors $\boldsymbol{x}$ and $\boldsymbol{X}_j^{(i)}$, $\boldsymbol{x}$ is an object under consideration and $\boldsymbol{X}_j^{(i)}$ is the jth object from the training sample set of the class π_i. Usually the normal density function with zero mean and the covariance matrix $\lambda_{PW}^2\boldsymbol{I}$, with $\boldsymbol{I}$ as the identity matrix, is used as the kernel function

$$K[H(\boldsymbol{x}, \boldsymbol{X}_j^{(i)})] = \frac{1}{(\sqrt{2\pi})^p \lambda_{PW}^p} \exp\left(-\frac{1}{2\lambda_{PW}^2}(\boldsymbol{x}-\boldsymbol{X}_j^{(i)})'(\boldsymbol{x}-\boldsymbol{X}_j^{(i)})\right) = N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{PW}^2\boldsymbol{I}) \quad . \tag{3.2}$$

Here p is the data dimensionality. The parameter λ_{PW} is called the *smoothing parameter*. It defines the weight of the input of each sample object $\boldsymbol{X}_j^{(i)}$ in the density estimate (3.1) at each separate point $\boldsymbol{x}$ of the multivariate space.

If the following conditions [35] for the kernel function are satisfied:

$$\int_{\Omega} K(\boldsymbol{z})d\boldsymbol{z} = 1 \quad , \tag{3.3}$$

$$\int_{\Omega} |K(z)|dz < \infty \,, \tag{3.4}$$

$$\sup_{z \in \Omega} |K(z)| < \infty \,, \tag{3.5}$$

$$\lim_{z \to \infty} |zK(z)| = 0 \,, \tag{3.6}$$

where $z = \dfrac{\boldsymbol{x} - \boldsymbol{X}_j^{(i)}}{\lambda_{PW}}$, and for the smoothing parameter λ_{PW}

$$\lim_{N \to \infty} \lambda_{PW}^{p} = 0 \,, \tag{3.7}$$

$$\lim_{N \to \infty} N\lambda_{PW}^{p} = \infty \,, \tag{3.8}$$

$$\lim_{N \to \infty} N\lambda_{PW}^{2p} = \infty \,, \tag{3.9}$$

then according to Parzen [74] the random function $f_{PW}(\boldsymbol{x}|\pi_i)$ converges to the real density function $f(\boldsymbol{x}|\pi_i)$, when $N \to \infty$. That is

$$\lim_{N \to \infty} \left| f(\boldsymbol{x}|\pi_i) - f_{PW}(\boldsymbol{x}|\pi_i) \right| = 0 \quad . \tag{3.10}$$

It appears that, if the training sample size $N \to \infty$, then the shape of the kernel function is not important. The asymptotic probability of misclassification, obtained for the sufficiently small fixed value of λ_{PW} and $N \to \infty$, is the same for different shapes of the kernel function. However, in practice *N* is a finite number. Therefore, for finite *N* the choice of the shape of the kernel function might be quite important. We consider this problem in Section 3.7.

If $\lambda_{PW} \to 0$, when $N \to \infty$, then the probability of misclassification (the classification error) P_N of the PWC approaches the Bayes error P_B [35].

For small values of the smoothing parameter $\lambda_{PW} \to 0$, the PWC approaches the *Nearest Neighbour Classifier* (NNC). For large values $\lambda_{PW} \to \infty$, the PWC with Gaussian kernel becomes the *Nearest Mean Classifier* (NMC) [87, 89].

If the conditions (3.3)-(3.6) for the kernel function $K(H)$ and the conditions (3.7)-(3.9) for the smoothing parameter λ_{PW} are satisfied, the estimate of the probability density function (3.1) $f_{PW}(\boldsymbol{x}|\pi_i)$ is asymptotically unbiased and consistent [35].

Let us now consider the small sample size properties of the PWC [90, 120]. The standard version of the PWC is based on sample estimates of the class conditional densities of the following form

$$f_{PW}(\boldsymbol{x}|\pi_i) = \frac{1}{N}\sum_{j=1}^{N} N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{PW}^2 \boldsymbol{I}) \quad . \tag{3.11}$$

At a fixed point $\boldsymbol{x}$ of the multivariate feature space Ω the value of the *Parzen window* (PW) distribution density estimate $f_{PW}(\boldsymbol{x}|\pi_i)$ depends on *N* random training vectors $\boldsymbol{X}_1^{(i)}, \boldsymbol{X}_2^{(i)}, \ldots, \boldsymbol{X}_N^{(i)}$. Therefore considering all possible training sets consisting of *N*

observations this density can be analysed as a random variable. When the training sample size N is still sufficiently large, the sum (3.11) of N random contribution terms $N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{PW}^2 \boldsymbol{I})$ approximates the Gaussian distribution (according to the central limit theorem). Thus, the conditional probability of misclassification $P(misclassification|\boldsymbol{x}, \boldsymbol{x} \in \pi_i) = P(f(\boldsymbol{x}|\pi_i) < f(\boldsymbol{x}|\pi_j)\big|\boldsymbol{x}, \boldsymbol{x} \in \pi_i)$ ($i = 1, 2$; $j = 3 - i$) at one particular point $\boldsymbol{x}$ (see Fig. 3.2) can be approximated by means E and variances V of the sample estimates of the class conditional densities $f(\boldsymbol{x}|\pi_1)$ and $f(\boldsymbol{x}|\pi_2)$ [87]

$$P(misclassification|\boldsymbol{x}, \boldsymbol{x} \in \pi_i) \approx \Phi\left\{ \frac{E\{f(\boldsymbol{x}|\pi_1)\} - E\{f(\boldsymbol{x}|\pi_2)\}}{\sqrt{V\{f(\boldsymbol{x}|\pi_1)\} + V\{f(\boldsymbol{x}|\pi_2)\}}} (-1)^i \right\}, \quad i = 1, 2, \quad (3.12)$$

where $\Phi\{u\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{u} e^{-\frac{t^2}{2}} dt$ is the standard cumulative Gaussian distribution $N(0, 1)$.

Let us consider the model of Gaussian data with a common covariance matrix (MGCC) with parameters μ_i and Σ. The conditional mean and the variance of the PW density estimate [35] (conditioned at a fixed point $\boldsymbol{x}$) *with respect to all possible training sets*, which consist of N observations, are

$$E\{f_{PW}(\boldsymbol{x}|\pi_i)\} = \frac{1}{N} \sum_{j=1}^{N} \int N(\boldsymbol{X}_j^{(i)}, \mu_i, \Sigma) N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{PW}^2 \boldsymbol{I}) d\boldsymbol{X}_j^{(i)} = N(\boldsymbol{x}, \mu_i, \Sigma + \lambda_{PW}^2 \boldsymbol{I}), \quad (3.13)$$

and

$$V\{f_{PW}(\boldsymbol{x}|\pi_i)\} = \frac{1}{N} \left[\frac{\left|2\Sigma + \lambda_{PW}^2 \boldsymbol{I}\right|^{1/2}}{\lambda_{PW}^p} \left(N(\boldsymbol{x}, \mu_i, 2\Sigma + \lambda_{PW}^2 \boldsymbol{I})\right)^2 - \left(E\{f_{PW}(\boldsymbol{x}|\pi_i)\}\right)^2 \right].$$

Let $\boldsymbol{T}$ be a $p \times p$ orthonormal matrix such that $\boldsymbol{T}\Sigma\boldsymbol{T}' = \boldsymbol{D}$ ($\boldsymbol{D}$ is a diagonal matrix of eigenvalues with elements $d_1, d_2, ..., d_p$). Then

$$V\{f_{PW}(\boldsymbol{x}|\pi_i)\} = \frac{1}{N} \left[\prod_{j=1}^{p} \sqrt{1 + \frac{2d_j}{\lambda_{PW}^2}} \left(N(\boldsymbol{x}, \mu_i, 2\Sigma + \lambda_{PW}^2 \boldsymbol{I})\right)^2 - \left(E\{f_{PW}(\boldsymbol{x}|\pi_i)\}\right)^2 \right]. \quad (3.14)$$

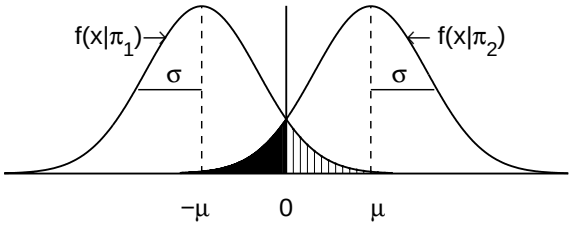


Fig. 3.2. Conditional densities $f(\boldsymbol{x}|\pi_i)$ and conditional probabilities of misclassification $P(misclassification|\boldsymbol{x}, \boldsymbol{x} \in \pi_i) = P(f(\boldsymbol{x}|\pi_i) < f(\boldsymbol{x}|\pi_j)\big|\boldsymbol{x}, \boldsymbol{x} \in \pi_i)$, $i = 1, 2$; $j = 3 - i$.

For $\lambda_{PW}^2 \to 0$ the variance of the PW density estimate is determined primarily by the term

$$\frac{1}{N}\prod_{j=1}^{p}\sqrt{1+\frac{2d_j}{\lambda_{PW}^2}}$$

This term decreases when the value of the smoothing parameter λ_{PW}^2 or/and the training sample size N increases. Let the eigenvalues of the covariance matrix Σ be equal: $d_1 = d_2 = \ldots = d_p = d$ and assume that the number of features p is increased. Then in order to keep the variance (3.14) (and also the misclassification error (3.12)) constant, the training sample size N should increase exponentially with p

$$N \equiv \left(1+\frac{2d}{\lambda_{PW}^2}\right)^{p/2} \tag{3.15}$$

Let us now assume that several eigenvalues of the covariance matrix Σ are very small $d_1 = d_2 = \ldots = d_r = d$, $d_{r+1} = d_{r+2} = \ldots = d_p = \varepsilon \to 0$. We call the number r *the intrinsic dimensionality* of the data for the MGCC model. For this data model we have instead of (3.15)

$$N \equiv \left(1+\frac{2d}{\lambda_{PW}^2}\right)^{r/2} \tag{3.16}$$

It means that *small training set properties of the Parzen window density estimate (3.11) are not determined by the formal data dimensionality p, but by the true - the intrinsic dimensionality r.* Thus the number of training vectors N required to design this classifier should increase exponentially with r.

With an increase in λ_{PW} the bias of the PW density estimate increases, the variance, however, decreases. Therefore in order to minimize the classification error one needs to find an optimal value of the smoothing parameter λ_{PW}.

3.3 Gaussian Noise Injection and the Parzen Window Estimate

Let us now consider a noise injection model in which we have N training pattern vectors $\boldsymbol{X}_j^{(i)}$ ($i = 1, 2$; $j = 1, \ldots, N$) from each of two pattern classes π_1 and π_2, and R independent p-variate random Gaussian noise vectors $\boldsymbol{Z}_{jr}^{(i)} \sim N(\boldsymbol{0}, \lambda_{NOISE}^2 \boldsymbol{I})$ ($r = 1, \ldots, R$) with zero mean and covariance matrix $\Sigma_N = \lambda_{NOISE}^2 \boldsymbol{I}$, which are added to each training vector $\boldsymbol{X}_j^{(i)}$. As a result, $2NR$ training vectors $\boldsymbol{U}_{jr}^{(i)} = \boldsymbol{X}_j^{(i)} + \boldsymbol{Z}_{jr}^{(i)}$ are obtained.

We will now demonstrate that by Gaussian noise injection to the training objects one actually generates data according to the Parzen window density estimate. In order to show this, let us consider the Rozenblatt's generalized histogram density estimate [104] constructed on the data $\boldsymbol{U}_{jr}^{(i)}$ enriched by noise and compare it with the Parzen window density estimate constructed on the original training vectors $\boldsymbol{X}_j^{(i)}$. In the generalized histogram approach one

calculates a number $n_i(\boldsymbol{x})$ of training vectors from the i-th class, which can be found in a neighbourhood $\Omega(\boldsymbol{x})$ of the vector $\boldsymbol{x}$. Let us define the neighbourhood $\Omega(\boldsymbol{x})$ by a hypercube with a width h and volume h^p, and let us analyse the generalized histogram classifier trained by $2NR$ "noisy" training vectors $\boldsymbol{U}_{jr}^{(i)}$. Then the density estimate is $f_{GH}(\boldsymbol{x}|\pi_i) = \frac{n_i(\boldsymbol{x})}{NRh^p}$, and the classification will perform according to the numbers $n_1(\boldsymbol{x})$ and $n_2(\boldsymbol{x})$ of the training vectors from classes π_1 and π_2, falling into the cell $\Omega(\boldsymbol{x})$. The number $n_i(\boldsymbol{x}) = \sum_{j=1}^{N} n_{ij}(\boldsymbol{x})$, where $n_{ij}(\boldsymbol{x})$ is the number of "noisy" vectors $\boldsymbol{U}_{jr}^{(i)}$ (r=1, 2, ..., R) generated around the training vector $\boldsymbol{X}_j^{(i)}$, falling into the cell $\Omega(\boldsymbol{x})$. The probability of a "noisy" vector falling into the cell $\Omega(\boldsymbol{x})$ with volume h^p is $P_{ij}(h,p) = N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I})h^p$. The numbers $n_{ij}(\boldsymbol{x})$ are random. For very small values of h the numbers $n_{ij}(\boldsymbol{x})$ are independent binomial random variables with mean $P_{ij}(h,p)R$ and with variance $P_{ij}(h,p)(1-P_{ij}(h,p))R \approx P_{ij}(h,p)R$. Thus, the expectation of the density estimate for the generalized histogram classifier with noise injection is

$$E\{f_{GH}(\boldsymbol{x}|\boldsymbol{X}_1^{(i)}, \boldsymbol{X}_2^{(i)}, \ldots, \boldsymbol{X}_N^{(i)}, \pi_i)\} = E\left\{\frac{n_i(\boldsymbol{x})}{NRh^p}\right\} = \frac{1}{NRh^p} E\{n_i(\boldsymbol{x})\} =$$

$$\frac{1}{NRh^p} E\left\{\sum_{j=1}^{N} n_{ij}(\boldsymbol{x})\right\} = \frac{1}{NRh^p} \sum_{j=1}^{N} E\{n_{ij}(\boldsymbol{x})\} = \frac{1}{NRh^p} \sum_{j=1}^{N} P_{ij}(h,p)R = \frac{1}{Nh^p} \sum_{j=1}^{N} P_{ij}(h,p) =$$

$$\frac{1}{Nh^p} \sum_{j=1}^{N} N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I})h^p = \frac{1}{N} \sum_{j=1}^{N} N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I}) = f_{PW}(\boldsymbol{x}|\pi_i) \quad .$$

The variance of the density estimate $f_{GH}(\boldsymbol{x}|\pi_i)$ can be calculated as follows:

$$V\{f_{GH}(\boldsymbol{x}|\boldsymbol{X}_1^{(i)}, \boldsymbol{X}_2^{(i)}, \ldots, \boldsymbol{X}_N^{(i)}, \pi_i)\} = V\left\{\frac{n_i(\boldsymbol{x})}{NRh^p}\right\} = \frac{1}{N^2R^2h^{2p}} V\{n_i(\boldsymbol{x})\} =$$

$$\frac{1}{N^2R^2h^{2p}} V\left\{\sum_{j=1}^{N} n_{ij}(\boldsymbol{x})\right\} = \frac{1}{N^2R^2h^{2p}} \sum_{j=1}^{N} V\{n_{ij}(\boldsymbol{x})\} \approx \frac{1}{N^2R^2h^{2p}} \sum_{j=1}^{N} P_{ij}(h,p)R =$$

$$\frac{1}{N^2Rh^{2p}} \sum_{j=1}^{N} N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I})h^p = \frac{1}{N^2Rh^p} \sum_{j=1}^{N} N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I}) = \frac{1}{NRh^p} f_{PW}(\boldsymbol{x}|\pi_i),$$

where the smoothing parameter $\lambda_{PW} = \lambda_{NOISE}$.

We see that, in expectation, the density estimate $f_{GH}(\boldsymbol{x}|\pi_i)$ of the generalized histogram classifier with Gaussian noise injection is equal to the Parzen window density estimate $f_{PW}(\boldsymbol{x}|\pi_i)$. When $R \to \infty$, $h \to 0$ and $N^2Rh^p \to \infty$, the variance of the density estimate $f_{GH}(\boldsymbol{x}|\pi_i)$ approaches zero. Therefore, for $R \to \infty$, $h \to 0$ and $N^2Rh^p \to \infty$, *the histogram approach with noise injection results in a density estimate, which is equivalent to the Parzen window estimate.*

Consequently, for the Parzen window estimate we have

$$\frac{E\{f_{PW}(\boldsymbol{x}|\pi_i)\}}{\sqrt{V\{f_{PW}(\boldsymbol{x}|\pi_i)\}}} = \frac{E\{f_{GH}(\boldsymbol{x}|\pi_i)\}}{\sqrt{V\{f_{GH}(\boldsymbol{x}|\pi_i)\}}} \approx \left(Rh^p \sum_{j=1}^{N} N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I}) \right)^{1/2} . \qquad (3.17)$$

When R is finite, there is an additional random factor which increases the variance of the class conditional density estimate. This can degrade the small sample properties of the classifier.

In Chapter 2, we have shown that asymptotically Gaussian noise injection to the training objects in the *Fisher's linear discriminant* function (FLD) is equivalent to the regularized FLD. However, when the number of noise injections R is finite, the FLD constructed on the "noisy" training data $\boldsymbol{U}_{jr}^{(i)}$ will differ from the regularized FLD constructed on the original training objects $\boldsymbol{X}_j^{(i)}$. For small R a random nature of noise vectors $\boldsymbol{Z}_{jr}^{(i)}$ may distort the distribution of the training vectors $\boldsymbol{X}_j^{(i)}$ and increase the generalization error.

Consequently, one can conclude that *the number of noise injections R strongly affects the effectiveness of the noise injection technique in the training of statistical classifiers.*

3.4 The Effect of the Intrinsic Data Dimensionality

In order to analyse the effect of the intrinsic data dimensionality on the effectiveness of noise injection, let us study the histogram classifier with noise injection to the training objects and compare it with the Parzen window estimate.

Suppose the intrinsic dimensionality is small in the neighbourhood $\Omega(\boldsymbol{x})$, i.e. the nearest training vector $\boldsymbol{X}_j^{(i)}$ of the vector $\boldsymbol{x}$ is in a subspace $S(\boldsymbol{x})$ with dimensionality r. The neighbourhood $\Omega(\boldsymbol{x})$ of the vector $\boldsymbol{x}$ is defined as a region, where $N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I}) > \varepsilon$ with sufficiently small value of ε. The intrinsic dimensionality r is defined by assuming that the $s = p - r$ last components of vectors $\boldsymbol{Y}_j^{(i)} = \boldsymbol{T}(\boldsymbol{x})(\boldsymbol{x} - \boldsymbol{X}_j^{(i)}) = (Y_{j1}^{(i)}, Y_{j2}^{(i)}, \ldots, Y_{jr}^{(i)}, Y_{jr+1}^{(i)}, \ldots, Y_{jp}^{(i)})'$ are equal or very close to zero. $\boldsymbol{T}(\boldsymbol{x})$ is an orthonormal transformation matrix.

Under the subspace assumption the vector $\boldsymbol{x}$ to be classified is located in the subspace $S(\boldsymbol{x})$, where the last s components of the vector $\boldsymbol{y} = \boldsymbol{T}(\boldsymbol{x})\boldsymbol{x}$ are equal or very close to zero. Let us denote $\boldsymbol{Y}_{j1}^{(i)} = (Y_{j1}^{(i)}, Y_{j2}^{(i)}, \ldots, Y_{jr}^{(i)})'$, $\boldsymbol{Y}_{j2}^{(i)} = (Y_{jr+1}^{(i)}, \ldots, Y_{jp}^{(i)})'$, and $\boldsymbol{y}_1 = (y_1, y_2, \ldots, y_r)'$, $\boldsymbol{y}_2 = (y_{r+1}, \ldots, y_p)'$. Then

$$P_{ij}(h, p) = N(\boldsymbol{x}, \boldsymbol{X}_j^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I}) h^p = N(\boldsymbol{y}_1, \boldsymbol{Y}_{j1}^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I}) N(\boldsymbol{y}_2, \boldsymbol{Y}_{j2}^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I}) h^p .$$

Under the subspace assumption $\boldsymbol{y}_2 - \boldsymbol{Y}_{j2}^{(i)} \approx 0$. Thus,

$$P_{ij}(h, p) = \left(\frac{h}{\sqrt{2\pi}\lambda_{NOISE}} \right)^s h^r N(\boldsymbol{y}_1, \boldsymbol{Y}_{j1}^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I}) .$$

Consequently, we obtain

$$E\{f_{GH}(\boldsymbol{x}|\boldsymbol{X}_1^{(i)}, \boldsymbol{X}_2^{(i)}, \ldots, \boldsymbol{X}_N^{(i)}, \pi_i)\} = \frac{1}{N}\left(\frac{1}{\sqrt{2\pi}\lambda_{NOISE}}\right)^s \sum_{j=1}^{N} N(\boldsymbol{y}_1, \boldsymbol{Y}_{j1}^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I})$$

and

$$V\{f_{GH}(\boldsymbol{x}|\boldsymbol{X}_1^{(i)}, \boldsymbol{X}_2^{(i)}, \ldots, \boldsymbol{X}_N^{(i)}, \pi_i)\} \approx \frac{1}{N^2 R h^p}\left(\frac{1}{\sqrt{2\pi}\lambda_{NOISE}}\right)^s \sum_{j=1}^{N} N(\boldsymbol{y}_1, \boldsymbol{Y}_{j1}^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I}) \quad .$$

Following (3.12), in the analysis of the conditional generalization error $P(misclassification|\boldsymbol{x}, \boldsymbol{x} \in \pi_i)$ we are interested in the ratio $\frac{E\{f_{GH}(\boldsymbol{x}|\pi_i)\}}{\sqrt{V\{f_{GH}(\boldsymbol{x}|\pi_i)\}}}$. Under the subspace assumption we obtain

$$\frac{E\{f_{GH}(\boldsymbol{x}|\pi_i)\}}{\sqrt{V\{f_{GH}(\boldsymbol{x}|\pi_i)\}}} \approx \left(R^* h^r \sum_{j=1}^{N} N(\boldsymbol{y}_1, \boldsymbol{Y}_{j1}^{(i)}, \lambda_{NOISE}^2 \boldsymbol{I})\right)^{1/2} , \tag{3.18}$$

where $R^* = \left(\frac{h}{\sqrt{2\pi}\lambda_{NOISE}}\right)^s R$. The parameter R^* is called the *effective* number of noise injections. Comparing (3.18) with (3.17) indicates that for $h<\lambda_{NOISE}$ the factor $\left(\frac{h}{\sqrt{2\pi}\lambda_{NOISE}}\right)^s$ reduces the influence of the number of noise injections R. Therefore, the generalization error of the histogram classifier with noise injection increases in comparison with the Parzen window classifier. In order to reduce the generalization error *we need to increase R*. An alternative way is to use instead of spherical noise the "directed" ("singular") noise with a small or zero value of λ_{NOISE} in directions of zero eigenvalues of the conditional covariance matrix $\boldsymbol{S}_N(\boldsymbol{x})$ in the nearest neighbourhood of the vector $\boldsymbol{x}$. Small values of λ_{NOISE} in $s = p - r$ directions increase $\left(\frac{h}{\sqrt{2\pi}\lambda_{NOISE}}\right)^s$ and do not destroy the intrinsic dimensionality of the data. One of the possibilities is to estimate the covariance matrix $\boldsymbol{S}_N(\boldsymbol{x})$ and to use its eigenvectors to determine the directions of noise injection. This approach has been used in a prediction algorithm ZET [138]. An alternative is to use *k*-NN noise injection as suggested by Duin [24].

The approach of *k*-NN directed noise injection consists in generating noise only in the direction of the *k*-nearest neighbours from the same pattern class of the object under consideration (see Fig. 3.1). Let $\boldsymbol{X}_j^{(i)}$ be an object under consideration and $\boldsymbol{Q}_{j1}^{(i)}, \boldsymbol{Q}_{j2}^{(i)}, \ldots, \boldsymbol{Q}_{jk}^{(i)}$ be its *k* nearest neighbours from the same pattern class. We determine the *k*-NN directed noise vectors as $\boldsymbol{Z}_{jr}^{(i)} = \lambda \times \frac{1}{k} \sum_{l=1}^{k} \xi_l (\boldsymbol{X}_j^{(i)} - \boldsymbol{Q}_{jl}^{(i)})$, where $\xi_l \sim N(0, 1)$, *k* is the number of the nearest neighbours used and λ is a scaling parameter.

The idea of using *k*-NN directed noise is based on the assumption that in generating

noise around a training object, it is useful to take into consideration the distribution of the *k*-nearest neighbours of this object, especially in the case of a low intrinsic data dimensionality, when data objects lie in the subspace of the feature space. The *k*-NN directed noise injection makes the training data set more complete in a somewhat different way than bootstrapping [26] does (no data copies are made). It does not destroy the low intrinsic data dimensionality, as the new objects are generated in the subspace determined by the *k*-nearest neighbours. Therefore, in comparison with spherical Gaussian noise injection it is less likely that the *k*-NN directed noise injection distorts the training set. In addition, the *k*-NN directed noise is more "economical": one adds noise only in useful directions, saving computer time and memory. Another advantage of the *k*-NN directed noise in comparison with ZET and similar algorithms, based on the normal data distribution assumption, is that it does not depend on knowledge of the data distribution. Therefore, the *k*-NN directed noise injection could be successfully applied to the data of any distribution (e.g. not normally distributed data or clustered data) if it is known that intrinsic data dimensionality is much smaller than the feature size.

Thus the conclusion follows: *the directions of noise injection as well as the number of noise injections R are very important factors which determine the effectiveness of the noise injection technique in statistical classifiers training.*

3.5 Noise Injection in Multilayer Perceptron Training

Noise injection (NI), when noise vectors are normally distributed $N(\mathbf{0}, \lambda^2 \boldsymbol{I})$, is equivalent to the ridge estimate of the covariance matrix in the standard FLD [95], as well as to weight decay in linear SLP training [70, 11, 65] (see also Chapter 2) and is similar to the smoothing in the Parzen window classifier. Noise injection helps to fill in gaps between training objects, smoothing the generalization error and stabilizing the training procedure [95, 117, 106, 42]. In the previous section it was shown that the intrinsic data dimensionality influences the effectiveness of noise injection for statistical classifiers. When the intrinsic data dimensionality r is small, more noise injections R are required in order to reduce the generalization error. One way to reduce R is to add noise only in useful directions (the *k*-NN directed noise injection), ignoring the directions where no data are located. It is reasonable to suppose that similar effects are also valid for MLP's. As theoretical analysis for MLPs requires multiple approximations and leads to tedious algebra, a simulation is used in order to confirm for MLPs the conclusions made for statistical classifiers. At first, in Section 3.5.1, the role of the number of the nearest neighbours k in *k*-NN directed noise is studied. Then, in Sections 3.5.2 and 3.5.3, we investigate the effect of intrinsic dimensionality r and of the number of noise injections R on the efficiency of noise injection in MLP training.

Four artificial data sets, described in Chapter 1, are used for our experimental investigations. They are 8-dimensional banana-shaped data, 8-dimensional banana-shaped data

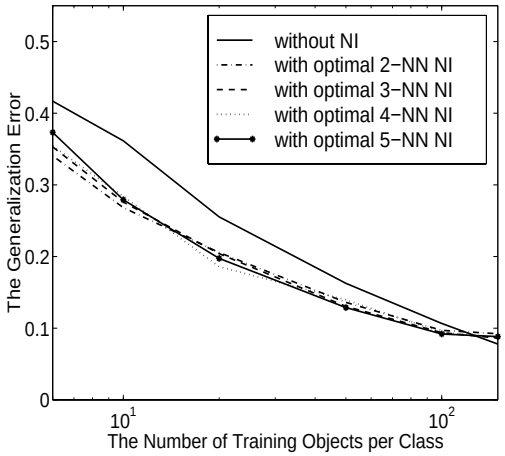
with the intrinsic dimensionality of two, 3-dimensional sinusoidal data and 12-dimensional uniformly distributed data. The last three data sets are concentrated in a subspace of the feature space, because we are interested in data sets with low intrinsic dimensionality. In our simulations we use the perceptron with three hidden units which is trained by the Levenberg-Marquardt learning rule [10] (see also Chapter 1). All results are averaged over 10 independent training sets and 5 different initializations (50 independent experiments in total). Noise injection on the training set is performed in such a way that in each training sweep the training data set is exchanged by R noise vectors generated from the original training set vectors. In our experiments we choose R=100, for each training set size. The variance for Gaussian noise injection and the scaling parameter λ for k-NN directed noise injection are optimized separately on each training data set with respect to the smallest apparent error.

3.5.1 The Role of the Number of Nearest Neighbours in *k*-NN Noise Injection

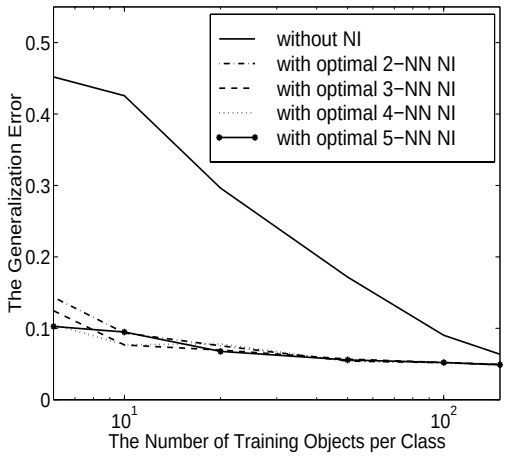
In order to study the influence of the number of nearest neighbours k on the efficiency of k-NN directed noise injection, we consider the performance of the multilayer perceptron with k-NN directed noise injection to the training set using the number of nearest neighbours k = 2, 3, 4, 5. The averaged and optimized results are presented in Fig. 3.3a,b,c,d for 8-dimensional banana-shaped data with high intrinsic dimensionality, for 8-dimensional banana-shaped data with low intrinsic dimensionality, for 3-dimensional sinusoidal data and for 12-dimensional uniform distributed data, respectively. The standard deviations of the mean generalization error vary approximately from 0.015 to 0.003, from 0.02 to 0.001, from 0.015 to 0.013 and from 0.025 to 0.0003, when increasing the training sample size, for all data sets mentioned above, respectively.

Surprisingly, for all data sets, the results obtained for different numbers (k = 2, 3, 4, 5) of nearest neighbours in k-NN noise injection are similar. Rather often small training sets misrepresent the distribution of the entire data set. Any training object could be misleading, giving wrong directions for Gaussian noise injection. Appending an additional nearest neighbour could worsen the situation. On the other hand, large training sets represent the distribution of the entire data set more accurately. Thus, two nearest neighbours of the object are the same representative as three or more nearest neighbours. In any case the directions for Gaussian noise injection are defined correctly. *The effectiveness of the k-NN directed noise injection does not depend on the number k of nearest neighbours*. However, it should be realized that our method contains a built-in optimization of the variance of noise, which may compensate for a possible influence of k.

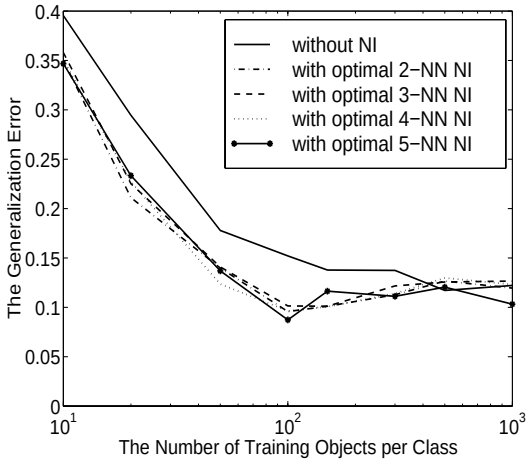
As the effectiveness of the k-NN directed noise injection does not depend on the number k of nearest neighbours, we have chosen k=2 for k-NN directed noise injection in the remaining experiments.



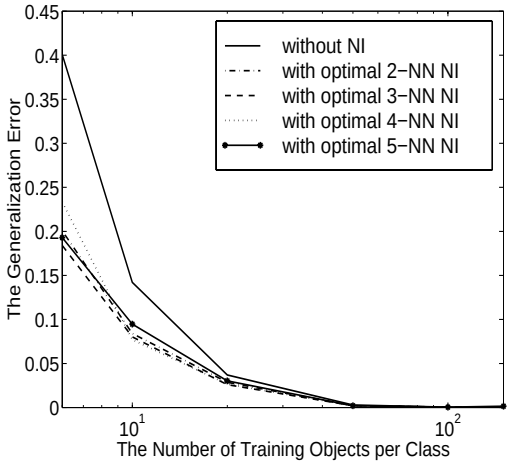
a) for 8-dimensional banana-shaped data with high intrinsic dimensionality



b) for 8-dimensional banana-shaped data with low intrinsic dimensionality



c) for 3-dimensional sinusoidal data with an intrinsic dimensionality of two



d) for 12-dimensional uniformly distributed data with an intrinsic dimensionality of two

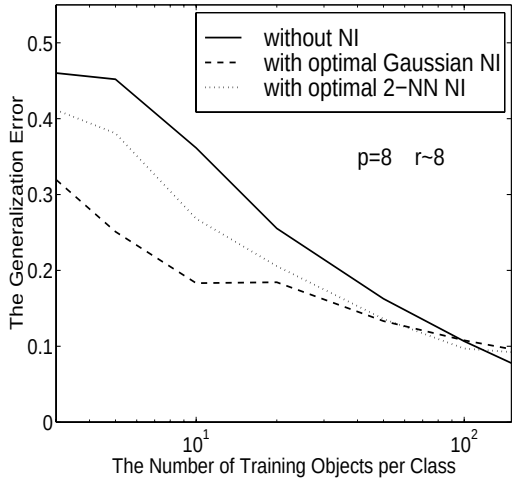
Fig. 3.3. The generalization error of the perceptron with three hidden neurons with k-NN noise injection versus the number of training objects per class.

3.5.2 The Effect of the Intrinsic Dimensionality on Noise Injection

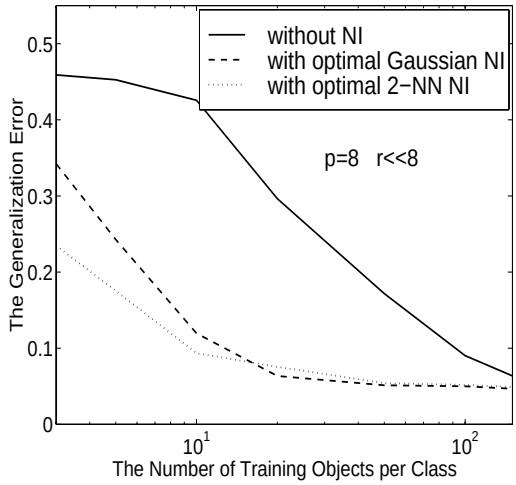
Let us consider the performance of the multilayer perceptron without noise injection, with Gaussian noise injection and with 2-NN directed noise injection. The averaged and optimized results are presented in Fig. 3.4a,b,c,d for all four considered data sets.

By considering two sets of 8-dimensional banana-shaped data (see Fig. 3.4a,b), one with high and another one with low intrinsic dimensionality, it can be seen that the intrinsic data dimensionality influences, both, the classifier performance and the effectiveness of noise

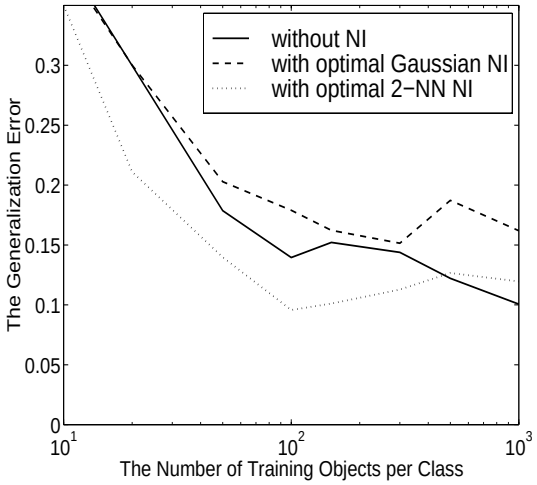
injection. When the training data set is small and situated in a subspace of the feature space, the perceptron tries to build the discriminant function in the feature space without taking into account the intrinsic data dimensionality. For this reason, the number of local minima of the pattern cost function grows together with the chance to be trapped into the bad local minima (with a high generalization error). Therefore, the low intrinsic dimensionality of the data can cause a poorer performance of the perceptron. As noise injection smoothes the surface of the perceptron cost function, one gets less deep (shallower) local minima and a more stable



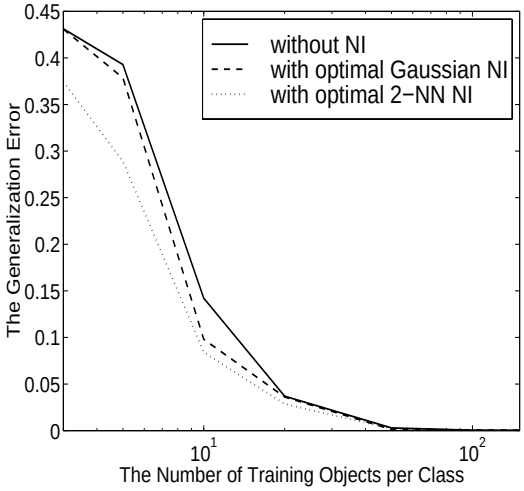
a) for 8-dimensional banana-shaped data with high intrinsic dimensionality



b) for 8-dimensional banana-shaped data with low intrinsic dimensionality



c) for 3-dimensional sinusoidal data with an intrinsic dimensionality of two



d) for 12-dimensional uniformly distributed data with an intrinsic dimensionality of two

Fig. 3.4. The generalization error of the perceptron with three hidden neurons without noise injection, with Gaussian noise injection and with 2-NN noise injection versus the number of training objects per class.

classifier. The small sample size properties of such a classifier depend on the intrinsic data dimensionality. The perceptron with noise injection has a better performance for the data with the smaller intrinsic dimensionality. In other words, *when the intrinsic data dimensionality is smaller, a smaller training set size is needed to achieve by noise injection the same results as for the data with high intrinsic dimensionality.*

It can also be observed that the multilayer perceptron with *2*-NN directed noise injection outperforms the multilayer perceptron with Gaussian noise injection for the data with low intrinsic dimensionality for the small sample size (see Fig. 3.4b,c,d). This does not happen for the data with high intrinsic dimensionality (see Fig. 3.4a). We see, that *the directions of noise injection become important for small sample sizes when the intrinsic data dimensionality is small*.

3.5.3 The Effect of the Number of Noise Injections on the Efficiency of Noise Injection

Let us compare the performance of the multilayer perceptron with Gaussian spherical noise and with 2-NN directed noise injection versus the number of noise injections R for two data sets with low intrinsic dimensionality: 8-dimensional banana-shaped data with an intrinsic dimensionality of two and with the training sample size per class equal to 50 and 12-dimensional uniformly distributed data with an intrinsic dimensionality of two and with the training sample size per class equal to 20. The averaged and optimized results are presented in Fig. 3.5 and Fig. 3.6 for 8-dimensional banana-shaped data and for 12-dimensional uniformly distributed data, respectively. For the banana-shaped data set and for the uniformly distributed data set, the standard deviations of the mean generalization error vary approximately from 0.026 to 0.002 and from 0.026 to 0.01, when increasing the number of noise injections R, respectively.

Both figures show the following.

1) Generalization error EP_N decreases with an increase in the number of noise injections R and stops decreasing when $R \to \infty$. It supports the theoretical conclusions obtained for parametric and non-parametric statistical classifiers discussed in Section 3.3.

2) The multilayer perceptron with 2-NN directed noise injection has a better performance. We see, that for small numbers of noise injections R the multilayer perceptron with 2-NN directed noise injection outperforms the multilayer perceptron with Gaussian spherical noise injection, because in 2-NN directed noise injection Gaussian noise is generated only in useful directions, in which the data are located. For large values of the number of noise injections R the results for both noise injection methods become similar. For very large values of R the spherical Gaussian noise vectors $\boldsymbol{Z}_{jr}^{(i)} \sim N(\mathbf{0}, \lambda^2 \boldsymbol{I})$ ($r = 1, \ldots, R$) become symmetrically distributed in all directions around a training vector $\boldsymbol{X}_j^{(i)}$. Therefore the negative effect of the finiteness of R

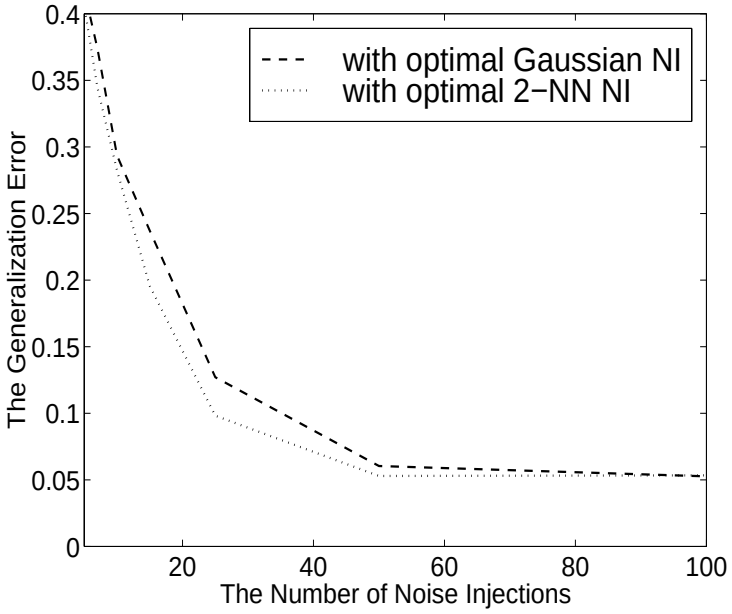


Fig. 3.5. The generalization error of the perceptron with three hidden neurons with noise injection versus the number of noise injections for 8-dimensional banana-shaped data with an intrinsic dimensionality of two and with the training sample size per class equal to 50.

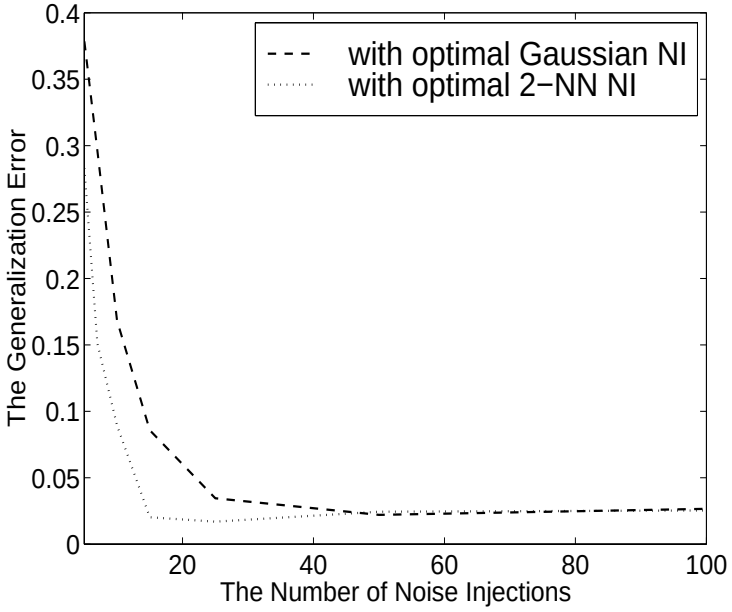


Fig. 3.6. The generalization error of the perceptron with three hidden neurons with noise injection versus the number of noise injections for 12-dimensional uniformly distributed data with an intrinsic dimensionality of two and with the training sample size per class equal to 20.

vanishes.

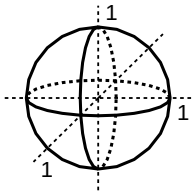
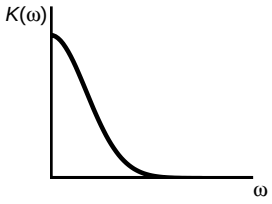
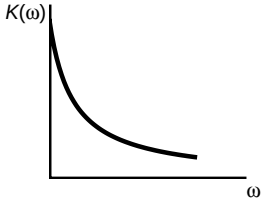
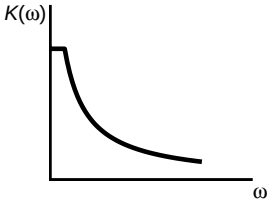
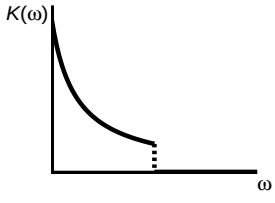
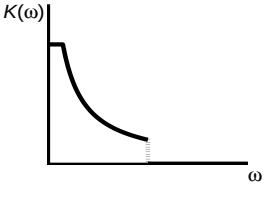
The considered examples demonstrate, that *the effectiveness of noise injection in multilayer perceptron training depends not only on the type of noise injection, but also on the number of noise injections R*.

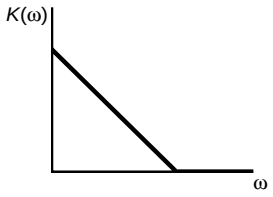
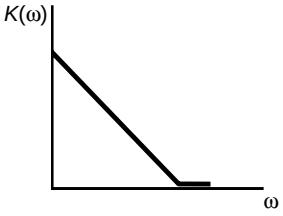
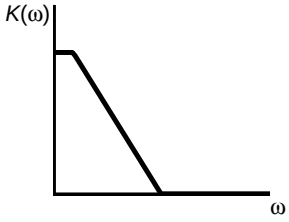
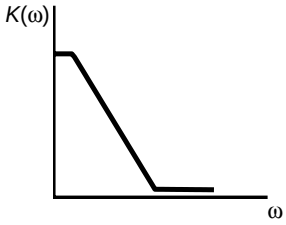
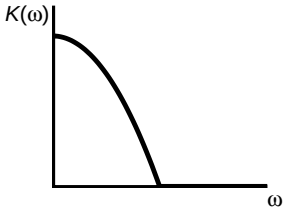
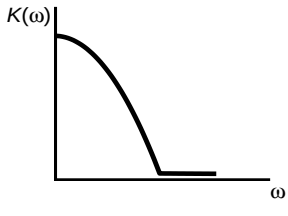
3.6 The Role of the Shape of Noise

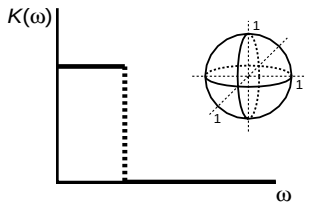
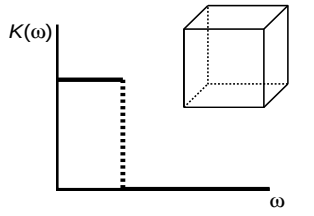
It was shown (see Section 3.3) that Gaussian noise injection to the training objects is similar to the Parzen Window estimate of the data set. Therefore, instead of considering noise injection models with different shape of noise, we may consider the Parzen Window Classifier (3.1) having different shapes for the kernel function. For large training sample sizes $N \to \infty$ (see (3.10)), the shape of the kernel function $K(H)$ (and/or the shape of noise) is not important. However, for the finite training sample sizes N, it might affect the performance of the classifier.

Many kernel functions $K(H)$ exist that satisfy the conditions (3.3)-(3.6). Rosenblatt [104] has suggested to use a rectangular shape of the kernel function. Epanechnikov [27] has also considered kernels of a different shape. He found the optimal shape of the kernel by minimizing the relative *Mean Integral Square Error* (MISE) $\text{MISE} = \int (f(x) - \hat{f}(x))^2 \, dx$ [27]. It is known that the classification error characterizes the quality of the classifier. The criterion of MISE is defined only by the first two statistical moments, without taking into account higher order asymmetry, which strongly affects the probability of misclassification [35, 87]. Therefore, this criterion characterizes an accuracy of the reconstruction of a multivariate density. However, it does not completely characterize the probability of misclassification [87]. It is considered, that the shape of the kernel function does not influence very much the performance of the PWC. The performance of the PWC having different kernels (normal and rectangular) has been studied by Duin [23]. He has considered one data set consisting of two 5-dimensional normally distributed classes. In a simulation study he found that the performance of the PWC with normal and rectangular (uniform) kernels is similar for optimal values of the smoothing parameter λ_{PW}. However, the behaviour of the classification error via all values λ_{PW} was quite different. The conclusion has been made that for a rectangular kernel the choice of the smoothing parameter is more critical than for a normal kernel. In monograph [89] the unpublished work by Grabauskas is mentioned. According to this monograph, Grabauskas has compared 20 different shapes of the kernel function for one problem. By simulation study he has shown that the performance of the PWC is similar for all considered shapes of the kernel function. However, both Duin and Grabauskas have performed their study considering only one problem. Therefore, it is still of interest to investigate the influence of the shape of the kernel function on the performance of the PWC in the case of small training sample sizes, when more data sets are considered.

Table 3.1. The estimates of the density function $f(\boldsymbol{x}|\pi_i)$ with the kernel function $K(z)$, where $\int_{\Omega} cK(z)dz = 1$, $z = \dfrac{\boldsymbol{x} - \boldsymbol{X}_j^{(i)}}{\lambda_{PW}}$, $K = K[\omega(z)]$ and $\omega = \sum_{k=1}^{p} z_k$.

The shape of the density function 		
No.	**The kernel shape**	**The analytical formula**
1		$K(\omega) = \exp(-\omega^2)$
2		$K(\omega) = \dfrac{77}{150\omega + 70}$
3		$K(\omega) = \begin{cases} 0.9, & \omega \le 0.3; \\ \dfrac{441}{1100\omega + 160}, & \omega > 0.3 \end{cases}$
4		$K(\omega) = \begin{cases} \dfrac{77}{150\omega + 70}, & \omega \le 2; \\ 0, & \omega > 2 \end{cases}$
5		$K(\omega) = \begin{cases} 0.9, & \omega \le 0.3; \\ \dfrac{441}{1100\omega + 160}, & 0.3 < \omega < 2; \\ 0, & \omega \ge 2 \end{cases}$

No.	The kernel shape	The analytical formula
6		$K(\omega) = \begin{cases} -0.36\,\omega + 0.9, & \omega < 2.5; \\ 0, & \omega \geq 2.5 \end{cases}$
7		$K(\omega) = \begin{cases} -0.36\,\omega + 0.9, & \omega < 2.4997; \\ \dfrac{1}{\omega + 9256.76}, & \omega \geq 2.4997 \end{cases}$
8		$K(\omega) = \begin{cases} 0.9, & \omega \leq 0.3636; \\ -0.55\omega + 1.1, & 0.4 < \omega < 2; \\ 0, & \omega \geq 2 \end{cases}$
9		$K(\omega) = \begin{cases} 0.9, & \omega \leq 0.3636; \\ -0.55\omega + 1.1, & 0.4 < \omega < 1.9998; \\ \dfrac{1}{\omega + 9088.91}, & \omega \geq 1.9998 \end{cases}$
10		$K(\omega) = \begin{cases} -\frac{1}{4}\omega^2 + 1, & \omega < 2; \\ 0, & \omega \geq 2 \end{cases}$
11		$K(\omega) = \begin{cases} -\frac{1}{4}\omega^2 + 1, & \omega < 1.9998; \\ \dfrac{1}{\omega + 4998.0002}, & \omega \geq 1.9998 \end{cases}$

No.	The kernel shape	The analytical formula
12		$K(\omega) = \begin{cases} 0.8, & \omega \le 1.25; \\ 0, & \omega > 1.25 \end{cases}$ with the spherical shape of the density function
13		$K(\omega) = \begin{cases} 0.8, & \omega \le 1.25; \\ 0, & \omega > 1.25 \end{cases}$ with the squared shape of the density function $Q(z) = \begin{cases} 1, & z_i \le 1, i = \overline{1,p} \\ 0, & otherwise \end{cases}$

When the normal density function is used as the kernel function, the expected classification error EP_N of the PWC can be expressed analytically [87]. However, it is impossible to obtain an analytical expression of the EP_N for many other kernel functions. The formula obtained for the case of the normal density function, is also exact only for the limiting case, when $N \to \infty$. In addition, it requires time consuming calculations. Therefore, even for this case, it is impossible to find an exact value of the expected classification error EP_N , when training sample sizes are small. Under these circumstances, it is reasonable to use simulation study in order to investigate the performance of the PWC with different shapes of the kernel function.

A common choice for the kernel function in the non-parametric PWC estimate of the density function $f_{PW}(\boldsymbol{x}|\pi_i)$ is the normal density function (3.2). Sometimes a rectangular kernel is used. We will consider 13 different kernel functions, which are presented in Table 3.1. In 12 cases the estimate of the density function has a highly dimensional spherical form with kernel functions having a different shape. In one case the estimate of the density function has a form of a highly dimensional cube with the kernel function having a rectangular shape. In all cases the coefficients in formula for kernel functions $K(\omega)$ are chosen in such a way that the square described by the kernel function is close to 1, $\int_{\Omega} cK(z)dz = 1$, where c is a normalizing coefficient.

Six real data sets and one artificial data set are used in our simulation study (see Table 3.2). Real data were obtained by solving practical problems in the field of medical diagnosis in Lithuania (data sets III-VI) and in acoustic pattern recognition (data set VII). The data set II describes the language structure defined by the frequency of the appearance of a certain letter in the text written in different languages. These data were used in the competition on Pattern Recognition in 1976. The data set I is the artificially generated data set that consists of the first

Table 3.2. The description of the data sets.

The name of the data set	# classes	# features	# training objects	# test objects	# experiments
I - BANANA	2	2	50	1000	10
II - LETTERS	2	5	100	500	5
III - LEICOSIS	2	5	25	100	2
IV - RITHMS	2	7	40	100	2
V - HEART ATTACKS	2	4	20	39	3
VI - INJURIES	3	13	20	60	2
VII - SOUNDS	3	8	15	40	2

two features of the 8-dimensional banana-shaped data set, described in Chapter 1 and used in our previous simulation study.

Each data set is divided into two parts: a training part and a test part. The test part is used as a test data set. The size of the test set is fixed for each data set. Just as the training part is divided into a couple of equal parts - training sets. The number of independent training setvaries from 2 to 10 depending on the amount of the available objects in the training part.Therefore, experiments are repeated several times (from 2 to 10 times for different data sets). The training is performed each time on the independent training set, and the testing is done always on the same test set. The obtained results are averaged over the number of repetitions.

The smoothing parameter λ_{PW} in the PWC changes in a large range: from 0.001 up to 1000. The optimal value of the smoothing parameter λ_{PW}^{opt} (in the sense of the minimal classification error on the training set) is found for each shape of the kernel function and for every data set. It is also done experimentally.

The averaged classification errors P_{ij} ($i=\overline{1,13}$, $j=\overline{1,7}$) of the PWC and their standard deviations (in brackets), which correspond to the optimal value of the smoothing parameter λ_{PW}^{opt}, are presented in Table 3.3 for 13 different shapes of the kernel function and for 7 different data sets. As the obtained results differ very much, it is difficult to compare them directly and to give a clear answer as to which kernel shape is the best. Therefore, we use two generalizing criteria for the postprocessing of the results, suggested by K. Čeponis [private communication]. The reasonable criteria are the criteria that allow us to pick out those kernel functions, which use in the PWC does not lead to a very bad performance (however, it is not necessary that the performance of the PWC using these kernel functions is the best).

Table 3.3. The minimal averaged classification errors and their standard deviations (in brackets) of the PWC for 13 different kernel functions and for 7 different data sets.

Data / Kernels	I	II	III	IV	V	VI	VII
1	**4.82** (0.15)	28.94 (0.64)	5.75 (1.16)	**32.0** (2.35)	9.83 (1.94)	35.56 (2.5)	**0.0** (0.0)
2	5.98 (0.17)	29.28 (0.62)	5.75 (1.16)	32.25 (2.35)	9.4 (1.90)	36.11 (2.5)	1.25 (0.71)
3	5.95 (0.17)	29.1 (0.64)	5.75 (1.16)	33.25 (2.35)	9.4 (1.90)	36.11 (2.5)	1.25 (0.71)
4	5.33 (0.16)	30.3 (0.65)	5.75 (1.16)	33.25 (2.35)	9.83 (1.94)	37.22 (2.5)	1.25 (0.71)
5	5.14 (0.156)	30.14 (0.65)	6.75 (1.25)	33.25 (2.35)	9.4 (1.9)	36.39 (2.5)	2.09 (0.9)
6	5.16 (0.156)	30.12 (0.65)	5.75 (1.16)	33.75 (2.35)	9.83 (1.94)	**34.17** (2.5)	2.5 (1.0)
7	4.93 (0.153)	28.72 (0.64)	5.75 (1.16)	32.25 (2.35)	9.83 (1.94)	**34.17** (2.5)	0.84 (0.9)
8	5.17 (0.156)	30.3 (0.65)	6.5 (1.23)	33.25 (2.35)	9.97 (1.87)	36.95 (2.5)	2.92 (1.1)
9	5.1 (0.156)	**28.56** (0.64)	**5.5** (1.14)	32.25 (2.35)	**8.97** (1.87)	36.95 (2.5)	**0.0** (0.0)
10	5.3 (0.16)	30.5 (0.65)	6.0 (1.19)	33.25 (2.35)	9.83 (1.94)	36.11 (2.5)	3.34 (1.15)
11	5.15 (0.156)	28.74 (0.64)	5.75 (1.16)	32.5 (2.35)	9.83 (1.94)	36.11 (2.5)	**0.0** (0.0)
12	5.29 (0.16)	32.0 (0.66)	9.0 (1.43)	34.75 (2.35)	**8.97** (1.87)	38.34 (2.5)	4.58 (1.35)
13	5.36 (0.16)	30.65 (0.65)	9.75 (1.48)	33.0 (2.35)	11.11 (2.0)	66.67 (2.5)	2.5 (1.0)

Criterion A. Separately for each of seven data sets, we find the minimal value $P_j^{min} = \min_i P_{ij}$ ($j=\overline{1,7}$) and the maximal value $P_j^{max} = \max_i P_{ij}$ ($j=\overline{1,7}$) of the minimal averaged classification error among 13 introduced values for each shape of the kernel function. Then we set these values equal to 0 and 1, respectively for P_{ij}^{min} and P_{ij}^{max}, and transform all other values of the minimal averaged classification error P_{ij} ($i=\overline{1,13}$) into the interval [0,1]:

Table 3.4. The values for the criterion A.

Data / Kernels	I	II	III	IV	V	VI	VII	K^A
1	0	0.11	0.059	0	0.402	0.043	0	**0.088**
2	1	0.209	0.059	0.091	0.203	0.06	0.273	**0.271**
3	0.974	0.157	0.059	0.455	0.201	0.06	0.273	**0.311**
4	0.435	0.506	0.059	0.091	0.402	0.094	0.273	**0.266**
5	0.276	0.459	0.294	0.455	0.201	0.069	0.455	**0.316**
6	0.293	0.454	0.059	0.636	0.402	0	0.546	**0.341**
7	0.095	0.047	0.059	0.091	0.4	0	0.182	**0.125**
8	0.302	0.506	0.235	0.455	0	0.086	0.637	**0.317**
9	0.237	0	0	0.091	0	0.086	0	**0.059**
10	0.414	0.564	0.118	0.455	0.402	0.06	0.728	**0.392**
11	0.285	0.052	0.059	0.182	0.4	0.06	0	**0.148**
12	0.401	1	0.824	1	0.002	0.128	1	**0.622**
13	0.466	0.611	1	0.364	1	1	0.546	**0.712**

$$P_{ij}^{A} = \frac{P_{ij} - P_j^{min}}{P_j^{max} - P_j^{min}}, \quad i=\overline{1,13}, j=\overline{1,7}. \tag{3.19}$$

In this way we obtain Table 3.4, which consists of the values P_{ij}^{A} belonging to the interval [0,1]. Afterwards, for each kernel function we calculate the mean, looking over all data sets. As a result, we obtain a criterion $K_i^A = \frac{1}{7}\sum_{j=1}^{7} P_{ij}^{A}$, $i=\overline{1,13}$. The best kernel function has the smallest value of the criterion K_i^A.

Criterion B. Separately for each of seven data sets, we consider 13 introduced values of the minimal averaged classification error P_{ij} ($i=\overline{1,13}$) for each kernel function. Throwing aside 2-4 the best and the worst values, we calculate the mean of P_{ij}. Then we normalize each value P_{ij} ($i=\overline{1,13}$) by the mean value P_j^{mean}:

$$P_{ij}^{B} = \frac{P_{ij}}{P_j^{mean}}, \quad i=\overline{1,13}, j=\overline{1,7}. \tag{2.20}$$

In this way we obtain Table 3.5, which consists of the normalized values P_{ij}^{B}. Then, for each kernel function we calculate the mean criterion value looking over all data sets. Thus, we obtain a criterion $K_i^B = \frac{1}{7}\sum_{j=1}^{7} P_{ij}^{B}$, $i=\overline{1,13}$. The best kernel function has the smallest value of the criterion K_i^B.

Table 3.5. The values for the criterion B.

Data / Kernels	I	II	III	IV	V	VI	VII	K^B
1	0.985	0.973	0.981	0.977	1.034	0.977	0	**0.847**
2	1.894	0.985	0.981	0.985	0.989	0.992	0.75	**1.082**
3	1.773	0.978	0.981	1.015	0.989	0.992	0.75	**1.068**
4	0.985	1.019	0.981	0.985	1.034	1.023	0.75	**0.968**
5	0.924	1.013	1.152	1.015	0.989	1	1.251	**1.049**
6	0.924	1.013	0.981	1.031	1.034	0.939	1.5	**1.06**
7	1.03	0.966	0.981	0.985	1.034	0.939	0.501	**0.919**
8	0.955	1.019	1.109	1.015	0.944	1.015	1.749	**1.115**
9	1.06	0.96	0.939	0.985	0.944	1.015	0	**0.843**
10	0.924	1.026	1.024	1.015	1.034	0.992	2.001	**1.145**
11	1.121	0.966	0.981	0.992	1.034	0.992	0	**0.869**
12	0.97	1.076	1.536	1.061	0.944	1.053	2.447	**1.302**
13	1.015	1.031	1.664	1.008	1.169	1.832	1.5	**1.317**

To get a better view of the obtained results, we have illustrated the averaged estimates of the criteria K_i^A and K_i^B, $i=\overline{1,13}$, from Table 3.4 and Table 3.5 by bars in Fig. 3.7. Both, left and right plots in Fig. 3.7 show that the kernels with numbers 9, 1, 7, and 11 have the best criterion estimates (with the smallest values). The mean estimates of criteria A and B for the kernel 9 (the kernel of trapezoid-logistic shape) are even better than for the kernel 1 (the normal kernel). Therefore, using the kernel with the trapezoid-logistic shape seems to be more preferable than using the normal kernel. Moreover, the computational time is twice smaller when using the trapezoid-logistic function.

The computational costs of the kernel functions with rectangular shape (functions 12 and 13) are the smallest. However, plots in Fig. 3.7 show that these functions have the worst criterion estimates in the case of small training sample sizes. Therefore, we do not recommend using them as the kernels in the PWC when the training sample sizes are small.

It is noticed that the best (with respect to the smallest classification error) kernel functions (e.g., the functions 1, 7, 9, 11) rapidly approach zero, however, never reaching it. On the other hand, kernel functions that reach zero (the functions 4, 5, 6, 8, 10, 12) or kernel functions that slowly converge to zero without reaching it (the functions 2 and 3) lead to the worst performance of the PWC.

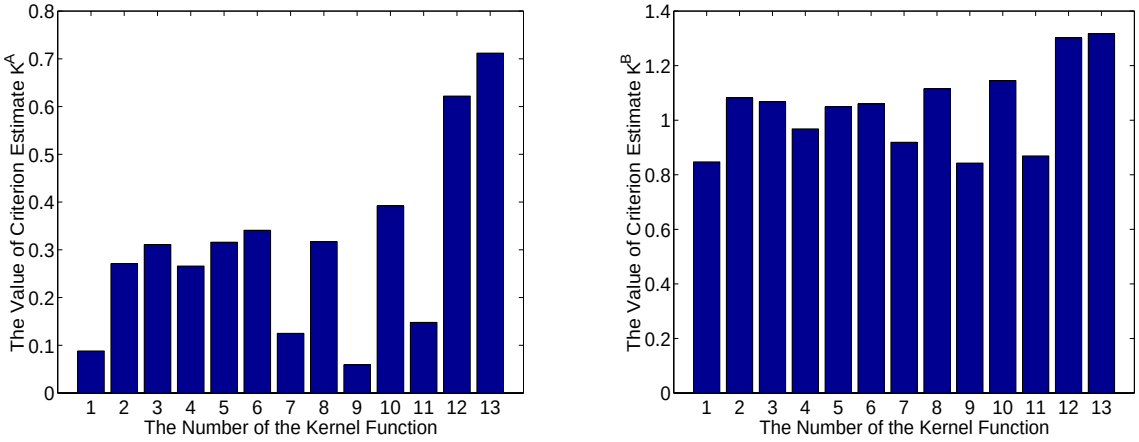


Fig. 3.7. The values of the averaged criterion estimate K^A *(left plot) and of the averaged criterion estimate* K^B *(right plot). The smaller the value of the criterion estimate, the better the kernel function.*

Simulation study also shows that there is no the kernel function shape which is the best for all data. Every data set has its own best shape(s) for the kernel function in the PWC, which gives the minimal classification error. For instance, the best kernel function for the data set I is the normal density function (1), for the data sets II and III - the trapezoid-logistic function (9), for the data set IV - the kernel function of the triangular shape (6 and 7).

Thus, one can see that the shape of the kernel function in the PWC is important when the small training sample sizes are used. The best kernel functions are the normal density function and the trapezoid-logistic function, which both give the best performance (the smallest classification error) of the PWC. It could also be recommended to use the trapezoid-logistic function as the kernel function in the PWC, since it reduces the computational costs of the PWC compared with the common case when the normal density function is used.

3.7 Scale Dependence of Noise Injection

Noise injection helps to solve the small sample size problem. However, the noise injection is data dependent and therefore scale dependent. If the noise injection is more dependent on data distribution, it will be less dependent on data scaling. It is possible to generate noise taking into account the distribution of the whole training data set (for example, the means and covariances of data classes) [80, 52]. Let us call this model of noise generation *the variances based noise*. For the large training sample sizes the injection of such noise will be almost scale independent. In the Gaussian noise injection model, when the Gaussian distributed noise is generated locally around each training object, noise is independent of the

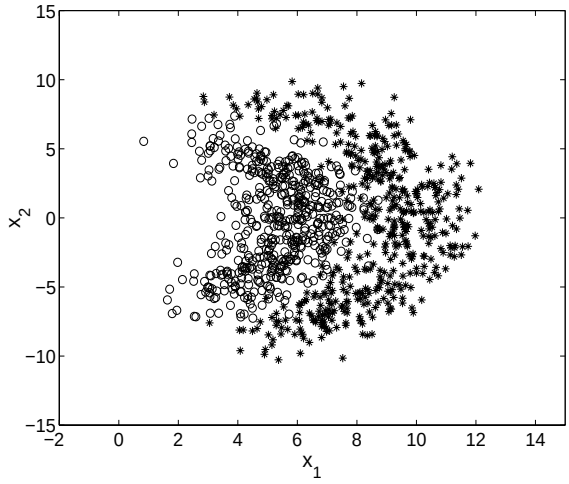


Fig. 3.8. The scatter plot of the first two features of 8-dimensional banana-shaped data.

distribution of the training set. However, it depends strongly on the location of each training object. Therefore, data rescaling strongly influences the effectiveness of the Gaussian noise injection. The advantage of the k-NN directed noise is that this noise is more data dependent. Generating the k-NN directed noise the distribution of the k nearest neighbours of the object is taken into account. This way, the injection of the k-NN directed noise is less scale dependent.

Let us consider an example of the 8-dimensional banana-shaped data with the high intrinsic dimensionality, described in Chapter 1. In order to demonstrate how data scaling affects different models of noise injection, we just rescale the first two data features (see Fig. 3.8) after data normalization and add noise to the training objects. The variances of the noise are chosen equal to 1.2, 1.2 and 1 for the Gaussian noise, for the 2-NN directed noise and for the variances based noise, respectively. In our simulation study we use a perceptron with three hidden units, trained by the Levenberg-Marquardt learning procedure. Eight objects from each data class are used for training and 492 objects from each class are used for testing. We run thirty independent experiments (using 10 independent training sets and 3 different weights initializations). The averaged results over these thirty independent experiments are presented in Fig. 3.9. In correspondence with the discussion above, one can see that scaling strongly affects data enriched by Gaussian noise. Adding Gaussian distributed noise with the same variance in all directions from the training object gives worse results when data features have large scale differences. Data enriched by the 2-NN directed noise and by the variances based noise are almost independent of scaling.

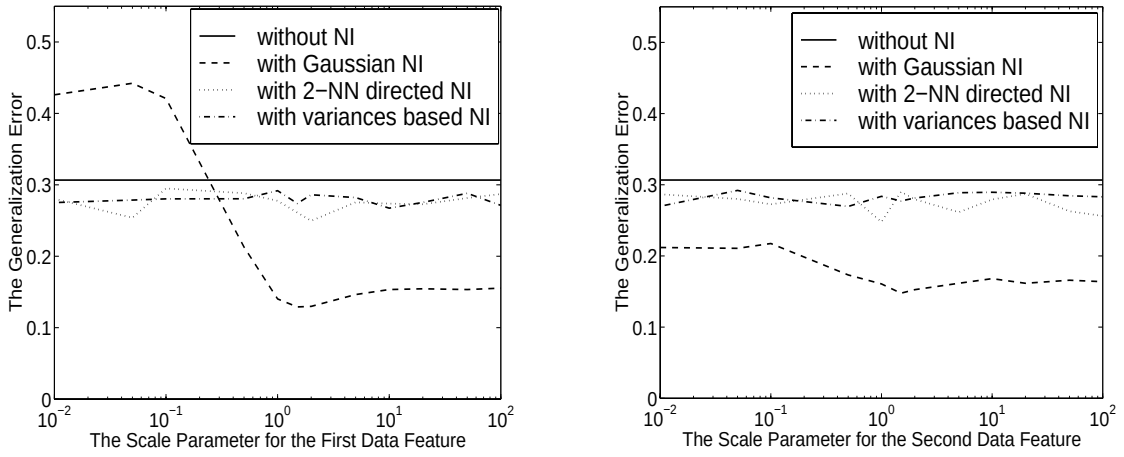


Fig. 3.9. The generalization error of the perceptron with three hidden neurons for the 8-dimensional banana-shaped data (with the training sample size per class equal to 8) without and with noise injection versus the data scaling parameter.

3.8 Conclusions

Theoretical results obtained for one parametric and two nonparametric statistical classifiers combined with our simulation study carried out for MLP's show the following.

1) Noise injection acts as a regularization factor which helps to reduce the generalization error.

2) There exists an optimal value of the noise variance λ.

3) The number of noise injections R should be sufficiently large. However, too large values of R do not diminish the generalization error and increase computing time.

4) A sufficient number of noise injections R^* depends on the data dimensionality p.

5) If the intrinsic data dimensionality r is small, R^* can be reduced by avoiding the injection of noise in "unnecessary" directions. The k-NN directed noise injection is one of possible effective means to do this.

6) The noise shape may only be important for small training sample sizes. The best shapes are Gaussian and trapezoid-logistic.

7) Noise is scale dependent: the more independent the noise injection model is on the distribution of the training set, the more affected it will be by data scaling.

Chapter 4

Regularization by Adding Redundant Features

4.1 Introduction

As a rule, with a small number of exceptions, statistical parametric classifiers are sensitive to the training sample size. When the training sample size is small, some classifiers, such as the *Fisher Linear Discriminant* (FLD) [28, 29] or the *Quadratic Classifier* (QC) [35], cannot be constructed without problems. They require the inverse of the covariance matrix, which is impossible to perform when the number of training objects is less than the data dimensionality. Other classifiers, such as the *Pseudo Fisher Linear Discriminant* (PFLD) [35], the *Regularized Fisher Linear Discriminant* (RFLD) [30, 95] or the *Regularized Quadratic Classifier* (RQC) [30] manage to overcome the small sample size problem. However, they may become very unstable and have a peaking effect for the generalization error when the training sample size n is comparable with the data dimensionality p [25, 115, 117]. With an increase in the training sample size, the generalization error first decreases, reaching a minimum, then increases, reaching a maximum at the point where the training sample size is equal to the data dimensionality, and afterwards begins again to decrease (see Fig. 4.1). For the RFLD and the RQC this happens when the value of regularization parameter λ is very small, close to zero.

In the past, the following ways have been studied to solve this problem:

1. Removing features (decreasing p) by some feature selection method. By this the relative training sample size increases. This brings the classifier out of the instable region $n=p$ because the learning curve of the classifier (the dependency of its generalization error, as a function, on the training sample size) shifts to the left with respect to the learning curve obtained for the original data dimensionality (see Fig. 4.1).
2. Adding objects (increasing n), either by using larger training sets, or, if this is not possible, by generating additional objects (noise injection [95]).
3. Removing objects (decreasing n). This also brings the classifier out of the instable region. In this case, however, the learning curve shifts to the right (see Fig. 4.1) as the relative data dimensionality increases. This method has been studied by us [25, 115] and is also effectively used in the Support Vector Classifier [19].

In this chapter, we will show by some examples that the fourth way can also be effective:

4. Adding redundant features (increasing p) [118]. Here, ρ completely redundant noise features $y_{j1}, y_{j2}, \ldots, y_{j\rho}$ are added to each p-dimensional training object $\boldsymbol{X}_j = (x_{j1}, x_{j2}, \ldots, x_{jp})'$, $j = 1, \ldots, n$. By this, $(p + \rho)$-dimensional objects $\boldsymbol{Z}_j=(x_{j1}, x_{j2}, \ldots, x_{jp}, y_{j1}, y_{j2}, \ldots, y_{j\rho})'$, enriched by redundant noise features, are obtained. Like the third method, this brings the

classifier out of the instable region, but now by enlarging the dimensionality entirely by noise. This chapter is completely based on our previously published work [118, 119]. We will mainly concentrate on the injection of noise by adding redundant features to the data and on its effect on the performance of the PFLD (Section 4.2). This classifier is discussed in detail in Chapter 1. In Section 4.3 we will show that noise injection, by adding redundant features to the data, is similar to other regularization techniques (noise injection to the training objects and the ridge estimate of the covariance matrix) and helps to improve the generalization error of this classifier for critical training sample sizes. In Section 4.4 we will draw conclusions and present some discussions.

Two artificial data sets and one real data set are used for our experimental investigations. They are the 30-dimensional correlated Gaussian data set, the 30-dimensional Gaussian spherical data with unequal covariance matrices and the 256-dimensional cell data. All these data sets are described in Chapter 1. Training data sets with 3 to 200 samples per class (with 3 to 300 for cell data) are chosen randomly from the total set. The remaining data are used for testing. These and all other experiments are repeated 10 times for independent training sample sets. In all figures the averaged results over 10 repetitions are presented.

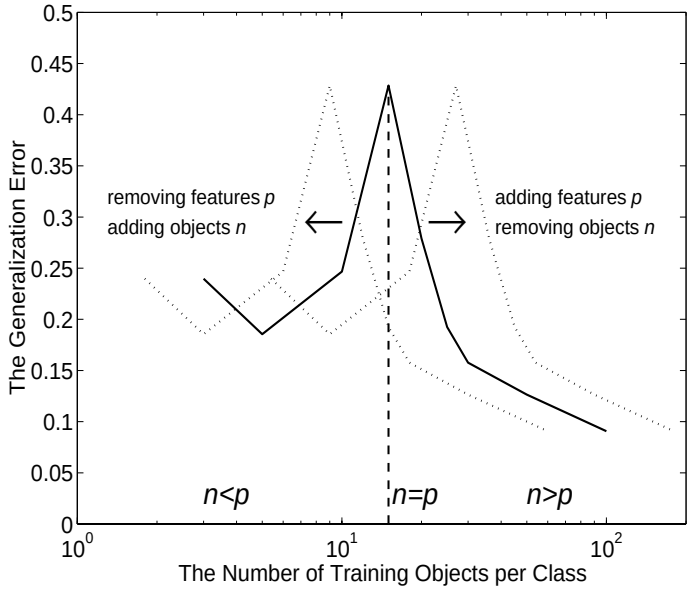


Fig. 4.1. The generalization error of the Pseudo Fisher linear discriminant (PFLD) versus the training sample size for 30-dimensional Gaussian correlated data. When the training sample size n increases or the data dimensionality p decreases, the learning curve of the PFLD shifts to the left with respect to the learning curve of the PFLD obtained on the original training set. When the training sample size n decreases or the data dimensionality p increases, the learning curve of the PFLD shifts to the right. Both, decreasing and increasing either the training sample size n or the data dimensionality p, bring the PFLD out of the instable region n=p.

4.2 The Performance of Noise Injection by Adding Redundant Features

The generalization error of the PFLD shows a peaking behaviour when the number of training objects n is equal to the data dimensionality p (see Fig. 4.1). In order to improve the generalization error of the PFLD for critical values of the training sample size ($n=p$), a number of techniques could be used (see Section 4.1). One of the ways to solve this problem involves generating more training objects by noise injection into the training data. Usually, spherical Gaussian distributed noise is generated around each training object. However, this method requires us to know quite precisely the variance of the injected noise in order to get good results (a low generalization error). The optimal value of the injected noise variance (with respect to the minimum generalization error) depends on many factors such as the training sample size, the data dimensionality and the data distribution (see Chapter 2) [95]. It could vary dramatically for different data. As a rule, it is computationally expensive to find the optimal value of the noise variance.

To demonstrate the influence of the injected noise variance λ^2 on the generalization error of the PFLD, we consider the 30-dimensional Gaussian correlated data. The averaged results for some values of λ^2 are presented in Fig. 4.2. We can see that the performance of the PFLD strongly depends on the variance of the noise.

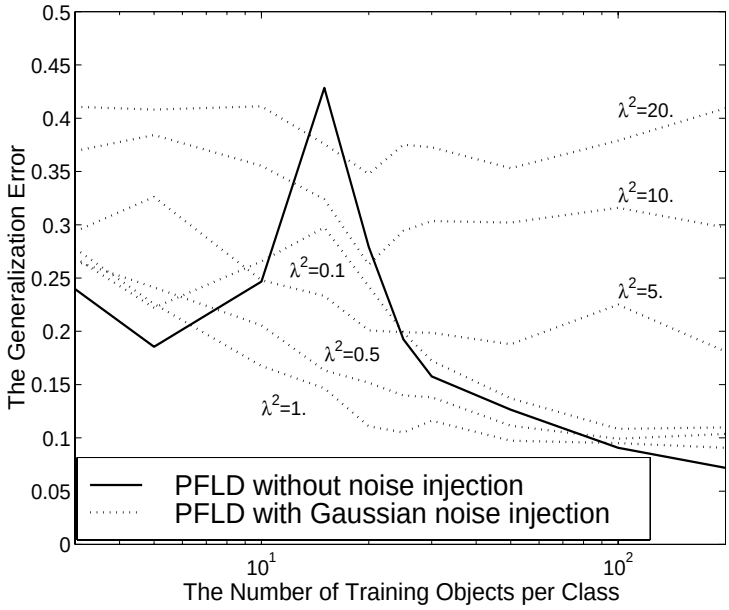


Fig. 4.2. The generalization error of the PFLD without and with noise injection to the training objects with different values of the noise variance λ^2 versus the training sample size for 30-dimensional Gaussian correlated data.

Considering small sample size properties (a learning curve) of the PFLD, one can approach another solution: decrease the number of training objects in order to avoid the critical training sample size problem. It could be also performed by noise injection into the data feature space instead of adding noise to the training objects. In this case, the data dimensionality is enlarged by adding Gaussian distributed features $N(0, \lambda^2)$ while the number of training objects remains the same (see Fig. 4.3). When increasing the data dimensionality *p*, the training sample size *n* relatively decreases, leaving the critical area *n*=*p*, where the PFLD has a high generalization error. For values $n < p$ the PFLD performs much better than for the critical sizes of the training set.

Let us now investigate the effectiveness of noise injection by adding redundant features for the three data examples mentioned above. To study the influence of injection of noise features to the data, for each considered data set, ρ additional redundant noise features were generated having Gaussian distributions with zero mean and unit variance $N(0, 1)$ for both classes. This enlarges the dimensionality from *p* to *p*+ρ. The generalization error of the PFLD for 30-dimensional Gaussian correlated data and 30-dimensional Gaussian spherical data with unequal covariance matrices without noise injection in the feature space and with 20, 70 and 170 additional redundant noise features is presented in Fig. 4.4 and Fig. 4.5, respectively. The generalization error of the PFLD obtained on the cell data without noise injection in the feature space and on the cell data with 44, 100, 144 and 200 redundant features is presented in Fig. 4.6.

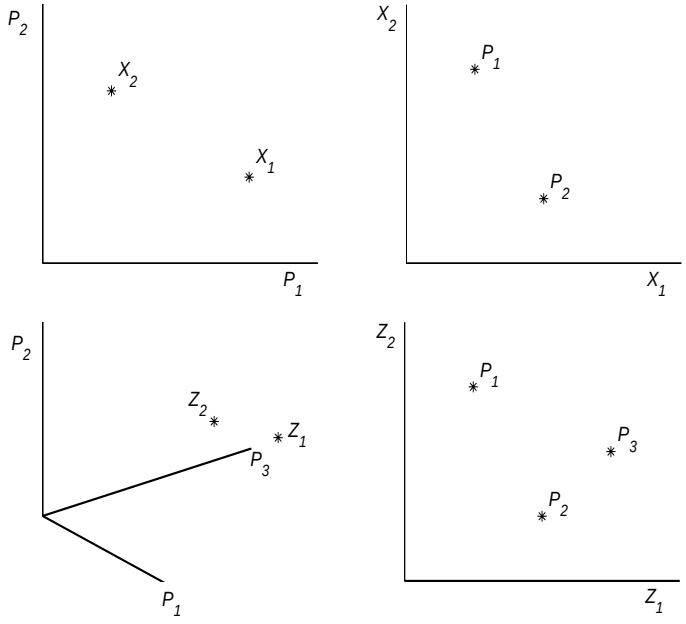


Fig. 4.3. Imaginary example of adding one feature $\boldsymbol{Y} = (y_1 \;\; y_2)$ *to the data set* $\boldsymbol{X} = (\boldsymbol{X}_1 \;\; \boldsymbol{X}_2) = \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \end{pmatrix} = \begin{pmatrix} \boldsymbol{P}_1 \\ \boldsymbol{P}_2 \end{pmatrix}$. *By this, the data dimensionality increases while the number of objects remains the same* $\begin{pmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 \\ y_1 & y_2 \end{pmatrix} = (\boldsymbol{Z}_1 \;\; \boldsymbol{Z}_2) = (\boldsymbol{P}_1 \;\; \boldsymbol{P}_2 \;\; \boldsymbol{P}_3)'$.

For all data sets the PFLD shows a critical behaviour with a high maximum of the generalization error around critical training sample size $n = p$. Figures 4.4-4.6 demonstrate nicely that noise injection into the data feature space helps to avoid the peaking effect of the generalization error of the PFLD for a given number of training objects. We can see that enlarging the data dimensionality twice by adding noise features has already doubled the performance of the classifier at the point $n=2N=p$, where N is the number of training objects per class. For cell data, it was enough to add 44-100 noise features for the same improvement. When the number of added noise features was 4-5 times larger than the original dimensionality of the data, the peak of the generalization error was smoothed almost completely: the generalization error was reduced in a whole region around the critical training sample size. However, for very small training sample sets, adding redundant features was useless. The reason is the following. Features of the cell data set have small variances. The largest variance is 0.12 (for the 137th feature). The variances of other features are smaller than 0.01. Therefore, adding noise with a relatively large variance ($\lambda^2 = 1$) to a highly dimensional feature space with only a few objects makes the training data set too "noisy" to represent the entire data set correctly. In this case, it becomes difficult or even impossible to build a good discriminant function. All data considered demonstrate nicely that the more noise added to the data by adding redundant features, the larger the generalization error obtained in the case of very small training sample sizes. A smaller noise variance should probably be used to get better results for small training set sizes. For critical training data sizes, adding redundant features helps to avoid the peaking effect of the generalization error of the PFLD.

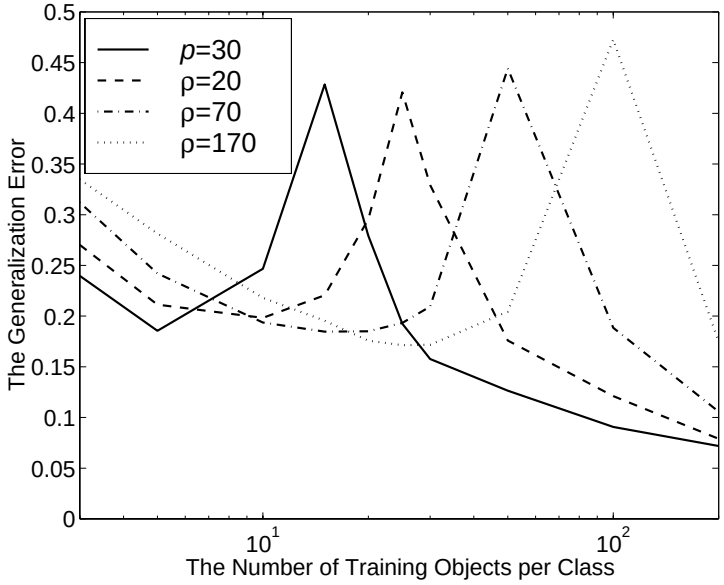


Fig. 4.4. The generalization error of the PFLD versus the training sample size for Gaussian correlated data without noise injection in the feature space (p=30) and with ρ=20, 70, 170 additional redundant noise features (p+ρ=50, 100, 200).

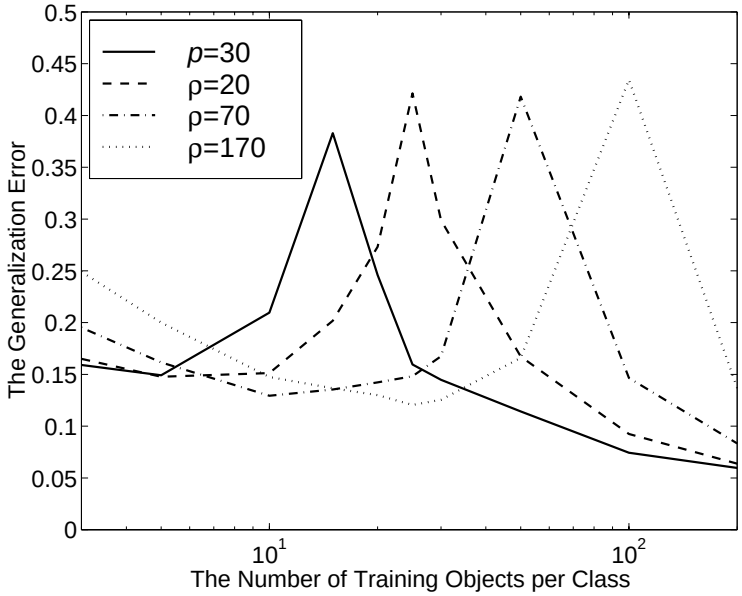


Fig. 4.5. The generalization error of the PFLD versus the training sample size for Gaussian spherical data with unequal covariance matrices without noise injection in the feature space (p=30) and with ρ=20, 70, 170 additional redundant noise features (p+ρ=50, 100, 200).

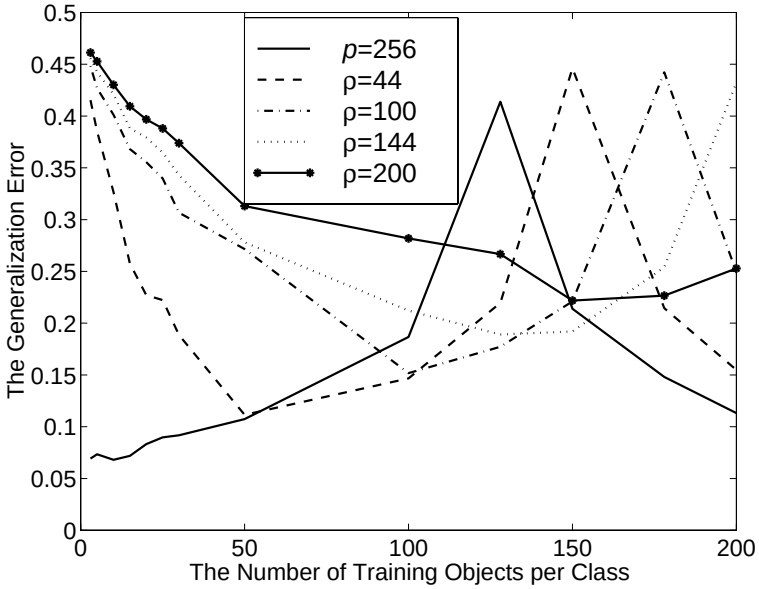


Fig. 4.6. The generalization error of the PFLD versus the training sample size for 256-dimensional cell data without noise injection (p=256) and with ρ=44, 100, 144, 200 additional redundant noise features (p+ρ=300, 356, 400, 456).

One can notice that the improvement obtained in the generalization error depends on the number ρ of redundant features used. It is also reasonable to suppose that the generalization error depends on the noise variance in redundant features. A mathematical attempt to understand this relation is made in the next section. Figures 4.4-4.6 suggest that the relationship between the number of redundant features ρ and the noise variance λ^2 in the redundant features depends on the training sample size, and may also depend on the intrinsic data dimensionality. Obviously, this question requires a more careful investigation in the future. Nevertheless, our simulation study shows the possible usefulness of noise injection in the data feature space in order to reduce the generalization error of the PFLD for critical training sample sizes.

4.3 Regularization by Adding Redundant Features in the PFLD

In this section we make two attempts to understand how the addition of redundant features to the data affects the performance of the PFLD. When the data dimensionality p is larger than the number of training objects n, the PFLD constructs the linear discriminant in the linear subspace, in which the training data are located, and perpendicularly to all other dimensions where no training data are present. Therefore, we will first try to understand what happens with the training data set in the linear subspace found by the PFLD when adding redundant features to the data. Considering that adding redundant features is actually noise injection in the feature space, it is logical to suppose that it should be similar to other regularization techniques. In Section 4.3.2 we illustrate this by some mathematical analysis and a simulation study.

4.3.1 The Inter-Data Dependency by Redundant Features

Let $\{\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n\}$ be the original training data set of n objects $\boldsymbol{X}_i$, $i = 1, \ldots, n$, where each $\boldsymbol{X}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})'$ is a p-dimensional vector, and $\mu = (\mu_1, \mu_2, \ldots, \mu_p)'$ is the mean of this training data set. Suppose that ρ redundant features y_{ij}, $j = 1, \ldots, \rho$, having a normal distribution $N(0, \lambda^2)$, are added to each training vector $\boldsymbol{X}_i$. These features y_{ij} constitute an ρ-dimensional vector of redundant features $\boldsymbol{Y}_i = (y_{i1}, y_{i2}, \ldots, y_{i\rho})'$ for each training vector $\boldsymbol{X}_i$, resulting in a set of $(p+\rho)$-dimensional vectors $\{\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots, \boldsymbol{Z}_n\}$ with $\boldsymbol{Z}_i = (\boldsymbol{X}_i' \ \boldsymbol{Y}_i')' = (x_{i1}, \ldots, x_{ip}, y_{i1}, \ldots, y_{i\rho})'$, $i = 1, \ldots, n$. Now $\boldsymbol{X} = (\boldsymbol{X}_1 \ \boldsymbol{X}_2 \ \ldots \ \boldsymbol{X}_n)$, $\boldsymbol{Y} = (\boldsymbol{Y}_1 \ \boldsymbol{Y}_2 \ \ldots \ \boldsymbol{Y}_n)$ and $\boldsymbol{Z} = (\boldsymbol{Z}_1 \ \boldsymbol{Z}_2 \ldots \ \boldsymbol{Z}_n)$ are data matrices with sizes of $p \times n$, $\rho \times n$ and $(p + \rho) \times n$.

When adding redundant noise features to the data, the data dimensionality is increased. When the data dimensionality is larger than the number of training objects, the PFLD constructs the linear discriminant in the linear subspace, spanned by the training objects, and

perpendicularly to all other directions where no training objects are present. In order to understand what happens with the training objects in the linear subspace found by the PFLD when adding redundant features to the data, let us consider the data dependency matrix which describes objects averaged over features. Let us define the *data dependency matrix* as $\boldsymbol{G}(\boldsymbol{X}) = E\{(\boldsymbol{X}-\mu)'(\boldsymbol{X}-\mu)\}$. It shows the inter-dependency between the data objects (multidimensional vectors) characterizing distances between data objects and their mean in the counterbalance the usual *covariance matrix* $\boldsymbol{C}(\boldsymbol{X}) = E\{(\boldsymbol{X}-\mu)(\boldsymbol{X}-\mu)'\} = \boldsymbol{G}(\boldsymbol{X}')$, which shows the dependency between data features (describing features averaged over objects). Adding redundant features transforms the data in multidimensional space. To see how the data are transformed, let us now compare the data dependency matrix $\boldsymbol{G}(\boldsymbol{Z})$ of the extended data $\boldsymbol{Z}$ with the data dependency matrix $\boldsymbol{G}(\boldsymbol{X})$ of the original data $\boldsymbol{X}$.

The inner product $(\boldsymbol{Z}-\boldsymbol{M})'(\boldsymbol{Z}-\boldsymbol{M})$ can be written as a sum of inner products

$$(\boldsymbol{Z}-\boldsymbol{M})'(\boldsymbol{Z}-\boldsymbol{M}) = ((\boldsymbol{X}-\mu)' \ \boldsymbol{Y}')\begin{pmatrix}\boldsymbol{X}-\mu\\ \boldsymbol{Y}\end{pmatrix} =$$

$$(\boldsymbol{X}-\mu)'(\boldsymbol{X}-\mu) + (\boldsymbol{X}-\mu)'\boldsymbol{Y} + \boldsymbol{Y}'(\boldsymbol{X}-\mu) + \boldsymbol{Y}'\boldsymbol{Y}.$$

Since $\boldsymbol{X}$ and $\boldsymbol{Y}$ are statistically independent and $E\{\boldsymbol{Y}\} = 0$ (as $\boldsymbol{Y}_i \sim N(\boldsymbol{0}, \lambda^2\boldsymbol{I})$, $i = 1, \ldots, n$), the data dependency matrix of $\boldsymbol{Z}$ can be expressed as follows:

$$\boldsymbol{G}(\boldsymbol{Z}) = \boldsymbol{C}(\boldsymbol{Z}') = E\{(\boldsymbol{Z}-\boldsymbol{M})'(\boldsymbol{Z}-\boldsymbol{M})\} =$$

$$E\{(\boldsymbol{X}-\mu)'(\boldsymbol{X}-\mu) + (\boldsymbol{X}-\mu)'\boldsymbol{Y} + \boldsymbol{Y}'(\boldsymbol{X}-\mu) + \boldsymbol{Y}'\boldsymbol{Y}\} =$$

$$E\{(\boldsymbol{X}-\mu)'(\boldsymbol{X}-\mu)\} + E\{(\boldsymbol{X}-\mu)'\}E\{\boldsymbol{Y}\} + E\{\boldsymbol{Y}'\}E\{\boldsymbol{X}-\mu\} + E\{\boldsymbol{Y}'\boldsymbol{Y}\} = \boldsymbol{G}(\boldsymbol{X}) + \boldsymbol{G}(\boldsymbol{Y}).$$

Considering that components y_{ij} of the matrix $\boldsymbol{Y}$ are statistically independent variables with normal distribution $N(0, \lambda^2)$, the diagonal elements of the matrix $\boldsymbol{Y}'\boldsymbol{Y}$ have $\lambda^2\chi^2_\rho$ distributions. Therefore, $\boldsymbol{G}(\boldsymbol{Y}) = \boldsymbol{C}(\boldsymbol{Y}') = E(\boldsymbol{Y}'\boldsymbol{Y}) = \lambda^2\rho\boldsymbol{I}$, where $\boldsymbol{I}$ is the $n \times n$ identity matrix. That gives us

$$\boldsymbol{G}(\boldsymbol{Z}) = \boldsymbol{G}(\boldsymbol{X}) + \lambda^2\rho\boldsymbol{I}. \tag{4.1}$$

Formula (4.1) shows that, when adding redundant features to the data, the distances of the training objects to their mean increase and tend to be equal (the training data set is actually decorrelated) as variances $\lambda^2\rho$ of $\boldsymbol{G}(\boldsymbol{Z})$ become larger.

When the training sample size n is smaller than the data dimensionality p, the PFLD finds the linear subspace of dimensionality $n-1$, which covers all training samples, estimates the data distribution parameters there and builds a discriminant function in this linear subspace. If the original data are enlarged by redundant features, noise is added in the feature space. According to formula (4.1), by adding noise features, the distances between training objects and their mean increase. The larger the variance λ^2 and the more noise in the feature space is added, the more the data objects are "decorrelated", losing their structure (distances between data points and their mean increase and tend to be equal). For very large values of

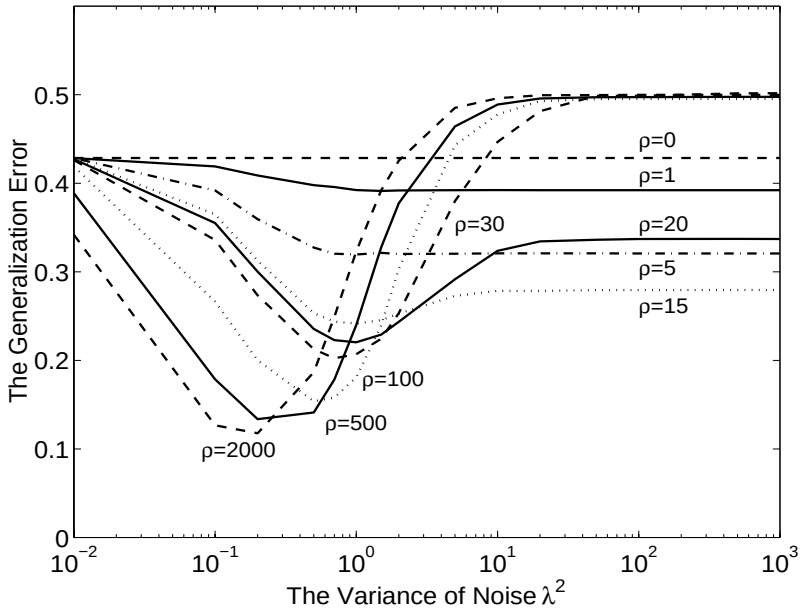


Fig. 4.7. The generalization error of the PFLD versus the noise variance λ^2 for different number of redundant features ρ for 30-dimensional Gaussian correlated data with the training sample size n=15+15=p.

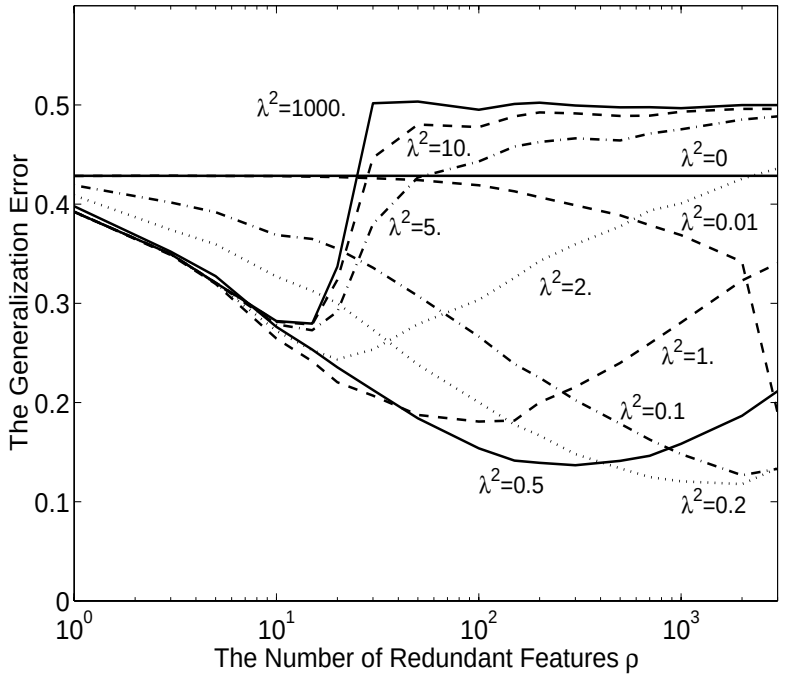


Fig. 4.8. The generalization error of the PFLD versus the number of redundant features ρ for different values of the noise variance λ^2 for 30-dimensional Gaussian correlated data with the training sample size n=15+15=p.

λ , the structure of the original data is lost. The classifier trained on such data is thus poor.

Formula (4.1) shows that the data decorrelation depends on two parameters: the number of redundant features ρ and the noise variance λ^2. Partial derivatives of $\boldsymbol{G}(\boldsymbol{Y}) = \lambda^2 \rho \boldsymbol{I}$

$$\frac{\partial}{\partial \rho} \boldsymbol{G}(\boldsymbol{Y}) = \lambda^2 \boldsymbol{I} = c_1 \boldsymbol{I}$$

and

$$\frac{\partial}{\partial \lambda} \boldsymbol{G}(\boldsymbol{Y}) = 2\lambda\rho \boldsymbol{I} = c_2 \lambda \boldsymbol{I} \ ,$$

show that the "speed" of the data decorrelation is affected more by the noise variance than by the number of redundant features. This fact is nicely illustrated by Fig. 4.7 and Fig. 4.8, where we consider the 30-dimensional Gaussian correlated data with the critical training sample size n=15+15=p. One can see clearly that the influence of the noise variance λ^2 on the generalization error of the PFLD is stronger than the influence of the number of redundant features ρ. Figures 4.7 and 4.8 also show that for each value of the noise variance an optimal value of the number of redundant features exists, and *vice versa*. Obviously, more study is required in this direction. However, it seems that in order to get a smaller generalization error by adding redundant features, it is better to add many redundant features with a small variance than a few redundant features with a large variance of noise.

4.3.2 Regularization by Noise Injection

To understand why adding redundant features can improve the performance of the PFLD, let us consider the sample covariance matrix and its decomposition used in the PFLD. As it was mentioned before, in the PFLD the pseudo inverse of the sample covariance matrix $\boldsymbol{S}$ is used. A sense of the pseudo-inverse consists in a singular value decomposition of $\boldsymbol{S}$: $\boldsymbol{TST}' = \boldsymbol{D}$, where $\boldsymbol{T}$ is an orthogonal matrix. Then the pseudo inverse of matrix $\boldsymbol{S}$

$$\boldsymbol{S}^{-1} = \boldsymbol{T}'\boldsymbol{D}^{-1}\boldsymbol{T}$$

is used instead of the direct inverse of $\boldsymbol{S}$.

We now consider what happens with the sample covariance matrix $\boldsymbol{S}$ in the PFLD when one adds redundant features to the data.

Let us keep the same definitions as above, and let $\boldsymbol{S} = \boldsymbol{C}(\boldsymbol{X})$ be the $p \times p$ sample covariance matrix of the training data set $\boldsymbol{X}$. Since $\boldsymbol{X}$ and $\boldsymbol{Y}$ are statistically independent, that gives the $(p+\rho) \times (p+\rho)$ covariance matrix of $\boldsymbol{Z}$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{S} & \boldsymbol{0} \\ \boldsymbol{0} & \lambda^2 \boldsymbol{I} \end{bmatrix}.$$

Let $\boldsymbol{T} = \begin{bmatrix} \boldsymbol{T}_1 & \boldsymbol{T}_2 \\ \boldsymbol{T}_2' & \boldsymbol{T}_3 \end{bmatrix}$ be the $(p+\rho) \times (p+\rho)$ orthogonal transformation matrix, which diagonalizes the covariance matrix $\boldsymbol{C}$: $\boldsymbol{TCT}' = \boldsymbol{D}$. Notice that $\boldsymbol{T}_1$ and $\boldsymbol{T}_3$ are $p \times p$ and

$\rho \times \rho$ symmetrical matrices, respectively. The matrix $\boldsymbol{T}_2$ is a $p \times \rho$ matrix, and is not symmetrical.

As $\boldsymbol{T}$ is an orthogonal matrix satisfying the condition

$$\boldsymbol{T}\boldsymbol{T}' = \begin{bmatrix} \boldsymbol{T}_1\boldsymbol{T}_1' + \boldsymbol{T}_2\boldsymbol{T}_2' & \boldsymbol{T}_1\boldsymbol{T}_2 + \boldsymbol{T}_2\boldsymbol{T}_3' \\ \boldsymbol{T}_2'\boldsymbol{T}_1' + \boldsymbol{T}_3\boldsymbol{T}_2' & \boldsymbol{T}_2'\boldsymbol{T}_2 + \boldsymbol{T}_3\boldsymbol{T}_3' \end{bmatrix} = \boldsymbol{I} ,$$

the following equations should hold:

$$\boldsymbol{T}_1\boldsymbol{T}_1' + \boldsymbol{T}_2\boldsymbol{T}_2' = \boldsymbol{I} ,$$
$$\boldsymbol{T}_2'\boldsymbol{T}_2 + \boldsymbol{T}_3\boldsymbol{T}_3' = \boldsymbol{I} .$$

That leads to the expressions

$$\boldsymbol{T}_2\boldsymbol{T}_2' = \boldsymbol{I} - \boldsymbol{T}_1\boldsymbol{T}_1' , \tag{4.2}$$
$$\boldsymbol{T}_3\boldsymbol{T}_3' = \boldsymbol{I} - \boldsymbol{T}_2'\boldsymbol{T}_2 . \tag{4.3}$$

Therefore

$$\boldsymbol{D} = \boldsymbol{T}\boldsymbol{C}\boldsymbol{T}' = \begin{bmatrix} \boldsymbol{T}_1 & \boldsymbol{T}_2 \\ \boldsymbol{T}_2' & \boldsymbol{T}_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{S} & \boldsymbol{0} \\ \boldsymbol{0} & \lambda^2\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{T}_1 & \boldsymbol{T}_2 \\ \boldsymbol{T}_2' & \boldsymbol{T}_3 \end{bmatrix}' = \begin{bmatrix} \boldsymbol{T}_1\boldsymbol{S}\boldsymbol{T}_1' + \boldsymbol{T}_2\lambda^2\boldsymbol{I}\boldsymbol{T}_2' & \boldsymbol{T}_1\boldsymbol{S}\boldsymbol{T}_2 + \boldsymbol{T}_2\lambda^2\boldsymbol{I}\boldsymbol{T}_3' \\ \boldsymbol{T}_2'\boldsymbol{S}\boldsymbol{T}_1' + \boldsymbol{T}_3\lambda^2\boldsymbol{I}\boldsymbol{T}_2' & \boldsymbol{T}_2'\boldsymbol{S}\boldsymbol{T}_2 + \boldsymbol{T}_3\lambda^2\boldsymbol{I}\boldsymbol{T}_3' \end{bmatrix} .$$

As $\boldsymbol{D}$ is a diagonal matrix, it should be

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{T}_1\boldsymbol{S}\boldsymbol{T}_1' + \lambda^2\boldsymbol{T}_2\boldsymbol{T}_2' & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T}_2'\boldsymbol{S}\boldsymbol{T}_2 + \lambda^2\boldsymbol{T}_3\boldsymbol{T}_3' \end{bmatrix} .$$

Substituting equations (4.2) and (4.3) yields

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_1 + \lambda^2\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D}_2 + \lambda^2\boldsymbol{I} \end{bmatrix} , \tag{4.4}$$

where $\boldsymbol{D}_1 = \boldsymbol{T}_1(\boldsymbol{S} - \lambda^2\boldsymbol{I})\boldsymbol{T}_1'$ and $\boldsymbol{D}_2 = \boldsymbol{T}_2'(\boldsymbol{S} - \lambda^2\boldsymbol{I})\boldsymbol{T}_2$ are $p \times p$ and $\rho \times \rho$ matrices, respectively.

Let us now consider the ridge estimate of the $p \times p$ sample covariance matrix $\boldsymbol{S}^* = \boldsymbol{S} + \lambda^2\boldsymbol{I}$. It is known [95] that regularization by the ridge estimate of the covariance matrix is equivalent to Gaussian noise injection to the training objects in the FLD. Let $\tilde{\boldsymbol{T}}$ be the $p \times p$ orthogonal matrix which diagonalizes the sample covariance matrix $\boldsymbol{S}$. Then by applying the orthogonal transformation to the ridge estimate $\boldsymbol{S}^*$, we obtain

$$\boldsymbol{D}^* = \tilde{\boldsymbol{T}}\boldsymbol{S}^*\tilde{\boldsymbol{T}}' = \tilde{\boldsymbol{T}}(\boldsymbol{S} + \lambda^2\boldsymbol{I})\tilde{\boldsymbol{T}}' = \tilde{\boldsymbol{T}}\boldsymbol{S}\tilde{\boldsymbol{T}}' + \tilde{\boldsymbol{T}}\lambda^2\boldsymbol{I}\tilde{\boldsymbol{T}}' = \tilde{\boldsymbol{D}} + \lambda^2\boldsymbol{I} ,$$

where $\tilde{\boldsymbol{D}} = \tilde{\boldsymbol{T}}\boldsymbol{S}\tilde{\boldsymbol{T}}'$. Thus, the ridge estimate of the sample covariance matrix $\boldsymbol{S}$ is also presented in the diagonal matrix

$$\boldsymbol{D}^* = \tilde{\boldsymbol{D}} + \lambda^2\boldsymbol{I} , \tag{4.5}$$

obtained by singular value decomposition in the PFLD.

Comparing equations (4.4) and (4.5), one can see that adding redundant features to the data in the PFLD is similar (but not equivalent) to the ridge estimate of the sample covariance matrix $\boldsymbol{S}$. To illustrate the similarity of adding redundant features to regularization techniques, such as noise injection to training objects and the ridge estimate of the covariance matrix, we consider all the data mentioned in this chapter with the critical training sample size. A hundred redundant features with different values of noise variance (but the same one for each redundant feature) were added to each data set. The averaged results for the generalization error of the PFLD without regularization, with Gaussian noise injection to the training objects, with ridge estimate of the covariance matrix and when adding redundant features, are presented in Figures 4.9, 4.10 and 4.11 for 30-dimensional Gaussian correlated data, 30-dimensional Gaussian spherical data with unequal covariance matrices and 256-dimensional cell data, respectively. The results obtained for all data sets are similar. One should take into account that the cell data set has a large dimensionality. Therefore, the optimal values of regularization parameters for different types of regularization differ more for this data set than for other two data sets. For the same reason, for the cell data the generalization error of the PFLD with regularization by adding redundant features increases more slowly with an increase of the noise variance than for other data sets. Nevertheless, in all the figures, one can see that the generalization error of the PFLD with Gaussian noise injection to the training objects and the generalization error of the PFLD with 100 redundant features added to the data behave

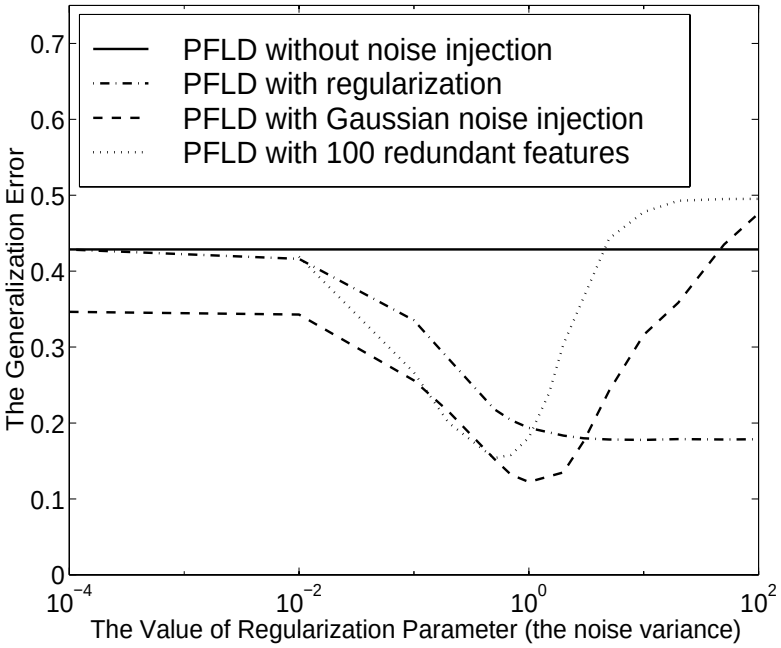


Fig. 4.9. The generalization error of the PFLD with the different type of regularization versus the value of the regularization parameter (the noise variance λ^2) for 30-dimensional Gaussian correlated data with the training sample size n=15+15=p.

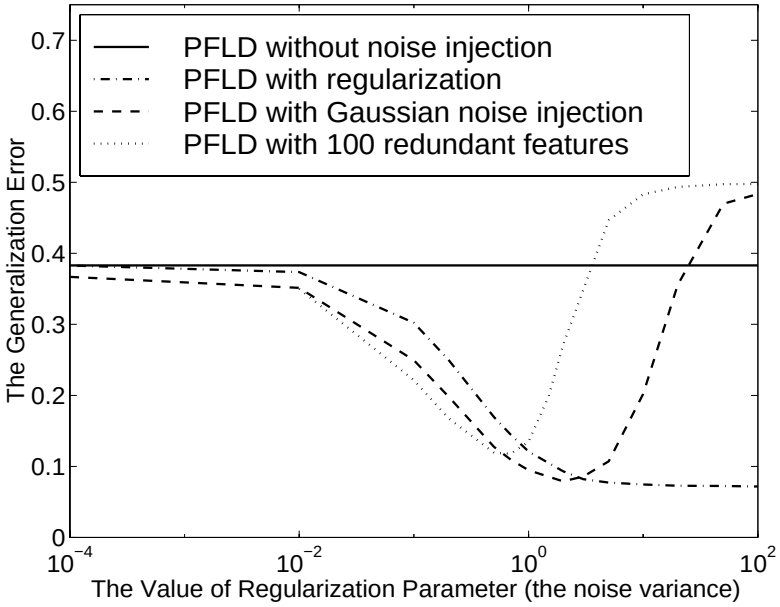


Fig. 4.10. The generalization error of the PFLD with different types of regularization versus the value of the regularization parameter (the noise variance λ^2) for 30-dimensional Gaussian spherical data with unequal covariance matrices and with the training sample size n=15+15=p.

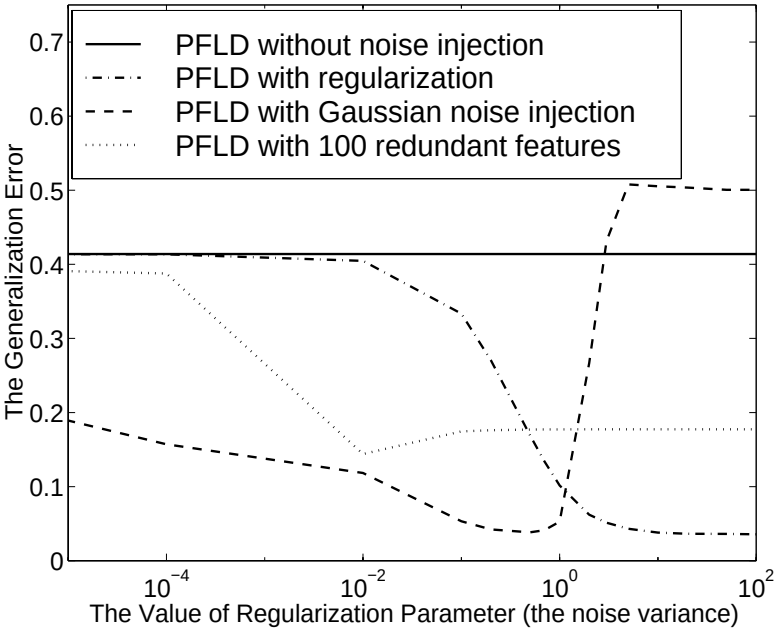


Fig. 4.11. The generalization error of the PFLD with different types of regularization versus the value of the regularization parameter (the noise variance λ^2) for 256-dimensional cell data with the training sample size n=128+128=p.

similarly. The generalization error of the PFLD with ridge estimate of the sample covariance matrix also behaves in a similar way for small values of the regularization parameter, and different for large values. However, the simulation study performed demonstrates nicely that *adding redundant features to the data is similar to other regularization techniques*.

4.4 Conclusions and Discussion

The PFLD may have a peaking behaviour of the generalization error for training sample sizes that are the same as the feature size. Based on the small sample size properties of the PFLD, in this chapter it has been suggested that injecting noise into the data feature space improves the generalization error of the PFLD for critical training sample sizes. Similar to bagging (see Chapter 5), adding redundant features to the data has a shifting effect on the learning curve, which describes a behaviour of the generalization error versus the training sample size. By adding redundant features into the data feature space, the generalization error of the classifier is shifted with reference to the generalization error of the classifier constructed on the training data set with the original dimensionality in the direction of the generalization error of the classifier constructed on a smaller training set with the original dimensionality. This approach was studied for two artificial data sets and one example of real data. Simulation results have shown that adding redundant noise features to the data allows us to dramatically reduce the generalization error of the PFLD in the region of critical training sample sizes.

Mathematical analysis and simulation studies have shown that adding noise by redundant features is similar to other regularization techniques, such as Gaussian noise injection to the training data and the ridge estimate of the covariance matrix. It was noticed that there appears to be an optimal relationship between the number of redundant features and the noise variance. This optimal relation might depend upon the size of the training data set and the intrinsic data dimensionality. However, it still needs more investigation to find an explicit expression.

Finally, let us note that some other classifiers (e.g. the regularized FLD and the regularized QC, both with a small value of the regularization parameter λ, close to zero) may also have a peaking behaviour of the generalization error in the region of critical training sample sizes. Therefore, it could be expected that adding redundant features could help to improve the performance of such classifiers constructed on critical training sample sizes. To verify our expectations let us consider the RQC with λ=0.001. To reduce computational costs in our experimental investigations, we have used only the first 10 features of the Gaussian correlated data and Gaussian spherical data with unequal covariance matrices. Figures 4.12 and 4.13 confirm that adding redundant features is also useful in the case of the RQC allowing us to considerably reduce the generalization error in the region of critical training sample sizes.

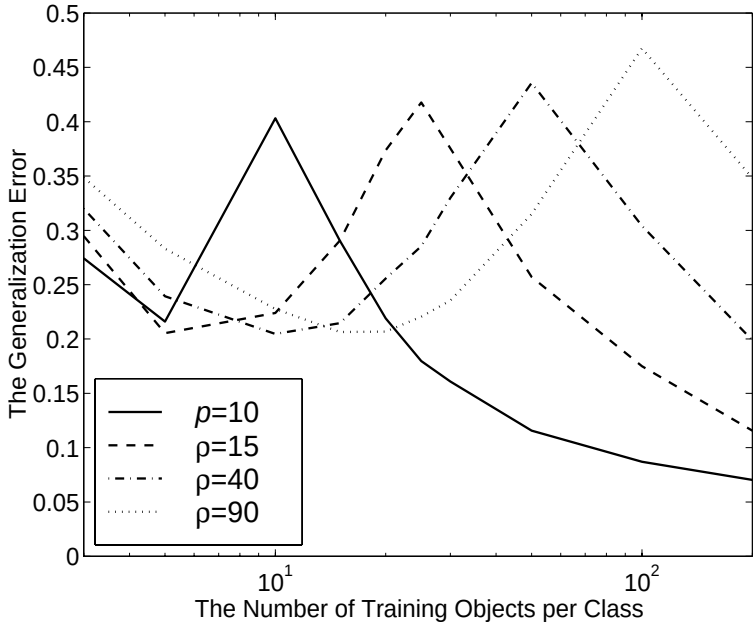


Fig. 4.12. The generalization error of the RQC (λ=0.001) versus the training sample size for 10-dimensional Gaussian correlated data without noise injection in the feature space (p=10) and with ρ=15, 40, 90 additional redundant noise features (p+ρ=25, 50, 100).

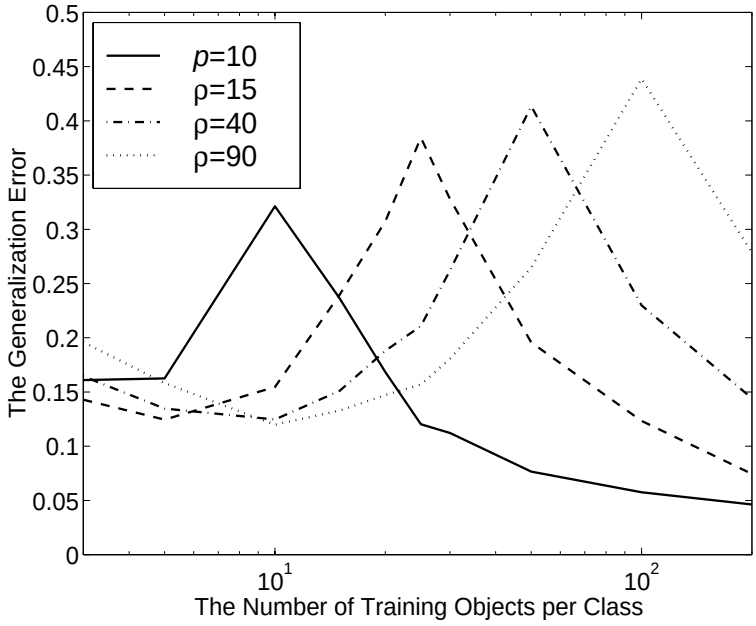


Fig. 4.13. The generalization error of the RQC (λ=0.001) versus the training sample size for 10-dimensional Gaussian spherical data with unequal covariance matrices without noise injection in the feature space (p=10) and with ρ=15, 40, 90 additional redundant noise features (p+ρ=25, 50, 100).

Chapter 5

Bagging for Linear Classifiers

5.1 Introduction

The main problem in building classifiers on small training sample sets is that it is impossible to estimate parameters of the data distribution properly. Moreover, a small training sample set may present the total data set incorrectly. Thus, classifiers built on small sample sets are biased, and may have a large variance in the probability of misclassification [54, 91, 23]. For this reason they are unstable. In order to make a classifier more stable a larger training sample set is needed or stabilizing techniques have to be used.

In many applications the total amount of data is not large and therefore the training data set is limited. In this case one can try to use different techniques to get more stable solutions. It still is an open question which stabilizing techniques are good: do they really stabilize the solution, and do they improve the performance of the classifier?

In the previous chapters we have investigated regularization techniques such as: 1) ridge estimate of the covariance matrix for the Fisher linear discriminant function; 2) weight decay in the training of a linear single layer perceptron; and 3) noise injection to the training objects. It has been shown that these techniques are indeed stabilizing (see Chapter 1).

In this chapter, *bagging* (**b**ootstrapping and **agg**regat**ing** [15]) is studied for a number of linear classifiers. It is well-known that the bootstrap estimate of the data distribution parameters is robust [103] and more accurate than the plug-in estimate [26] usually used in classification rules. So it might be promising to use bootstrapping and aggregating techniques to get a better classifier [126, 135] with a more stable solution. We discuss bagging, its use and a modification proposed by us, which we call “nice” bagging, in Section 5.2. Bagging is mainly investigated for regression and classification trees [14]. Studying bagging for decision trees, Breiman [15] has noticed that the efficiency of bagging depends on the stability of the prediction or classification rule. He and other researchers (see, for example, [15, 17, 21]) were mostly performing their simulation study on data sets which had very large training samples. For some classifiers, such as decision trees, it does not affect the stability of the classification rule. However, for other classifiers, e.g. linear ones, it does. Usually, linear classifiers are unstable, when they are constructed on small training samples, and stable, when large training samples are used. As large training samples were used in the simulation study performed by Breiman [17] on *Linear Discriminant Analysis* (LDA), this simulation study has not shown the usefulness of bagging.

In order to study the stability of classifiers and the effect of bagging and other regularization techniques, one needs to have a measure for the stability. One possible measure

is the standard deviation of the mean classification error. However, it measures more the stability of the classification error than the stability of the discriminant function. Therefore, we need a measure, which also depends on the size and the composition of the training set used to construct the classifier. In Section 1.5 of Chapter 1 such a possible measure is introduced, which we call the *instability*. This instability measure applied to linear classifiers is discussed by us in the relation to the performance of these classifiers in Section 5.3. It is demonstrated that the instability of linear classifiers often depends on the training sample size and is correlated with the classifier performance.

We study the influence of regularization and bagging on the stability and on the generalization error of linear classifiers: whether they really stabilize the classifier and whether they improve the generalization error of the classifier. The relation between the performance and the stability of bagged linear classifiers is investigated in Sections 5.4 and 5.5. Our simulation study shows that in comparison with regularization, which really stabilizes the solution of the classifier, bagging is not a stabilizing technique: bagging improves the performance of the classifier only for the critical training sample sizes (when the training sample size is comparable to the data dimensionality) and only in rather unstable situations.

Conclusions are summarized in Section 5.6.

5.2 Bagging

5.2.1 Standard Bagging Algorithm

Bagging is based on bootstrapping [26] (see Section 1.5 in Chapter 1) and aggregating concepts and presented by Breiman [15]. Both, bootstrapping and aggregating may be beneficial. Bootstrapping is based on random sampling with replacement. Therefore, taking a bootstrap replicate $\boldsymbol{X}^b = (\boldsymbol{X}_1^b, \boldsymbol{X}_2^b, \ldots, \boldsymbol{X}_n^b)$ (the random selection with replacement of n objects from the set of n objects) of the training set $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$, one can sometimes avoid or get less misleading training objects in the bootstrap training set. Consequently, a classifier constructed on such a training set may have a better performance. Aggregating actually means combining classifiers. Often a combined classifier gives better results than individual base classifiers in the ensemble, because of combining the advantages of the individual classifiers in the final classifier. Therefore, bagging might be helpful to build a better classifier on training sets with misleaders (objects that misrepresent the real data distribution). In bagging, bootstrapping and aggregating techniques are implemented in the following way.

1. Repeat for $b = 1, \ldots, B$.
   a) Take a bootstrap replicate $\boldsymbol{X}^b$ of the training data set $\boldsymbol{X}$.
   b) Construct a base classifier $C_b(\boldsymbol{x})$ (with a decision boundary $C_b(\boldsymbol{x}) = 0$) on $\boldsymbol{X}^b$.
2. Combine base classifiers $C_b(\boldsymbol{x})$, $b = 1, \ldots, B$, by the simple majority vote (the most often

predicted label) to a final decision rule $\beta(\boldsymbol{x}) = \underset{y \in \{-1, 1\}}{argmax} \sum_b \delta_{\mathrm{sgn}(C_b(\boldsymbol{x})), y}$, where $\delta_{i,j} = \begin{cases} 1, & i = j; \\ 0, & i \neq j; \end{cases}$ is Kronecker symbol, y is a possible decision (class label) of the classifier.

Let us note that bagging has been designed for decision trees, where it is difficult to imply combining rules other than voting. However, when applying bagging for statistical classifiers other combining rules may be used.

5.2.2 Other Combining Rules

In the final decision rule one may use combining rules based on the posteriori probabilities $P(\pi_i|\boldsymbol{x})$, $\boldsymbol{x} \in \boldsymbol{X}$, $i = 1, \ldots, \kappa$, related to the distances to the discriminant function $C(\boldsymbol{x}) = 0$. Usually, the posteriori probability is defined as the sigmoid of the output of the discriminant function $C(\boldsymbol{x})$

$$P(\pi_i|\boldsymbol{x}) = sigmoid(C(\boldsymbol{x})) = \frac{1}{1 + \exp(-C(\boldsymbol{x}))} . \tag{5.1}$$

However, for linear and polynomial classifiers, it may be necessary to scale (normalize) the coefficients of the discriminant function $C(\boldsymbol{x})$ in order to get the best posteriori probabilities $P(\pi_i|\boldsymbol{x})$ for all training objects $\boldsymbol{x} \in \boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$. This may be done by optimizing an additional scaling parameter α for the discriminant function $C(\boldsymbol{x})$ in the one-dimensional logistic classifier [5], which maximizes the likelihood of the sigmoid of the posteriori probabilities obtained on the training data set

$$\max_{\alpha} \prod_{\boldsymbol{x} \in \boldsymbol{X}} P(\pi_i|\boldsymbol{x}) = \max_{\alpha} \prod_{\boldsymbol{x} \in \boldsymbol{X}} sigmoid(\alpha C(\boldsymbol{x})) . \tag{5.2}$$

The most popular combining rules based on the posteriori probabilities are the minimum, the maximum, the mean and the product rules. When using the *minimum* (or the *maximum*) combining rules, the final decision of an ensemble of classifiers $\underset{\pi_i \in \Pi}{argmax} P_\beta(\pi_i|\boldsymbol{x})$, $\Pi = \{\pi_1, \ldots, \pi_\kappa\}$, is made by that base classifier in the ensemble which has the minimal (or the maximal) posteriori probability of the classified object $\boldsymbol{x}$ among all classifiers $C_b(\boldsymbol{x})$, $b = 1, 2, \ldots, B$, participating in the ensemble: $P_\beta(\pi_i|\boldsymbol{x}) = \min_b P_b(\pi_i|\boldsymbol{x})$ (or $P_\beta(\pi_i|\boldsymbol{x}) = \max_b P_b(\pi_i|\boldsymbol{x})$). In the *mean* combining rule, the decision is made according to the mean of the posteriori probabilities given by the base classifiers: $P_\beta(\pi_i|\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} P_b(\pi_i|\boldsymbol{x})$. In the *product* rule, the decision is made by the product of posteriori probabilities presented by the base classifiers: $P_\beta(\pi_i|\boldsymbol{x}) = \prod_{b=1}^{B} P_b(\pi_i|\boldsymbol{x})$.

When combining linear and polynomial classifiers, one may also use the *average* combining rule when one averages the coefficients of the base classifiers $C_\beta(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} C_b(\boldsymbol{x})$. However, before averaging, the coefficients of the classifiers $C_b(\boldsymbol{x})$ should be scaled in order to get, for all training objects $\boldsymbol{x} \in \boldsymbol{X}$, the best posteriori probabilities $P_b(\pi_i|\boldsymbol{x})$, $i = 1, \ldots, \kappa$. The scaling coefficients α_b for each classifier $C_b(\boldsymbol{x})$ may be found

by maximizing the likelihood of the sigmoid of the posteriori probabilities (5.2). "Combining by averaging" performs similarly to "combining by majority vote" [59]. However, the averaging has an advantage over voting and combining rules based on the posteriori probabilities: it is not necessary to store *all* classification results (class labels or posteriori probabilities) of all the base classifiers $C_b(\boldsymbol{x})$, $b = 1, 2, \ldots, B$. It is enough to store only the coefficients of the base classifiers. Additionally, one obtains just a single classifier as a final decision rule, which has the same complexity (the same number of parameters (coefficients)) as each of the base classifiers.

As the average combing rule performs similarly to voting and it does not increase the complexity of the final classifier, we will use this combining rule when applying bagging to linear classifiers.

5.2.3 Our Bagging Algorithm

Applying bagging to linear classifiers makes it possible to use the average combining rule. Thus, bagging is organized by us in the following way.

1. Repeat for $b = 1, 2, \ldots, B$.
   a) Take a bootstrap replicate $\boldsymbol{X}^b$ of the training data set $\boldsymbol{X}$.
   b) Construct a linear classifier $C_b(\boldsymbol{x})$ on $\boldsymbol{X}^b$.
2. Combine linear classifiers by averaging their coefficients into a final decision rule $C_\beta(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} C_b(\boldsymbol{x})$ (with the decision boundary $C_\beta(\boldsymbol{x}) = 0$).

It is also possible to modify bagging in some way. We introduce here just one modification which we call *"nice" bagging*. "Nice" bagging is bagging in which only "nice" base classifiers (bootstrap versions of the classifier) are averaged. "Nice" base classifiers are those classifiers for which the error on the training set (the apparent error) is not larger than the error of the original classifier on the same (not bootstrapped) training set. In other words, "nice" bagging builds a classifier based on the apparent classification error. The properties of bagging and "nice" bagging for linear classifiers are investigated in Section 5.5.

5.2.4 Why Bagging May Be Superior to a Single Classifier

As mentioned before, bagging is based on bootstrapping the training set and aggregating (combining) the bootstrap versions of the original classifier. When bootstrapping the training set $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$, the probability that the training object $\boldsymbol{X}_j$ ($j = 1, \ldots, n$) is selected $m = 0, 1, 2, \ldots$ times in a bootstrap sample $\boldsymbol{X}^b$ is $b\left(m \mid n, \frac{1}{n}\right) = C_n^m \left(\frac{1}{n}\right)^m \left(1 - \frac{1}{n}\right)^{n-m}$, $C_n^m = \frac{n!}{m!(n-m)!}$. For large n, binomial distribution can be approximated by Poisson

distribution $b\left(m|n,\frac{1}{n}\right)\approx\frac{e^{-1}}{m!}$. So each training object has probability approximately $\frac{1}{e}$ of being left out of a bootstrap sample. Therefore, on average, approximately 37% of the objects are not presented in the bootstrap sample. This means, that possible "outliers" in the training set sometimes do not show up in the bootstrap sample. By that, better classifiers (with a smaller apparent error) may be obtained by the bootstrap sample than by the original training set. These classifiers will appear "sharper" in the apparent error than those obtained on the training sets with outliers and, therefore, they will be more decisive than other bootstrap versions in the final judgement. Thus, aggregating classifiers (by voting or averaging) in bagging can sometimes give a better performance than individual classifiers. To illustrate this, let us consider the *Fisher Linear Discriminant* (FLD) (for a description of the FLD, see Chapter 1) applied to an imaginary two-class problem of one-dimensional data with one outlier. In Fig. 5.1, one can see that the FLD constructed on the full data set is not able to separate the data classes without error. Constructing classifiers on bootstrap samples of the training sets sometimes gives a better classifier, sometimes a worse one. The better classifier is sharper (its sigmoid (5.1) is steeper) in the domain of the posteriori probabilities. By that, it dominates in the final decision. Therefore, aggregating (combining) bootstrap versions of the original classifier allows us to get a better classifier than the original one.

Another example, presented in Fig. 5.2, shows the case, when the average of bootstrap versions of the classifier gives a solution (a discriminant function), that is not reached by separate bootstrap versions of the classifier. In this example, two-dimensional Gaussian correlated data are used (see Fig. 1.1 in Chapter 1). The *Nearest Mean Classifier* (NMC) (for description, see Chapter 1) is bagged on 100 bootstrap replicates of a training data set consisting of 10 objects. To represent linear discriminants as points in a two-dimensional plot, we have additionally scaled all coefficients $(\boldsymbol{w},w_0)$ (see formula (1.3) in Chapter 1) of each discriminant function by dividing them by a free coefficient w_0. A scatter plot of the NMC, its bootstrap versions and its bagged version in the space of normalized coefficients w_1/w_0 and w_2/w_0 is shown in Fig. 5.2. One can see that the bagged (aggregated) classifier gives a solution inside an empty space between different bootstrap versions of the classifier. Separate bootstrap versions of the classifier can hardly reach the solution obtained by aggregating (averaging) bootstrap versions of the classifier. The experiments described below show that the bagged version of the NMC yields a better expected performance in this example.

5.2.5 Discussion

Bagging, its different modifications and the performance of bagged classifiers were investigated by a number of researchers [15, 17, 21, 83, 118, 126, 135]. Bagging was usually studied for regression problems. Breiman [15] has shown that bagging could reduce the error

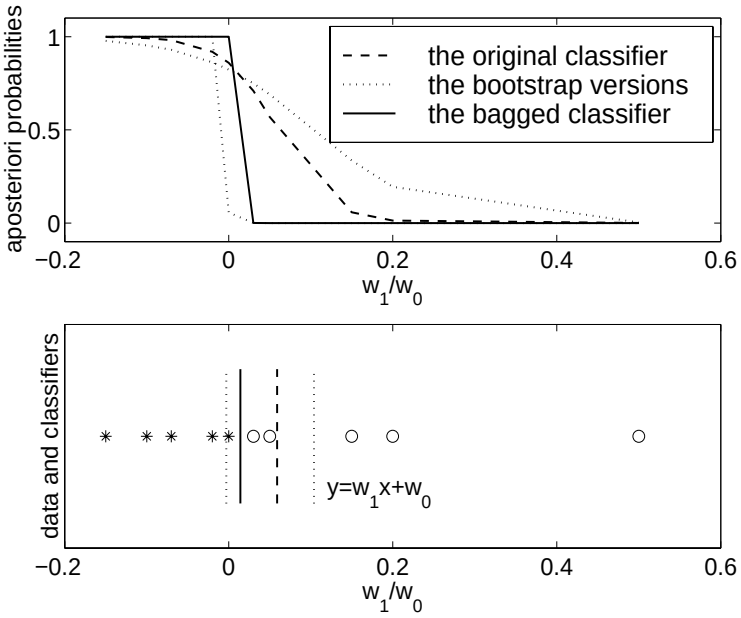


Fig. 5.1. Scatter plot of the one-dimensional data set with one outlier, the discriminant functions and the posteriori probabilities of the original training set, obtained for the FLD built on the original training set, for the two bootstrap versions of the FLD and for the bagged FLD constructed from these two bootstrap versions.

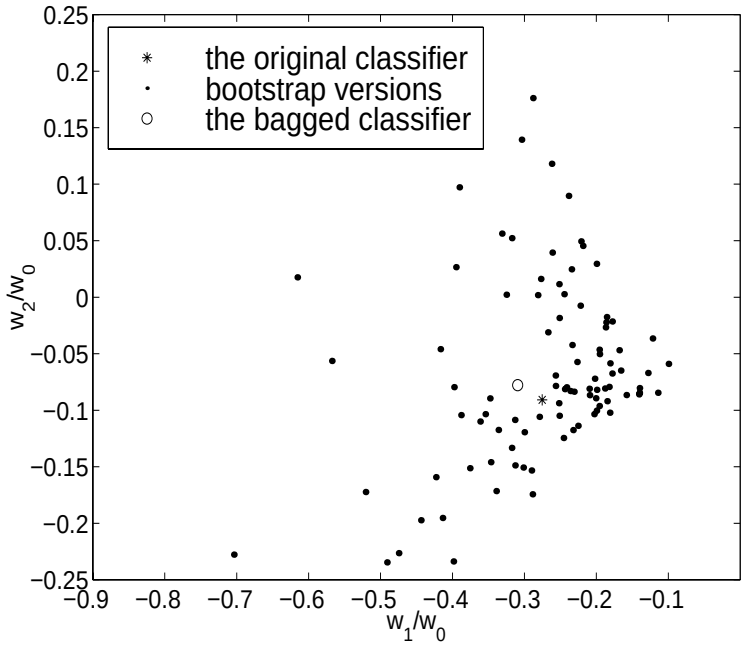


Fig. 5.2. Scatter plot of the two-dimensional projection of the Nearest Mean discriminant in the space of its normalized coefficients w_1/w_0 and w_2/w_0 for 2-dimensional Gaussian correlated data set.

of linear regression and the classification error of decision trees. He noticed that bagging is useful for unstable procedures only. For stable ones it could even deteriorate the performance of the classifier. He has shown that bagging for the Nearest Neighbour Classifier is not a good idea. Breiman [17] has also studied bagging for *Linear Discriminant Analysis* (LDA). He has found that bagging is useless for LDA, as LDA is stable in general. We disagree with this conclusion. In Section 5.3, we show that stability of linear classifiers depends on the training sample size. In most cases Breiman has used training sets of a huge size. Thus, he was considering the situations, when the LDA was very stable. In these stable situation bagging is indeed useless and unnecessary. The central problem has been solved by assembling a large data set.

Another popular aggregating technique is *boosting* (or arcing) [32, 109, 17, 31], which should not be confused with bagging. Boosting is proposed by Freund and Schapire [32]. Boosting consists in sequentially applying a classification algorithm to reweighted versions of the training data, and then taking a weighted majority vote of the sequence of classifiers produced. Initially, all weights are set equally, but on each round (iteration), the weights (probabilities) of incorrectly classified objects are increased by a factor that depends on the weighted training set error. By this, the classifier is forced to focus on the difficult objects in the training set, maximizing the margins of the training objects [107]. As a rule, difficult objects are objects situated on the border between data classes or outliers. Both, bagging and boosting, construct the set of base classifiers using the modified training set and then combine them in the final classification rule. However, they differ in the way these base classifiers are obtained. Boosting has two main differences in comparison with bagging: 1) *all* objects of the training data set are used (no bootstrapping) at each stage of boosting; 2) a classifier, obtained at each iteration of boosting, *strongly depends* on all of the previously obtained classifiers (in bagging, all obtained bootstrap versions are independent). Therefore, as one can see, boosting seems to be fundamentally different from bagging in the way that base classifiers are obtained. *Arcing* (**a**daptive **r**esampling and **c**ombin**ing**) proposed by Breiman [17] is a modification of boosting, which uses boosting-by-resampling (training samples are generated by selecting examples at random according to the distribution over the training set) instead of boosting-by-reweighting (when learning algorithm works directly with a weighted training sample). However, arcing is still a sequential algorithm with probabilistic resampling of the training set instead of independent resampling, which is used in bagging. We study boosting for linear classifiers and compare it with bagging in Chapter 6.

In connection with the above discussion, one can see that to study the dependence of the efficiency of bagging on the stability of a classifier and the stability of bagged classifiers is of great interest. We perform our study for linear discriminant analysis. One of the reasons to choose linear classifiers is their simplicity. Another reason is to show that they are not always as stable as one would like and, therefore, applying bagging can be useful in some situations.

5.3 Experimental Study of the Performance and the Stability of Linear Classifiers

We study five linear classifiers on three data sets. The studied classification rules are the *Nearest Mean Classifier* (NMC), the *Regularized Fisher Linear Discriminant* (RFLD), the *Pseudo Fisher Linear Discriminant* (PFLD), the *Small Sample Size Classifier* (SSSC) and the *Karhunen-Loeve Linear Classifier* (KLLC), which all are described in Chapter 1.

Two artificial data sets and one real data set are used for our experimental investigations. These data sets have a high dimension because we are interested in critical situations where classifiers have unstable solutions. These data are also described in Chapter 1. They are

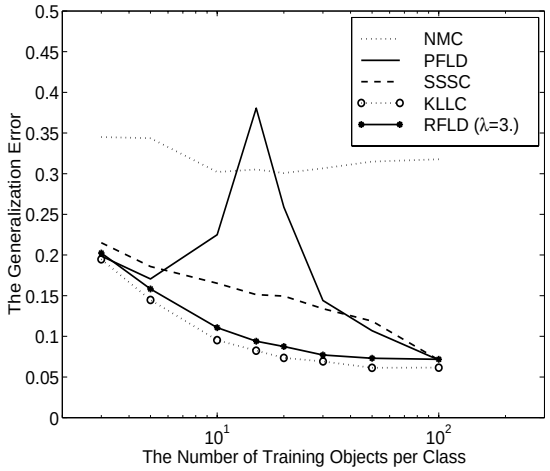
- the 30-dimensional correlated Gaussian data set;
- the 30-dimensional Non-Gaussian data set;
- the cell data set.

As the performance and stability of classifiers often depend on the training sample size, we consider training data sets of different sizes. Training sets with 3 to 100 objects (300 objects for the cell data) per class are chosen randomly from a total set. The remaining data are used for testing. All following experiments are repeated 50 times. Therefore, in all figures below, the averaged results over 50 repetitions are presented. We should notice that calculating the instability measure, which is described in details in Chapter 1, is very costly, because the bootstrapping of the training set is involved in the procedure. In order to evaluate the instability of the classifier on one separate training set, we must draw 25 bootstrap samples of this training set and construct a base classifier on each of them. This procedure should then be repeated 50 times on the independent training sets having the same size in order to get the averaged results.

Let us now consider the relation between the performance and the stability of the classifiers. In Fig. 5.3 the behaviour of the generalization error of the linear classifiers for three considered data sets are presented. For the same data sets, Fig. 5.4 introduces the instability of linear classifiers, measured by the test set, as well as the instability, measured by the training set. Both show how changes in the composition of the training set affect the classifier. The instability measured by the test set is more objective than the instability measured by the training set. However, it can not be used in practice. Therefore, it is of interest to study both of them.

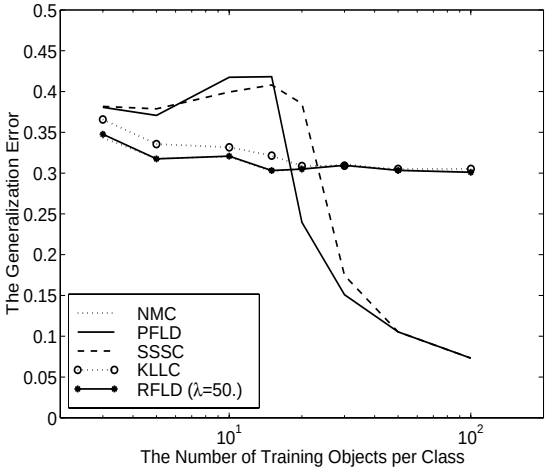
When comparing the performance of the classifiers and their instability measured on the test set, one can see that the performance and the instability of classifiers are correlated: more instable classifiers perform worse. Let us consider the PFLD. For all data, the PFLD shows a critical behaviour with a high maximum of the generalization error around $N=p$ [119] (see also Chapter 4). This maximum corresponds to the instability maximum exactly at the same point. Another example, which nicely illustrates that the performance and the instability of the classifier are related, is the RFLD (see Fig. 5.5). The RFLD represents the large family of

30-dimensional Gaussian correlated data



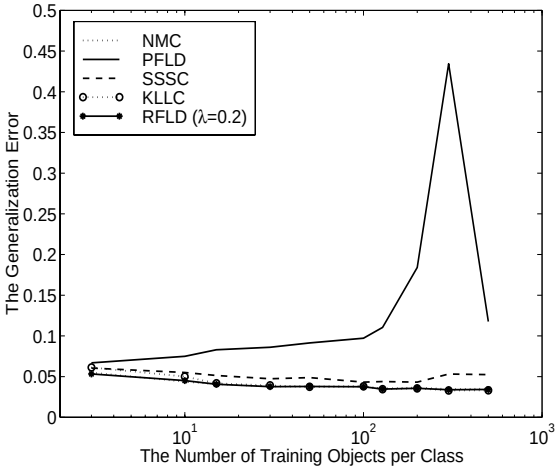
a)

30-dimensional Non-Gaussian data



b)

256-dimensional cell data



c)

Fig. 5.3. The generalization error of the NMC, the PFLD, the SSSC, the KLLC and the RFLD versus the training sample size for 30-dimensional Gaussian correlated data (a), for 30-dimensional Non-Gaussian data (b) and for 256-dimensional cell data (c), respectively.

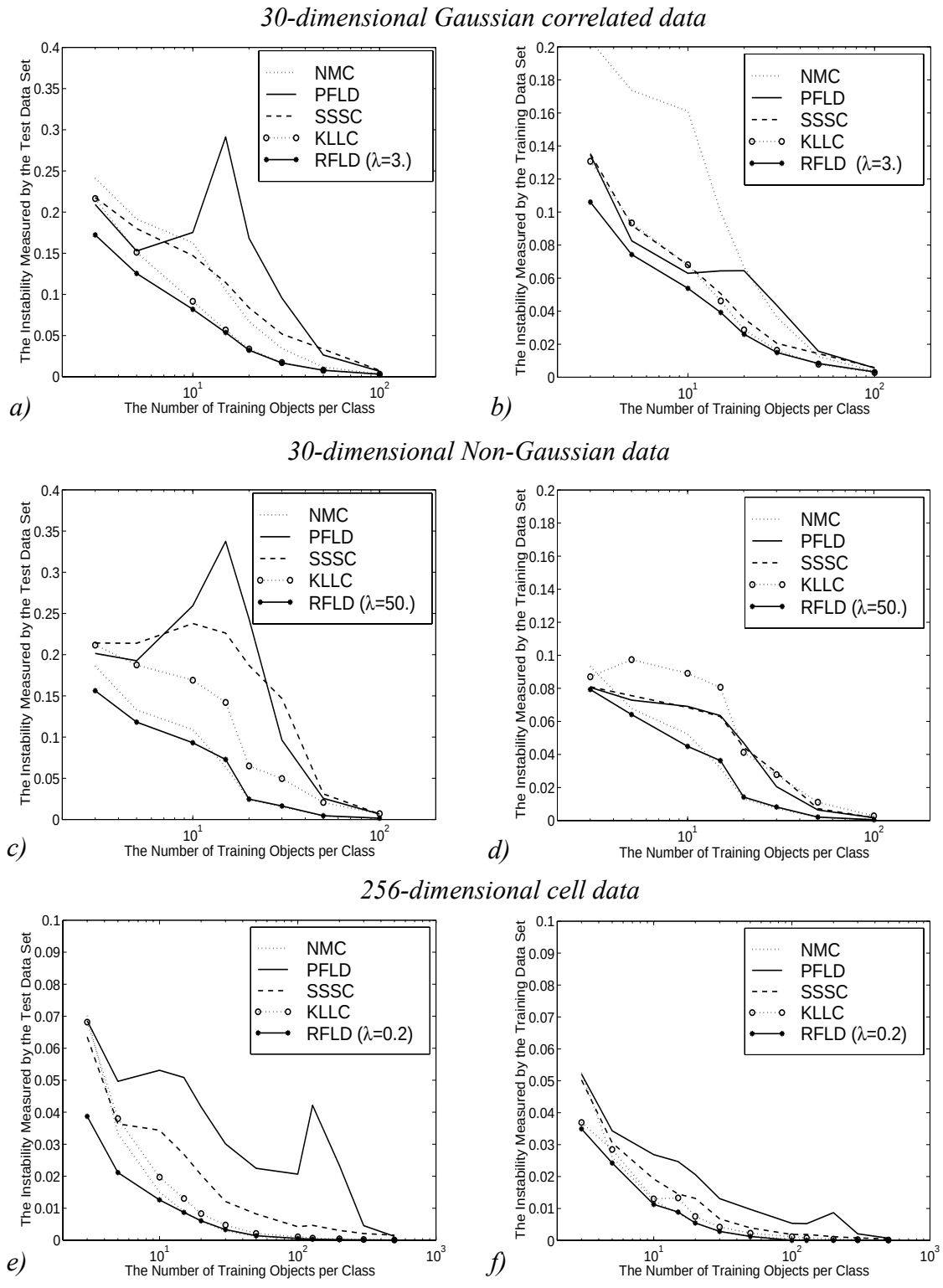


Fig. 5.4. The instability, measured by the test data set (left) and by the training data set (right), of the NMC, the PFLD, the SSSC, the KLLC and the RFLD for 30-dimensional Gaussian correlated data versus the training sample size (a,b), for 30-dimensional Non-Gaussian data (c,d) and for 256-dimensional cell data (e,f), respectively.

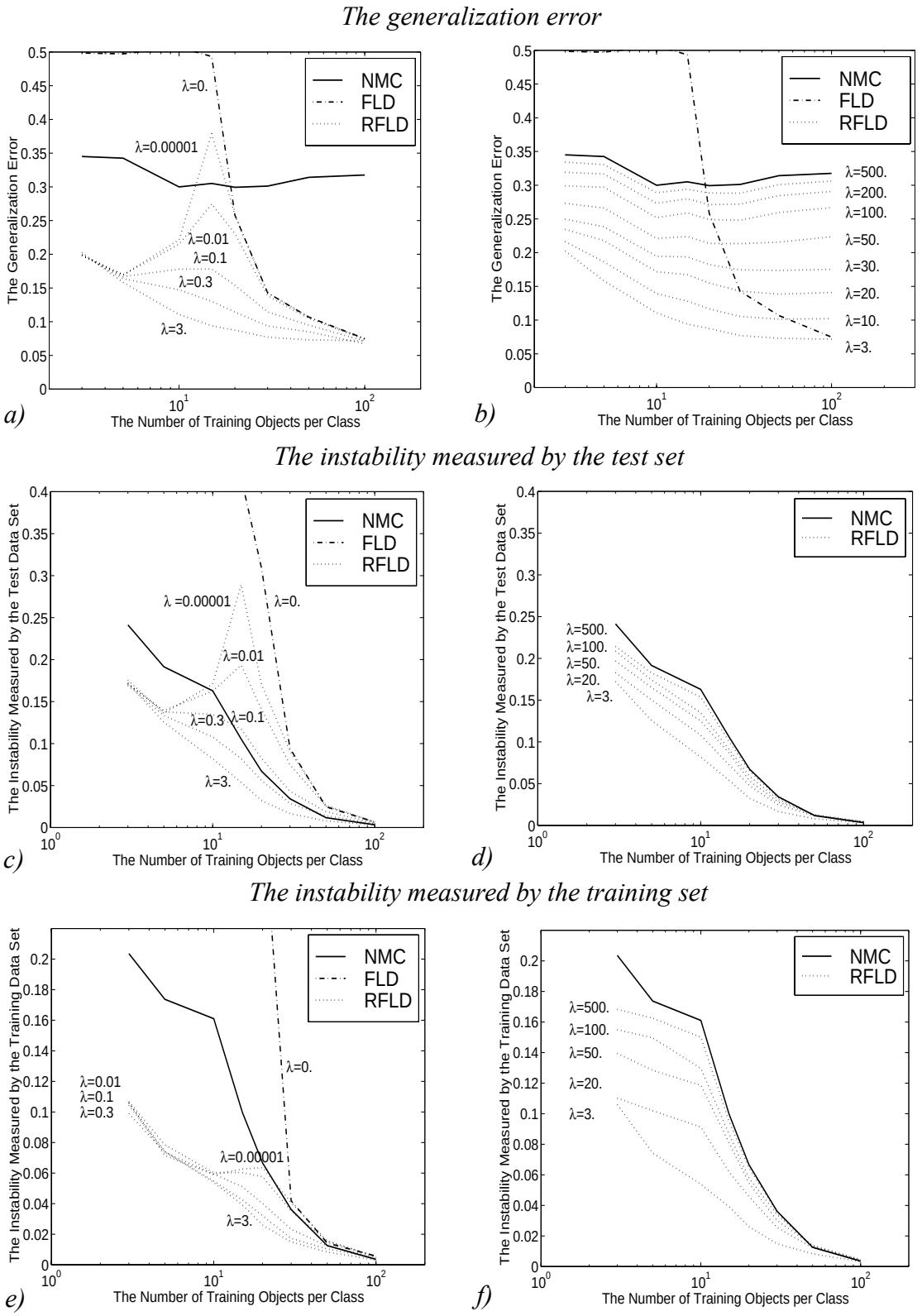


Fig. 5.5. The generalization error (a,b) and the instability of the RFLD, measured by the test set (c,d) and by the training set (e,f), with different values of the regularization parameter λ for 30-dimensional Gaussian correlated data versus the training sample size respectively.

linear classifiers starting from the FLD (when λ=0) and ending at the NMC (when $\lambda = \infty$). At first, with an increase in the value of the regularization parameter λ from 0 to 3, the instability and the generalization error of the RFLD decrease to their minimal values. Further increasing the value of the regularization parameter increases the instability and the generalization error of the RFLD up to the instability and the generalization error of the NMC. Thus, some values of the regularization parameter in the RFLD help to stabilize the classifier. In this way the generalization error of the regularized classifier decreases as well. Therefore, this example also shows that regularization by the ridge estimate of the covariance matrix is indeed a stabilizing technique.

In general, the instability of a classifier depends on many factors such as the complexity of the classifier, the distribution of data used to construct the classifier, and the sensitivity of the classifier to the size and the composition of the training data set.

As it was mentioned before, the instability, measured by the test set (Fig. 5.4a,c,e and Fig. 5.5c,d), is more objective. It shows, how much a discriminant function alters with the changes in the composition of the training set. When considering the instability measured by the test set, one can see that, for all data sets, the classifiers are the most unstable for small training sample sizes (with the exception of the PFLD discussed above). For small numbers of training objects, even small changes in the composition of the training set may heavily affect the discriminant function. After increasing the number of training objects above the critical value (the data dimensionality), the classifiers become more stable and their generalization errors decrease. The composition of the training set becomes less important if the training data set is sufficiently large to represent correctly the true distribution of the data set. Only the Nearest Mean Classifier is an exception due to its insensitivity to the number of training samples [89]. In spite of becoming more stable with an increasing training set, its generalization error does not change much.

The instability, measured by the training data set (Fig. 5.4b,d,f and Fig. 5.5e,f), is more biased, because the same training set is used for both constructing a classifier and estimating its instability. Let us notice, that in Fig. 5.4 a different scaling is used for plotting the instability, measured by the training set, and the instability, measured by the test set. However, one can see, that the instability, measured by the training set, shows the same behaviour as the instability, measured by the test set. The instability of the classifiers decreases when increasing the training sample size.

Unfortunately, the instability measured by the test set can not be observed in practice. For this reason, we will investigate in the next sections only the instability as measured by the training set. We will call it further *the instability*. The instability illuminates how much changes in the composition of the training set affect the discriminant function. Linear classifiers are unstable when the training sample size is small or comparable to the data dimensionality. In bagging, one combines classifiers obtained on the bootstrap versions of the training set.

Bootstrapping is the most effective for relatively small training sets, as small training sets are the most sensitive to perturbations in their composition. Bootstrap replicates of small training sets differ more from each other (by statistical characteristics) than bootstrap replicates of large training sets. Therefore, in bagging one may obtain more diverse classifiers when using small training sets than when using large ones. Combining diverse classifiers is more useful than combining similar ones. Therefore, it seems that bagging may be useful for classifiers constructed on relatively small training sets, when the training set and consequently the classifier are the most affected by perturbations in the composition of the training set. It implies that the instability of the classifier may be very promising in order to predict the usefulness of bagging.

In the next sections we will consider the relation of the instability to the performance of bagged linear classifiers and the relation of the instability to the usefulness of bagging.

5.4 Bagging Is Not a Stabilizing Technique

It is known [15, 17] that bagging improves the performance of classifiers only when they are unstable. However, it is still unclear what is the exact relation between bagging and the instability of classifiers: when bagging is useful and whether it stabilizes classifiers.

To study the effect of bagging on the performance and on the instability of classifiers, let us consider the NMC, constructed on the two training data sets with the sizes n=3+3 and n=10+10 of Gaussian correlated data. When applying bagging and "nice" bagging to these situations (see Fig. 5.6), one can see that, strictly speaking, *bagging does not stabilize the original classifier*. The bagged classifier could be both, less and more stable than the original classifier. It merely depends on the number of base classifiers (bootstrap replicates) used to build the bagged classifier. Taking a bootstrap sample of the original training set, the size of the training set actually becomes smaller. By that, the instability of the classifier constructed on the bootstrap sample of the training set increases. Therefore, the bagged classifier constructed only on a few bootstrap replicates is more unstable than the original classifier. The stability of the bagged classifier increases with an increase in the number of bootstrap replicates used to construct the classifier. It holds for both the bagged and the "nicely" bagged NMC. However, how many bootstrap versions should be used to get a more stable classifier than the original one is not clear. Sometimes it is enough to use 10 bootstrap replicates to get a more stable classifier than the original one. Sometimes 1000 bootstrap replicates are not enough to surpass the stability of the original classifier. It may depend on many factors such as the data distribution, the data dimensionality, the training sample size, the complexity of the classification rule and so forth.

Fig. 5.6 shows that *the generalization error and the instability of the bagged classifier are related*. The correlation between the generalization error and the instability for the bagged

and "nicely" bagged classifier on the Gaussian correlated data for the training sample size N=3 per class is 0.38 and 0.68, respectively. For the training sample size N=10 per class these correlations are 0.97 and 0.98, respectively. However, when the bagged or "nicely" bagged classifier starts to improve the generalization error of the original classifier, it can be both, less and more stable than the original classifier. One can see that the "nice" bagging performs better than bagging. The "nicely" bagged NMC is also more stable than the bagged NMC.

Let us consider now the influence of the number of base classifiers (bootstrap replicates) used to construct the bagged classifier on the performance and the instability of the bagged classifier. Fig. 5.6 shows that after a certain value the increase of the number of base classifiers does not help anymore either to stabilize the classifier or to improve its performance. This

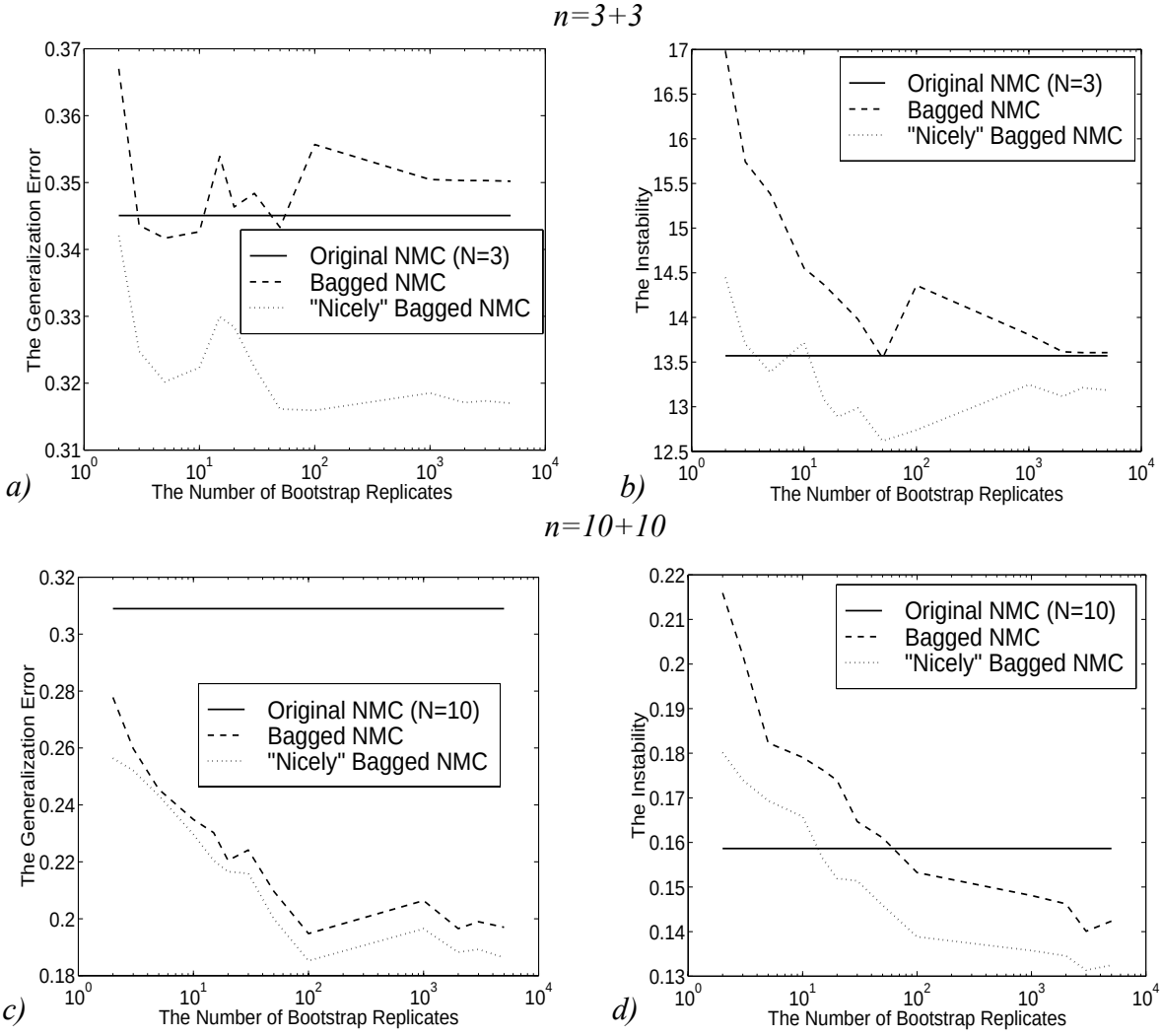


Fig. 5.6. The generalization error and the instability, measured by the training set, of the NMC, the "bagged" and "nicely bagged" NMC versus the numbers of bootstraps B for 30-dimensional Gaussian Correlated data with the training sample size N=3 per class (a,b) and with the training sample size N=10 per class (c,d).

means that *the number of base classifiers used in bagging should be limited*. Applying bagging in practical problems it is sufficient to choose some reasonable value of the number of base classifiers (say, from 10 to 100, depending on the classification rule and the data used). The use of additional bootstrap replicates in bagging will increase computation time and may give just a small improvement in generalization error or may even cause a deterioration in it.

5.5 Instability and Performance of the Bagged Linear Classifiers

Bagging has a different effect on the performance [17] and on the instability of different classifiers. It also may depend on the distribution of the data used. As one can see (Fig. 5.6), bagging performs differently on different sizes of the training data set. Therefore, the efficiency of bagging depends on the training sample size. For instance, Ali and Pazzani [2] have noticed that bagging does not perform well when the training data sample size is very large. Also applying bagging to linear discriminant analysis did not show the usefulness of bagging [17], probably because of a too-large training sample sizes used in the simulation study. Therefore, it seems important to study bagging and its usefulness in relation to the training sample size.

In this section we study bagging and "nice" bagging for the linear classifiers by a set of examples. As the performance and the instability of bagged classifiers are correlated (see previous section), we will also study the stabilizing effect of bagging and "nice" bagging on the classifiers. At first, in Section 5.5.1, we will consider the Nearest Mean Classifier and show that bagging is useful only for critical training sample sizes. We will also demonstrate that the instability of the classifier may help to predict the usefulness of bagging on critical training sample sizes. Then, studying bagging for the Pseudo Fisher Linear Discriminant in Section 5.5.2, we will establish that bagging has a shifting effect on the generalization error with respect to the size of the training set. In Section 5.5.3, we will study bagging for the Regularized Fisher Linear Discriminant and demonstrate that "nice" bagging has a compensating effect, helping to decrease high generalization errors of the RFLD with large values of the regularization parameter. Finally, in Section 5.5.4 we will discuss the usefulness of bagging for the Small Sample Size Classifier and the Karhunen-Loeve Linear Classifier.

5.5.1 Instability Helps to Predict the Usefulness of Bagging

At first, let us consider the effect of bagging on the performance of the Nearest Mean Classifier in the relation with the classifier instability. Considering Fig. 5.7, one can clearly see that bagging is useful only for critical training sample sizes when the classifier is still unstable. Bagging is useless on very small training sets, in spite of the high instability of the classifier. Very small training sets may be very misleading, when representing the real data distribution.

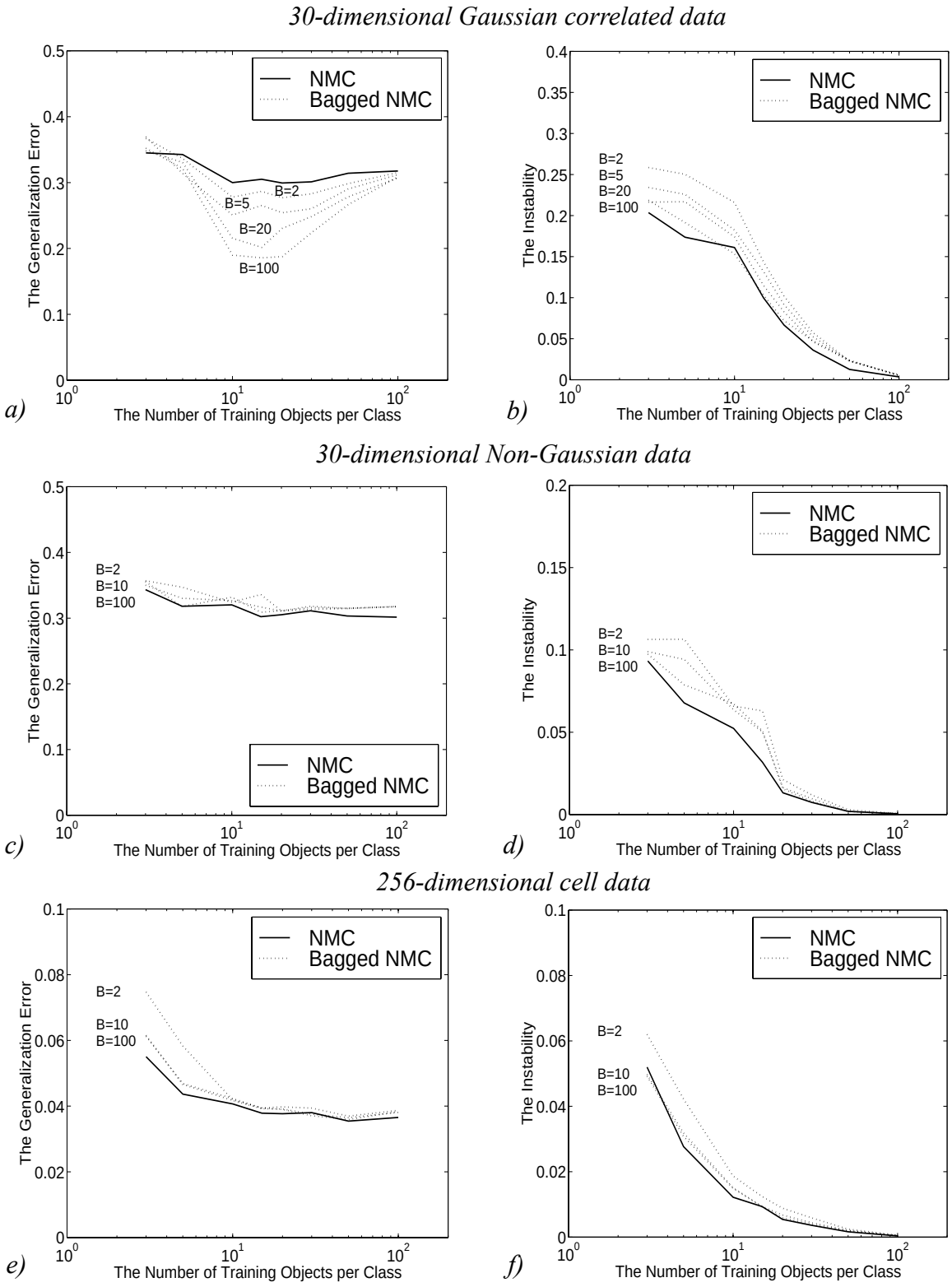
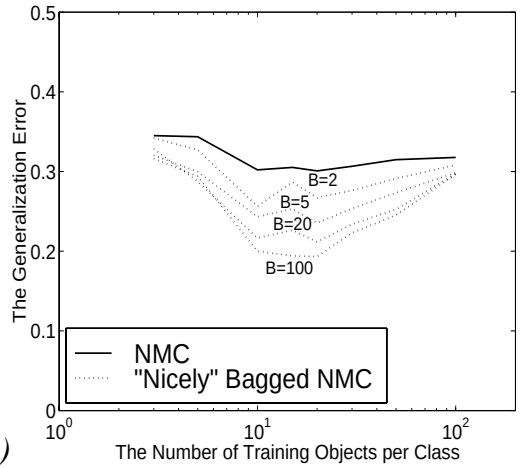
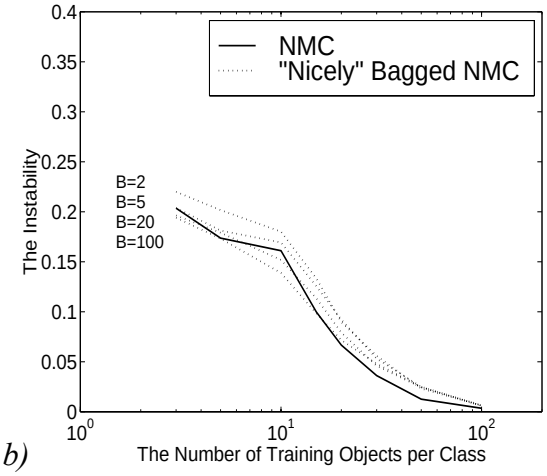


Fig. 5.7. The generalization error and the instability, measured by the training set, of the NMC and the "bagged" NMC for different numbers of bootstraps B versus the training sample size for 30-dimensional Gaussian correlated data (a,b), for 30-dimensional Non-Gaussian data (c,d) and for 256-dimensional cell data (e,f).

30-dimensional Gaussian correlated data

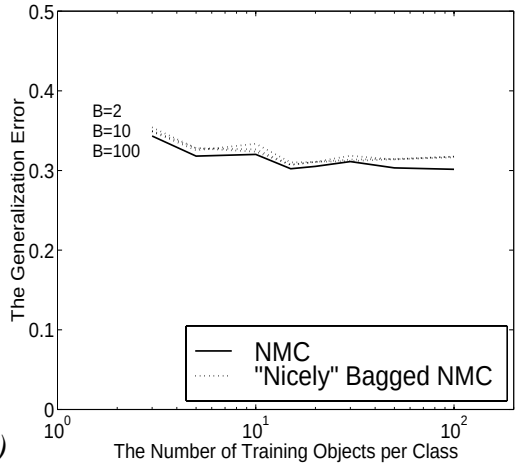


a)

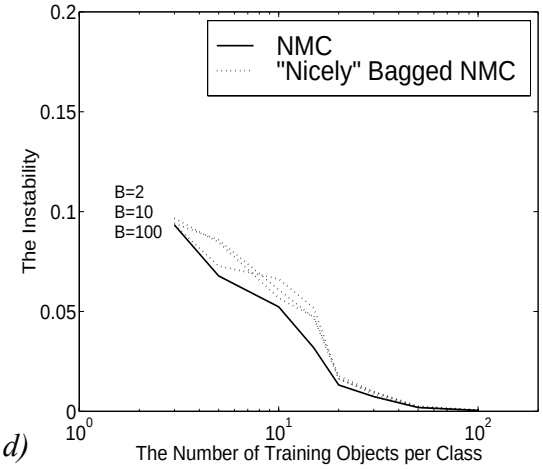


b)

30-dimensional Non-Gaussian data

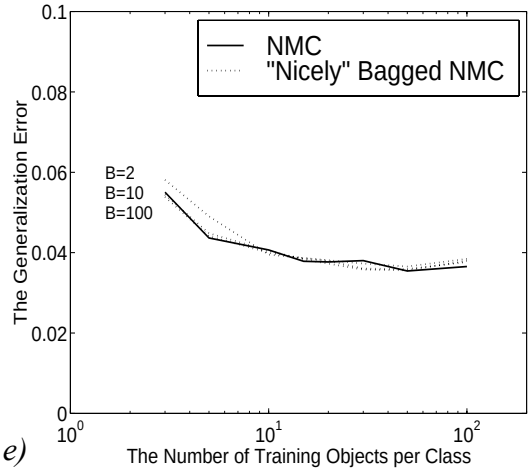


c)

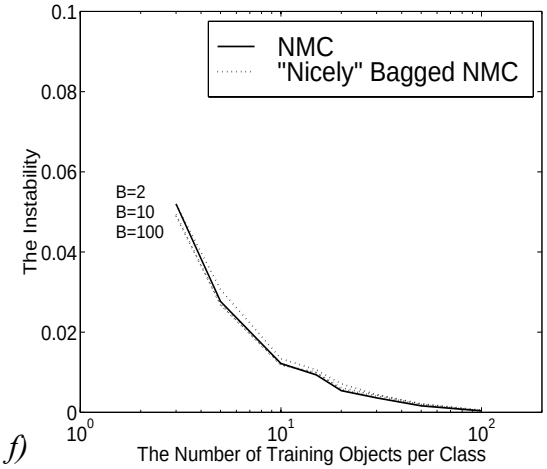


d)

256-dimensional cell data



e)



f)

Fig. 5.8. The generalization error and the instability, measured by the training set, of the NMC and the "nicely bagged" NMC for different numbers of bootstraps B versus the training sample size for 30-dimensional Gaussian correlated data (a,b), for 30-dimensional Non-Gaussian data (c,d) and for 256-dimensional cell data (e,f).

Bootstrapping will not improve their quality. A bootstrap sample of a very bad training set will also be bad. Additionally, bootstrapping a very small training set, one gets many similar bootstrap samples. By this, in bagging, one obtains many similar poorly performing classifiers. Combining such classifiers will not give much profit. For this reason, *bagging on principle cannot be useful for very small training sample sizes*. However, when the training sample size is larger, bootstrapping the training data becomes more effective. One may obtain many different bootstrap samples and, consequently, more diverse classifiers. Combining diverse classifiers may be useful. Therefore, *bagging* turns out to be *beneficial for critical training sample sizes*. However, increasing the training sample size, classifiers become more stable. Large training sets represent the real data distribution well. So, there is no need to improve the training set by bootstrapping. Bootstrapping large training sets is useless. The obtained bootstrap samples will be similar by statistical characteristics. The classifiers constructed on them will not differ much. Combining similar classifiers will not be beneficial. *Bagging is not useful when the training set is large*. Thus, bagging is useful only for critical training sample sizes, when it is possible to obtain diverse classifiers by bootstrapping the training set.

As the instability of classifiers shows the diversity of the classifiers obtained by bootstrapping the training data set, the instability may be used in order to predict the usefulness of bagging. Considering the instability of the NMC for the critical training sample sizes (Fig. 5.7b,d,f) and the performance of bagging for this classifier (Fig. 5.7a,c,e), one can see that bagging is useful only for the Gaussian correlated data set, when the classifier is still rather unstable (say, the instability is larger than 0.05). For the Non-Gaussian data set and the cell data, the NMC is more stable. Bagging appears to be useless for these cases. It even deteriorates the performance of the original NMC.

"Nice" bagging performs on average a bit better than standard bagging (see Fig. 5.8 compared with Fig. 5.7). It is also more stable. This is caused by the fact, that only "nice" base classifiers (that have the same or smaller classification error obtained on the training set than the apparent error of the original classifier), are combined in the final classifier. Similar to standard bagging, "nice" bagging is useful only for critical training sample sizes when the classifier is still rather unstable. Like standard bagging, it improves the performance of the original classifier only for the Gaussian correlated data. It appears to have no benefit in using "nice" bagging for the Non-Gaussian and the cell data.

Notice, that Fig. 5.7b,d,f and Fig. 5.8b,d,f also confirm the fact, that bagging and "nice" bagging are not stabilizing techniques.

5.5.2 Shifting Effect of Bagging

As was mentioned before, when using bootstrap samples of the training set, the number of actually used training objects is reduced (see Section 5.2.4). On average only $1 - \frac{1}{e} \approx 63.2\%$

of the training objects are present in the bootstrap sample. Ignoring the object repetition in a bootstrap sample, the training set becomes smaller. In this way, the "value" of the bootstrap sample is comparable with the "value" of a smaller training set. Therefore, the classifier constructed on a bootstrap sample should be similar to the classifier constructed on a smaller training set. So, the bagged classifier should also have some features of the classifier built on a smaller training set. This phenomenon is most evident for the PFLD due to its characteristic maximum of the generalization error for critical sizes of the training set (see Fig. 5.9). Indeed, considering the effect of bagging on the Pseudo Fisher Linear Discriminant (Fig. 5.9a,c,e), it is clearly seen that *the learning curve* (the dependence of the generalization error as a function on the training sample size) *of the bagged classifier is shifted* with reference to the learning curve of the original classifier in the direction of larger training sample sizes (the bagged classifier performs similar to the original classifier constructed on smaller training sets). Thus, one can expect that bagging improves the performance of the PFLD for training sample sizes that are comparable to data dimensionality.

Taking into account the phenomenon discussed above, we can conclude that when explaining the behaviour of the bagged classifier, it is not enough to study just the instability of the classifier. Also the *peculiarities of the classifier* (the behaviour of its learning curve), that depend on the training sample size, *are important* and should be considered. Studying bagging for the NMC, it was sufficient to consider just its instability, because the NMC is insensitive to the number of training objects. However, in investigating bagging for other linear classifiers, it is necessary to consider both, the instability and the peculiarities of the classifier.

Considering the effect of bagging on the PFLD, we can see that bagging is mostly useful for the critical training sample sizes, when the PFLD performs poorly and is still quite sensitive to the perturbations in the training set. Nevertheless, one could expect to perform bagging better for the training sample sizes a bit larger than $n=p$, where the classifier is still rather unstable. However, the shifting effect of bagging in combination with the instability nicely explains the bad performance of bagging for these training sample sizes. The performance of the classifier constructed on the smaller training sample sizes reflects in the bagged classifier constructed on a larger training sample size. Let us consider the behaviour of the generalization error and the instability of the PFLD (see Fig. 5.9). At first the generalization error of the PFLD may decrease a bit, when the training sample size increases. With the further increase of the training sample size up to the data dimensionality $n=p$, the generalization error increases. The classifier becomes more stable. However, around the point $n=p$, the instability of the PFLD does not change much and may even increase (see Fig. 5.9b,d,f). Bagging is useless for the PFLD constructed on very small training data sets, as bootstrapping such training sets is not advantageous. Usually very small training sets are bad as they are very misleading when representing the real data distribution. By bootstrapping them, one hardly gets better representation of the real data distribution. For critical training

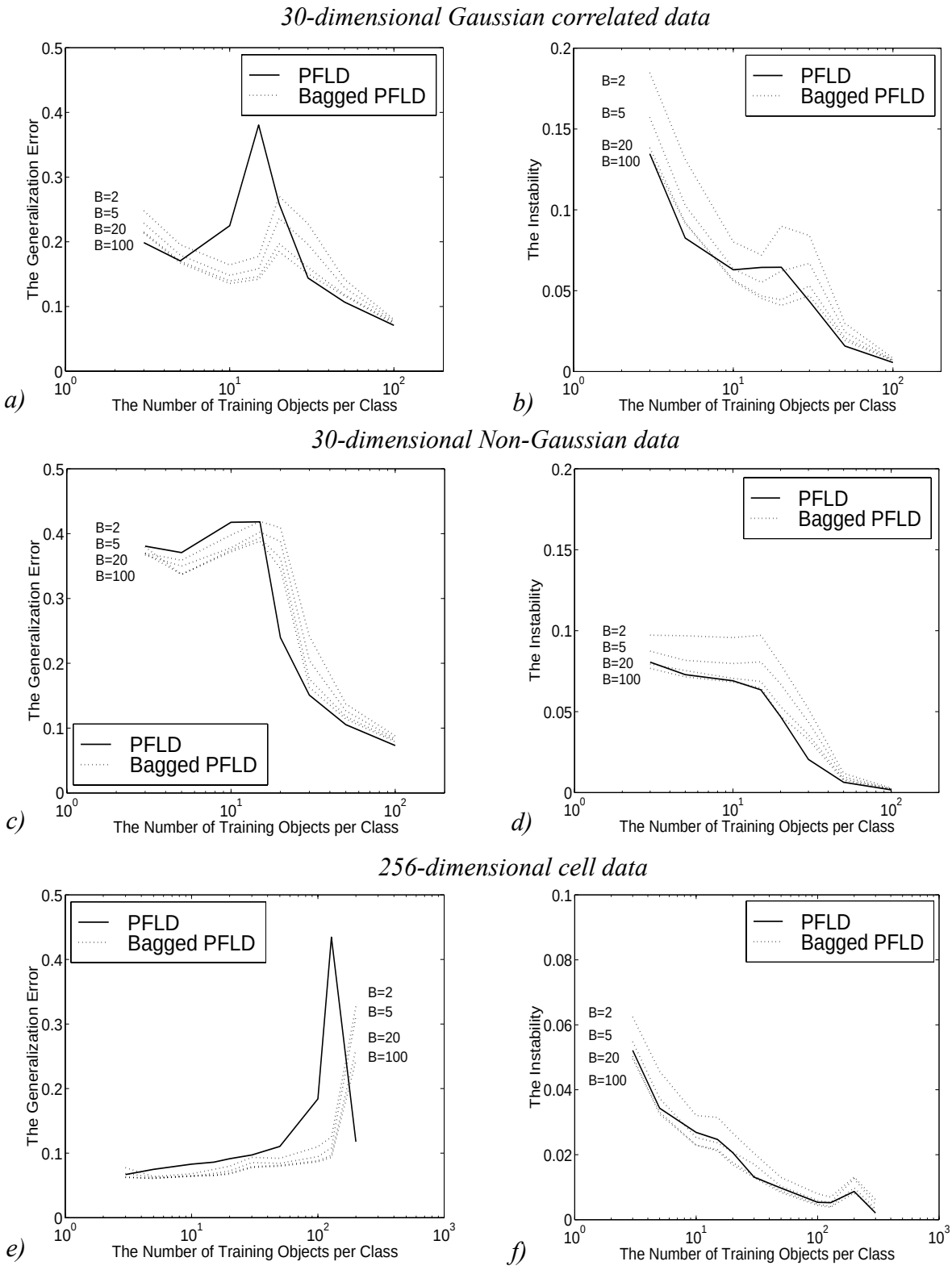


Fig. 5.9. The generalization error and the instability, measured by the training set, of the PFLD and the "bagged" PFLD for different numbers of bootstraps B versus the training sample size for 30-dimensional Gaussian correlated data (a,b), for 30-dimensional Non-Gaussian data (c,d) and for 256-dimensional cell data (e,f).

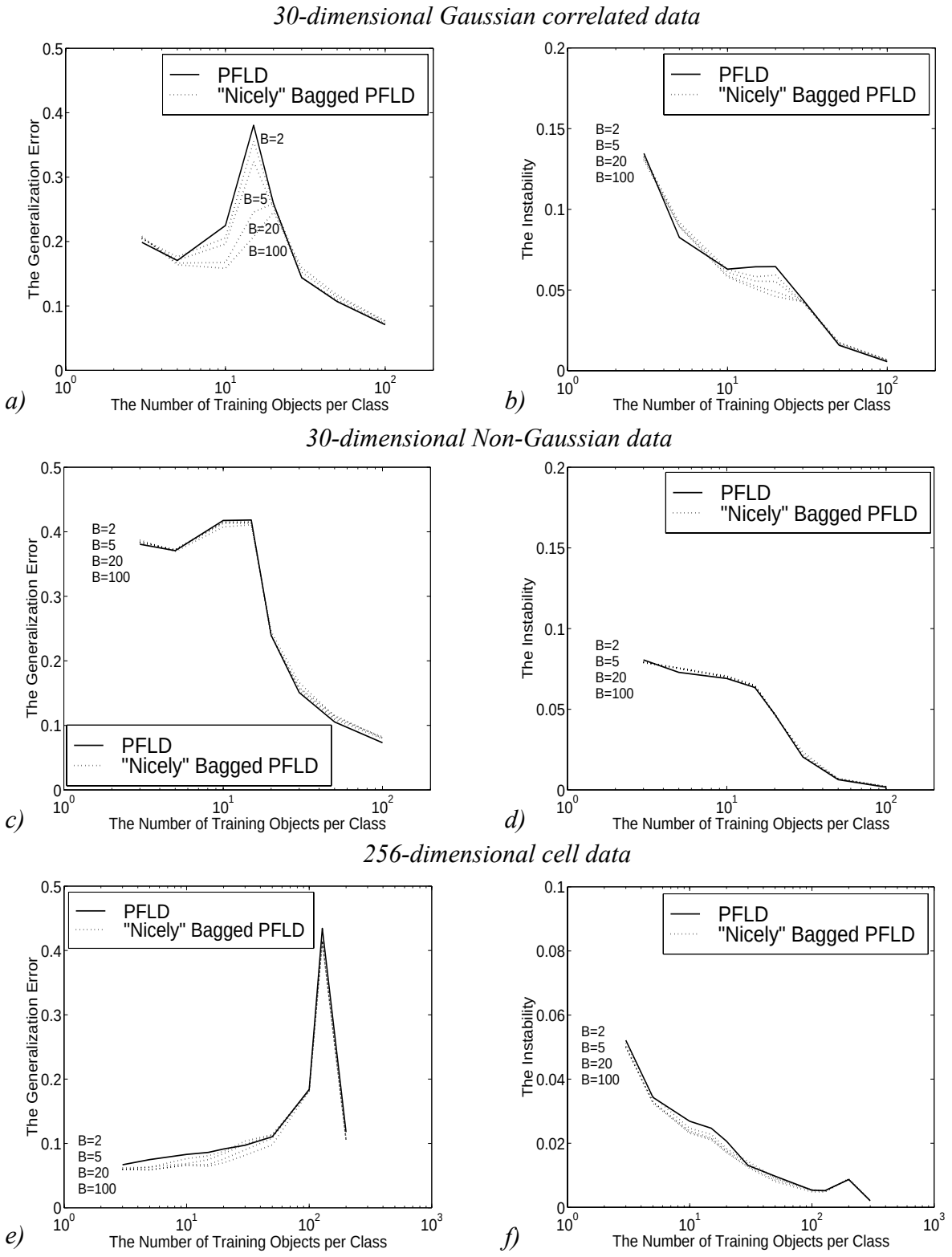


Fig. 5.10. The generalization error and the instability, measured by the training set, of the PFLD and the "nicely bagged" PFLD for different numbers of bootstraps B versus the training sample size for 30-dimensional Gaussian correlated data (a,b), for 30-dimensional Non-Gaussian data (c,d) and for 256-dimensional cell data (e,f).

sample sizes, bagging the PFLD is useful due to both, the high instability of the PFLD and the helpful shifting effect of bagging on the generalization error of the PFLD. However, from the point $n=p$, the generalization error of the PFLD starts to decrease. By that, the shifting effect of bagging becomes harmful. Bagging is less useful or even starts to degrade the performance of the original classifier. By increasing the training sample size further, the PFLD becomes also more stable. In combination with its harmful shifting effect on the generalization error, bagging becomes completely useless. Let us note, that the shifting effect of bagging on the generalization error is very powerful. The PFLD constructed on cell data is rather stable. Nevertheless, bagging is very useful due to its shifting effect on the generalization error (see Fig. 5.9e,f). Therefore, studying the learning curve of the classifier is very important in order to foresee the possible use of bagging.

Let us now consider the usefulness of "nice" bagging. When the classifier is "nicely" bagged, only "nice" bootstrap samples of the training set are used. The classifier built on "nice" bootstrap samples classifies the original training set with the same or smaller generalization error, than the original classifier does. As the generalization error of almost all classifiers decreases when the number of training objects increases up to a certain limit, "nice" bagging perhaps tries to keep only those bootstrap samples that contain the largest possible number of training objects represented in the original training set. A "nice" bootstrap sample of the training set uses more training objects than an ordinary bootstrap replicate. In this way, "nice" bagging has a smaller shifting effect than standard bagging. For this reason, "nice" bagging is less useful for the PFLD than standard bagging (see Fig. 5.10 compared with Fig. 5.9).

5.5.3 Compensating Effect of Bagging for the Regularized Fisher Linear Classifier

In this section the usefulness of bagging and "nice" bagging is studied for the Regularized Fisher Linear Discriminant with different values of the regularization parameter. It will be shown that for large training sample sizes "nice" bagging can correct an increase in the generalization error of the RFLD (with reference to the generalization error of the FLD) caused by using values for the regularization parameter in the RFLD that are too large.

The RFLD (see Chapter 1) represents a large family of linear classifiers (see Fig. 5.5), from the standard Fisher Linear Discriminant (when the regularization parameter λ is equal to zero) and the Pseudo Fisher Linear Discriminant (when the regularization parameter λ is very small) to the Nearest Mean Classifier (when the regularization parameter λ approaches infinity). Bagging has a different effect on each of these regularized classifiers. The generalization error of the bagged and "nicely" bagged RFLD with different values of the regularization parameter λ is presented in Fig. 5.11a-j for the 30-dimensional Gaussian

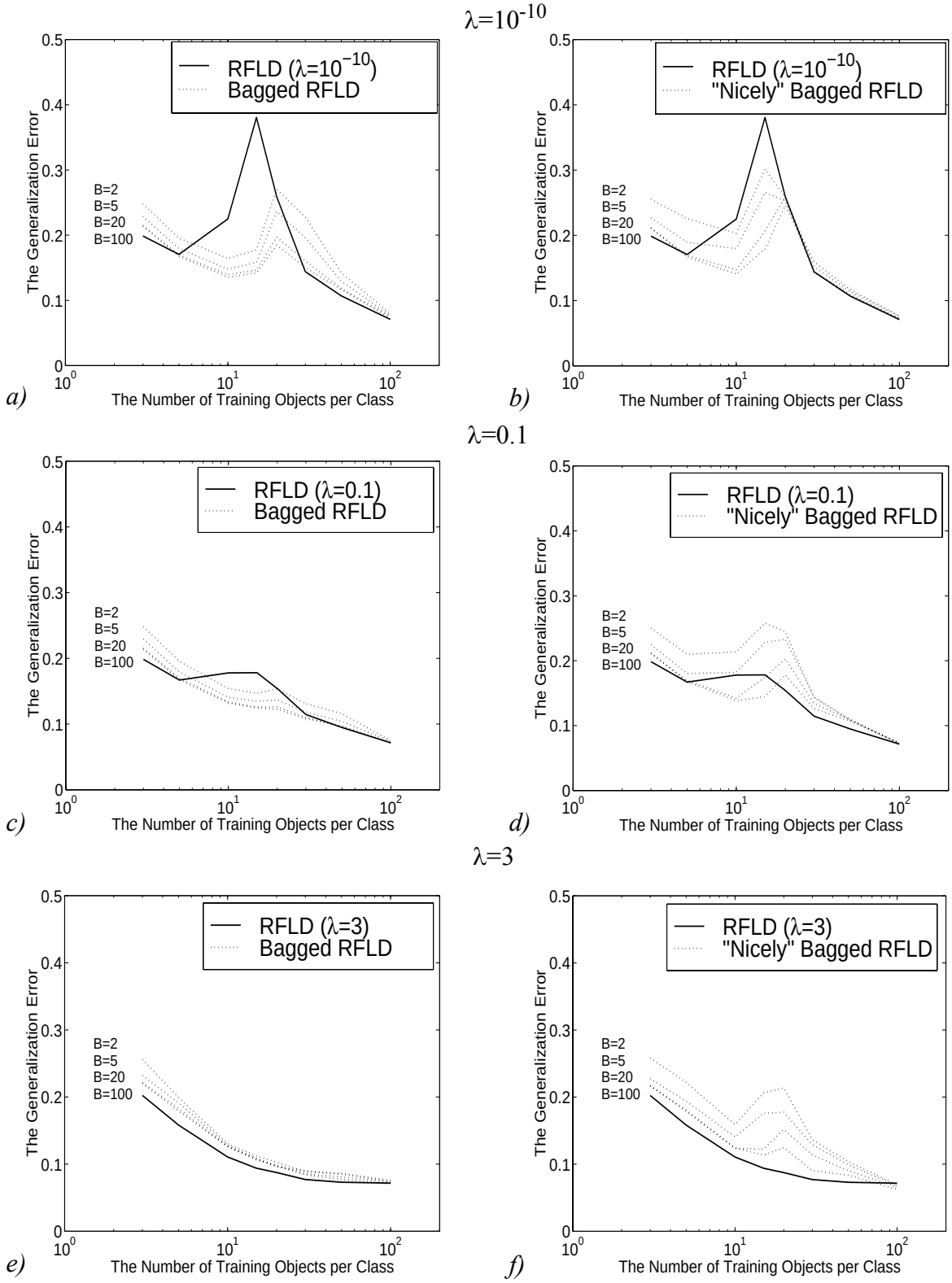


Fig. 5.11. The generalization error of the "bagged" and "nicely bagged" RFLD for different numbers of bootstraps B versus the training sample size for 30-dimensional Gaussian correlated data.

correlated data set. We considered only this case, because from our point-of-view this is the most interesting example, which demonstrates the relation between the stability of the classifier and the use of bagging for this classifier. The usefulness of bagging and "nice" bagging for the RFLD can be nicely explained by considering the instability of the RFLD (see Fig. 5.5c,d) and the shifting effect of bagging as discussed in the previous section.

The RFLD with the regularization parameter $\lambda = 10^{-10}$ is similar to the PFLD. Thus bagging and "nice" bagging behave in the same way as they do for the PFLD (Fig. 5.11a,b). The RFLD with the regularization parameter λ=0.1 is more stable than the RFLD with $\lambda = 10^{-10}$. However, due to the beneficial shifting effect of bootstrapping, bagging and "nice" bagging are still useful for critical sizes of the training set (Fig. 5.11c,d). As this RFLD is still close to the PFLD, bagging gives better results than "nice" bagging. The RFLD with λ=3 is

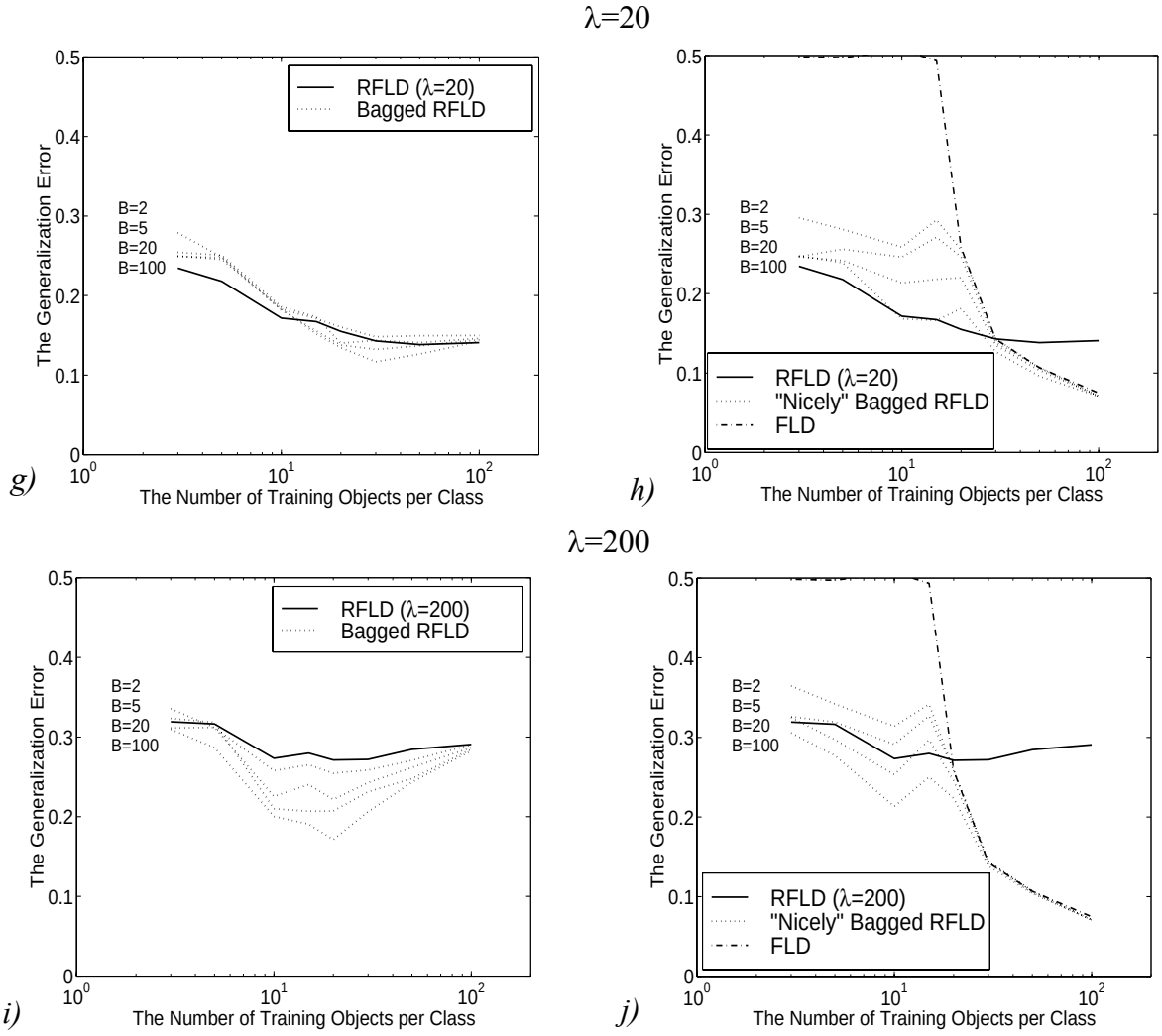


Fig. 5.11. The generalization error of the "bagged" and "nicely bagged" RFLD for different numbers of bootstraps B versus the training sample size for 30-dimensional Gaussian correlated data.

very stable. Both bagging and "nice" bagging are useless (Fig. 5.11e,f). When the regularization parameter increases over the value λ=3 to infinity, the RFLD approaches the Nearest Mean Classifier and becomes more unstable. Bagging again becomes useful. Bagging decreases the generalization error of the RFLD for λ=20 in case of critical and large training sample sizes (Fig. 5.11g,h). However, bagging is not very useful here, as the shifting effect of bagging is unfavorable for this classifier. The RFLD with λ=200 is rather close to the NMC. Bagging performs in the same way as for the NMC (Fig. 5.11i,j).

Surprisingly, "nice" bagging performs very well for the RFLD with large values of the regularization parameter in case of large training sample sizes. It decreases the generalization error of the RFLD by a factor two or more. This phenomenon can be explained as follows. The RFLD with large values of the regularization parameter approaches the NMC. However, the RFLD is still not the same classifier. It constructs the discriminant function in a different way than the NMC. On the other hand, "nice" bagging constructs the classifier by averaging the parameters (coefficients) of classifiers built on "nice" bootstrap replicates of the training set. "Nice" bootstrap versions of the classifier are chosen in such a way that their classification error on the original training set is equal to or smaller than the apparent error of the original classifier. Therefore, the original training set is used as the validation data set, in order to select "nice" bootstrap versions. However, classifiers selected to be "nice" on the training set are not necessarily nice for a test data set. When the number of objects in the training set is small, relatively bad classifiers might be selected to be "nice" bootstrap versions of the classifier. The "nicely" bagged classifier, constructed from these nonideal classifiers, will be bad as well. When the number of objects increases in the training set, the training set becomes more representative and the true error approaches the ideal classification error. The "nicely" bagged classifier approaches the ideal classifier with the Bayes classification error. This way, "nice" bagging constructs nice classifiers for large training sample sizes.

When, for Gaussian correlated data, the FLD is stabilized by regularization with large values of the regularization parameter, the generalization error of the FLD decreases for small training sample sizes and increases for large training sample sizes. As "nice" bagging builds an ideal classifier for large training sample sizes, *"nice" bagging neutralizes the negative effect of too large regularization* and decreases the generalization error of the regularized classifier up to the generalization error of the FLD and even slightly improves it.

5.5.4 Bagging for the Small Sample Size Classifier and for the Karhunen-Loeve's Linear Classifier

In studying bagging and "nice" bagging for the Small Sample Size Classifier (see Fig. 5.12 and Fig. 5.13) and for the Karhunen-Loeve's Linear Classifier (see Fig. 5.14), we do not consider 256-dimensional cell data, because bagging on these data leads to large computational

burdens. However, as the SSSC and the KLLC seem to be rather stable for this data set (Fig. 5.4f), it is reasonable to conclude that bagging is probably useless for these classifiers. We also skip figures demonstrating the behaviour of "nice" bagging for the KLLC, as it performs very similar to standard bagging.

Let us consider now the bagging experiments on the SSSC. For the Gaussian correlated data (Fig. 5.12a,b) the SSSC is more stable than the NMC or the PFLD. However, bagging is still useful for critical training sample sizes. Bagging becomes less useful for larger training sample sizes due to the unfavorable shifting effect of bagging. For the Non-Gaussian data (Fig. 5.12c,d), the SSSC is still unstable for critical training sample sizes. The shifting effect of bagging is beneficial for small and critical training sample sizes up to the training sample size

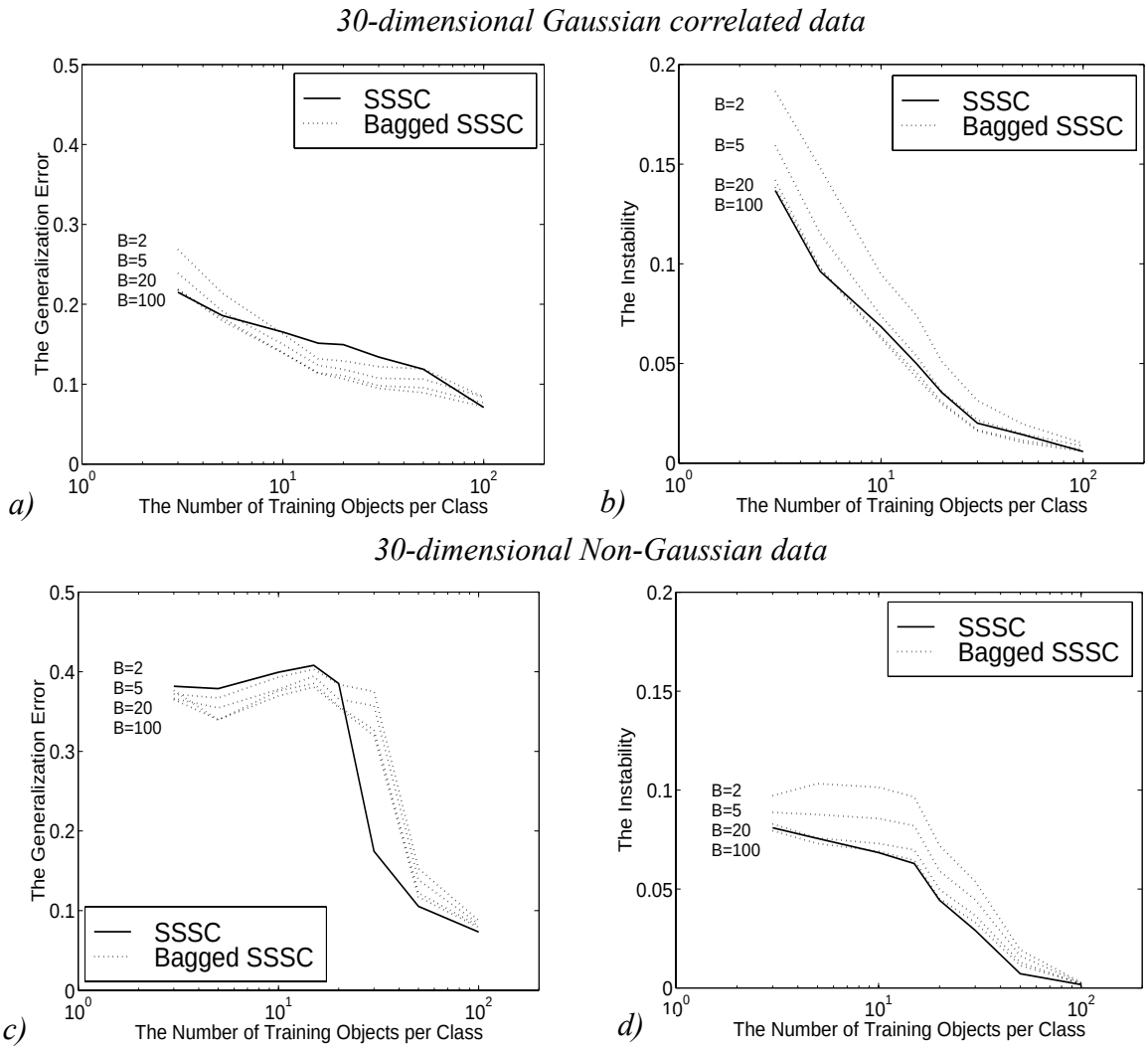


Fig. 5.12. The generalization error and the instability, measured by the training set, of the SSSC and the "bagged" SSSC for different numbers of bootstraps B versus the training sample size for 30-dimensional Gaussian correlated data (a,b) and for 30-dimensional Non-Gaussian data (c,d).

equal to the data dimensionality. Therefore, bagging is useful for these data sizes. For the training sample sizes larger than the data dimensionality, the generalization error of the SSSC decreases. By that, bagging has a harmful shifting effect on the generalization error and becomes useless.

As "nice" bagging builds nonideal classifiers on small training sets and its shifting effect is smaller than the shifting effect of ordinary bagging, "nice" bagging is less useful for the Gaussian correlated data (Fig. 5.13a,b) and useless for the Non-Gaussian data (Fig. 5.13c,d).

Let us consider now the usefulness of bagging for the KLLC. For Gaussian correlated data, the KLLC constructed on critical training sample sizes is rather stable (Fig. 5.14a,b). Additionally, bagging has an unfavorable shifting effect on the generalization error, as the

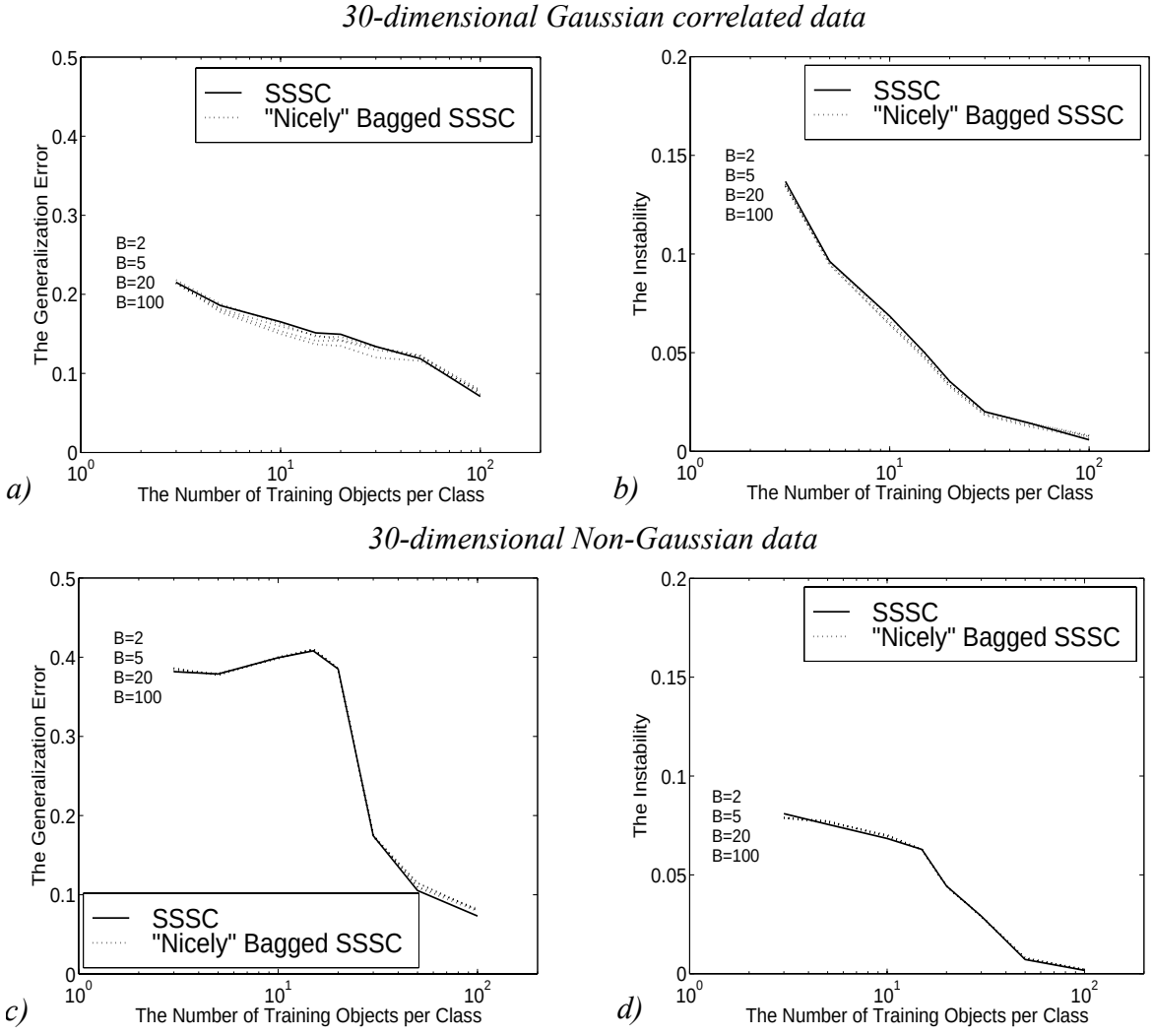


Fig. 5.13. The generalization error and the instability, measured by the training set, of the SSSC and the "nicely bagged" SSSC for different numbers of bootstraps B versus the training sample size for 30-dimensional Gaussian correlated data (a,b) and for 30-dimensional Non-Gaussian data (c,d).

generalization error of the KLLC decreases with an increase in the training sample size. Therefore, bagging is completely useless and even degrades the performance of the original classifier. For the Non-Gaussian data (Fig. 5.14c,d) the KLLC is more unstable for critical sizes of the training set. The generalization error of the KLLC does not change much when the training sample size increases. Therefore bagging is useful for critical training sample sizes. However, the effect of bagging is not large for this data set, because the KLLC constructs a discriminant function in the subspace of principal components defined by some of the largest eigenvalues of the data distribution. In this way the KLLC ignores some informative features with small variances. Changes in the composition of the training set do not help much to build a better classifier. Bagging cannot be very useful here.

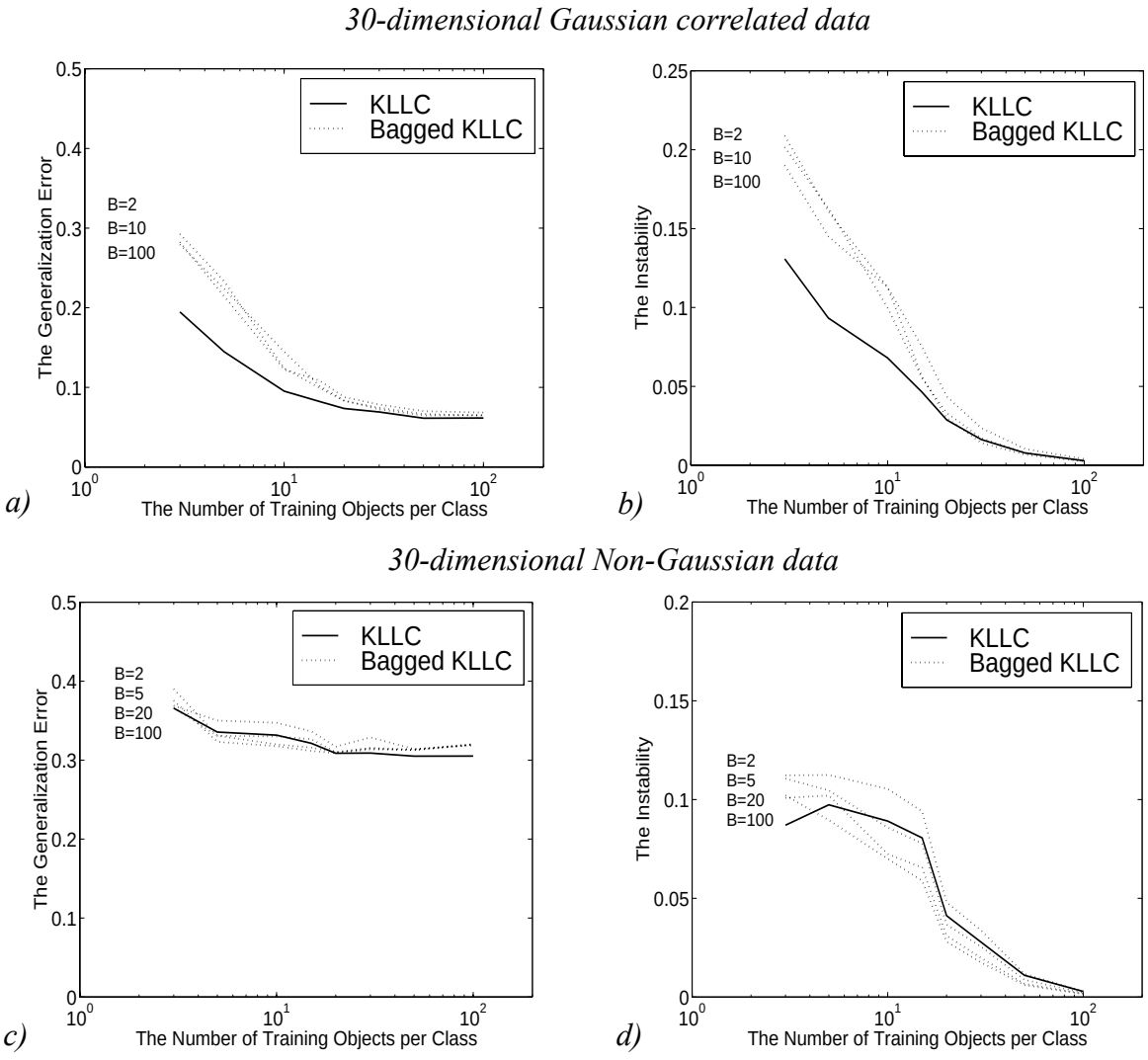


Fig. 5.14. The generalization error and the instability, measured by the training set, of the KLLC and the "bagged" KLLC for different numbers of bootstraps B versus the training sample size for 30-dimensional Gaussian correlated data (a,b) and for 30-dimensional Non-Gaussian data (c,d).

5.6 Conclusions

The bagging technique combines benefits of bootstrapping and aggregating. The bootstrap method, its implementation on the computer, its usefulness to get more accurate statistics and its application to some real data analysis problems are nicely described by Efron and Tibshirani [126]. Bagging was developed by Breiman [15] and investigated by a number of researchers. However, bagging was mainly studied for regression problems and much less in discriminant analysis (classification trees, the nearest neighbour classifier and a linear classifier were considered). Breiman has found that for unstable procedures bagging works well. However, his simulation study performed in linear discriminant analysis has shown a nonuse of bagging. This is probably caused by the large training sample sizes he used. We continued the investigation of bagging [117]. In the current chapter, which is based on our paper [117], bagging for linear classifiers, such as the Nearest Mean Classifier, the Pseudo Fisher Linear Discriminant, the Regularized Fisher Linear Discriminant, the Small Sample Size Classifier and the Karhunen-Loeve's Linear Classifier, has been studied. We also investigated the relation between the effect of bagging on the performance and on the instability of classifiers.

For our investigations a measure of stability of the classifier, called the instability, has been suggested (see Chapter 1). We have demonstrated that the performance and the instability of the classifier are related: the more stable the classifier, the smaller its generalization error. This is observed for the RFLD with different values of the regularization parameter (Fig. 5.5).

We proposed using the instability of the classifier measured by the training set to predict the possible efficacy of bagging. In bagging, one combines classifiers constructed on the bootstrap replicates of the training set into the final decision rule. The quality of the final classifier strongly depends on the diversity of aggregated classifiers. The instability of the classifier, measured by the training set, illuminates this diversity. By this, the instability and the usefulness of bagging should be related.

We have shown that *bagging is not a stabilizing technique* nor does it always improve the performance of the original classifier. The bagged classifier can be more as well as less stable than the original classifier. It depends on the number of bootstrap replicates used to construct the bagged classifier. However, the number of bootstrap replicates used has not to be very large. Increasing the number of bootstrap replicates in bagging above a certain value does not decrease the generalization error and the instability of the classifier anymore. Therefore, when one applies bagging to linear classifiers, it is reasonable to use about 20-100 bootstrap replicates (depending on the classifier and the data).

We have shown by some examples that *bagging may improve the performance of linear classifiers constructed on critical training sample sizes*. However, *the classifier should be rather unstable*. When the classifier is stable, bagging is usually useless. Bagging is worthless

for very small and very large training sample sizes. In the first case, the training set is usually bad giving a wrong representation of the real data distribution. Bootstrapping a small bad data set hardly ever gives a better representation of the real data distribution. Moreover, one cannot get many different bootstrap replicates on very small training sets. As a result, on such bootstrap replicates, one obtains similar and badly performing classifiers. Combining them is not beneficial. In the second case, bootstrap replicates of very large training sets have similar statistical characteristics, because very large training sets represent the real data distribution well and perturbations in their composition barely affect these characteristics. Classifiers constructed on such bootstrap replicates are also similar. Combining similar classifiers is not useful.

Simulation studies revealed that bagging has a shifting effect on the generalization error with respect to the training sample size. Due to the fact that the number of actually used training objects is reduced when one uses bootstrap samples of the training set, the performance of the bagged classifier is close to the performance of the classifier built on a smaller training set. So, the learning curve of the bagged classifier is shifted with reference to the learning curve of the original classifier in the direction of larger training sample sizes. Therefore, *in order to predict the usefulness of bagging for the classifier, it is necessary to take into consideration both the instability and the peculiarities of the classifier*, which depend on the training sample size.

As different linear classifiers have a different stability and behave differently on different kinds of data and for different training sample sizes, bagging does not work in the same way in different situations. The review of the use of bagging for linear classifiers in different situations was given in Section 5.5.

A modification of bagging called “nice” bagging has been studied. It is shown that the “nicely” bagged classifier is more stable than the bagged classifier. The shifting effect of “nice” bagging on the generalization error is also smaller. Sometimes “nice” bagging could be preferable to “ordinary” bagging. However, in general “nice” bagging does not give much better results than bagging. A better possibility than “nice” bagging to combine classifiers is to weight decisions of base classifiers in ensemble accordingly to their performance on the training data set. This possibility is considered by us in Chapter 6.

Chapter 6

Boosting for Linear Classifiers

6.1 Introduction

When data are highly dimensional, having small training sample sizes compared to the data dimensionality, it may be difficult to construct a good single classification rule. Usually, a classifier, constructed on small training sets is biased and has a large variance. Consequently, such a classifier may have a poor performance [54]. In order to improve a weak classifier by stabilizing its decision, a number of techniques could be used, for instance, regularization [30] or noise injection [4].

Another approach is to construct many weak classifiers instead of a single one and then combine them in some way into a powerful decision rule. Recently a number of such combining techniques have been developed. The most popular ones are bagging [15], boosting [32] and the random subspace method [47]. In bagging (see Chapter 5), one samples the training set, generating random independent bootstrap replicates [26], constructs the classifier on each of these and aggregates them by simple majority voting in the final decision rule. In boosting, classifiers are constructed on weighted versions of the training set, which are dependent on previous classification results. Initially, all objects have equal weights, and the first classifier is constructed on this data set. Then weights are changed according to the performance of the classifier. Erroneously classified objects get larger weights and the next classifier is "boosted" on the reweighted training set. In this way a sequence of training sets and classifiers is obtained, which are then combined by simple or weighted majority voting in the final decision. In the random subspace method (see Chapter 7) classifiers are constructed in random subspaces of the data feature space. Then, only classifiers with zero apparent error (the classification error on the training data set) are combined by simple majority voting in the final decision rule.

Bagging, boosting and the random subspace method have been designed for decision trees [14], where they often produce an ensemble of classifiers, which is superior to a single classification rule [107, 17, 31, 21, 47]. However, these techniques may also perform well for other classification rules than decision trees. For instance, it was shown that bagging and boosting may be useful for perceptrons (see, e.g. [110, 6]). In Chapter 5 we demonstrated that bagging may be beneficial in *Linear Discriminant Analysis* (LDA) for *critical training sample sizes* (when the number of training objects is comparable with data the dimensionality). In this chapter we intend to study the usefulness of boosting for linear classifiers and compare it with the usefulness of bagging. If applied to decision trees, it is difficult to imply other combining

rules than simple or weighted majority voting. Linear classifiers allow us to use other combining rules (see Section 5.2.2 in Chapter 5) such as the average rule, the mean rule and the product rule. It may happen that these combining rules perform better than voting, especially for bagging. However, the average rule (see Section 5.2.2 in Chapter 5) has an advantage to voting and other combining rules. When using voting or the combining rules based on the posteriori probabilities, one must store each combined classifier and their decisions (class labels or the posteriori probabilities) in order to compute the final decision. When aggregating classifiers by the average rule, it is sufficient to store only the coefficients of the combined classifiers in order to get the final decision. At the end one obtains just a single classifier having the same number of parameters (coefficients) as each of the combined classifiers.

In this chapter, which is based on our works [121, 122], we study the usefulness of boosting for a large group of linear classifiers and investigate its relation with the instability of classifiers and with the training sample size. The nearest mean classifier [35], the Fisher linear discriminant function [35] and the regularized Fisher linear discriminant function [30] are used in our study as a base classifier. This choice is made in order to observe many different classifiers with dissimilar instability and, by that, to establish whether the usefulness of boosting depends on the instability of the base classifier or on other peculiarities of the classifier. The chosen classification rules, their instability and the performance of boosting and bagging for these classifiers in relation with the training sample size are discussed in Section 6.3. First, a description of the boosting algorithm and some discussion on boosting are given in Section 6.2. The role of the combining rules for boosting and bagging on the example of the nearest mean classifier is studied in Section 6.4. Conclusions are summarized in Section 6.5.

6.2 Boosting

Boosting, proposed by Freund and Schapire [32], is another technique to combine weak classifiers having a poor performance in order to get a classification rule with a better performance. In boosting, classifiers and training sets are obtained in a strictly *deterministic* way. Both, training data sets and classifiers are obtained *sequentially* in the algorithm in contrast to bagging (see Chapter 5), where training sets and classifiers are obtained *randomly* and *independently* (in parallel) from the previous step of the algorithm. At each step of boosting, training data are reweighted in such a way that incorrectly classified objects get larger weights in a new, modified training set. By that, one actually maximizes margins between training objects. It suggests the connection between boosting and Vapnik's Support Vector Classifier (SVC) [107, 19], as the objects obtaining large weights may be the same as the support objects.

Boosting is organized by us in the following way. It is based on the "arc-fs" algorithm

described by Breiman [17], where we reweight the training set instead of resample it. The "arc-fs" algorithm is the improved version of the standard AdaBoost algorithm [32]. Additionally, we set initial weight values w_i^1, $i = 1, \ldots, n$, to 1 instead of $1/n$, in order to be independent of data normalization. Therefore, boosting is implemented by us as follows.

1. Repeat for $b = 1, \ldots, B$.
   a) Construct a base classifier $C_b(\boldsymbol{x})$ (with a decision boundary $C_b(\boldsymbol{x})=0$) on the weighted version $\boldsymbol{X}^* = (w_1^b \boldsymbol{X}_1, w_2^b \boldsymbol{X}_2, \ldots, w_n^b \boldsymbol{X}_n)$ of training data set $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$, using weights w_i^b, $i = 1, \ldots, n$ (all $w_i^b = 1$ for b=1).
   b) Compute probability estimates of the error $err_b = \frac{1}{n}\sum_{i=1}^{n} w_i^b \xi_i^b$ and combining weights $c_b = \frac{1}{2}\log\left(\frac{1-err_b}{err_b}\right)$, where $\xi_i^b = \begin{cases} 0, \textit{if } \boldsymbol{X}_i \textit{ is classified correctly} \\ 1, \textit{otherwise} \end{cases}$.
   c) If $0 < err_b < 0{,}5$, set $w_i^{b+1} = w_i^b \exp(-c_b \xi_i^b)$, $i = 1, \ldots, n$, and renormalize so that $\sum_{i=1}^{n} w_i^{b+1} = n$. Otherwise, set all weights $w_i^b = 1$, $i = 1, \ldots, n$, and restart the algorithm.
2. Combine base classifiers $C_b(\boldsymbol{x})$, $b = 1, 2, \ldots, B$, by the weighted majority vote with weights c_b to a final decision rule $\beta(\boldsymbol{x}) = \underset{y \in \{-1, 1\}}{\textit{argmax}} \sum_b c_b \delta_{\mathrm{sgn}(C_b(\boldsymbol{x})), y}$, where $\delta_{i,j}$ is Kronecker symbol and $y \in \{-1, 1\}$ is a possible decision (class label) of the classifier.

Boosting has been studied theoretically and experimentally by many researchers [6, 9, 17, 21, 22, 31, 32, 34, 47, 56, 79, 109] and mainly the weighted majority voting rule was used to combine classifiers. Schapire and Singer [108] have shown that the training error $\hat{P}_A$ of the final decision rule $\beta(\boldsymbol{x})$ obtained by combining weak learner $C_b(\boldsymbol{x})$ by the weighted majority voting is bounded. If each weak learner $C_b(\boldsymbol{x})$ is slightly better than random, then the training error drops exponentially fast. By that, boosting can efficientely convert a weak learning algorithm to a strong learning algorithm. Freund and Schapire [34] have shown that the generalization error $\hat{P}_N$ of the final rule $\beta(\boldsymbol{x})$ has also an upper bound which depends on its training error $\hat{P}_A$, on the training sample size N, on the Vapnik-Chervonenkis (VC) dimensionality d [129] of the parameter space of the weak learner and on the number of iterations B (the number of combined classifiers) used in boosting

$$\hat{P}_N \le \hat{P}_A + \tilde{O}\left(\sqrt{\frac{Bd}{N}}\right) . \tag{6.1}$$

One can see that the generalization error $\hat{P}_N$ will increase as the number B of classifiers combined in boosting increases. This means that the boosted classifier $\beta(\boldsymbol{x})$ will overfit (will be overtrained).

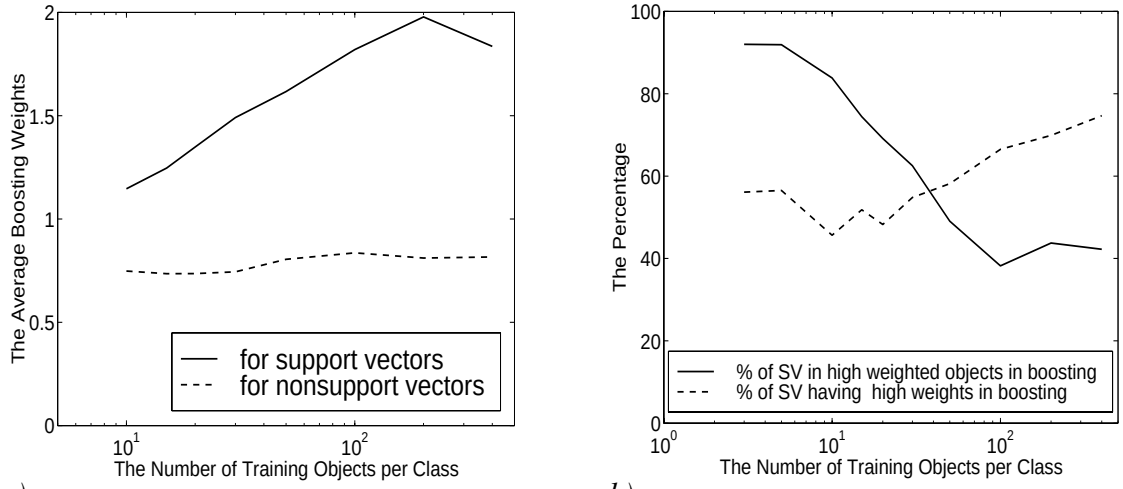
Schapire et al. [107] have also given an alternative analysis in terms of the margins of the training examples. They have shown that the generalization error $\hat{P}_N$ of the final rule $\beta(\boldsymbol{x})$ is bounded as

$$\hat{P}_N \le \hat{\Pr}(margin(\boldsymbol{x}) \le \theta)) + \tilde{O}\left(\sqrt{\frac{d}{N\theta^2}}\right) \tag{6.2}$$

for any $\theta > 0$ with high probability and with the margin of the object $\boldsymbol{x}$ (the distance of the object $\boldsymbol{x}$ to the final classifier $\beta(\boldsymbol{x})$) defined for a 2-class problem as

$$margin(\boldsymbol{x}) = \frac{\left|\sum_{b=1}^{B} c_b C_b(\boldsymbol{x})\right|}{\xi \sum_{b=1}^{B} |c_b|}, \qquad \xi = \begin{cases} 1, & if\ \boldsymbol{x}\ is\ classified\ correctly\ by\ \beta(\boldsymbol{x}) \\ -1, & if\ \boldsymbol{x}\ is\ classified\ erroreously\ by\ \beta(\boldsymbol{x}) \end{cases}.$$

Schapire et al. came to the conclusion that the larger the margins the lower the generalization error. Consequently, they suggested that the success of boosting is defined by its ability to produce large margins. In addition, it was proved that boosting is particularly aggressive at concentrating on the objects having small margins. By that, boosting has much in common with the support vector machines [19], as they aim to maximize the minimum margin. Although boosting and the SVC have the same goal, computationally they perform in a different way. In the SVC one performs the global optimization in order to maximize the minimal margin, while in boosting one maximizes the margin locally for each training object. In order to illustrate the similarity and the difference between boosting and the SVC, let us consider the example of the 30-dimensional Gaussian correlated data set described in Chapter 1. In Fig. 6.1a one can see that, on average, support vectors, found by the SVC, get larger



a) *b)*

Fig. 6.1. a) The average weights, obtained in boosting the NMC after 250 steps, for non-support and support vectors, found by the linear SVC, versus the training sample size;
b) The percentage of support vectors (SV) in training objects, that obtain high weights in boosting the NMC, and the percentage of support vectors having high weights in boosting versus the training sample size.

weights in boosting than non-support vectors. However, objects getting large weights in boosting are not always the same as the support vectors (see Fig. 6.1b). When the training sample size is small, almost all training objects are found to be the support vectors, while only part of training objects get large weights in boosting. When the training sample size increases, the set of objects having large weights in boosting becomes larger as well. It also contains more support vectors (but not all of them) found by the SVC. On the other hand, the number of support vectors found is smaller than the total number of training objects obtaining large weights in boosting. It means, that objects with large weights in boosting are not identical to the support vectors found by the SVC.

Analyzing formulas (6.1) and (6.2), we can see that boosting is the most effective for large training sample sizes N and for small VC-dimensionality d of the classification rules to be combined (e.g. for simple classification rules with less parameters). Breiman [16, 18] doubted that Schapire's et al. explanation of the success of boosting is complete. He generalized the margin function by another function applicable to multiclass problems, that he calls the edge. By studying the edge and the margin functions for different boosting modifications, he has found that the bound (6.2) obtained by Schapire et al. is not supported by empirical results, i.e. sometimes boosting algorithms having larger margins (while keeping the VC-dimension fixed) may have a worse performance. Additionally, formula (6.2) does not explain the effect of overfitting in boosting. However, in spite of the fact that formula (6.2) is not very practical, it may be still helpful in understanding the performance of boosting.

Boosting was also studied from the bias-variance point of view [17, 9, 31]. It was shown that boosting reduces both the variance and the bias of the classifier, while bagging is the most aggressive in reducing the variance. In [31] connections were made between boosting, logistic regression and additive models. Namely, it was shown that boosting can be viewed as a method for fitting an additive model optimizing a criterion similar to binomial log likelihood. It has also been suggested to speed up boosting by weight trimming, in which one deletes the objects having very low weights from the training set separately at each step of boosting.

Another direction in studying boosting has been the game theory [33, 18]. It has been shown that boosting can be viewed as an algorithm for playing repeated games and for approximately solving a game. This also shows that boosting is related to linear programming.

Many modifications of boosting have been developed [16, 31, 17, 56]. One of them is *arcing* (**a**daptive **r**esampling and **c**ombin**ing**) [17], in which one resamples the training data set $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$ at each iteration using updated probabilities w_i^b ($i = 1, \ldots, n$) instead of reweighting the training set with weights w_i^b. By this, arcing becomes a bit similar to bagging. However, it is fundamentally different from bagging in the way the base classifiers are obtained. Due to adaptive resampling instead of adaptive reweighting, arcing is more widely applicable than boosting. Nevertheless, the performed simulation studies [17, 9, 110] did not show a better performance for arcing when compared to boosting. The modification of

boosting proposed in [56] combines base classifiers consisting of randomly selected hyperplanes and using a different method for adaptive resampling and unweighted voting.

Boosting has been investigated experimentally and compared to bagging and other combining techniques by many researchers [6, 9, 17, 21, 22, 31, 32, 47, 79]. It has been shown that boosting is beneficial for regression and classification trees [9, 17, 21, 22, 31, 32, 47, 79] and perceptrons [110, 6] often giving a better performance than bagging. In [17] it was stated that bagging and boosting are useful for unstable classifiers. It has been demonstrated [17, 9] on several examples that both, bagging and boosting, reduce the variance of the classifier. It was shown [107], however, that a large variance of the classifier is not a requirement for boosting to be effective. Also some simulation studies were performed on the usefulness of bagging and boosting in LDA [17]. Just one linear classifier (probably, the Fisher linear discriminant) was considered, and the relation between the training sample size and the data dimensionality was not taken into account. From this study the conclusion was made that neither bagging nor boosting is beneficial for linear classifiers because in general linear classifiers are stable. Chapter 5 contradicts this conclusion demonstrating that the instability of linear classifiers depends on the training sample size and on their complexity. In [17] large training sample sizes are used. Indeed, the Fisher linear discriminant is rather stable when it is trained on large training data sets, and bagging is not beneficial in that case. We should mention that as a rule no systematic study (for decision trees and other classifiers) was performed on the performance of boosting and bagging and on their comparison with respect to the training sample size. Usually large training sample sizes were considered (e.g., taking 90% of available objects for training and 10% for testing). In our study, the training sample size plays an important role. In Chapter 5, we investigate the usefulness of bagging for a wide range of linear classifiers in relation with the training sample size and with the instability of classifiers. In the next section we intend to investigate the performance of boosting in LDA in relation to the above mentioned questions and we will compare it with the performance of bagging.

6.3 Boosting versus Bagging for Linear Classifiers

Bagging for linear classifiers was studied by us in Chapter 5. Bagging is based on bootstrapping and aggregating and, by that, incorporates benefits of both approaches. It was established that bagging may be useful for linear classifiers when they are unstable. This happens when the training sample size is critical, that is when the number of training objects is comparable to the data dimensionality. For very small and also for very large training sample sizes bagging is usually useless. In bagging, the actual training sample size is reduced, because on average only $1 - \frac{1}{e} \approx 63.2\%$ of the training objects is used in each bootstrap sample of the training data set (see Section 5.2.4 in Chapter 5). Therefore, the usefulness of bagging also

depends on the small sample size properties of the classifier as bagging has a shifting effect on the generalization error in the direction of the generalization error obtained on the smaller training sets (see Chapter 5). Similar to bagging, boosting also combines multiple classifiers in the final decision rule. However, boosting significantly differs from bagging in the way training sets and base classifiers are obtained at each step of the algorithm. In boosting, classifiers obtained at each step strongly depend on the previously obtained classifiers, while in bagging all obtained classifiers are independent. As boosting maximizes margins between training objects, it is similar to the support vector classifier. Formula (6.1) shows that boosting is the most effective for classifiers having a simple complexity (with the low VC-dimensionality d) and for large training sample sizes N. Therefore, it may be expected that boosting will be useful for the nearest mean classifier (as the most simple linear classifier) constructed on large training sets.

In order to confirm our expectations, let us study the *Regularized Fisher Linear Discriminant* function (RFLD) [30] (see Chapter 1 for more details) which represents a large group of linear classifiers: from the *Fisher Linear Discriminant* (FLD) [35] (when the regularization parameter $\lambda = 0$) and the *Pseudo Fisher Linear Discriminant* (PFLD) [35] (when λ is very small) to the *Nearest Mean Classifier* (NMC) [35] ($\lambda = \infty$) (see Fig. 6.2).

One artificial data set and two real data sets are chosen to perform a simulation study. All data sets represent the 2-class problem. The artificial data set is the 30-dimensional *correlated Gaussian data* set described in Chapter 1. Two real data sets (also described in Chapter 1) are taken from the UCI Repository [12]. The first is the 34-dimensional *ionosphere* data set with 225 and 126 objects belonging to the first and the second data class, respectively. The second is the 8-dimensional *(pima-)diabetes* data set consisting of 500 and 268 objects from the first and the second data class, respectively. Let us note, that the latter two data sets are used in [17], when studying bagging and boosting for decision trees. The diabetes data set is used when bagging and boosting were studied for LDA [17]. Training data sets with 3 to 400, with 3 to 100 and with 3 to 200 samples per class are chosen randomly from the Gaussian correlated data set, the ionosphere data set and the diabetes data set, respectively. The remaining data are used for testing. These and all other experiments are repeated 50 times on independent training sample sets. In all figures the averaged results over 50 repetitions are presented. The results of bagged and boosted linear classifiers are presented after 250 steps, that is 250 base classifiers are combined (however, often 100 classifiers in the ensemble are more than enough to obtain good classification results). In both, bagging and boosting, weighted majority voting (see Section 6.2) is used as a combining rule, in order to perform a fair comparison (see next section). The standard deviations of the mean generalization errors for single, bagged and boosted linear classifiers are of similar order for each data set. When increasing the training sample size, they are decreasing approximately from 0.015 to 0.004, from 0.014 to 0.007 and from 0.018 to 0.004 for the Gaussian correlated data set, for the ionosphere data set and for the

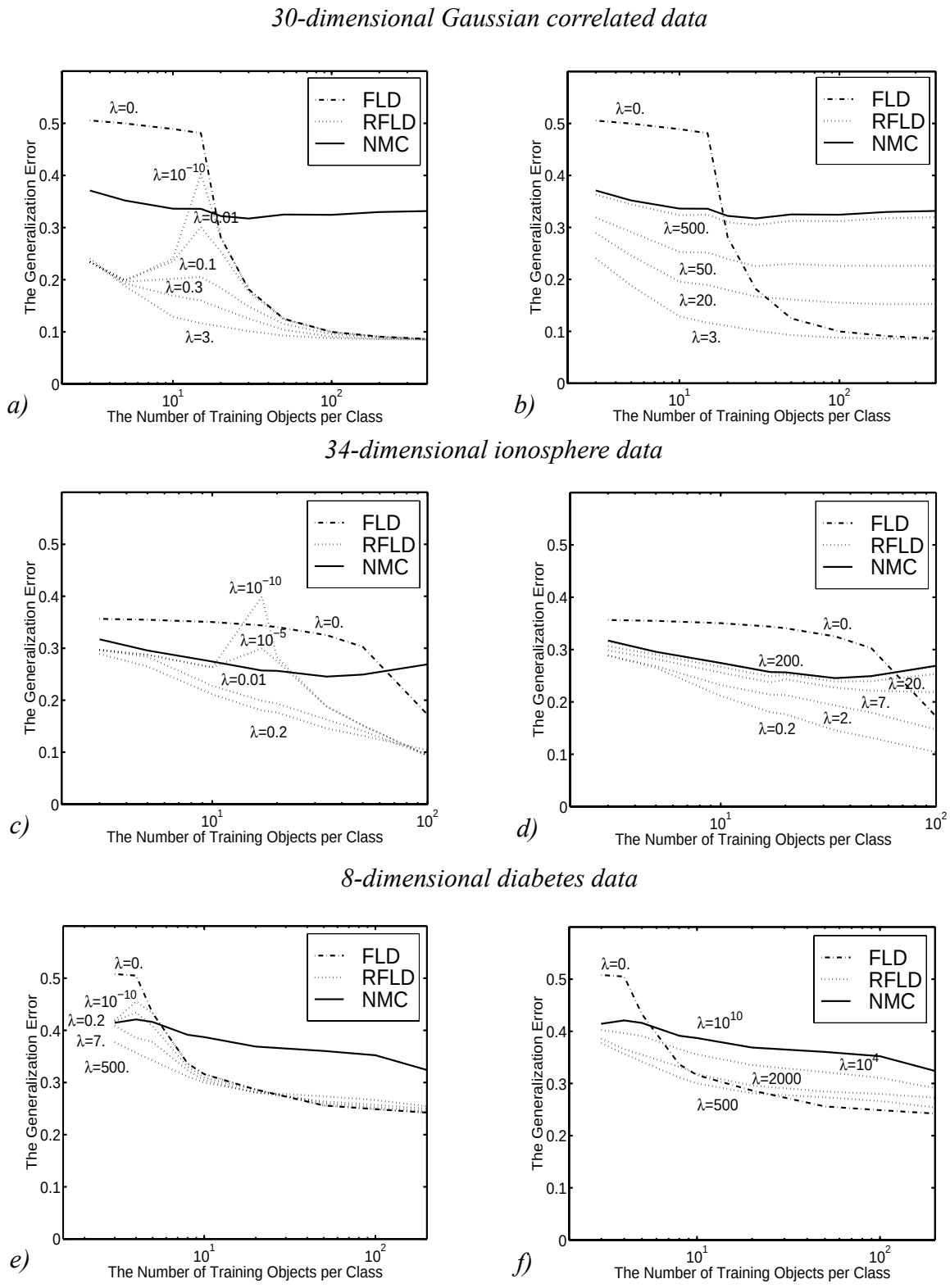


Fig. 6.2. Dependence of the generalization error of the RFLD on the regularization parameter λ for 30-dimensional Gaussian correlated data (a,b), for 34-dimensional ionosphere data set (c,d) and 8-dimensional diabetes data set (e,f).

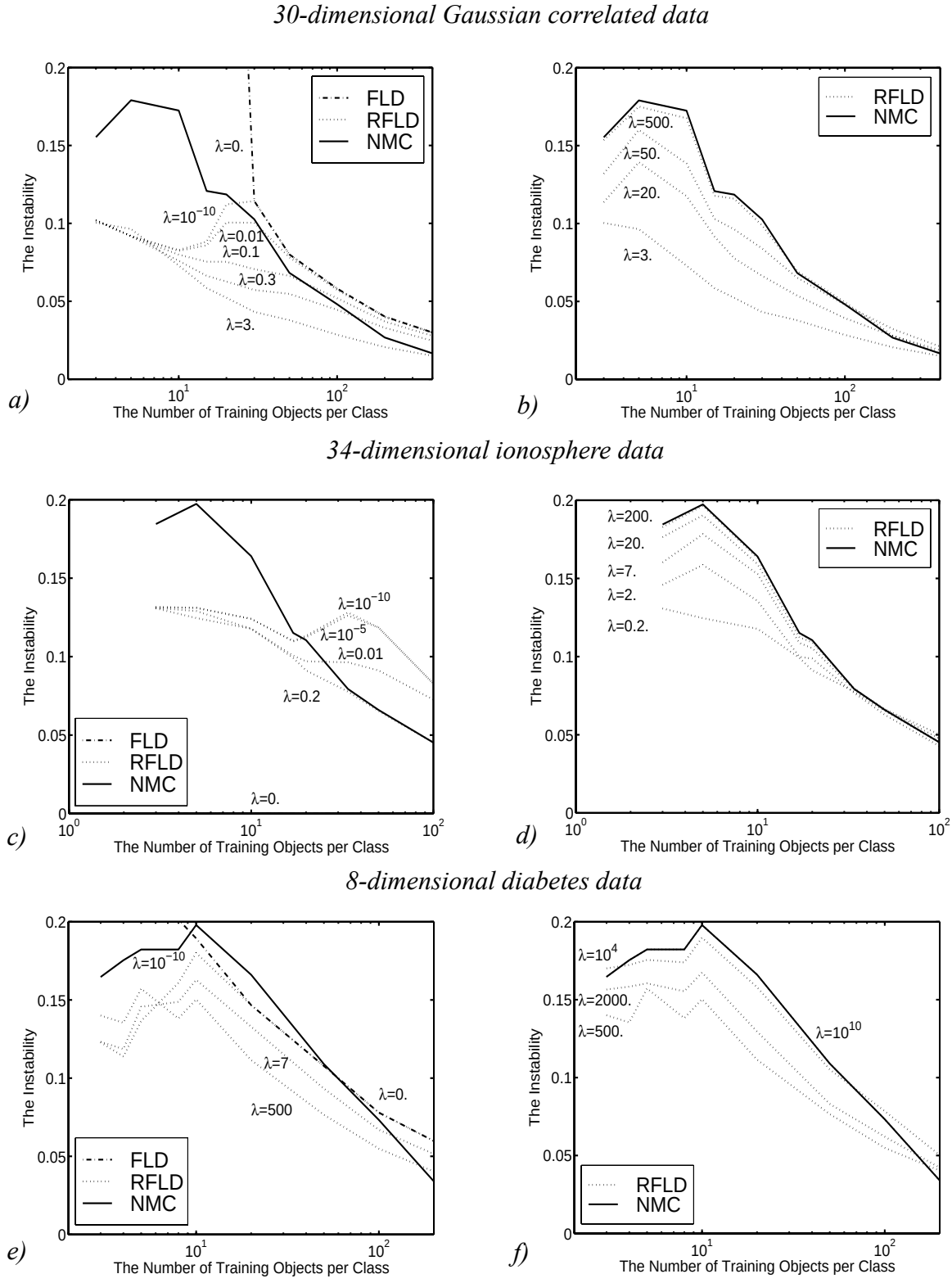


Fig. 6.3. Dependence of the instability of the RFLD on the regularization parameter λ for 30-dimensional Gaussian correlated data (a,b), for 34-dimensional ionosphere data set (c,d) and 8-dimensional diabetes data set (e,f).

diabetes data set, respectively. When the mean generalization error of the single, bagged or boosted RFLD shows a peaking behaviour on the ionosphere data set (see Fig. 6.5), its maximal standard deviation is about 0.03.

In order to understand better when boosting may be beneficial, it is useful to consider the instability of a classifier [117] (see Section 1.5 in Chapter 1). The mean instability of linear classifiers (on 50 independent training sets) is presented in Fig. 6.3. One can see that the instability of the classifier decreases when the training sample size increases. The instability and the performance of a classifier are correlated: usually more stable classifiers perform better than less stable ones. However, the performance of the NMC does not depend on the training sample size. In contrast to other classifiers, it remains a weak classifier for large training sample sizes, while its stability increases. The theory of boosting has been developed for weak classifiers that are simple in complexity and trained on large training sample sizes. Therefore, one can expect that boosting should be beneficial for the NMC.

We will now study the performance of boosting in LDA on the example of the NMC, the FLD and the RFLD in relation with the instability of these classifiers and compare it with the performance of bagging.

6.3.1 Boosting and Bagging for the NMC

Simulation results (see Fig. 6.4g, Fig. 6.5f and Fig. 6.6f) show that bagging and boosting are useful for the NMC, especially for the 30-dimensional Gaussian correlated data set. Bagging may outperform boosting on small training sample sizes ($n=2N<p$). Boosting is more beneficial for training sample sizes larger than the data dimensionality, $n>p$.

Bagging is useless for very small training sample sizes and for large training sample sizes (see Chapter 5). In boosting, wrongly classified objects get larger weights. Mainly, these are objects on the border between classes, which are difficult to classify. Therefore, boosting performs the best for large training sample sizes, when the border between classes becomes more informative and gives a good representation of the real distribution of the data classes. In this case (for large training sample sizes), boosting the NMC performs similarly to the linear SVC [19]. However, when the training sample size is large, the NMC is stable. This brings us to the idea that, in contrast to bagging, the usefulness of boosting does not depend directly on the stability of the classifier. It depends on the "quality" (distribution) of the "difficult" objects and on the potential ability of the classifier (its complexity) to distinguish them correctly.

6.3.2 Boosting and Bagging for the FLD

Considering boosting for the FLD (see Fig. 6.4a, Fig. 6.5a and Fig. 6.6a), one can see that boosting is useless for this classifier. The performance and the stability of the FLD depends on the training sample size. For small training sample sizes, the classifier is very

30-dimensional Gaussian correlated data

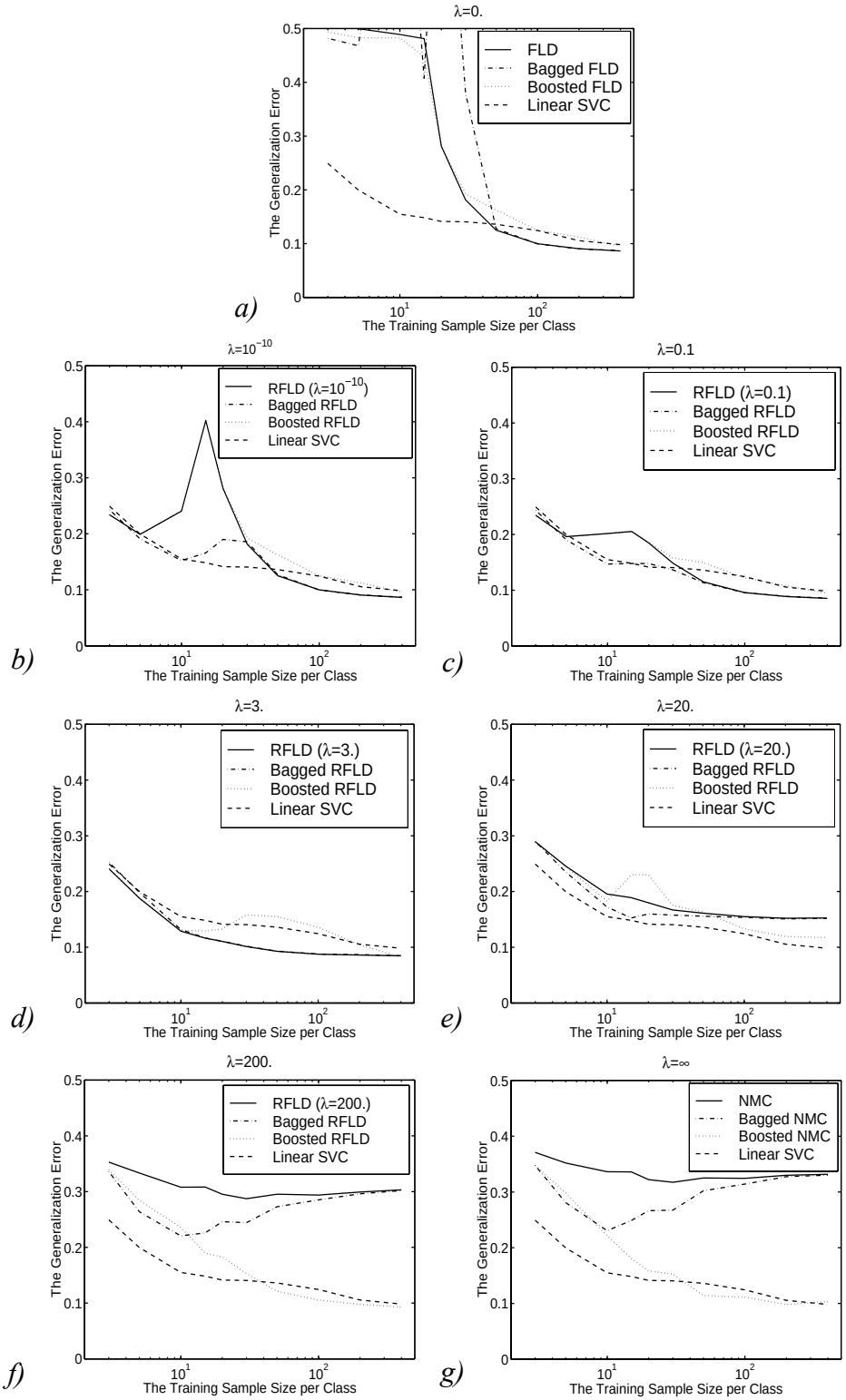


Fig. 6.4. The generalization error of the bagged and boosted linear classifiers (B=250) for 30-dimensional Gaussian correlated data. Boosting becomes useful, when increasing regularization and the RFLD becomes similar to the NMC.

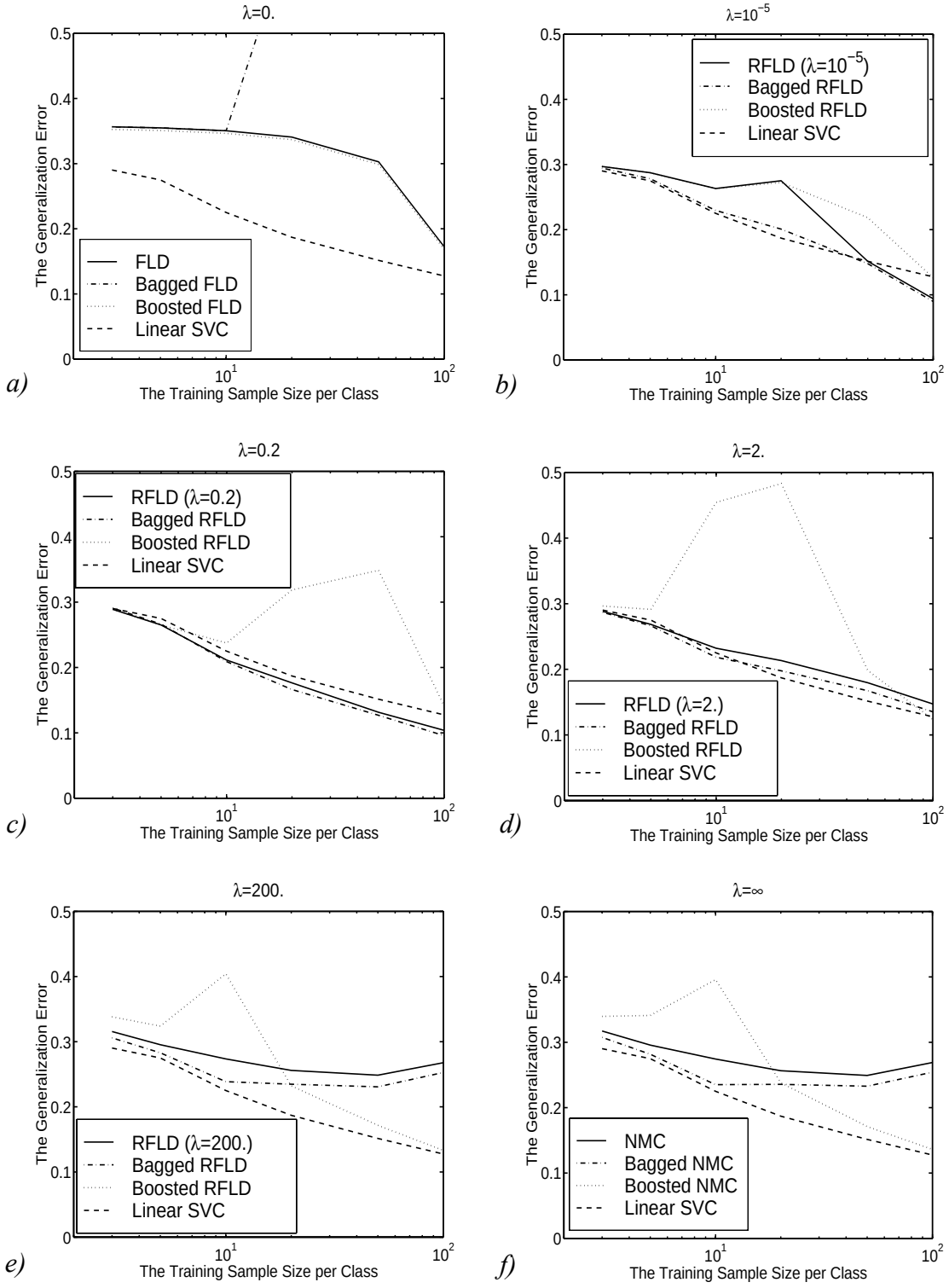


Fig. 6.5. The generalization error of the bagged and boosted linear classifiers (B=250) for 34-dimensional ionosphere data. Boosting becomes useful, when increasing regularization and the RFLD becomes similar to the NMC.

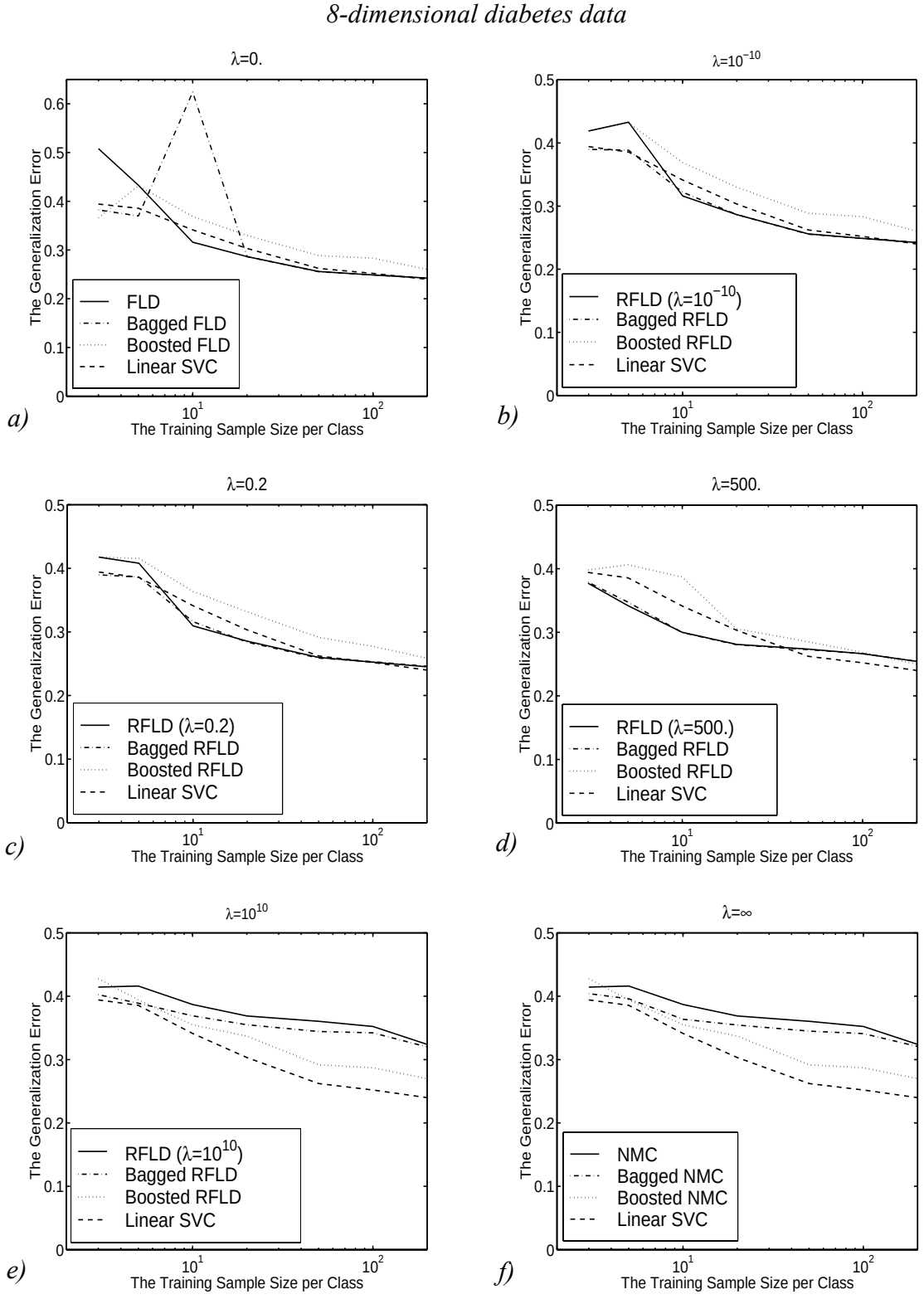


Fig. 6.6. The generalization error of the bagged and boosted linear classifiers (B=250) for 8-dimensional diabetes data. Boosting becomes useful, when increasing regularization and the RFLD becomes similar to the NMC.

unstable and has a poor performance, as sample estimates of means have a large bias and a sample estimate of a common covariance matrix is singular or nearly singular. When increasing the training sample size, the sample estimates are less biased, and the classifier becomes more stable and performs better. In boosting, "difficult" objects on the border between data classes get larger weights. By that, the number of actually used training objects decreases. When the training sample size is smaller than the data dimensionality, all or almost all objects lie on the border. Therefore, almost all training objects are used at each step of the boosting algorithm. One may get many similar classifiers that perform badly [91]. The combination of such classifiers does not improve the FLD. When the training sample size increases, the FLD performs better. In this case, boosting may perform similar to a single FLD (if the number of objects on the border is sufficiently large to construct a good FLD) or may worsen the situation (if the number of actually used training objects at each step of boosting is not sufficiently large to define a good FLD).

Bagging is also useless for the FLD. When the training sample size is relatively small, sample estimates obtained on bootstrap replicates may be biased. Additionally, the sample estimate of the covariance matrix will be singular or nearly singular. The FLD constructed on bootstrap replicates will have a poor performance. Some of these poorly performing classifiers will get a larger weight in voting and will dominate in the final decision. For this reason, bagging is useless and sometimes may even deteriorate the performance of a single classifier as in the case for the ionosphere and the diabetes data sets (see Fig. 6.5a and Fig. 6.6a). Due to the shifting effect of generalization error in bagging (see Section 5.5.2 in Chapter 5), bagging will also be disadvantageous for critical training sample sizes. For large training sample sizes, any bootstrap replicate of the training set gives the correct representation of the true distribution of the data classes. One obtains many similar classifiers. Combining them will hardly improve the single classifier.

6.3.3 Boosting and Bagging for the PFLD

Let us consider boosting for the PFLD, which is similar to the RFLD with a very small value of the regularization parameter (see Fig. 6.4b, Fig. 6.5b and Fig. 6.6b) and compare it with bagging. One can see that due to the shifting effect of the generalization error, bagging is useful for critical training sample sizes (see Section 6.5.2) while boosting is useless for any training sample size. For the training sample sizes larger than the data dimensionality, the PFLD, maximizing the sum of distances to all given samples, is equivalent to the FLD. For the training sample sizes smaller than the data dimensionality, however, the Pseudo Fisher rule finds a linear subspace, which covers all the data samples. The PFLD constructs a linear discriminant perpendicular to this subspace in all other directions for which no samples are given. For these training sample sizes, the apparent error (the classification error on the

training set) of the PFLD is always zero, as in the subspace a perfect separation is always possible. It means that the weights for training objects are not changed by the boosting algorithm, and the same classifier is obtained at each boosting step. Thus boosting is completely useless for the PFLD.

Let us note that the last two examples of linear classifiers, the FLD and the PFLD, also demonstrate nicely that the usefulness of boosting does not depend on the instability of classifiers. The FLD and the PFLD are very unstable, the first for small training sample sizes and the second for critical training sample sizes. However, boosting does not improve the performance of these unstable and poorly performing classifiers.

6.3.4 Boosting and Bagging for the RFLD

Considering the RFLD with different values of the regularization parameter λ, one can see that boosting is also not beneficial for these classifiers with the exception of the RFLD with very large values of λ, which performs similar to the NMC. For small training sample sizes, when all or almost all training objects have similar weights at each step of the boosting algorithm, the modified training set is similar to the original one, and the boosted RFLD performs similarly to the original RFLD. For critical training sample sizes, the boosted RFLD may perform worse or even much worse (having a high peak of the generalization error) than the original RFLD. This is caused by two effects. The first is that the modified training sets used at each step of boosting have actually fewer training objects than the original training data set. Smaller training sets give more biased sample estimates of the class means and covariance matrix than larger training sets. Therefore, the RFLD constructed on the smaller training set usually has a worse performance. Combining the worse quality classifiers constructed on the smaller training sets may result in a classifier that performs worse than the single classifier constructed on the larger training set. The second effect is that the "difficult" objects on the border between data classes (which are getting larger weights in the boosting algorithm) have usually another distribution than the original training data set. Therefore, on such a modified training set, the RFLD with a fixed value of the regularization parameter λ may perform differently than the same RFLD on the original training set. Regularization may not be sufficient, causing a peak in the generalization error similar to the RFLD with very small values of λ. However, for large training sample sizes, boosting may be beneficial for the RFLD, when a single RFLD performs worse than a linear Support Vector Classifier. This happens when the RFLD has large or very large values for the regularization parameter. Consequently, boosting is useful only for the RFLD with large values of the regularization parameter λ and for large training sample sizes.

Bagging is useful for the RFLD ($\lambda \neq 0$) when it is constructed on critical training sample sizes and when the RFLD is unstable (this happens when the regularization parameter λ is not close to its optimal value) (see Section 5.5.3 in Chapter 5).

6.4 The Effect of the Combining Rule in Bagging and Boosting

In the previous section we have established that boosting in LDA is only useful for the NMC or for the RFLD with large values of the regularization parameter that approximates the NMC. Boosting is the most effective for large training sample sizes. The effectiveness of boosting does not depend on the instability of the classifier. On the contrary, bagging is useful for critical training sample sizes when linear classifiers are unstable. Therefore, when one compares the performance of boosting and bagging, the training sample size should be taken into account. Another important issue, when comparing both techniques is the combining rule used. Usually, in boosting one combines classifiers by weighted majority voting, while in bagging the simple majority vote combining rule is used. As both boosting and bagging are techniques that combine multiple classifiers into the final decision rule, it seems unfair to compare their performance when using unequal combining rules in both methods. In order to investigate this question, let us consider now how boosting and bagging perform for the NMC, when using different combining rules.

We consider five different combining rules: the average, the mean, the product (see Section 5.2.2 in Chapter 5 for description of these three rules), the simple majority voting (see Section 5.2.1 in Chapter 5) and the weighted majority vote (see Section 6.2). The simulation results on boosting and bagging for the NMC using these combining rules are presented in Fig. 6.7 for all three data sets studied above. Left side plots show results obtained for bagging, right side plots show results for boosting. One can see that the choice of the combining rule may strongly affect the performance of bagging and boosting. The simple majority voting is the worst among all studied combining rules for both bagging and boosting. The average combining rule is the best for bagging. The product and the weighted majority vote combining rules are the second-best combining rules for bagging. The mean combining rule follows them. For the ionosphere and the diabetes data sets, the choice of the combining rule is not that important (Fig. 6.7c,e) as for the 30-dimensional Gaussian correlated data set (Fig. 6.7a), where the choice of the combining rule strongly affects the performance of bagging. However, one can see that the product combining rule may degrade the performance of bagging on small training sample sizes due to overtraining (see Fig. 6.8a and Fig. 6.9a). For boosting, the weighted majority vote combining rule is the best, especially on the ionosphere data set (Fig. 6.7d). The average, the product and the mean combining rules are the second-best combining rules.

Considering Fig. 6.7d, one can see that on the ionosphere data set boosting performs well only for the weighted majority vote combining rule. It implies that boosting overfits if other combining rules are used. In order to study the effect of overtraining for bagging and boosting and whether it is affected by the choice of the combining rule, let us consider the performance of the bagged and boosted NMC with respect to the number of combined classifiers. In Fig.

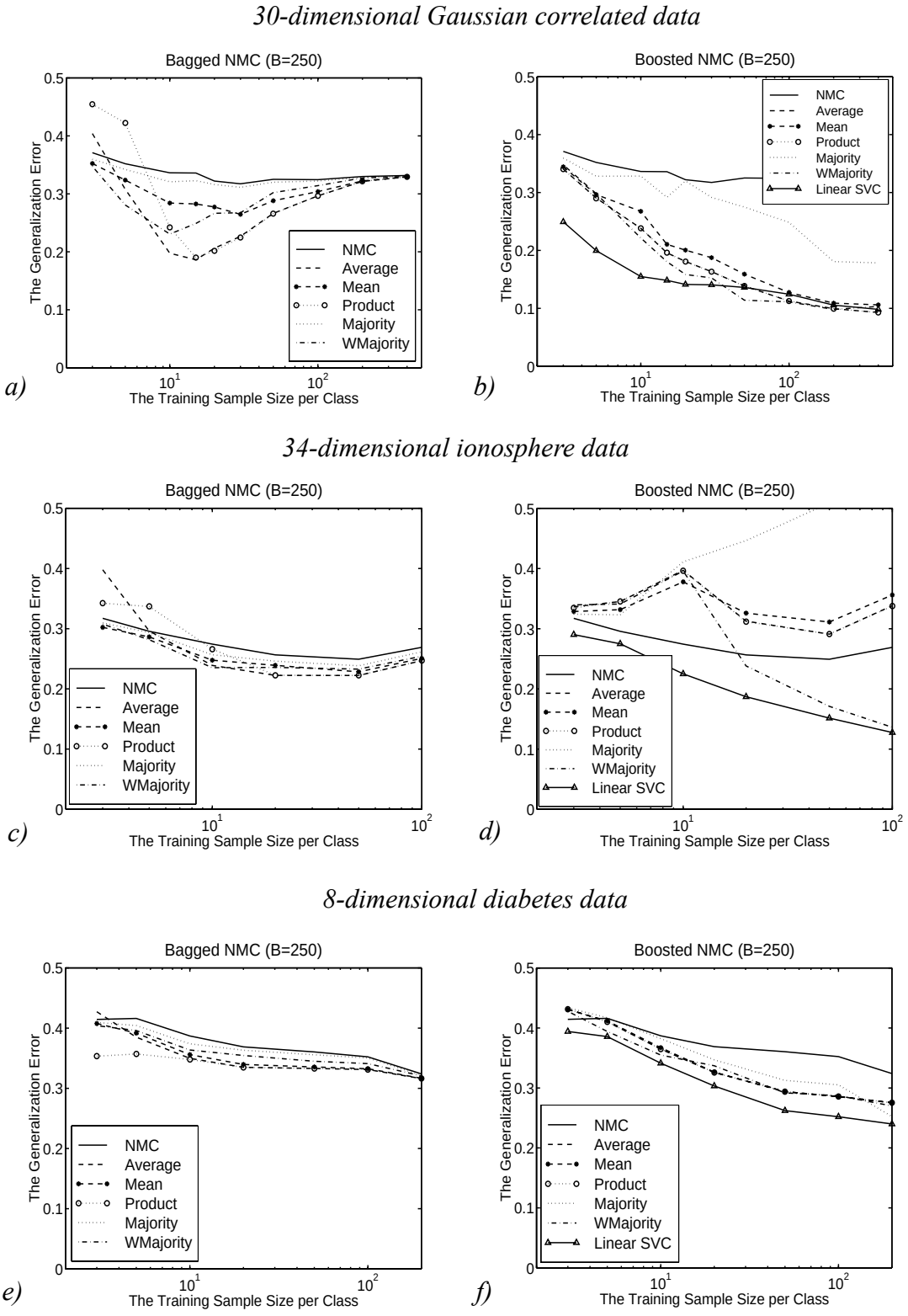


Fig. 6.7. The generalization error of the NMC, the bagged NMC (left plots) and the boosted NMC (right plots) using different combining rules for 30-dimensional Gaussian correlated data (a,b), for 34-dimensional ionosphere data (c,d) and for 8-dimensional diabetes data (e,f).

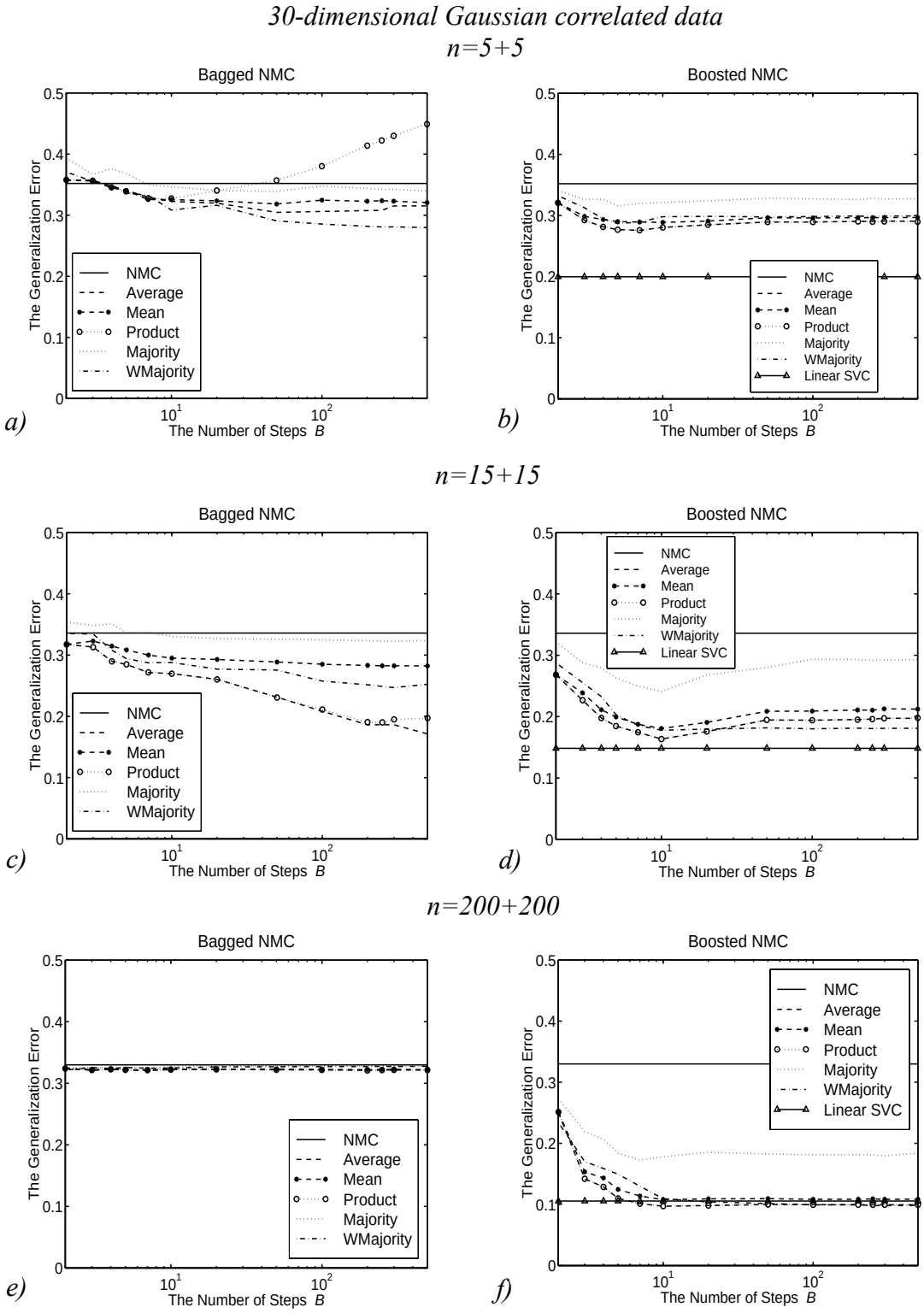


Fig. 6.8. The generalization error of the NMC, the bagged NMC (left plots) and the boosted NMC (right plots) using different combining rules for 30-dimensional Gaussian correlated data versus the number of steps (combined classifiers).

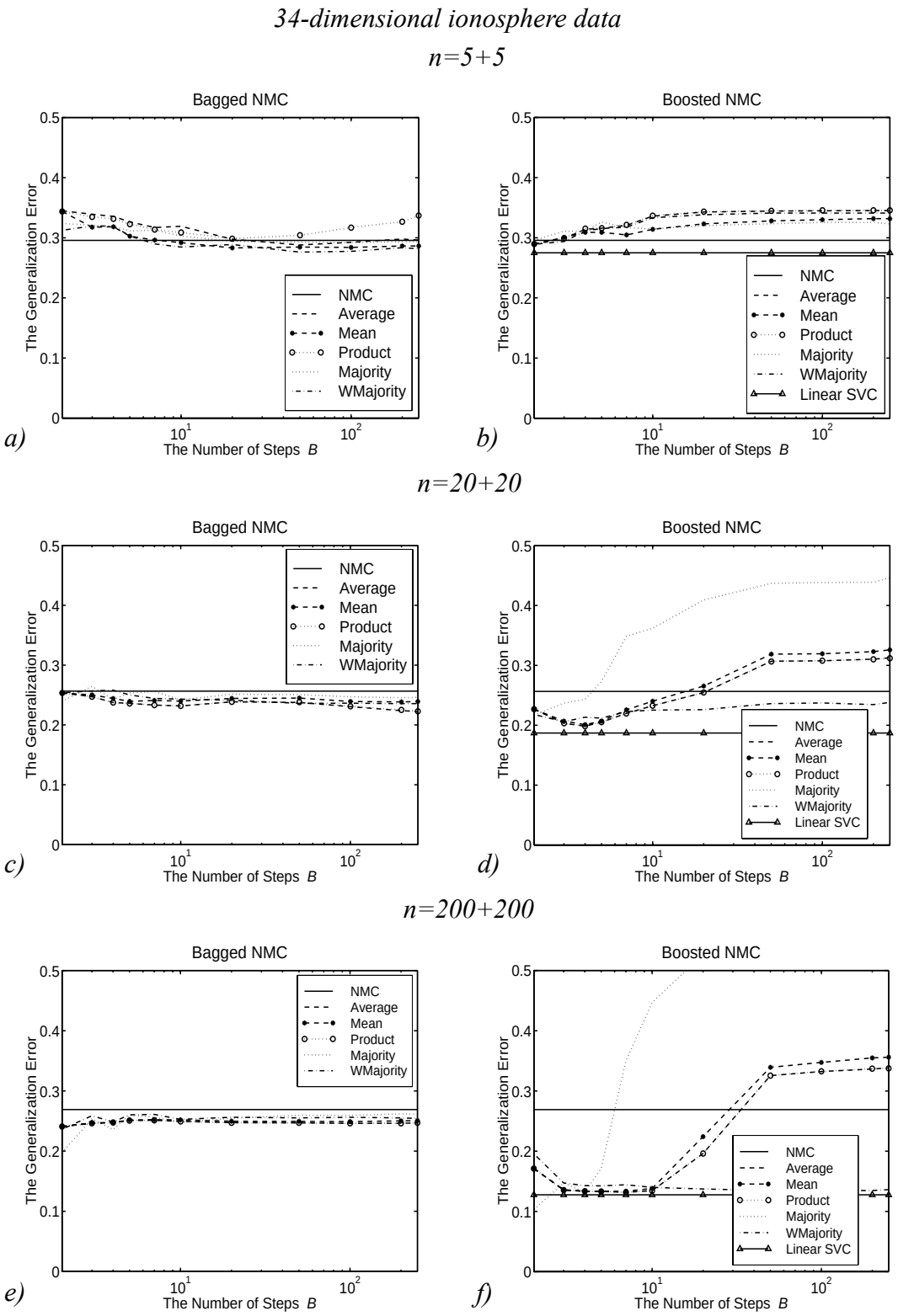


Fig. 6.9. The generalization error of the NMC, the bagged NMC (left plots) and the boosted NMC (right plots) using different combining rules for 34-dimensional ionosphere data versus the number of steps (combined classifiers).

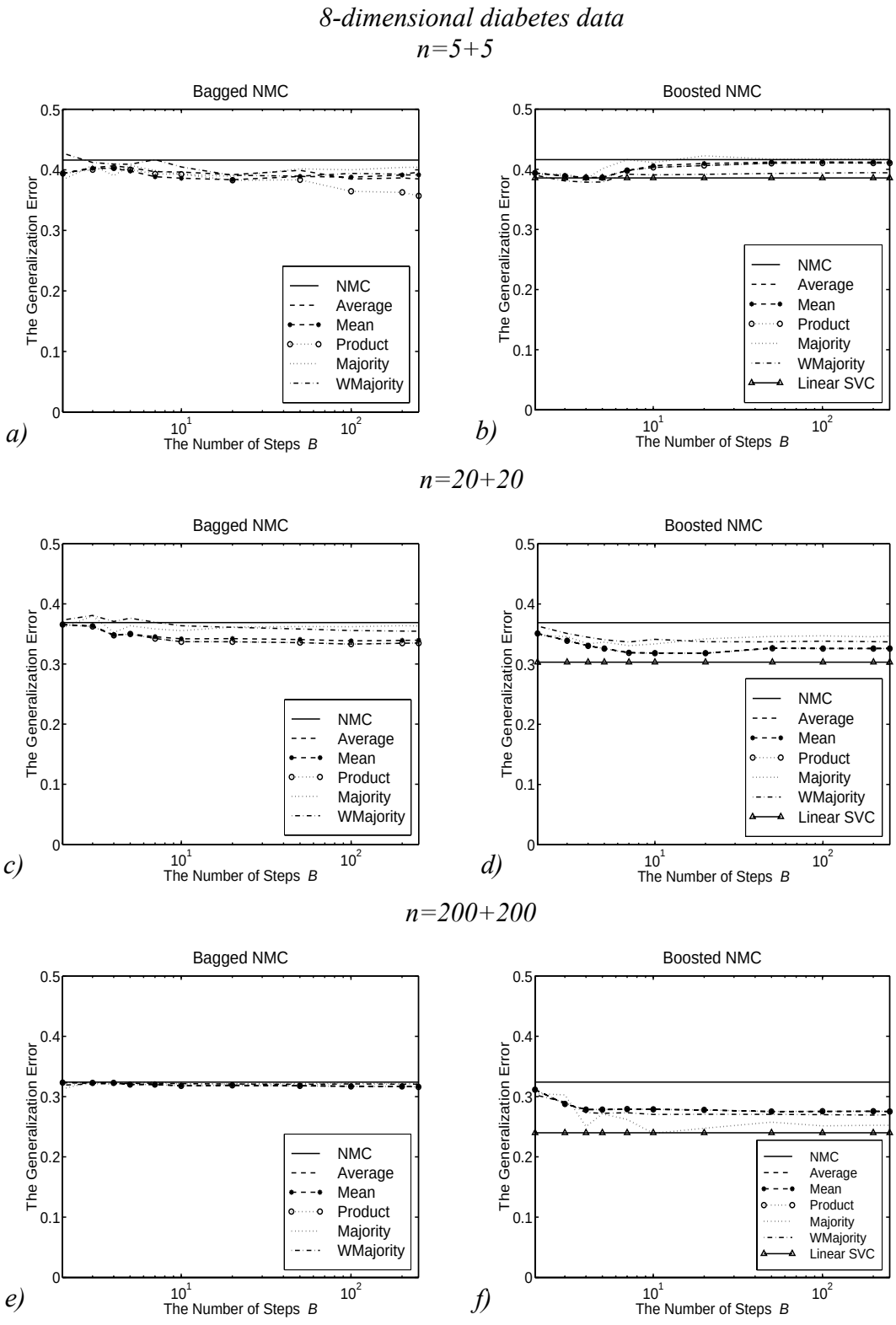


Fig. 6.10. The generalization error of the NMC, the bagged NMC (left plots) and the boosted NMC (right plots) using different combining rules for 8-dimensional diabetes data versus the number of steps (combined classifiers).

6.8-6.10 we consider small (plots a, b), critical (plots c, d) and large (plots e, f) training sample sizes for each data set. The obtained plots show that bagging is almost unaffected by overtraining. Only when the training sample size is small and the product combining rule is used, bagging produces overfitting (Fig. 6.8a and Fig. 6.9a). The explanation as to why bagging is not affected by overtraining is the following. In bagging, the training data set is modified randomly and independently at each step. The bootstrap resampling used in bagging is known as a robust technique. Therefore, in expectation, the distribution of a bootstrap sample (which consists of 63,2% of training data) becomes similar to the real data distribution. In this way, bagging is prevented from overtraining. In agreement with formula (6.1), Fig. 6.8-6.10 show that boosting may overfit for the NMC. In boosting, the training data set is sequentially modified at each step, emphasizing the objects erroreously classified at the previous step. Sometimes these objects may have a different distribution than the real data. Taking many steps in boosting, it may happen that after some time one obtains only modified training sets consisting of a few "difficult" objects, which represent the real data distribution incorrectly. If the number of base classifiers constructed on deteriorated training sets is too large or some of their vote weights are too large, then boosting may overfit (the generalization error of the boosted classifier becomes worse). Sometimes (although not always, e.g. see Fig. 6.8b,d,f), when increasing the training sample size, the effect of overtraining may increase (see right side plots in Fig. 6.9b,d,f). It depends on the data distribution and on the distribution of the "difficult" objects. Figures 6.8-6.10 demonstrate that boosting overfits less when the weighted majority vote combining rule is used than when other combining rules are used in boosting. One can also clearly see that when the training sample size is large, the boosted NMC approaches the performance of the SVC (see Fig. 6.8f, 6.9f, 6.10f).

6.5 Conclusions

Summarizing the simulation results presented in the previous section, we can conclude the following.

Boosting may be useful in linear discriminant analysis for classifiers that perform poorly on large training sample sizes. Such classifiers are the Nearest Mean Classifier and the Regularized Fisher's Linear Discriminant with large values of the regularization parameter λ, which approximates the NMC.

Boosting is useful only for *large training sample sizes* (while bagging helps in unstable situations, for critical training sample sizes), if the objects on the border give a better representation of the distributions of the data classes than the original data classes distribution and the classifier is able (by its complexity) to distinguish them well.

As boosting is useful only for large training sample sizes, when classifiers are usually stable, *the performance of boosting does not depend on the instability of the classifier.*

It was shown theoretically and experimentally for decision trees [107] that boosting increases the margins of the training objects. In this way, boosting is similar to maximum margin classification [19], based on minimizing the number of support vectors. In this chapter, we have experimentally shown that *boosted linear classifiers may achieve the performance of the linear support vector classifier* when training sample sizes are large compared with the data dimensionality.

The choice of the combining rule may be *important*. However, it strongly depends on the data and the training sample size.

When comparing the performance of bagging and boosting, it should be done in a "fair" context, when the same combining rule is used in both methods.

As a rule, simple majority voting is the worst possible choice for the combining rule. The weighted majority vote rule is often a good choice for bagging as well as for boosting. However, in order to compute the final decision, one should store the weights and the decisions of each base classifier in the ensemble. The average, the mean and the product combining rules may also perform well and sometimes better than the weighted majority vote combining rule. The advantage of the average rule is that it is sufficient to store just the coefficients of the base classifiers. The final classifier is a single classifier with the same number of parameters as each of the base classifiers.

The success of boosting depends on many factors including the training sample size, the choice of the weak classifier (decision tree, the FLD, the NMC or other), the exact way that the training set is modified, the choice of the combining rule [121] and, finally, the data distribution. Thus, it is difficult to establish universal criteria predicting the usefulness of boosting. Obviously, this question needs more investigation in future.

Chapter 7

The Random Subspace Method for Linear Classifiers

7.1 Introduction

When the number of training objects is smaller than the data dimensionality, the sample estimates of the classifier parameters are usually biased. As a result, the classifier obtained is unstable having a large variance. It may happen that such classifiers have a bad performance [91]. In order to improve weak classifiers, one may either stabilize them by regularization techniques (see Chapters 2-4), or apply combining techniques (see Chapters 5 and 6) without directly stabilizing the discriminant function.

When applying combining techniques, one modifies the training set, constructs classifiers on these modifications and then combines them in the final decision rule. Lately, several combining techniques have been designed for decision trees, where they often perform better than a single classification rule [17, 21, 47]. The most popular ones are bagging [15], boosting [32] and the Random Subspace Method (RSM) [47].

Bagging, boosting and the RSM are combining techniques. However, they differ in the way the modified training sets and the base classifiers are obtained. In *bagging*, one constructs base classifiers on *random independent* bootstrap replicates of the training set. In *boosting*, the training sets and the classifiers are obtained in a strictly *deterministic* way. In the *RSM*, one constructs base classifiers in *random subspaces* of the data feature space. Therefore, the obtained classifiers are *random* and *independent* as in bagging. However, in bagging one samples the training objects while in the RSM one samples the features of the data set.

In the RSM, by using random subspaces, one actually decreases the data dimensionality while the training sample size remains the same. When the data have many redundant features, one may obtain better classifiers in random subspaces than in the original complete feature space. The combined decision of such classifier may be superior to a single classification rule constructed in the original complete feature space. Thus, one may expect that the RSM will be useful for linear classifiers which suffer from the curse of dimensionality.

In Chapters 5 and 6, we have studied bagging and boosting for linear classifiers. In this chapter, which is based on our works [124, 125], we study the usefulness of the RSM in LDA for 2-class problems and in particular to investigate its relation to the training sample size and the small sample size properties of the base classifier. We also compare the performance of the RSM with the performances of bagging and boosting. All mentioned combining techniques are designed for weak classifiers. The most popular linear classifiers are the Nearest Mean Classifier (NMC) [35] and the Fisher Linear Discriminant function (FLD) [35]. However,

when the number of training objects is smaller than the data dimensionality, the sample estimate of the covariance matrix is singular. In these circumstances, the FLD cannot be constructed as it requires the inverse of the covariance matrix. In order to overcome the direct inverse of an ill-conditioned covariance matrix, one may perform the Moore-Penrose Pseudo inverse, which is used in the Pseudo Fisher Linear Discriminant function (PFLD) [35]. In addition, both the NMC and the PFLD are weak classifiers (see Chapter 1). The NMC is usually weak when data classes have another distribution than Gaussian distribution with equal variances. The PFLD as such is weak for critical training sample sizes (when the training sample size is comparable with the data dimensionality) and its use as a single classifier is not recommended. Our previous study (see Chapters 5 and 6) has shown that combining techniques are useful for these two classifiers. Therefore, we have chosen the NMC and the PFLD as the base classifiers for our simulation study on the RSM.

This chapter is organized as follows. First, a description of the RSM algorithm with some discussion is given in Section 7.2. Simulation results on the performance of the RSM for linear classifiers are discussed in Section 7.3. In Sections 7.3.1 and 7.3.2, we compare the performance of the RSM with the performance of bagging and boosting. The effect of the redundancy in the data feature space on the performance of the RSM is studied in Section 7.3.3. Whether the RSM is a stabilizing technique is considered in Section 7.3.4. Conclusions are summarized in Section 7.4.

7.2 The Random Subspace Method

The random subspace method is the combining technique proposed by Ho [47]. In the RSM, one also modifies the training data as in other combining techniques. However, this modification is performed in the feature space. Let each training object $\boldsymbol{X}_i$ ($i = 1, \ldots, n$) in the training sample set $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$ be a p-dimensional vector $\boldsymbol{X}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. In the RSM, one randomly selects $p^* < p$ features from the data set $\boldsymbol{X}$. By this, one obtains the p^*-dimensional random subspace of the original p-dimensional feature space. Therefore, the modified training set $\tilde{\boldsymbol{X}}^b = \left(\tilde{\boldsymbol{X}}_1^b, \tilde{\boldsymbol{X}}_2^b, ..., \tilde{\boldsymbol{X}}_n^b\right)$ consists of p^*-dimensional training objects $\tilde{\boldsymbol{X}}_i^b = (x_{i1}^b, x_{i2}^b, \ldots, x_{ip^*}^b)$ ($i = 1, \ldots, n$), where p^* components x_{ij}^b ($j = 1, \ldots, p^*$) are randomly selected from p components x_{ij} ($j = 1, \ldots, p$) of the training vector $\boldsymbol{X}_i$ (the selection is the same for each training vector). One then constructs base classifiers in the random subspaces $\tilde{\boldsymbol{X}}^b$ (of the same size), $b = 1, \ldots, B$, and combines them by simple majority voting in the final decision rule. The RSM is organized in the following way.

1. Repeat for $b = 1, 2, \ldots, B$.
   a) Select the p^*-dimensional random subspace $\tilde{\boldsymbol{X}}^b$ from the original p-dimensional feature space $\boldsymbol{X}$.
   b) Construct a base classifier $C_b(\boldsymbol{x})$ (with a decision boundary $C_b(\boldsymbol{x}) = 0$) in $\tilde{\boldsymbol{X}}^b$.

2. Combine classifiers $C_b(\boldsymbol{x})$, $b = 1, 2, \ldots, B$, by simple majority voting to a final decision rule $\beta(\boldsymbol{x}) = \underset{y \in \{-1, 1\}}{arg\,max} \sum_b \delta_{\mathrm{sgn}(C_b(\boldsymbol{x})), y}$, where $\delta_{i,j}$ is Kronecker symbol, $y \in \{-1, 1\}$ is a possible decision (class label) of the classifier.

The RSM may benefit from using both random subspaces for constructing the classifiers and aggregating the classifiers. In the case, when the number of training objects is relatively small as compared with the data dimensionality, by constructing classifiers in random subspaces one may solve the small sample size problem. The subspace dimensionality is smaller than in the original feature space while the number of training objects remains the same. By this, the training sample size increases relatively. When the data set has many redundant features, one may obtain better classifiers in random subspaces than in the original feature space. The combined decision of such classifiers may be superior to a single classifier constructed on the original training set in the complete feature space.

The RSM is developed for Decision Trees (DT's) [14] in order to improve the performance of weak classifiers. The RSM is expected to perform well when there is a certain redundancy in the data feature space [47]. It was noticed that the performance of the RSM is affected by the problem complexity (feature efficiency, the length of class boundary etc.) [49].

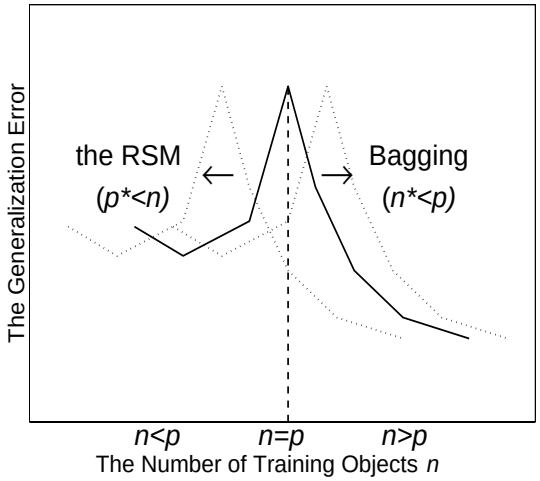


Fig. 7.1. The shifting effect of generalization error of the PFLD for the RSM and for bagging. In the RSM, the learning curve of the final classifier shifts with respect to the learning curve of the original classifier in the direction of smaller training sample sizes (the final classifier obtained in the RSM performs similar to the original classifier constructed on a larger training set). In bagging, the learning curve shifts with respect to the learning curve of the original classifier in the direction of larger training sample sizes (the bagged classifier performs similar to the original classifier constructed on a smaller training set). Here, n is the number of training objects and p is the data dimensionality.

When applying to DT's, the RSM is superior to a single classifier and may outperform both, bagging and boosting [47]. It is also shown that the RSM is beneficial for k Nearest Neighbours (NN) classifiers [48]. The RSM has not been yet used in LDA. The performance of the RSM (for DT's and k-NN classifiers) has not been studied with respect to the training sample size either. However, it seems very promising to apply the RSM to linear classifiers which often suffer from the curse of dimensionality and to investigate its usefulness in relation to the training sample size.

In the RSM, by using random subspaces, the number of training objects relatively increases. This implies that the performance of the RSM in LDA will be affected by the small sample size properties of the base classifier. The training set in the subspace is comparable to a larger training set in the complete feature space. Therefore, classifiers constructed in them are expected to perform similarly. By this, the combined classifier in the RSM will also have similar characteristics to the classifier built on the larger training set. Thus, as in bagging (see Section 5.5.2 in Chapter 5), in the RSM, *the learning curve* (the generalization error as a function of the training sample size) *of the final classifier* will be *shifted with respect to the learning curve of the original classifier* (see Fig. 7.1). However, this shift will be in the opposite direction: *in the direction of smaller training sample sizes* (that is the final classifier obtained in the RSM will perform like the original classifier constructed on a larger training set). Therefore, one may expect that the RSM will be useful for classifiers having a clearly *decreasing learning curve* (e.g., for the FLD) and will be useless for the NMC since its performance does not depend on the training sample size. We also expect that the RSM will be effective for the PFLD constructed on critical training sample sizes ($n \sim p$). As well, it may be expected that the RSM is more beneficial than bagging for this classifier, as the PFLD performs better on large training sample sizes than on small ones.

In connection with the above discussion, one can see that to study the efficiency of the RSM for linear classifiers in relation to the training sample size and the small sample size properties of the base classifier is of great interest.

7.3 The RSM for Linear Classifiers

In order to observe the performance of the RSM in different situations and compare it with the performance of bagging and boosting, we consider two linear classifiers and several artificial and real data sets. The linear classifiers are the NMC and the PFLD (which is equivalent to the FLD for training sample sizes larger than the data dimensionality, $n>p$). We have chosen them for two reasons. The first one is that they are both weak classifiers. The other one is that they have quite different small sample size properties that allow us to study the effect of these on the performance of the RSM. In addition, only these two classifiers may benefit from bagging and boosting (see Chapters 5 and 6).

Most data sets used in our experimental investigations are highly dimensional, because we are interested in unstable situations (where classifiers are weak) when the data dimensionality is smaller than or comparable to the training sample size. In our study, we use artificial data sets which have many redundant features since the RSM is expected to be the most effective for data with a small intrinsic dimensionality. They are the 200-dimensional *Gaussian correlated data* set and the 200-dimensional *Gaussian spherical data* set, described in Chapter 1. Real data sets are needed to show that these techniques may be effective when solving real world problems. The real data sets are taken from the UCI Repository [12]. These are the 34-dimensional *ionosphere* data set, the 8-dimensional *pima-diabetes* data set, the 60-dimensional *sonar* data set, the 30-dimensional *wdbc* data set and the 24-dimensional *german* data set, which have been described in Chapter 1. These data sets were used in [17, 47], when studying bagging, boosting and the RSM for decision trees. The diabetes data set was also used when bagging and boosting were applied in LDA [17].

Training sets are chosen randomly from a total data set. The remaining data are used for testing. All experiments are repeated 50 times on independent training sets. Therefore, in all figures the averaged results over 50 repetitions are presented. The standard deviations of the mean generalization errors for single and aggregated linear classifiers are of similar order for each data set. When increasing the training sample size, they are decreasing approximately from 0.018 to 0.005.

Bagging, boosting and the RSM are techniques which combine multiple classifiers. They fundamentally differ from each other in the ways the base classifiers in the ensemble are obtained. By using a different combining rule in each of the combining techniques, we make

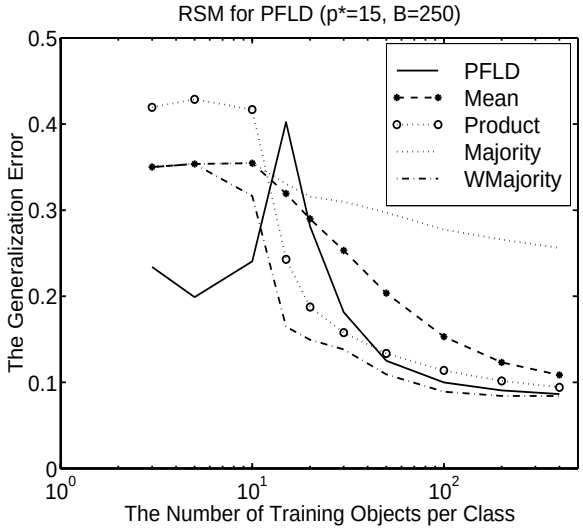


Fig. 7.2. The generalization error of the PFLD in random subspaces (p =15) using different combining rules (the mean, the product, simple majority and weighted majority voting) for 30-dimensional Gaussian correlated data.*

the difference between them even larger. Our previous study [122] has revealed that the choice of the combining rule may be very important when combining classifiers in bagging and boosting (see Fig. 6.7 in Chapter 6). Fig. 7.2 shows that it also affects the performance of the RSM. Therefore, in order to perform a fair comparison of the RSM with bagging and boosting, we have decided to use the weighted majority combining rule (as it is defined in boosting, see Section 6.2) in the RSM and in two other combining techniques. In addition, when using this combining rule, bagging and boosting usually do not overfit, while they often overfit when using other combining rules (see Fig. 6.8-6.10 in Chapter 6).

7.3.1 The RSM versus Bagging for the PFLD

From the conclusions obtained in Chapter 5, in Fig. 7.3 we see that bagging may be useful for the PFLD constructed on critical training sample sizes. Boosting the PFLD is completely useless (see Section 6.3.3 in Chapter 6).

The RSM may be very useful for the PFLD (and therefore, for the FLD, as they are equivalent for the training sample sizes larger than the data dimensionality). Due to the shifting effect of the RSM on the learning curve of the final classifier in the direction of smaller training sample sizes (see Fig. 7.1) (the final classifier obtained in the RSM performs similar to the original classifier constructed on a larger training set), the RSM may significantly improve the performance of the PFLD constructed on critical training sample sizes. When the training sample size is large, the PFLD is not a weak classifier and the RSM becomes less useful.

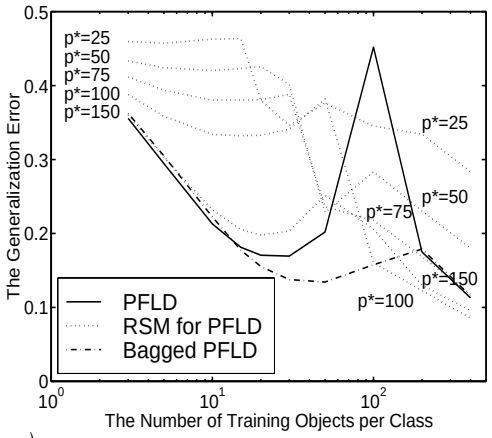
Usually the RSM performs better than bagging for critical training sample sizes $n \sim p$ (see Fig. 7.3b-g) because the *bagged* classifier performs similar to the original classifier constructed on *smaller* training sets and the final classifier in the *RSM* performs like the original classifier constructed on *larger* training sets (see Fig. 7.1). However, sometimes bagging may outperform the RSM for training sample sizes smaller than the data dimensionality $n < p$ (see Fig. 7.3a).

Obviously, the efficient dimensionality of random subspaces depends on the level of redundancy in the data feature space. Further, the usefulness of the RSM is affected by the relation between the number of informative features, the total data dimensionality and the subspace dimensionality. This subject is discussed in Section 7.3.3.

7.3.2 The RSM versus Bagging and Boosting for the NMC

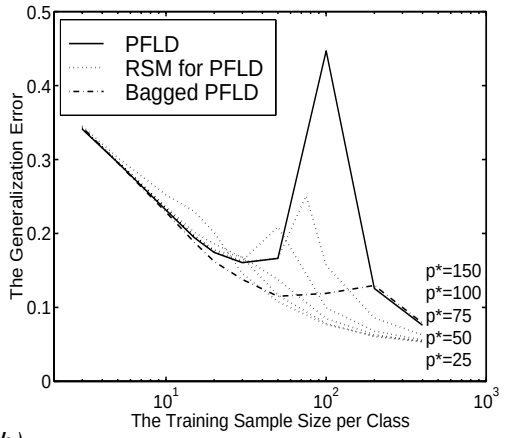
Simulation results (see Fig. 7.4) show that the combining techniques may be useful for the NMC. Bagging may improve the generalization error of the NMC for critical training sample sizes, when the classifier is unstable (see Fig. 7.4a,c,d) (see Chapter 5). Boosting performs the best for large training sample sizes (see Chapter 6), when the border between data

200-dimensional Gaussian correlated data



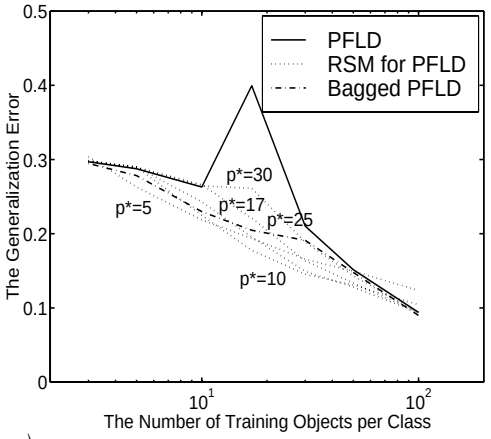
a)

200-dimensional Gaussian spherical data



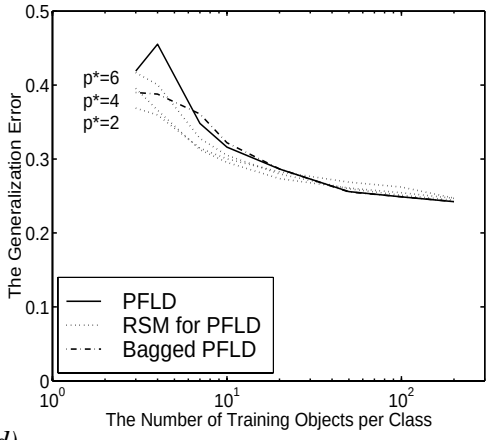
b)

34-dimensional ionosphere data



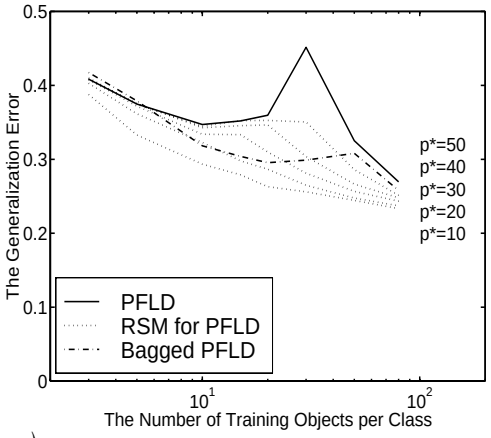
c)

8-dimensional pima-diabetes data



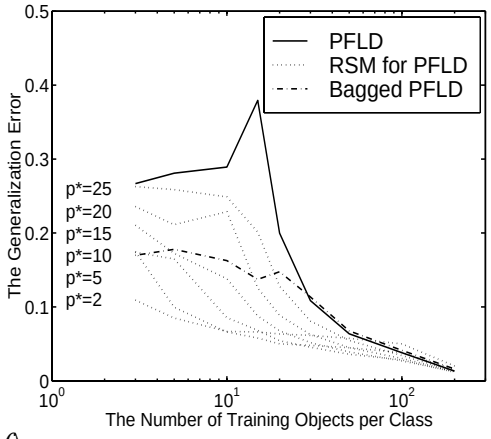
d)

60-dimensional sonar data



e)

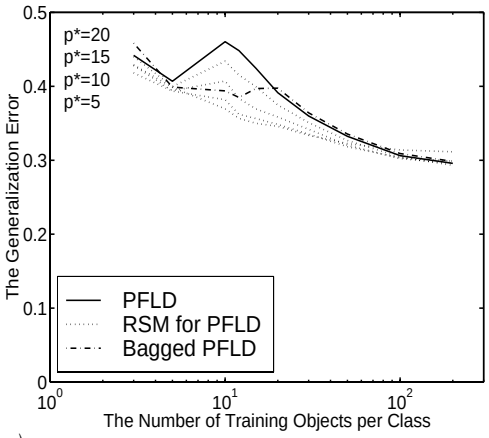
30-dimensional wdbc data



f)

Fig. 7.3. The performance of the RSM and bagging (B=250) for the PFLD.

24-dimensional german data



g)

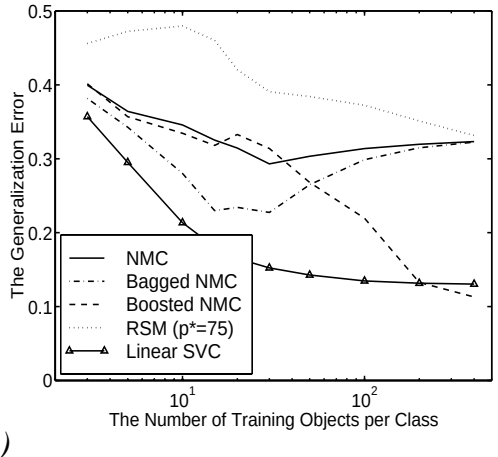
Fig. 7.3. The performance of the RSM and bagging (B=250) for the PFLD.

classes becomes more informative and gives a good representation of the real distribution of the data classes (see Fig. 7.4a,c,d,e). In this case (for large training sample sizes), boosting the NMC performs similar to the linear SVC. However, the performance of the combining techniques is affected by the data distribution and by the potential ability of the base classifier to distinguish data classes well. Therefore, it may happen that for some data sets bagging and boosting are useless (see Fig. 7.4b,f,g).

The RSM is usually useless for the NMC (see Fig. 7.4a-c,e,f). However, sometimes, if some data features are redundant and the learning curve of the NMC decreases with an increase of the training sample size, the RSM may improve the performance of the NMC and outperform bagging and boosting (see Fig. 7.4d,g). It happens because the relative number of training objects increases when constructing classifiers in random subspaces. In addition, sometimes it may be easier to separate data classes in the subspaces than in the complete feature space. If classifiers obtained in random subspaces are diverse and perform better than the original classifier constructed in the complete feature space, combining such classifiers is useful.

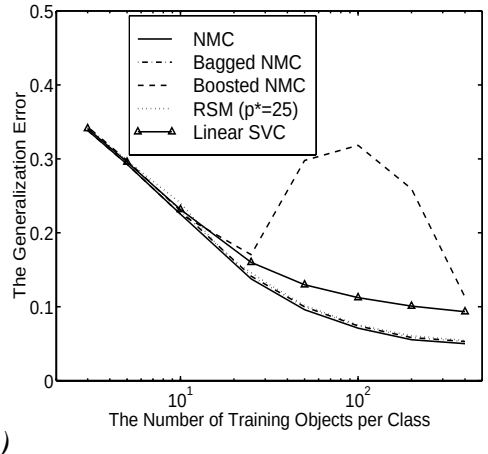
Let us note that bagging, boosting and the RSM are completely useless for the NMC constructed on the Gaussian spherical data set and on the wdbc data set (see Fig. 7.4b,f). The NMC is not a weak classifier for these data sets. By modifying the training set it is difficult to get better classifiers than the single classifier obtained on the original training set. Combining such classifiers is not beneficial.

200-dimensional Gaussian correlated data



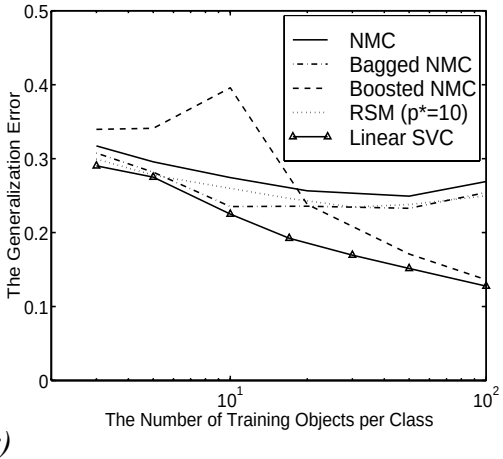
a)

200-dimensional Gaussian spherical data



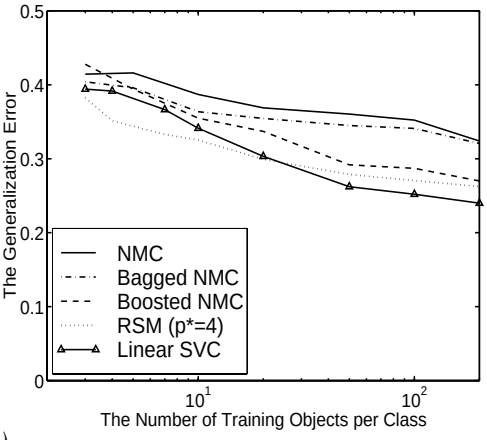
b)

34-dimensional ionosphere data



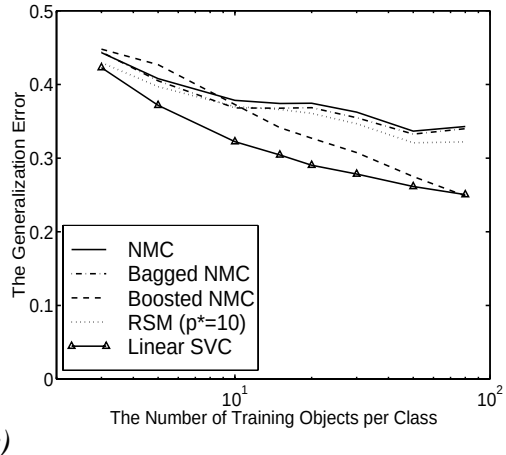
c)

8-dimensional pima-diabetes data



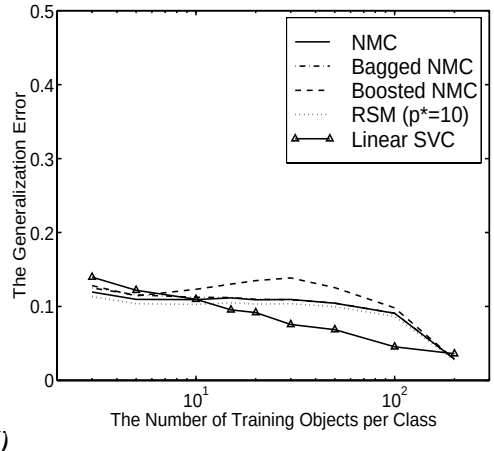
d)

60-dimensional sonar data



e)

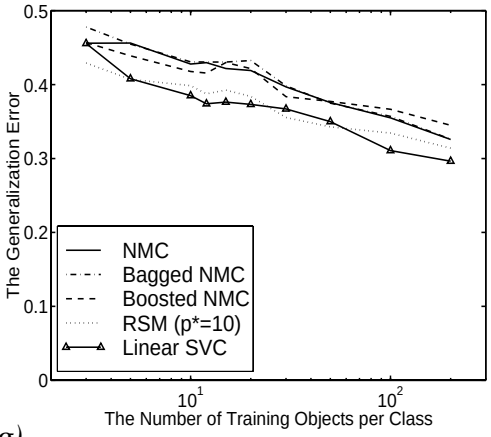
30-dimensional wdbc data



f)

Fig. 7.4. The performance of The RSM, bagging and boosting (B=250) for the NMC.

24-dimensional german data



g)

Fig. 7.4. The performance of the RSM, bagging and boosting (B=250) for the NMC.

7.3.3 The Effect of the Redundancy in the Data Feature Space

When studying the RSM for DT's, the conclusion was made [47] that the RSM is expected to be useful when there is a certain redundancy in the dataset, especially in the collection of features. It was shown [49] that the RSM performs relatively better when a discrimination power is distributed evenly over many features. Let us now consider how the redundancy presented in the data feature space affects the performance of the RSM and bagging applied to the PFLD (both the RSM and bagging are beneficial for the PFLD).

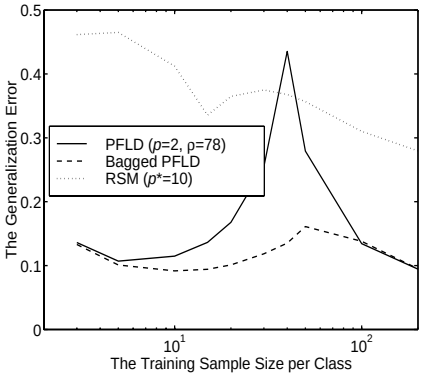
In order to study the effect of redundancy in the data feature set on the performance of bagging and the RSM, we have modified the considered data sets by artificially increasing their redundancy in the data feature set. In order to investigate whether redundancy representation in the data feature space affects the performance of the combining techniques, we modify the original data sets in two different ways: 1) just adding ρ completely redundant noise features distributed normally $N(0,10^{-4})$ to the original p-dimensional data set (by this, we obtain p+ρ-dimensional data sets with many completely redundant noise features, where the discrimination power is condensed in the subspace described by the original p-dimensional data set), 2) additionally rotating the data set (by using a Hadamard matrix [38]) in the whole (p+ρ)–dimensional space after injecting ρ completely redundant noise features to the original data set (by this, the discrimination power is spread over all p+ρ features).

In order to reduce computational costs in bagging and the RSM, we consider ensembles consisting of 100 classifiers. The size p^* of random subspaces used in the RSM is 10. All figures show the averaged results obtained over 50 repetitions on independent training sets. The standard deviations of the mean generalization errors for the single and combined PFLD's

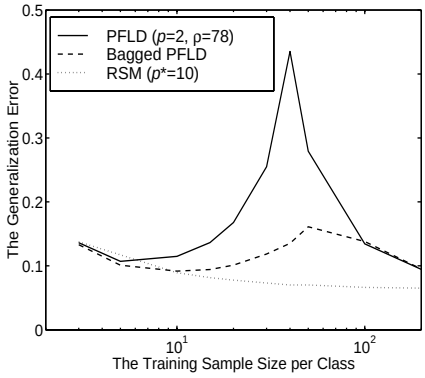
the discrimination power is condensed in p features (data have ρ completely redundant noise features)

the discrimination power is spread over all p+ρ features (rotated data)

Gaussian correlated data

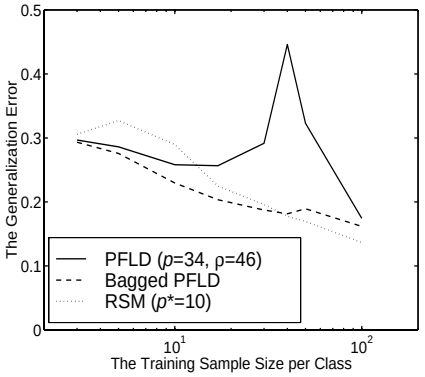


a)

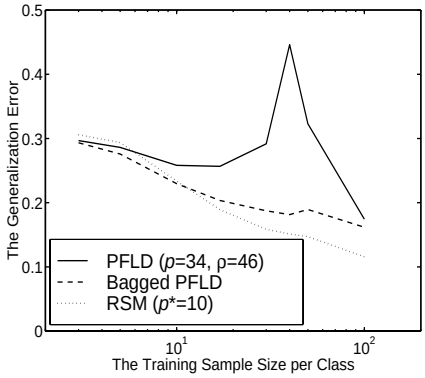


b)

ionosphere data

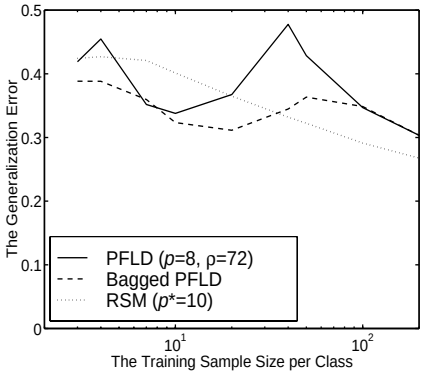


c)

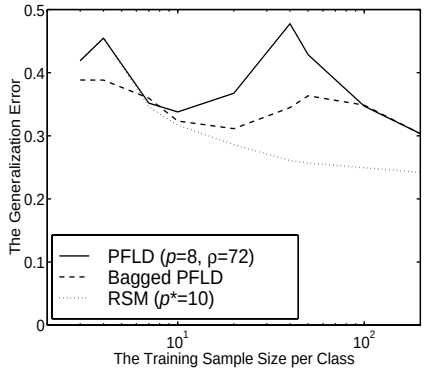


d)

pima-diabetes data



e)



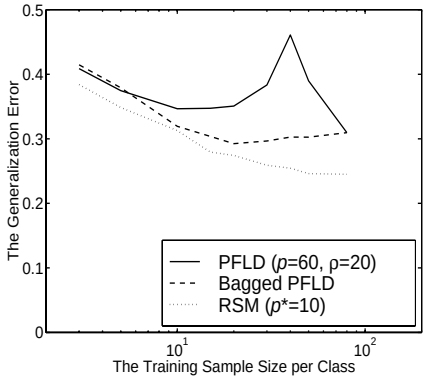
f)

Fig. 7.5. Learning curves of the single PFLD, bagging and the RSM (p=10) for the 80-dimensional data sets (p+ρ=80). Left plots show the case when discrimination power is condensed in p features (data have ρ completely redundant noise features). Right plots show the case when discrimination power is spread over all features (after adding ρ redundant features to the p-dimensional original data set, the data are rotated in all p+ρ=80 directions).*

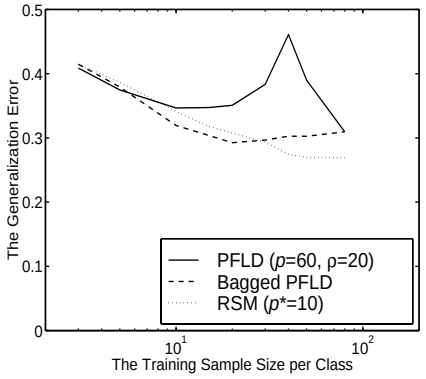
the discrimination power is condensed in p features (data have ρ completely redundant noise features)

the discrimination power is spread over all p+ρ features (rotated data)

sonar data

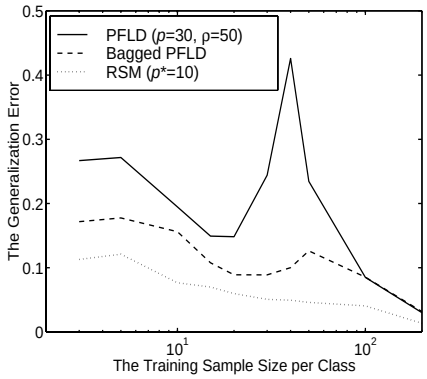


g)

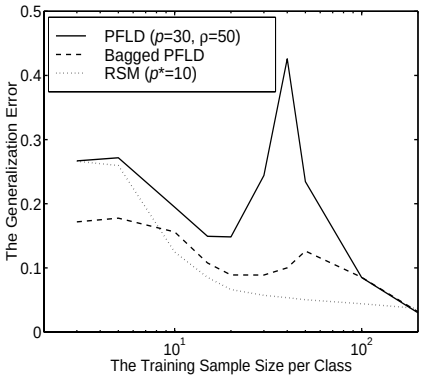


h)

wdbc data

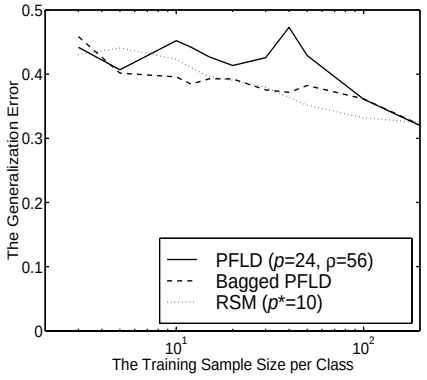


i)

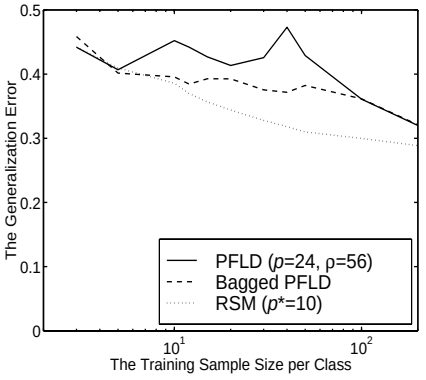


j)

german data



k)



l)

Fig. 7.5. Learning curves of the single PFLD, bagging and the RSM (p=10) for the 80-dimensional data sets (p+ρ=80). Left plots show the case when discrimination power is condensed in p features (data have ρ completely redundant noise features). Right plots show the case when discrimination power is spread over all features (after adding ρ redundant features to the p-dimensional original data set, the data are rotated in all p+ρ=80 directions).*

are around 0.01 for each data set.

Figure 7.5 shows learning curves of the single PFLD, bagging and the RSM for the 80-dimensional data sets (p+ρ=80). Figure 7.6 represents the generalization errors versus the number of redundant noise features ρ when the training sample size is fixed to 40 objects per class (80 training objects in total). Left plots in Fig. 7.5 and Fig. 7.6 show the case when the classification ability is concentrated in p features. Right plots represent the case when the discrimination power is spread over all p+ρ features. Figures show that both combining techniques are useful in highly redundant feature spaces. The RSM performs relatively better when the discrimination power is spread over many features than when it is condensed in few features (this coincides with results obtained by Ho for DT's [49]). When all data features are informative, the increasing redundancy in the data feature set does not affect the performance of the RSM. This can be explained as follows. In order to construct good classifiers in random subspaces, it is important that each subspace would contain as much as possible useful information. This could be achieved only when information is "uniformly" spread over all features (it is especially important when small random subspaces are used in the RSM). If useful information is condensed in few features, many random subspaces will be "empty" of useful information, the classifiers constructed in them will be bad and may worsen the combined decision.

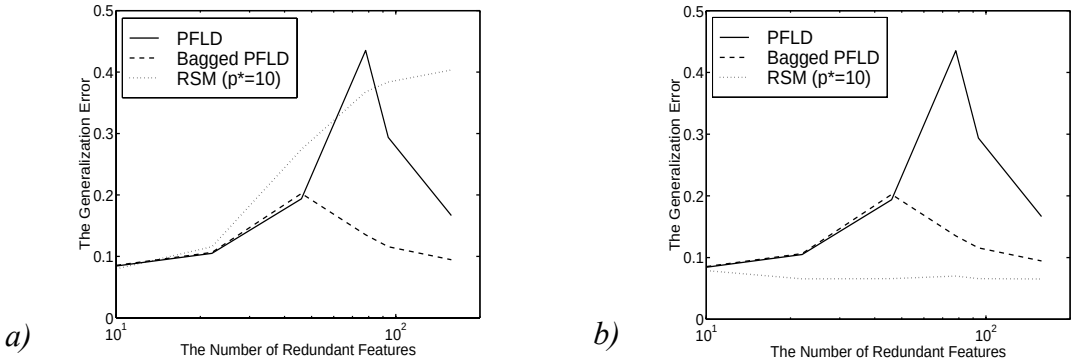
In contrast to the RSM, the performance of bagging is affected neither by the redundancy representation in the data feature set, nor by the increasing feature redundancy (it is affected by the data dimensionality referred to the training sample size). It happens because all features are kept in bagging when training objects are sampled. Rotation does not change the informativity of the bootstrap replicate of the training set. Thus, bagging may perform better than the RSM for the highly redundant feature spaces when the discrimination power is condensed in few features and the training sample size is small. However, the RSM outperforms bagging when the discrimination power is distributed over all data features.

As well, besides the redundancy in the data feature set, other factors (e.g. the class overlap, the data distribution etc.) affect the performance of the PFLD, bagging and the RSM. Usually many factors act simultaneously, assisting and counteracting each other. Sometimes, some factors may have a stronger influence than the redundancy in the data feature set. For instance, the sonar data set represents time signals where the order of features jointly with their values are very important. When rotating this data set enriched by redundant noise features, some important information is lost. By this reason, the RSM performs worse on the rotated sonar data set than when no rotation is performed (see Fig. 7.5g,h and Fig. 7.6g,h). Another example, where rotation does not improve the performance of the RSM, is the modified wdbc data set (see Fig. 7.5i,j and Fig. 7.6i,j). All features of the original data set are strongly correlated and it has the largest Mahalanobis distance (d=14.652) among all considered data sets.

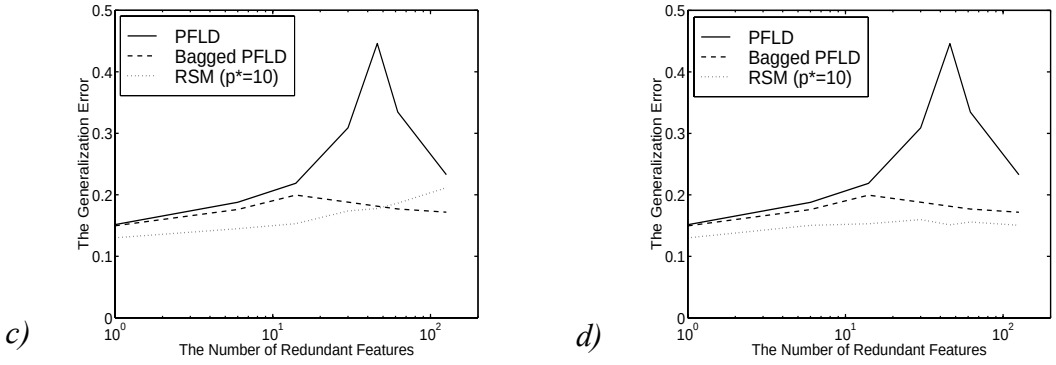
the discrimination power is condensed in p features (data have ρ completely redundant noise features)

the discrimination power is spread over all p+ρ features (rotated data)

Gaussian correlated data



ionosphere data



pima-diabetes data

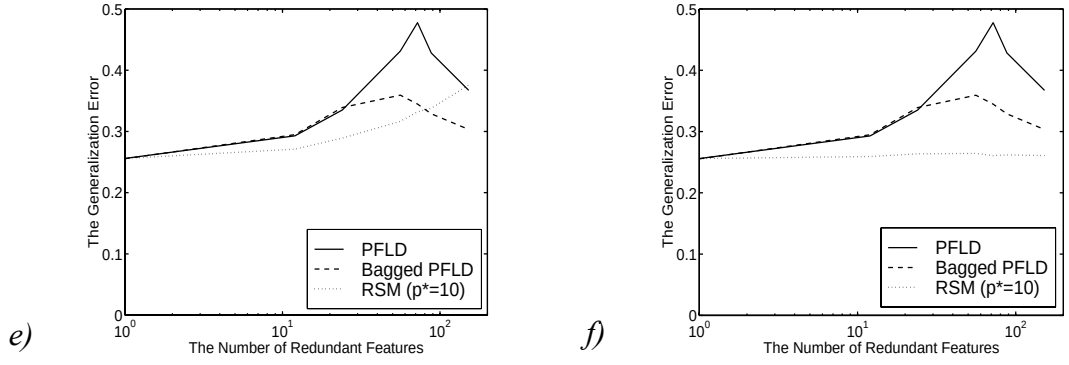
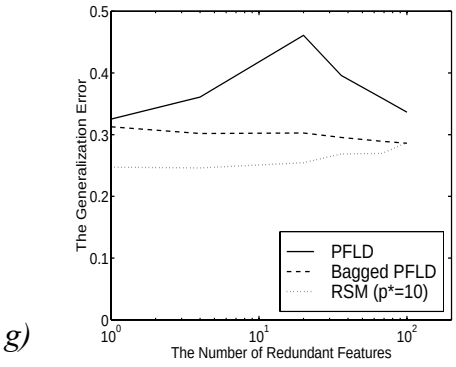


Fig. 7.6. The generalization error of the single PFLD, bagging and the RSM (p=10) versus the number of redundant noise features ρ added to the original p-dimensional data sets. Left plots show the case when the discrimination power is condensed in p features (data have ρ completely redundant noise features). Right plots represent the case when the discrimination power is spread over p+ρ features (after adding ρ redundant features to p-dimensional data set, the data are rotated in all p+ρ directions). In total, 80 training objects are used for the single classifier and the combining techniques.*

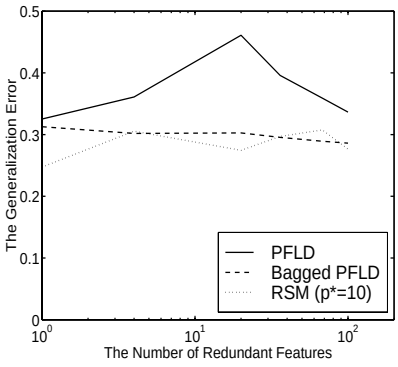
the discrimination power is condensed in p features (data have ρ completely redundant noise features)

the discrimination power is spread over all p+ρ features (rotated data)

sonar data

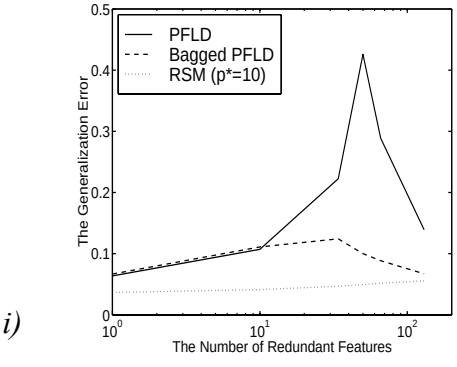


g)

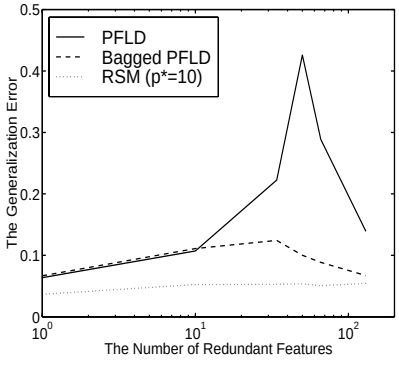


h)

wdbc data

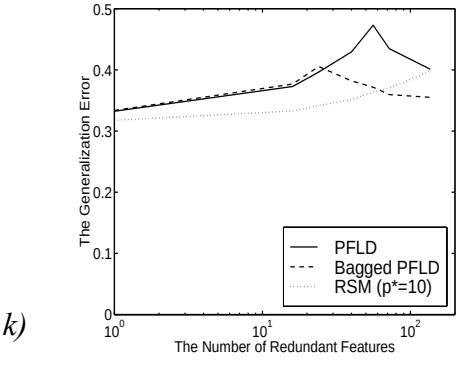


i)

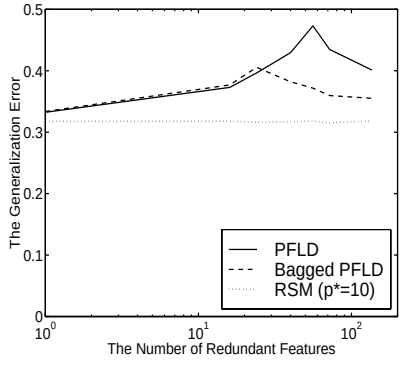


j)

german data



k)



l)

Fig. 7.6. The generalization error of the single PFLD, bagging and the RSM (p=10) versus the number of redundant noise features ρ added to the original p-dimensional data sets. Left plots show the case when the discrimination power is condensed in p features (data have ρ completely redundant noise features). Right plots represent the case when the discrimination power is spread over p+ρ features (after adding ρ redundant features to p-dimensional data set, the data are rotated in all p+ρ directions). In total, 80 training objects are used for the single classifier and the combining techniques.*

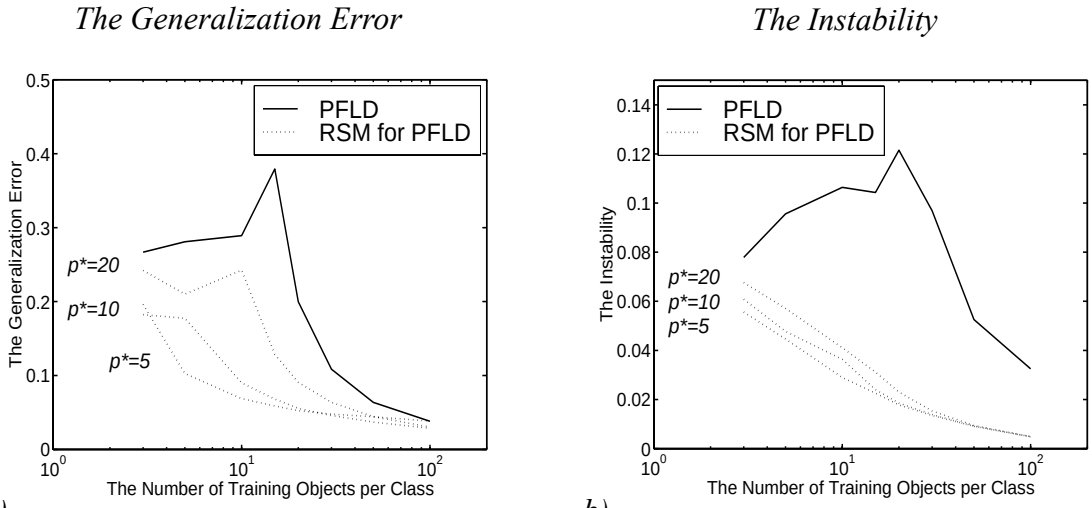


Fig. 7.7. The performance (plot a) and the instability (plot b) of the PFLD and of the RSM applied to the PFLD (B=50) for the 30-dimensional wdbc data set.

7.3.4 The RSM Is a Stabilizing Technique

The performance and the instability of linear classifiers depend on the training sample size. Usually, the classifiers are more stable and perform better when the training sample size increases. In the RSM, by constructing classifiers in the random subspaces, one actually increases the training sample size. By this, one obtains more stable classifiers than the original classifier constructed in the complete feature space. This implies, that the final classifier obtained by the RSM should be also more stable than the original classifier.

In order to demonstrate that the RSM stabilizes the original classifier, we consider the example of the PFLD constructed on the 30-dimensional wdbc data set. We should note that, in this example, we use fewer classifiers (50 instead of 250) in the ensemble obtained by the RSM, in order to decrease high computational costs associated with the instability (see Section 1.5 in Chapter 1). The generalization errors of the original PFLD and of the final classifiers obtained by the RSM (using p^*-dimensional random subspaces) are presented in Fig. 7.7a. The instabilities are presented in Fig. 7.7b. One can clearly see in Fig. 7.7 that indeed *the RSM is a stabilizing technique*.

7.4 Conclusions

Summarizing simulation results presented in the previous section, we can conclude that *the random subspace method may be useful in LDA*. Its efficiency depends on the training sample size, on the choice of the base classifier and on the level of redundancy in the data feature set.

The RSM may be useful for weak linear classifiers obtained on *small and critical training sample sizes*, because when constructing classifiers in random subspaces, one relatively increases the number of training objects. By this, one may improve the performance of linear classifiers which often suffer from the curse of dimensionality.

The RSM has a *shifting effect on the learning curve* of the final classifier with respect to the learning curve of the original classifier *in the direction of smaller training sample sizes* (i.e., the final classifier obtained in the RSM performs similar to the original classifier constructed on larger training sets). Therefore, *the RSM is useful for classifiers having decreasing learning curves*. Examples of such classifiers are the FLD and the PFLD (which is equivalent to the FLD for training sample sizes larger than the data dimensionality).

The RSM is a stabilizing technique since the classifiers obtained in random subspaces are more stable than the original classifier obtained in the complete feature space.

The usefulness of the RSM depends on the level of redundancy in the data feature set and on the way this redundancy is presented. When applied to the Pseudo Fisher linear classifier, *the RSM performs relatively better when the classification ability* (the discrimination power and also the redundancy) *is spread over many features* (i.e., for the data sets having many informative features) *than when the classification ability is condensed in few features* (i.e., for the data sets with many completely redundant noise features). *When the discrimination power is spread over all features, the RSM is resistant to the increasing redundancy in the data feature set*.

For linear classifiers with decreasing learning curves, *the RSM often outperforms bagging and boosting*.

Unlike the RSM, *the performance of bagging*, when applied to the PFLD, *depends neither on the redundancy representation nor on the level of redundancy in the data feature set* (it depends on the data dimensionality related to the training sample size). Therefore, *bagging may perform better than the RSM for the highly redundant feature spaces where the discrimination power is condensed in few features.* However, *when the discrimination power is spread over all features, the RSM outperforms bagging.*

Chapter 8

The Role of Subclasses in Machine Diagnostics

8.1 Introduction

In machine diagnostics one is interested in obtaining a full representation of all possible states of machine functioning: the normal behaviour and the abnormal behaviour -both for a wide range of loads, speeds, environmental conditions and in time (for different ages of the machine). However, in practice it is impossible to collect learning data from all states of machine functioning. Training data sets are usually not representative for the conditions under which an automatic diagnosis system has to operate: the probability density function for machine states and environmental conditions is badly defined. In order to get a well defined dataset for analyzing machine operating modes (subclasses), one has to fill gaps in the domain. It also becomes important to evaluate the significance of missing data for machine diagnostics. It is reasonable to assume that some subclasses may overlap. If missing subclasses overlap with the collected data, their absence will not significantly alter the quality of machine diagnostics. However, some missing subclasses may have unique information that is not presented in the collected data. Consequently, their absence will affect machine diagnostics.

It is also logical to assume that the data, describing machine functioning for gradually varying conditions (e.g., increasing or decreasing the running speed or the machine load), will also change slowly in the data feature space, creating a trajectory of samples. Clustering the samples on the trajectory, one obtains a sequence of subclasses. It might be of interest to study the cost of missing data in this sequence of subclasses for machine diagnostics, and also the possibility to regenerate missing data by noise injection. Other methods (unsupervised clustering and using temporal information) to recover lost information (limited to contextual) have been considered by Turney [127]. Gaussian noise is usually used in order to enrich available data. Here, Gaussian distributed objects are generated around each data object (see Section 1.6.2 in Chapter 1). However, when data samples are located in a subspace of the feature space where they are described, Gaussian noise injection can deteriorate the real data distribution. In this case the *k*-Nearest Neighbours (*k*-NN) directed noise injection becomes preferable [120] (see also Chapter 3) as it takes into account the distribution of available objects. By that, the new data objects are generated within borders of the distribution of existing objects, filling in missing links between observed objects.

In this chapter, which is based on our work [123], we investigate two problems: the role of subclasses in machine diagnostics and the effectiveness of noise injection in regenerating missing subclasses. We perform our study on the data obtained from a water-pump, which we

describe in Section 8.2. In Section 8.3 we consider several situations, where some machine operating modes are available for learning and other ones are missing. We study the cost of missing data in these situations for machine diagnostics. This study is carried out for 2-class and 4-class problems. To evaluate the quality of machine diagnostics, the Parzen Window Classifier [74] and the Karhunen-Loeve's Linear Classifier [35] are used. When studying the effectiveness of noise injection in regenerating missing subclasses in the data, we compare two kinds of noise injection, 2-Nearest Neighbours (2-NN) directed noise injection (NI) and Gaussian noise injection (GNI). The conclusions of our study are summarized in Section 8.4.

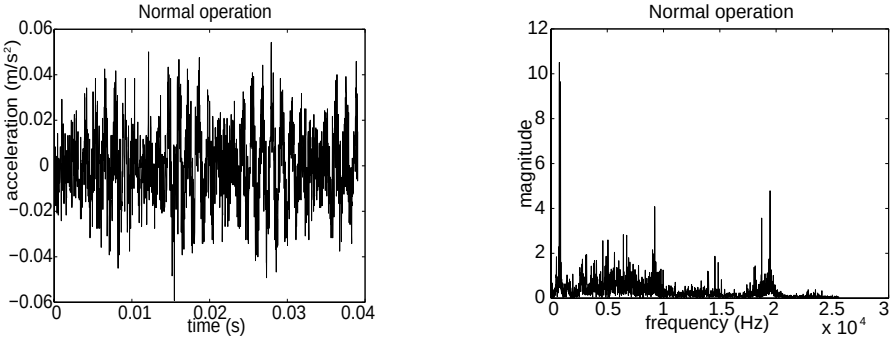
8.2 Pump Data

The data set has been obtained during the research carried out by Alexander Ypma under the project "Machine Diagnostics by Neural Networks" of the Dutch Technology Foundation (STW). The aim of the project was to develop a general, flexible machine monitoring and diagnosis system that can be used for predictive maintenance of mechanical equipment. In order to develop such a system, a real data set of machine vibration patterns representing normal and abnormal behaviour of machine functioning under different conditions was required. In order to collect the data, a one-channel impeller submersible pump (see Fig. 8.1) was maintained in the basement of the Laboratory of Applied Physics of Delft University of Technology. The pump was lowered in a concrete basin until it was completely under water. The vibration was measured with acceleration underwater sensors. More details about the experimental setup may be found in [137].

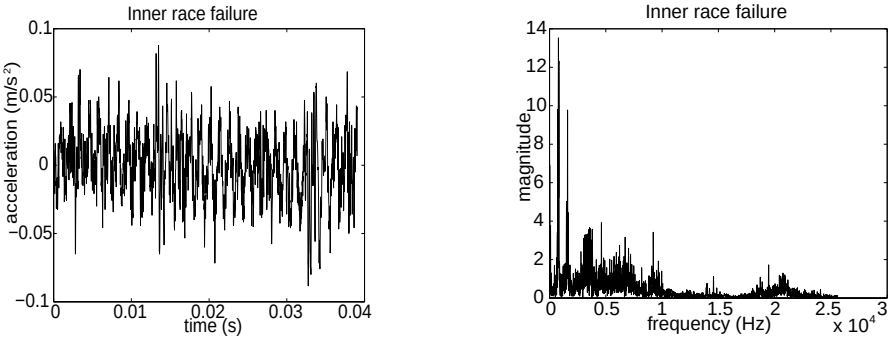
We use the data set which consists of the measurements obtained from four kinds of water-pump operating states: normal behaviour (NB), a bearing fault (BF) (a fault in the outer ring of the uppermost single bearing), an imbalance failure (IF) and a loose foundation failure

Fig. 8.1. The picture of the water-pump from which the data are collected.

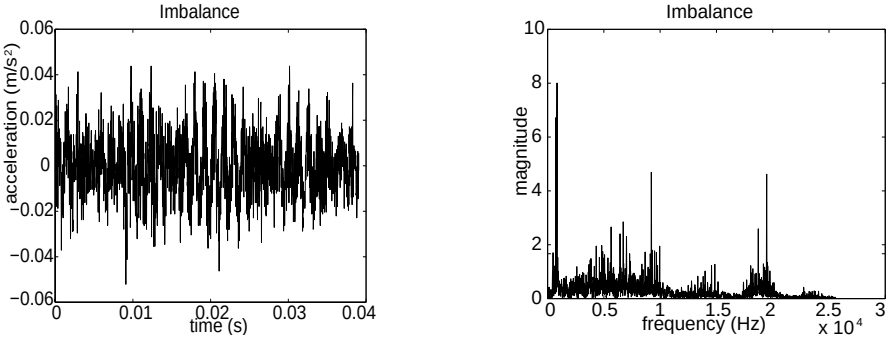
Normal Behaviour



Bearing Fault



Imbalance Failure



Loose Foundation Failure

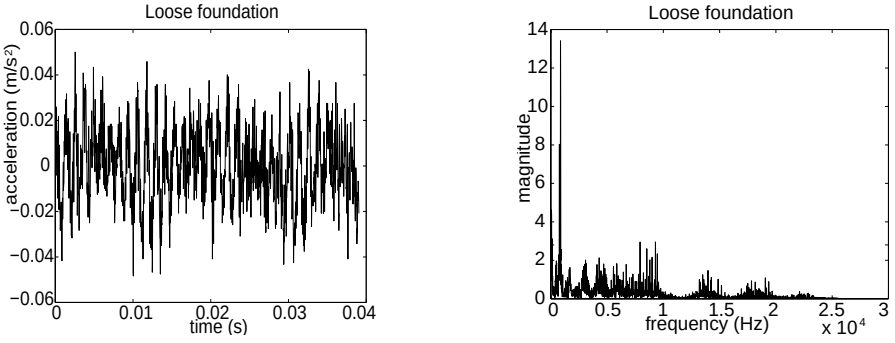


Fig. 8.2. The examples of obtained time series of the pump vibration patterns for four different operating states of the water-pump.

(LFF), where the running speed (46, 48, 50, 52 and 54 Hz) and the machine load (25, 29 and 33 KW) are varied. The examples of obtained time series of the pump vibration patterns are shown in Fig. 8.2 for four different operating states of the water-pump. In order to "normalize" raw signals, the mean is subtracted from each time signal. Then, for the obtained time series of the pump vibration patterns, which have a length of 4096 points, we determined the coefficients of an order 128 autoregressive model. The 128 coefficients of this model are used as the features describing the pump vibration patterns. For each operating mode 15 128-dimensional vectors are obtained. Then the data are combined either in 4 classes (normal behaviour, a bearing fault, an imbalance failure and a loose foundation failure) consisting of 225 observations each, or in 2 classes (normal behaviour and abnormal behaviour). In the latter case, the normal behaviour class consisted of 225 observations and the abnormal behaviour class consisted of 675 observations representing all three failures: bearing, imbalance and loose foundation. The class posterior probabilities are set according to the number of observations in the class.

The data dimensionality p=128 is large, compared to the number of available objects per subclass. In order to reduce the data dimensionality, we applied principal component analysis (PCA) to the combined data set consisting of all data classes. It shows (see Fig. 8.3) that 99 percent of total information (the total variance) is kept by the first 64 principal components. The analysis also reveals that the data are clustered and located in a subspace of the feature domain (see Fig. 8.4). Therefore, it is possible to reduce the data dimensionality. We decided to keep the 64 features with the largest variance.

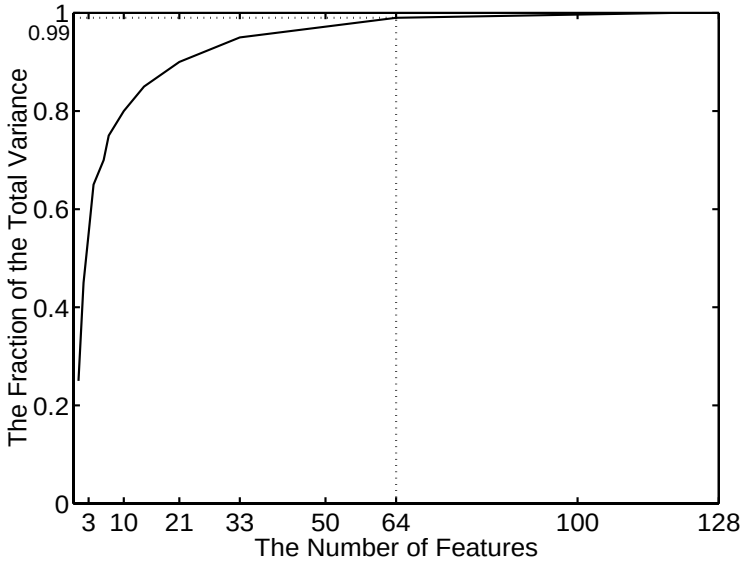


Fig. 8.3. The fraction of the total variance versus the number of features for the pump data.

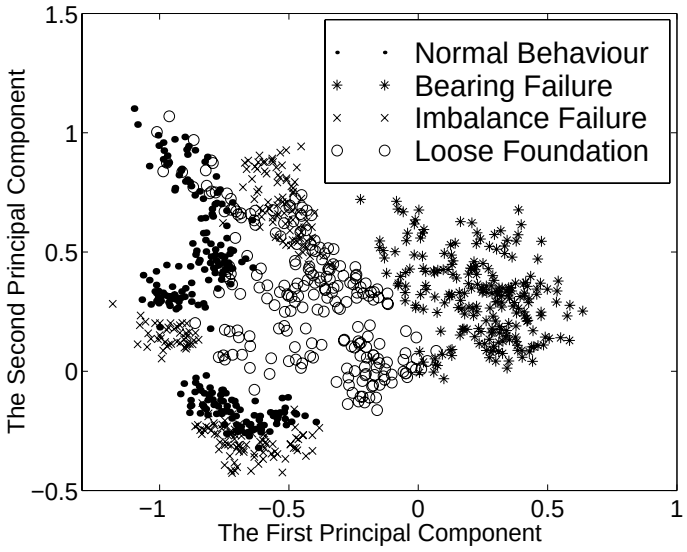


Fig. 8.4. The scatter plot of the first two principal components of the autoregressive model coefficients for four operating states.

8.3 Removing Data Gaps by Noise Injection

In order to study the importance of subclasses in machine diagnostics, we consider 15 different machine functioning modes (for 5 varied machine speeds and for 3 varied machine loads, see Fig. 8.5) presented by our data set described above. We contemplate several situations where we assume that some of the subclasses are available for learning and other ones are missing. Firstly, we consider the performance of the classifier in the worst case when it is trained on available subclasses and tested on missing ones. The ideal case is when the classifier is trained and tested on all subclasses. By comparing the performance of the classifier in the worst and ideal cases we can evaluate the importance of missing subclasses for machine diagnostics.

When studying the second problem - the usefulness of noise injection in regenerating missing data gaps (subclasses), we try to enrich incomplete data (the worst case) by noise injection. We train the classifier on the enriched training data set and test it on missing subclasses. Comparing the performance of the classifier trained on the enriched training data with the performance of the classifier trained on the complete data set (the ideal case), we can judge the effectiveness of noise injection.

This section is organized in the following way. First, in Section 8.3.1 we introduce the used noise injection models. Then we describe the experimental setup in Section 8.3.2. The results of our simulation study are presented and discussed in Section 8.3.3.

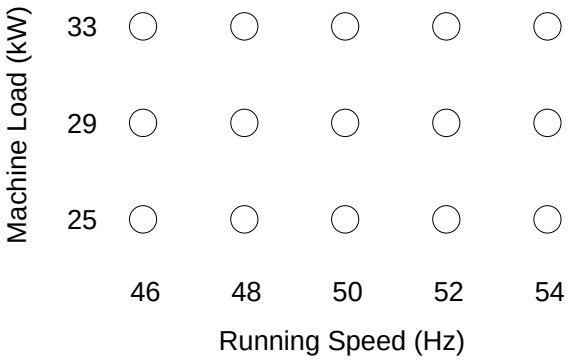
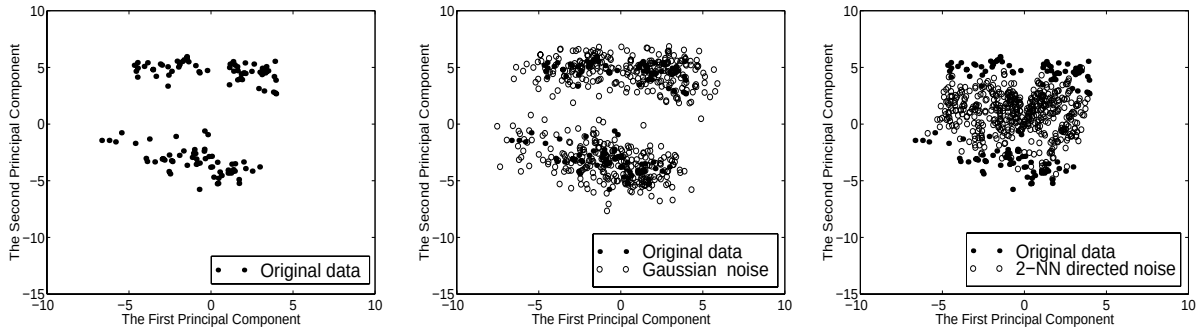


Fig. 8.5. The considered subclasses which represent 15 different operating states for 3 varied loads (25, 29 and 33 KW) and 5 varied speeds (46, 48, 50, 52 and 54 Hz).

8.3.1 Noise Injection Models

Studying the usefulness of noise injection in reconstructing missing subclasses, we consider two kinds of noise injection: Gaussian noise injection (see Section 3.3 in Chapter 3) and 2-NN directed noise injection (see Section 3.4 in Chapter 3). When injecting Gaussian noise to the training data, each training vector is replaced by the Gaussian distributed "cloud" of noisy vectors (see Fig. 8.6b). However, when the intrinsic data dimensionality is small as compared to the number of features describing the data, the Gaussian noise injection may deteriorate the real data distribution (see Chapter 3). In these circumstances, the *k*-NN directed noise injection becomes preferable.

The *2-NN directed noise injection* is applied in the following way. At first, all available training objects are clustered by the k-means procedure [45] inside each class κ. We used four clusters. Then, for the randomly chosen training object $\boldsymbol{X}_j^{(i)}$ ($i = 1, \ldots, \kappa$; $j = 1, \ldots, N$), its two nearest objects, $\boldsymbol{Q}_{j1}^{(i)}$ and $\boldsymbol{Q}_{j2}^{(i)}$, are found, which belong to another cluster than $\boldsymbol{X}_j^{(i)}$. Noise is generated only in positive directions of the nearest neighbours (see Fig. 8.6c). It is defined as $\boldsymbol{Z}_{jr}^{(i)} = \frac{1}{2}\sum_{k=1}^{2} \xi_k(\boldsymbol{Q}_{jk}^{(i)} - \boldsymbol{X}_j^{(i)})$ ($r = 1, \ldots, R$), where the components of ξ_k are uniformly distributed $U(0, 1)$, R is the desired number of noise injections. We generated 100 noisy vectors $\boldsymbol{Z}_{jr}^{(i)}$ per class (*NR*=100) for 2-NN directed noise injection, as well as for Gaussian noise injection. When applying the Gaussian noise injection, we optimized the noise variance λ_{NOISE} (see Section 3.3 in Chapter 3) on the training data set. From several values of λ_{NOISE} we chose as the optimal one the value which gives the smallest apparent error (the classification error on the training set).



a) *b)* *c)*

Fig. 8.6. The example of the original data (plot a), the original data enriched by Gaussian noise injection (plot b) and the original data enriched by 2-NN directed noise injection (plot c).

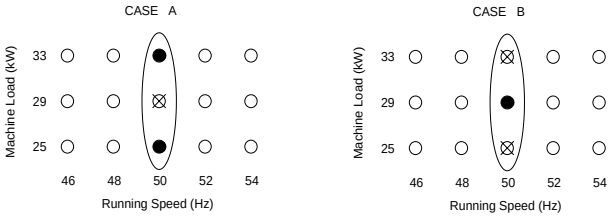
8.3.2 Experimental Setup

In order to study the importance of subclasses in machine diagnostics, let us consider a number of situations (see Fig. 8.7), where some subclasses (black ones) are selected for learning a classifier and other subclasses (crossed out) are used for testing. We intend to compare the quality of machine diagnostics on complete data (the ideal case, when both black and crossed out subclasses are used for learning and testing the classifier), on incomplete data (the worst case, when black subclasses are used for learning, and crossed out subclasses are used for testing) and on incomplete data enriched by noise injection (when learning is performed on black subclasses enriched by noise injection, and testing is done on crossed out subclasses). In order to evaluate the quality of machine diagnostics, we use the Parzen Window Classifier (PWC) (see Section 3.2) with the window width λ=0.1 and the Karhunen-Loeve's Linear Classifier (KLLC) (see Section 1.3), which constructs a linear discriminant in the subspace of the 8 features with the largest eigenvalues. For the ideal case, training sets are chosen randomly from all considered subclasses (black and crossed out) and the rest of the data is used as the test set. For all cases, experiments are repeated 25 times on independent randomly chosen training sets. The mean classification errors obtained on the test set and their standard deviations (in brackets) are presented in Tables 8.1-8.4.

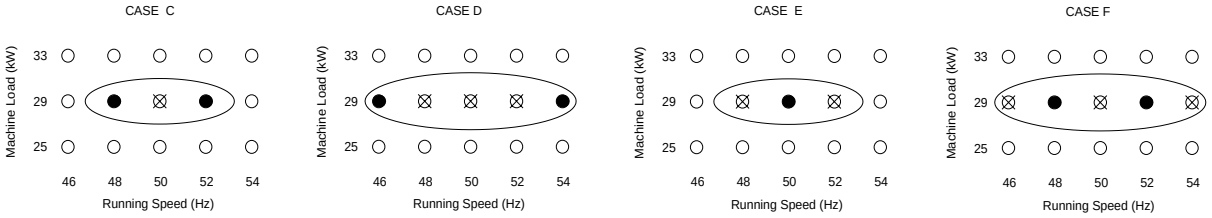
8.3.3 Results

When studying the role of subclasses and the effectiveness of noise injection in machine diagnostics, we consider four different situations: interpolation (applying the classifier to inside area) and extrapolation (applying the classifier to outside area) of machine loads (cases A, B), interpolation and extrapolation of machine speeds (cases C-F), interpolation of both

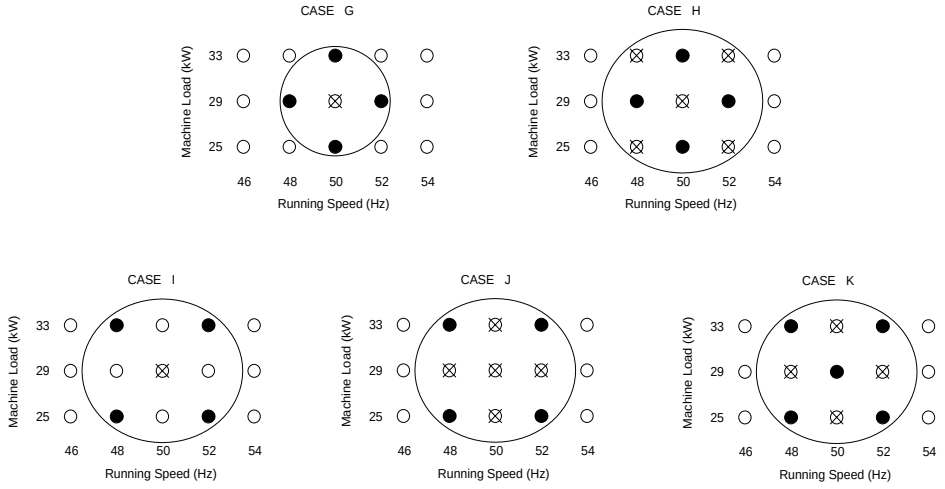
Interpolating and Extrapolating Loads



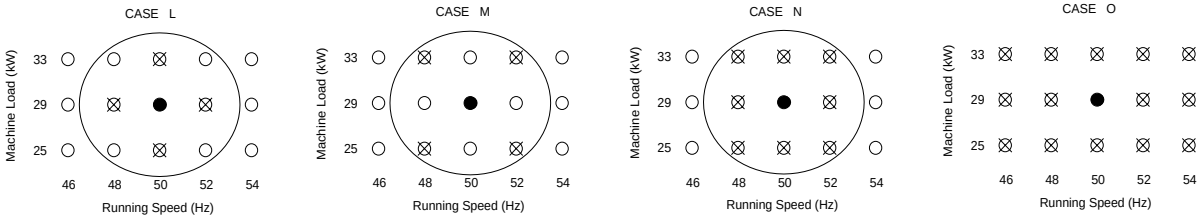
Interpolating and Extrapolating Speeds



Interpolating from Bound Subclasses



Extrapolating from a Typical Operating Mode



Increasing "extrapolation"

Fig. 8.7. The cases with incomplete data, where black subclasses are selected for learning, crossed out subclasses are missing and used only for testing, white subclasses are not considered.

machine loads and speeds from border subclasses (cases G-K), and extrapolation of machine speeds and loads from a typical operating mode (cases L-O). By interpolation we mean that the classification results are applied to the inside area of border subclasses. Extrapolation means that we apply the classification results to the outside area. The averaged performances of the PWC and the KLLC are presented in Tables 8.1-8.4.

Interpolating and extrapolating machine loads (see Table 8.1).

The results of the simulation study show that missing machine loads do not cause problems in machine diagnostics. It turns out that the subclasses representing different machine loads (for the same speed) are usually distributed nearby each other and partly overlapping (see Fig. 8.8a). Therefore, it is easy to interpolate and even extrapolate the loads.

Table 8.1. The mean classification error and its standard deviation (in brackets) of the PWC and the KLLC for the 4-class and the 2-class problem with 10 training objects per class, when interpolating and extrapolating machine loads.

case	ideal	worst	2-NN NI	GNI
4-class problem				
PWC				
A	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
B	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
KLLC				
A	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
B	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
2-class problem				
PWC				
A	0.0010 (0.0010)	0.0040 (0.0030)	0.005 (0.004)	0.0040 (0.0020)
B	0.0005 (0.0003)	0.0003 (0.0003)	0.009 (0.001)	0.0003 (0.0003)
KLLC				
A	0.002 (0.001)	0.003 (0.002)	0.003 (0.002)	0.001 (0.001)
B	0.005 (0.003)	0.003 (0.001)	0.003 (0.002)	0.003 (0.001)

Interpolating and extrapolating machine speeds (see Table 8.2).

The obtained results show that classification with missing running speeds (cases C-F) is much more difficult than with missing machine loads. This happens because the subclasses representing different running speeds (for the same load) may sometimes be far from each other. They also may be distributed in the feature space inconsistently to the values of the running speed (see Fig. 8.8b). For this reason, even interpolation of running speeds (cases C

and D) seems to be a real problem. One can see, that 2-NN directed noise injection can improve the situation and sometimes solve the problem (e.g., case C for 4-class problem), while Gaussian noise injection is everywhere useless. With respect to 2-class problems, 2-NN directed noise injection is effective for both the interpolation and the extrapolation of speeds, when using the PWC. For the KLLC, noise injection is less useful than for the PWC (completely useless for the 2-class problem), because linear classifiers are less flexible in discriminating surface (it is always a plane) than non-parametric classifiers.

Table 8.2. The mean classification error and its standard deviation (in brackets) of the PWC and the KLLC for the 4-class and the 2-class problem with 10 training objects per class, when interpolating and extrapolating machine speeds.

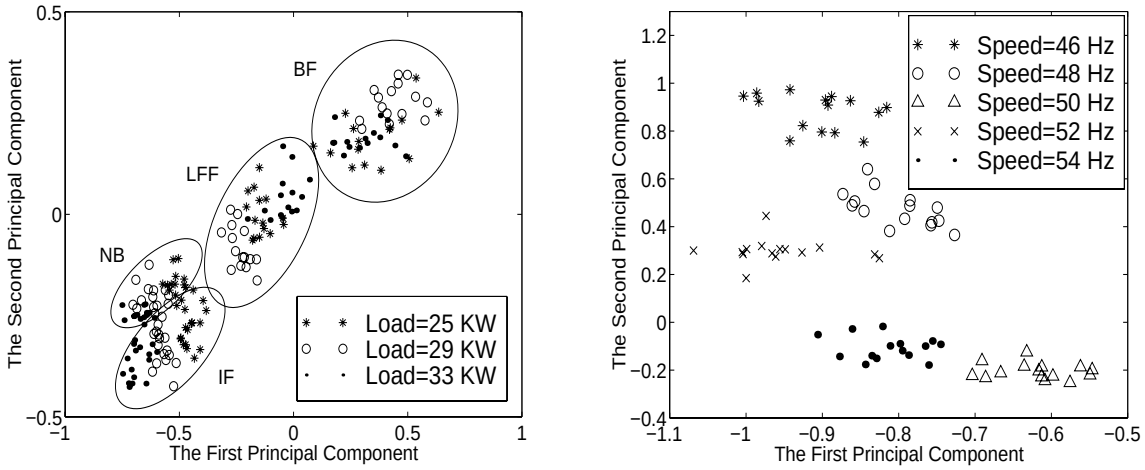
case	ideal	worst	2-NN NI	GNI
4-class problem				
PWC				
C	0.033 (0.013)	0.271 (0.009)	0.087 (0.014)	0.273 (0.010)
D	0.076 (0.007)	0.059 (0.006)	0.151 (0.008)	0.166 (0.006)
E	0.021 (0.008)	0.129 (0.002)	0.138 (0.004)	0.129 (0.002)
F	0.076 (0.007)	0.303 (0.005)	0.252 (0.006)	0.306 (0.006)
KLLC				
C	0.033 (0.070)	0.191 (0.022)	0.092 (0.02)	0.205 (0.024)
D	0.166 (0.011)	0.189 (0.009)	0.170 (0.007)	0.188 (0.009)
E	0.034 (0.007)	0.139 (0.006)	0.138 (0.007)	0.142 (0.008)
F	0.150 (0.010)	0.277 (0.008)	0.228 (0.009)	0.284 (0.010)
2-class problem				
PWC				
C	0.107 (0.011)	0.220 (0.013)	0.143 (0.017)	0.221 (0.013)
D	0.173 (0.010)	0.167 (0.009)	0.141 (0.008)	0.169 (0.009)
E	0.093 (0.013)	0.112 (0.012)	0.060 (0.010)	0.112 (0.012)
F	0.182 (0.010)	0.241 (0.009)	0.168 (0.012)	0.240 (0.009)
KLLC				
C	0.096 (0.011)	0.111 (0.018)	0.101 (0.019)	0.109 (0.020)
D	0.188 (0.010)	0.181 (0.005)	0.176 (0.007)	0.188 (0.004)
E	0.07 (0.011)	0.105 (0.012)	0.11 (0.017)	0.115 (0.016)
F	0.181 (0.013)	0.164 (0.010)	0.151 (0.010)	0.163 (0.012)

Table 8.3. The mean classification error and its standard deviation (in brackets) of the PWC and the KLLC for the 4-class and the 2-class problem with 25 training objects per class, when interpolating both machine loads and speeds from bound subclasses.

case	ideal	worst	2-NN NI	GNI
4-class problem				
PWC				
G	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
H	0.0008 (0.0)	0.007 (0.002)	0.012 (0.003)	0.008 (0.002)
I	0.0012 (0.000)	0.264 (0.002)	0.135 (0.010)	0.263 (0.003)
J	0.0015 (0.000)	0.178 (0.001)	0.089 (0.005)	0.176 (0.002)
K	0.0014 (0.001)	0.002 (0.001)	0.008 (0.002)	0.003 (0.001)
KLLC				
G	0.008 (0.003)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
H	0.022 (0.002)	0.040 (0.003)	0.030 (0.002)	0.045 (0.004)
I	0.027 (0.003)	0.119 (0.015)	0.092 (0.014)	0.127 (0.017)
J	0.025 (0.001)	0.096 (0.007)	0.087 (0.006)	0.104 (0.008)
K	0.025 (0.002)	0.041 (0.003)	0.031 (0.003)	0.042 (0.003)
2-class problem				
PWC				
G	0.029 (0.007)	0.003 (0.0016)	0.021 (0.011)	0.002 (0.0014)
H	0.027 (0.009)	0.023 (0.008)	0.040 (0.010)	0.023 (0.008)
I	0.021 (0.007)	0.215 (0.004)	0.163 (0.016)	0.213 (0.015)
J	0.022 (0.006)	0.118 (0.002)	0.106 (0.007)	0.118 (0.003)
K	0.009 (0.004)	0.034 (0.010)	0.044 (0.010)	0.033 (0.010)
KLLC				
G	0.03 (0.006)	0.013 (0.009)	0.011 (0.010)	0.019 (0.013)
H	0.053 (0.009)	0.038 (0.005)	0.059 (0.010)	0.045 (0.008)
I	0.065 (0.009)	0.052 (0.014)	0.090 (0.020)	0.062 (0.015)
J	0.046 (0.006)	0.078 (0.009)	0.096 (0.013)	0.088 (0.009)
K	0.045 (0.006)	0.064 (0.009)	0.063 (0.009)	0.066 (0.009)

Interpolating from border subclasses (see Table 8.3).

These examples (see cases G-K in Fig. 8.7) show that having only the border subclasses is not always enough to interpolate missing data. One can see that the subclasses representing different machine speeds are much more important than the subclasses representing different machine loads. Therefore, to interpolate (and extrapolate) machine loads (cases G, H) from



a) *b)*

Fig. 8.8. a) The scatter plot of different loads for four operating states with the running speed 50Hz and b) the scatter plot of different running speeds for normal behaviour with the machine load 29kW in the first two principal components domain.

border subclasses is much easier than interpolate machine speeds (cases I, J). In case K, compared to case J, we use an additional subclass (speed mode) for learning. It remarkably improves machine diagnostics performed by the PWC. As enough different speeds are presented in the data, one can interpolate the missing subclasses well. One can also observe that 2-NN directed noise injection is helpful when recovering missing machine speeds (cases I, J), while Gaussian noise injection is completely useless.

Extrapolating from a typical operating mode (see Table 8.4).

Let us now consider the situation when only one typical operating mode is available for learning and all other operating modes (subclasses) are missing (cases L-O in Fig. 8.7). Table 8.4 shows the extrapolation results for the PWC and the KLLC applied to 2-class and 4-class problems. Cases L, M and N again demonstrate that it is easier to compensate for missing machine loads than for missing machine speeds. Case O shows an interesting result (for the 2-class problem), which demonstrates that even one properly chosen subclass (typical running speed and load) allows us to successfully extrapolate a large number of missing subclasses. Concerning two different types of noise injection, one can see that 2-NN noise injection was very helpful only for the PWC applied to the 2-class problem in regenerating missing subclasses (see Fig. 8.9).

We should also point out that the performance of the classifiers (Tables 8.1-8.4) for the 2-class problem was somewhat worse than for the 4-class problem. This happens due to the fact that classes representing abnormal situations do not overlap, but the normal behaviour class

Table 8.4. The mean classification error and its standard deviation (in brackets) of the PWC and the KLLC for the 4-class and the 2-class problem with 10 training objects per class, when extrapolating both machine loads and speeds from a typical operating mode.

case	ideal	worst	2-NN NI	GNI
4-class problem				
PWC				
L	0.032 (0.007)	0.065 (0.001)	0.069 (0.002)	0.065 (0.001)
M	0.019 (0.005)	0.144 (0.001)	0.148 (0.004)	0.145 (0.001)
N	0.009 (0.001)	0.104 (0.001)	0.109 (0.003)	0.105 (0.001)
O	0.091 (0.008)	0.182 (0.001)	0.177 (0.001)	0.182 (0.001)
KLLC				
L	0.041 (0.004)	0.070 (0.003)	0.069 (0.003)	0.071 (0.004)
M	0.052 (0.007)	0.141 (0.005)	0.144 (0.007)	0.150 (0.009)
N	0.040 (0.004)	0.105 (0.004)	0.106 (0.005)	0.110 (0.006)
O	0.166 (0.007)	0.194 (0.003)	0.194 (0.003)	0.197 (0.004)
2-class problem				
PWC				
L	0.113 (0.016)	0.056 (0.006)	0.034 (0.005)	0.056 (0.006)
M	0.081 (0.016)	0.083 (0.011)	0.072 (0.007)	0.084 (0.011)
N	0.089 (0.014)	0.070 (0.008)	0.053 (0.006)	0.070 (0.008)
O	0.166 (0.012)	0.141 (0.008)	0.085 (0.004)	0.141 (0.008)
KLLC				
L	0.089 (0.014)	0.054 (0.006)	0.056 (0.009)	0.059 (0.008)
M	0.075 (0.013)	0.109 (0.008)	0.114 (0.012)	0.111 (0.010)
N	0.090 (0.012)	0.081 (0.007)	0.085 (0.010)	0.085 (0.009)
O	0.171 (0.009)	0.153 (0.006)	0.157 (0.008)	0.154 (0.007)

has overlaps with other classes (see Fig. 8.4). Therefore, combining abnormal situations in one class has led to the deterioration of machine diagnostics. Additionally, for the 2-class problem, we apply the same classifier as for the 4-class problem. More complex classifiers could be more beneficial for the 2-class problem by taking into account subclasses.

Comparing the performance of the PWC and the KLLC (Tables 8.1-8.4), one can see that sometimes the KLLC performs poorly for the ideal situation. However, in some cases (e.g., cases C, F, I and J), it gives relatively better results than the PWC in the worst situation. Noise injection is not very useful for the KLLC.

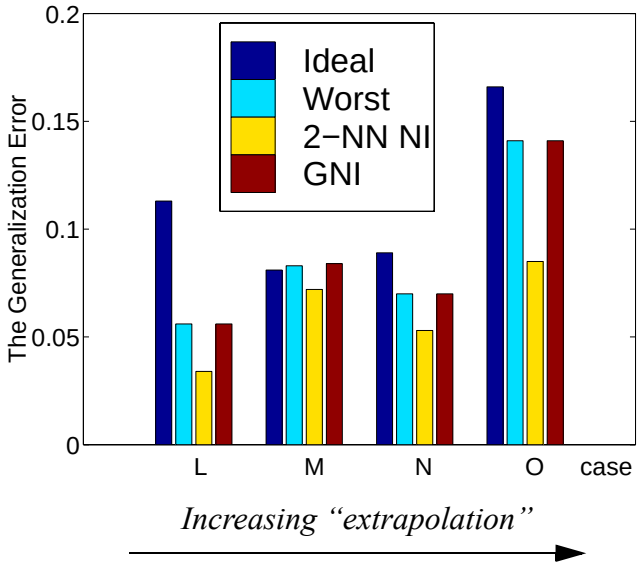


Fig. 8.9. The classification error of the PWC for two-class problem (with 10 training objects per class) when extrapolating machine loads and speeds from a typical operating mode. The complexity of the task increases from the case L, via the cases M and N, to the case O.

8.4 Conclusions

When performing machine diagnostics, it is impossible to have all possible operating modes for learning. Usually some operating modes are missing. Therefore, it is important to understand which operating modes are important to have in the training set for successful machine diagnostics, and which operating modes can be reconstructed from the available modes by noise injection. The simulation study performed on the coefficients of the autoregressive model constructed from the water-pump vibration patterns has shown that running speeds are much more important than machine loads in water-pump diagnostics. Consequently, the regeneration of missing operating modes between machine loads is easier than between running speeds. In order to reconstruct missing subclasses in the data, 2-NN directed noise injection can be used. It can reasonably improve the performance of the PWC, when it is used for machine diagnostics.

Chapter 9

Conclusions

The main goal of this thesis has been to study *alternative possibilities to improve the performance of weak classifiers*. The classifier may perform poorly when it is constructed on small training sample sizes as compared to the data dimensionality. In this case, one faces the small sample size problem since the training data set is not large enough to estimate properly the parameters of the classification rule. These estimates are biased. Further, the classifier obtained may have a large variance when it is constructed on different training sets of the same size. As a result, it is unstable. In order to evaluate the instability of classifiers, an instability measure is suggested (Chapter 1). Usually the stability and the performance of classifiers are correlated. Stable classifiers perform better than less stable ones. In addition, the performance and the instability of classifiers depend on the training sample size. Therefore, *the training sample size plays an important role in our study.*

In order to stabilize the classifier and, by this, to improve its performance, regularization techniques can be used. In this thesis we study several of them: noise injection to the training objects (Chapters 2, 3 and 8), noise injection in the feature space (Chapter 4), ridge estimate of the covariance matrix (in the FLD) (Chapter 2) and the weight decay (in perceptron training) (Chapter 2).

One may also, however, improve the performance of the weak classifier without directly stabilizing the discriminant function. In this approach, instead of a single decision of one weak classifier, one considers a combined decision of an ensemble of many weak classifiers. If classifiers in the ensemble are diverse, such a combined decision may be superior to the decision of a single classifier. Recently, a number of combining techniques (e.g., bagging, boosting and the Random Subspace Method (RSM)) were designed for and successfully applied to decision trees. In this thesis, we apply bagging (Chapter 5), boosting (Chapter 6) and the RSM (Chapter 7) to linear classifiers and study their effectiveness and stabilizing effect on these classifiers.

The **general conclusions** of the thesis, carried out from the performed theoretical and simulation studies are:

1. The training sample size defines the instability and the performance of linear classifiers (Chapter 1).

2. Regularization techniques are most effective for small and critical training sample sizes when linear classifiers are most unstable (Chapter 1). They stabilize the discriminant function.

3. Under certain conditions, different regularization techniques (ridge estimate of the covariance matrix in the FLD, weight decay in linear single layer perceptron training and noise

injection to the training objects) are equivalent (Chapter 2). The optimal value of the regularization parameter depends on the training sample size, on the data dimensionality and on the data distribution.

4. When data sets have a small intrinsic dimensionality as compared to the dimensionality of the feature space in which they are described (i.e. the data are located in a subspace), the directions of noise injection become very important (Chapter 3). Under these conditions, we suggest using the *k*-NN directed noise injection instead of Gaussian noise injection in order to prevent the original data distribution from distortion. We demonstrate the effectiveness and the advantage of the *k*-NN directed noise injection in a machine diagnostics problem (Chapter 8), when it helps to cover missing data gaps between the measurements obtained from a water-pump.

5. In order to avoid a high peak in the generalization error of the PFLD constructed on critical training sample sizes, one may either increase the number of training objects by noise injection, decrease the data dimensionality by feature selection or remove training objects. However, it is also possible to add redundant features. Adding redundant noise features dramatically improves the performance of the PFLD constructed on critical training sample sizes (Chapter 4). It is similar to other regularization techniques - the ridge estimate of the covariance matrix and noise injection to the training objects.

6. Combining techniques - bagging, boosting and the RSM - may be useful in LDA: bagging for critical training sample sizes (Chapter 5), boosting for large training sample sizes (Chapter 6) and the RSM for small and critical training sample sizes (Chapter 7).

7. Bagging and boosting are not stabilizing techniques as they relatively decrease the training sample size (Chapters 5 and 6). In bagging, the training set is reduced by using bootstrap replicates of the training set. In boosting, one decreases the training sample size by assigning large weights to objects on the border between data classes and small weights to remote objects. In addition, boosting is useful only for large training sample sizes when classifiers are stable.

8. The RSM is a stabilizing technique as the training sample size is relatively increased by using random subspaces (Chapter 7).

9. Bagging, boosting and the RSM in LDA are useful in different situations (see User Guide in Table 9.1). Their usefulness is strongly affected by the choice of the base classifier and by the training sample size. Bagging is useful for weak and unstable classifiers with a non-decreasing learning curve (the NMC and the PFLD, both constructed on critical training sample sizes) (Chapter 5). Boosting is beneficial only for weak, simple classifiers, with a non-decreasing learning curve, constructed on large training sample sizes (the NMC) (Chapter 6). The RSM is advantageous for weak and unstable classifiers, that have a decreasing learning curve and are constructed on small and critical training sample sizes (Chapter 7).

10. The usefulness of the RSM depends on the level of redundancy in the data feature set and on the way this redundancy is presented (Chapter 7). The RSM performs relatively better when the classification ability (the discrimination power and also the redundancy) is spread over many features (i.e., for the data sets having many informative features) than when the classification ability is condensed in few features (i.e., for the data sets with many completely redundant noise features). When the discrimination power is spread over all features, the RSM is resistant to the increasing redundancy in the data feature set.

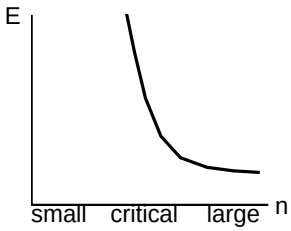
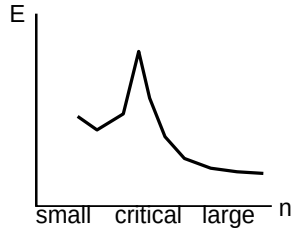
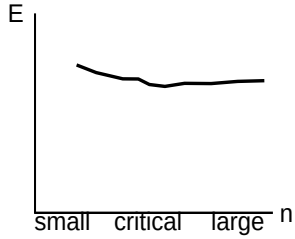
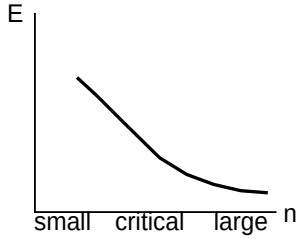
11. The performance of bagging depends neither on the redundancy representation nor on the level of redundancy in the data feature set (it depends on the data dimensionality related to the training sample size). Therefore, bagging may perform better than the RSM for the highly redundant feature spaces where the discrimination power is condensed in few features. However, when the discrimination power is spread over all features, the RSM outperforms bagging (see Chapter 7).

12. The performance of the combining techniques may depend on the choice of the combining rule. Usually the weighted majority voting is a good choice for combining base classifiers in bagging, boosting and the RSM (Chapters 6 and 7). This combining rule, however, is time consuming since one has to perform the classification B times (B is the number of base classifiers in the ensemble) in order to compute the final decision. It may be possible to optimize the weights for the weighted majority voting rule, but we have not studied this.

A summary of the possible usefulness of regularization and combining techniques for linear classifiers is given in Table 9.1. The abbreviations and notations used in the table have the following meanings: RE is regularization by Ridge-Estimate of the covariance matrix, NI is Noise Injection to the training objects, ARF is Adding Redundant Features, RSM is the Random Subspace Method, R is the number of noise injections, n is the training sample size, p is the data dimensionality.

The training sample size, the small sample size properties of the classifier (the behaviour of its classification error as a function of the training sample size) as well as the data distribution and the level of redundancy in the data feature set (the intrinsic dimensionality) strongly affect the performance of regularization and combining techniques. Therefore, it is very important to take them into account when applying these techniques in order to improve the performance of weak classifiers. Without this, one can hardly benefit from using regularization or combining techniques in discriminant analysis.

Table 9.1. The possible usefulness of regularization and combining techniques for linear classifiers.

THE USER GUIDE			
the linear classifier	**the training sample size**		
	small ($n<p$)	**critical ($n\sim p$)**	**large ($n>3p$)**
FLD (a decreasing learning curve) 	**Regularization:** RE, NI **Combining:** RSM	**Regularization:** RE, NI **Combining:** RSM	**Regularization:** not useful **Combining:** not useful
PFLD (a learning curve with the peaking behaviour) 	**Regularization:** RE, NI ($R*n>p$) **Combining:** not useful	**Regularization:** RE, NI, ARF **Combining:** bagging, RSM	**Regularization:** not useful **Combining:** not useful
NMC (a non-decreasing learning curve) 	**Regularization:** NI **Combining:** not useful	**Regularization:** not useful **Combining:** bagging	**Regularization:** not useful **Combining:** boosting
NMC (a decreasing learning curve) 	**Regularization:** NI **Combining:** RSM	**Regularization:** NI **Combining:** RSM	**Regularization:** not useful **Combining:** not useful

References

[1] S.A. Aivazian, V.M. Buchstaber, I.S. Yenyukov and L.D. Meshalkin, *Applied Statistics*: *Classification and Reduction of Dimensionality*. Moscow: Finansy i Statistika, 1989 (in Russian).

[2] K.M. Ali and M.J. Pazzani, Error Reduction through Learning Multiple Descriptions, *Machine Learning*, vol. 24, no. 3, pp. 173-202, 1986.

[3] S. Amari, A Theory of Adaptive Pattern Classifiers, *IEEE Transactions on Electronic Computers*, vol. 16, pp. 299-307, 1967.

[4] G. An, The Effects of Adding Noise During Backpropagation Training on a Generalization Performance, *Neural Computation*, vol. 8, pp. 643-674, 1996.

[5] J. A. Anderson, Logistic Discrimination, in *Handbook of Statistics 2: Classification, Pattern Recognition and Reduction of Dimensionality*, P.R. Krishnaiah and L.N. Kanal, Eds., Amsterdam: North Holland, 1982.

[6] R. Avnimelech and N. Intrator, Boosting Regression Estimators, *Neural Computation*, vol. 11, pp. 499-520, 1999.

[7] D.M. Barsov, Optimal Ridge Estimates of the Covariance Matrix in Highdimensional Discriminant Analysis, *Theory of Probability and Applications*, no. 8, pp. 820-821, 1982.

[8] M.S. Bartlett, An Inverse Matrix Adjustment Arising in Discriminant Analysis, *Annals of Mathematical Statistics*, vol. 22, no. 1, p. 107, 1952.

[9] E. Bauer and R. Kohavi, An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting and Variants, *Machine Learning*, vol. 36, no. 1/2, pp. 105-142, 1999.

[10] C.M. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995.

[11] C.M. Bishop, Training with Noise is Equivalent to Tikhonov Regularization, *Neural Computation*, vol. 7, no. 1, pp. 108-116, 1995.

[12] C.L. Blake, E. Keogh and C.J. Merz, *UCI Repository of Machine Learning Databases* (http://www.ics.uci.edu/~mlearn/MLRepository.html). Irvine, CA: University of California, Department of Information and Computer Sciences, 1998.

[13] B. Boser, I. Guyon and V. Vapnik, A Training Algorithm for Optimal Margin Classifiers, in *Proceedings of the 5th Annual Workshop on Computational Learning Theory (COLT'92)*, vol. 5, Pittsburgh: ACM, pp. 144-152, 1992.

[14] L. Breiman, J. Freidman, R. Olshen and C. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984.

[15] L. Breiman, Bagging Predictors, *Machine Learning Journal*, vol. 24, no. 2, pp. 123-140, 1996.

[16] L. Breiman, Arcing the Edge, Technical Report 486, Statistics Department, University of California, Berkeley, 1997.

[17] L. Breiman, Arcing Classifiers, *Annals of Statistics*, vol. 26, no. 3, pp. 801-849, 1998.

[18] L. Breiman, Prediction Games and Arcing Algorithms, *Neural Computation*, vol. 11, pp. 1493-1517, 1999.

[19] C. Cortes and V. Vapnik, Support-Vector Networks, *Machine Learning*, vol. 20, pp. 273-297, 1995.

[20] H. Dai and C. MacBeth, Effects of Learning Parameters on Learning Procedure and Performance of a BPNN, *Neural Networks*, vol. 10, no. 8, pp. 1505-1521, 1997.

[21] T.G. Dietterich, An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization, *Machine Learning*, vol. 40, no. 2, pp. 139-157, 2000.

[22] H. Drucker and C. Cortes, Boosting Decision Trees, *Advances in Neural Information Processing Systems*, vol. 8, pp. 479-485, 1995.

[23] R.P.W. Duin, *On the Accuracy of Statistical Pattern Recognizers*. PhD dissertation, Delft University of Technology, 1978.

[24] R.P.W. Duin, Nearest Neighbour Interpolation for Error Estimation and Classifier Optimization, in *Proceedings of the 8th Scandinavian Conference on Image Analysis*, K.A. Hogda, B. Braathen, K. Heia, Eds., Tromso, Norway, pp. 5-6, 1993.

[25] R.P.W. Duin, Small Sample Size Generalization, in *Proceedings of the 9th Scandinavian Conference on Image Analysis*, vol. 2, G. Borgefors, Ed., Uppsala, Sweden, pp. 957-964, 1995.

[26] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*. New York: Chapman &Hall, 1993.

[27] V.A. Epanechnikov, Nonparametrical Estimate of the Multivariate Probability Density, *Probability Theory and its Apllication*, vol. 14, no. 1, Moscow: Nauka, 1969 (in Russian).

[28] R.A. Fisher, The Use of Multiple Measurements in Taxonomic Problems, *Annals of Eugenics*, vol. 7, part II, pp. 179-188, 1936.

[29] R.A. Fisher, The Precision of Discriminant Functions, *Annals of Eugenics*, vol. 10, no. 4, 1940.

[30] J.H. Friedman, Regularized Discriminant Analysis, *Journal of the American Statistical Association* (JASA), vol. 84, pp. 165-175, 1989.

[31] J. Friedman, T. Hastie and R. Tibshirani, Additive Logistic Regression: a Statistical View of Boosting, Technical Report, 1998.

[32] Y. Freund and R.E. Schapire, Experiments with a New Boosting Algorithm, in *Proceedings of the 13th International Conference Machine Learning*, pp. 148-156, 1996.

[33] Y. Freund and R.E. Schapire, Game Theory, On-line Prediction and Boosting, in *Proceedings of the 9th Annual Conference on Computational Learning Theory (COLT'96)*, pp. 325-332, 1996.

[34] Y. Freund and R.E. Schapire, A Decision-Theoretic Generalization of Online Learning and an Application to Boosting, *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119-139, 1997.

[35] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. San-Diego: Academic Press, 1990.

[36] N.P. Galatsanos and A.K. Katsaggelos, Methods for Choosing the Regularization Parameter and Estimating the Noise Variance in Image Restoration and Their Relation, *IEEE Transactions on Image Processing*, vol. 1, no. 3, pp. 322-336, 1992.

[37] P. Gallinari, S. Thiria, F. Badran and F. Fogelman-Soulie, On the Relations between Discriminant Analysis and Multi-Layer Perceptrons, *Neural Networks*, vol. 4, pp. 349-360, 1991.

[38] S.W. Golomb and L.D. Baumert, The Search for Hadamard Matrices, *American Mathematics Monthly*, vol. 70, pp. 12-17, 1963.

[39] P. Gorman and T.J. Sejnowski, Learned Classification of Sonar Targets Using a Massively Parallel Network, *IEEE Transactions on ASSP*, vol. 36, no. 7, pp. 1135-1140, 1988.

[40] K. Gouhara and Y. Uchikawa, Shape of Learning Surface in Multilayer Neural Network, *Proceedings of International Joint Conference on Neural Networks*, vol. 2, Seattle, pp. 971, 1991.

[41] Y. Grandvalet and S. Canu, Comments on "Noise Injection into Inputs in Back-Propagation Learning", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 4, pp. 678-681, 1995.

[42] Y. Grandvalet, S. Canu and S. Boucheron, Noise Injection: Theoretical Prospects, *Neural Computation*, vol. 9, no. 5, pp. 1093-1108, 1997.

[43] A. Gupta and S.M. Lam, Weight Decay Backpropagation for Noisy Data, *Neural Networks*, vol. 11, pp. 1127-1137, 1998.

[44] Y. Hamamoto, Y. Mitani and S. Tomita, On the Effect of Noise Injection in Small Training Sample Size Situations, in *Proceedings of International Conference on Neural Information Processing*, Seoul, pp. 626-628, 1994.

[45] J.A. Hartigan, *Clustering Algorithms*. New York: John Wiley & Sons, 1975.

[46] J.A. Hertz and A. Krogh, Statistical Dynamics of Learning, in *Proceedings of the Joint Conference on Artificial Neural Networks (JCANN-91)*, T. Kohonen, K. Mäkisara, O. Simula and J. Kangas, Eds., Amsterdam: Elsevier, vol. 1, pp. 125-131, 1991.

[47] T.K. Ho, The Random Subspace Method for Constructing Decision Forests, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832-844, 1998.

[48] T.K. Ho, Nearest Neighbours in Random Subspaces, in *Proceedings of the 2nd International Workshop on Statistical Techniques in Pattern Recognition (SPR'98)*, Sydney, Australia, pp. 640-648, 1998.

[49] T.K. Ho, Complexity of Classification Problems and Comparative Advantages of Combined Classifiers, in *Multiple Classifier Systems, Proceedings of the 1st International Workshop on Multiple Combining Systems (MCS'2000)*, J. Kittler, F. Roli, Eds., Lecture Notes in Computer Science, vol. 1857, Berlin: Springer-Verlag, pp. 97-106, 2000.

[50] A. Hoekstra, H. Netten and D. de Ridder, A Neural Network Applied to Spot Counting, in *Proceedings of the 2nd Annual Conference of the Advanced School for Computing and Imaging (ACSI'96)*, Lommel, Belgium, pp. 224-229, 1996.

[51] A.E. Hoerl and R.W. Kennard, Ridge-regression: Biased Estimation for Nonorthogonal Problems, *Technometrics*, vol. 12, pp. 55-67, 1970.

[52] L. Holmström and P. Koistinen, Using Additive Noise in Back-Propagation Training, *IEEE Transactions on Neural Networks*, vol. 3, no. 1, pp. 24-38, 1992.

[53] K. Hornik, M. Stinchcombe and H. White, Multilayer Feedforward Networks are Universal Approximators, *Neural Networks*, vol. 2, pp. 359-366, 1989.

[54] A.K. Jain and B. Chandrasekaran, Dimensionality and Sample Size Considerations in Pattern Recognition Practice, in *Handbook of Statistics*, vol. 2, P.R. Krishnaiah and L.N. Kanal, Eds., Amsterdam: North-Holland, pp. 835-855, 1987.

[55] C. Ji, R.R. Sanapp and D. Psaltis, Generalizing Smoothness Contstraints from Discrete Samples, *Neural Computation*, vol. 2, pp. 188-197, 1990.

[56] C. Ji and S. Ma, Combinations of Weak Classifiers, *Special Issue of Neural Networks and Pattern Recognition*, *IEEE Transactions on Neural Networks*, vol. 8, pp. 32-42, 1997.

[57] L.N. Kanal and B. Chandrasekaran, On Dimensionality and Sample Size in Statistical Pattern Classification, *Pattern Recognition*, vol. 3, pp. 238-255, 1971.

[58] A. Khotanzad and Y.H. Hong, Rotation Invariant Pattern Recognition Using Zernike Moments, in *Proceedings of the International Conference on Pattern Recognition*, Rome, Italy, pp. 326-328, 1988.

[59] J. Kittler, M. Hatef and R.P.W. Duin, Combining Classifiers, in *Proceedings of ICPR*, Vienna, Austria, pp. 897-901, 1996.

[60] J.S. Koford and G.F. Groner, The Use of an Adaptive Threshold Element to Design a Linear Optimal Pattern Classifier, *IEEE Transactions on Information Theory*, vol. 12, pp. 42-50, 1966.

[61] J.F. Kolen and J.B. Pollack, Back Propagation is Sensitive to Initial Conditions, *Advances in Neural Information Processing Systems*, vol. 3, Lippman, Moody and Touretzky, Eds., San Mateo: Morgan Kaufmann, pp. 860-866, 1991.

[62] G. Korn and T. Korn, *Handbook on Mathematics for Researchers and Engineers*. Moscow: Nauka, 1973 (in Russian).

[63] A. Krogh and J.A. Hertz, A Simple Weight Decay Can Improve Generalization, in *Advances in Neural Information Processing*, vol. 4, J.E. Moody, S.J. Hanson, and R.P. Lippmann, Eds., pp. 950-957, 1992.

[64] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard and L.D. Jackel, Back Propagation Applied to Handwritten Zip Code Recognition, *Neural Computation*, vol. 1, pp. 541-551, 1989.

[65] T.K. Leen, From Data Distributions to Regularization in Invariant Learning, *Neural Computation*, vol. 7, no. 5, pp. 974-981, 1995.

[66] K. Levenberg, A Method for the Solution of Certain Non-linear Problems in Least Squares, *Quarterly Journal of Applied Mathematics*, vol. II, no. 2, pp. 164-168, 1944.

[67] Wei-Lem Loh, On Linear Discriminant Analysis with Adaptive Ridge Classification Rules, *Journal of Multivariate Analysis*, no. 53, pp. 264-278, 1995.

[68] G.D. Magoulas, M.N. Vrahatis and G.S. Androulakis, Effective Backpropagation Training with Variable Stepsize, *Neural Networks*, vol. 10, no. 1, pp. 69-82, 1997.

[69] J. Mao and A. Jain, Regularization Techniques in Artificial Neural Networks, in *Proceedings of the World Congress on Neural Networks*, vol. IV, Portland: IEEE Computer Society Press, pp. 75-79, 1993.

[70] K. Matsuoka, Noise Injection into Inputs in Back-Propagation Learning, *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 3, pp. 436-440, 1992.

[71] H. Netten, I.T. Young, M. Prins, L.J. van Vliet, H.J. Tanke, J. Vrolijk and W. Sloos, Automation of Fluorescent Dot Counting in Cell Nuclei, in *Proceedings of the 12th International Conference on Pattern Recognition,* vol. 1, Jerusalem, Israel, pp. 84-87, 1994.

[72] G.B. Orr and K.R. Muller, Eds., *Neural Networks: Tricks of the Trade*. Berlin: Springer-Verlag, 1998.

[73] Yoh-Han Pao, *Adaptive Pattern Recognition and Neural Networks*. Reading, Massachusetts: Addison-Wesley, 1989.

[74] E. Parzen, On Estimation of a Probability Density Function and Mode, *Annals of Mathematical Statistics*, vol. 33, pp. 1065-1076, 1962.

[75] D.W. Patterson and R.L. Mattson, A Method of Finding Linear Discriminant Functions for a Class of Performance Criteria, *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 380-387, 1966.

[76] R. Peck and J. Van Ness, The Use of Shrinkage Estimators in Linear Discriminant Analysis, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 4, pp. 530-537, 1982.

[77] D. Plaut, S. Nowlan and G. Hinton, Experiments on Learning by Back-Propagation, Technical Report CMU-CS-86-126, Carnegie Mellon University, Pittsburg, PA, 1986.

[78] D. Pomerleau, ALVINN: An Autonomous Land Vehicle in a Neural Network, *Advances in Neural Information Processing Systems*, vol. 1, Touretzky, ed., San Mateo: Morgan Kaufmann, pp. 305-313, 1989.

[79] J.R. Quinlan, Bagging, Boosting, and C4.5, in *Proceedings of the 13th National Conference on Artificial intelligence*, AAAI Press and the MIT Press, pp. 725-730, 1996.

[80] J.W. Raatgever and R.P.W. Duin, On the Variable Kernel Method for Multivariate Nonparametric Density Estimation, in *Compstat 1978,* L.C.A. Corsten, J. Hermans, Eds., pp. 524-533, 1978.

[81] R.H. Randless, J.D. Brofitt, I.S. Ramberg and R.V. Hogg, Generalized Linear and Quadratic Discriminant Functions Using Robust Estimates, *Journal of American Statistical Association (JASA)*, vol. 73, no. 363, pp. 564-568, 1978.

[82] C.R. Rao, On Some Problems Arising of Discrimination with Multiple Characters, *Sankya*, vol. 9, pp. 343-365, 1949.

[83] J.S. Rao and R. Tibshirani, The Out-of-bootstrap Method for Model Averaging and Selection,Technical Report, University of Toronto, Canada, 1997.

[84] Š. Raudys, On Determining the Training Sample Size of Linear Classifier, in *Computing Systems*, vol. 28, N.G. Zagoruiko, Ed., Novosibirsk: Nauka, pp. 79-87, 1967 (in Russian).

[85] Š. Raudys, On the Problems of Sample Size in Pattern Recognition, in *Detection, Pattern Recognition and Experiment Design*: *Proceedings of the 2nd All-Union Conference on Statistical Methods in Control Theory*, V.S. Pugatchiov, Ed., Moscow: Nauka, pp. 64-67, 1970 (in Russian).

[86] Š. Raudys, On the Amount of a Priori Information in Developing a Classification Algorithm, *Izvestija Akademii Nauk SSSR, serija Tekhnicheskaja Kibernetika*, no. 4, 1972 (in Russian).

[87] Š. Raudys, The Investigation of Nonparametric Classifiers when a Sample Size is Limited, *Statistical Methods of Control*, vol. 14, Vilniuse: Institute of Physics and Mathematics Press, pp. 117-126, 1976 (in Russian).

[88] Š. Raudys and V. Pikelis, On Dimensionality, Sample Size, Classification Error and Complexity of Classification Algorithm in Pattern Recognition, *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 2, no. 3, pp. 242-252, 1980.

[89] Š. Raudys, Statistical Pattern Recognition: Small Design Sample Problems, a manuscript, Vilnius: Institute of Mathematics and Cybernetics Press, 1984 (in Russian).

[90] Š. Raudys, On the Effectiveness of Parzen Window Classifier, *Informatica*, Vilnius: MII Press, vol. 2, no. 3, pp. 434-454, 1991.

[91] Š. Raudys and A.K. Jain, Small Sample Size Effects in Statistical Pattern Recognition: Recommendations for Practitioners, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 252-264, 1991.

[92] Š. Raudys, On Shape of Pattern Error Function, Initializations and Intrinsic Dimensionality in ANN Classifier Design, *Informatica*, vol. 3, no. 3-4, Vilnius: MII Press, pp. 360-383, 1993.

[93] Š. Raudys and M. Skurichina, The Role of the Number of Training Samples and Weights Initialization of Artificial Neural Net Classifier, in *Neuroinformatics and Neurocomputers*, *Proceedings of Russian Neural Network Society and IEEE Symposium on Neuroinformatics and Neurocomputers*, Rostov-na-Donu: IEEE Press, pp. 343-353, 1992.

[94] Š. Raudys and M. Skurichina, Small Sample Properties of Ridge Estimate of the Covariance Matrix in Statistical and Neural Net Classification, in *New Trends in Probability and Statistics: Multivariate Statistics and Matrices in Statistics*, *Proceedings of the 5th Tartu Conference*, vol. 3, E.M. Tiit, T. Kollo and H. Niemi, Eds., Vilnius: TEV and Utrecht: VSP, pp. 237-245, 1994.

[95] Š. Raudys, M. Skurichina, T. Cibas and P. Gallinari, Optimal Regularization of Neural Networks and Ridge Estimates of the Covariance Matrix in Statistical Classification, *Pattern Recognition and Image Analysis: Advances in Mathematical Theory and Applications* (an International Journal of Russian Academy of Sciences), vol. 5, no. 4, pp. 633-650, 1995.

[96] Š. Raudys and R.P.W. Duin, On Expected Classification Error of the Fisher Linear Classifier with Pseudo-Inverse of the Covariance Matrix, *Pattern Recognition Letters,* vol. 19, no. 5-6, pp. 385-392, 1998.

[97] Š. Raudys, Evolution and Generalization of a Single Neuron: I. Single-layer Perceptron as Seven Statistical Classifiers, *Neural Networks*, vol. 11, no. 2, pp. 283-296, 1998.

[98] Š. Raudys, Classifier's Complexity Control While Training Multilayer Perceptrons, in *Advances in Pattern Recognition*, *Proceedings of Joint International Workshops SSPR'2000 and SPR'2000*, F.J. Ferri, J.M. Inesta, A. Amin and P. Pudil, Eds., Lecture Notes in Computer Science, vol. 1876, Berlin: Springer-Verlag, pp. 32-44, 2000.

[99] Š. Raudys, *Statistical and Neural Classifiers*, Springer, 2001.

[100] R. Reed, Pruning Algorithms - A Survey, *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740-747, 1993.

[101] R. Reed, R.J. Marks II and S. Oh, Similarities of Error Regularization, Sigmoid Gain Scaling, Target Smoothing, and Training with Jitter, *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 529-538, 1995.

[102] B.D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press, 1996.

[103] W.P.J. Rousseeuw and A.M. Leroy, Robust Regression and Outlier Detection, in *Applied Probability and Statistics*, New York: John Wiley & Sons, 1987.

[104] M. Rozenblatt, Remarks on Some Nonparametric Estimates of a Density Function, *Annals of Mathematical Statistics*, vol. 27, pp. 832-837, 1956.

[105] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning Internal Representations by Error Propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. I, D.E. Rumelhart, J.L. McClelland, Eds., Cambridge, MA: Bradford Books, pp. 318-362, 1986.

[106] Hidetsugu Sakagushi, Stochastic Dynamics and Learning Rules in Layered Neural Networks, *Progress of Theoretical Physics*, vol. 83, no. 4, pp. 693-700, 1990.

[107] R. Schapire, Y. Freund, P. Bartlett and W. Lee, Boosting the Margin: a New Explanation for the Effectiveness of Voting Methods, in *Proceedings of the 14th International Conference on Machine Learning*, pp. 322-330, 1997.

[108] R.E. Schapire and Y. Singer, Improved Boosting Algorithms Using Confidence-Rated Predictions, in *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pp. 80-91, 1998.

[109] R.E. Schapire, Theoretical Views of Boosting, in *Proceedings of the 4th European Conference on Computational Learning Theory (EuroCOLT'99)*, pp. 1-10, 1999.

[110] H. Schwenk and Y. Bengio, Training Methods for Adaptive Boosting of Neural Networks for Character Recognition, in *Advances in Neural Information Processing Systems*, vol. 10, S. Solla and M. Jordan, Eds., Cambridge, MA: MIT Press, pp. 647-653, 1998.

[111] V.I. Serdobolskij, On Minimal Probability of Misclassification in Discriminant Analysis, *Lectures of the Academy of Sciences of the USSR*, vol. 270, no. 5, pp. 1066-1070, 1983 (in Russian).

[112] J. Sietsma and R.J.F. Dow, Neural Network Pruning - Why and How, in *Proceedings of IEEE International Conference on Neural Networks*, vol. 1, pp. 325-333, 1988.

[113] J. Sjöberg and L. Ljung, Overtraining, Regularization and Searching for a Minimum in Neural Networks, Technical Report S-581 83, Department of Electrical Engineering, Linkoping University, Sweden, 1991.

[114] M. Skurichina, The Effect of the Kernel Function Form on the Quality of Nonparametric Parzen Window Classifier, *Statistical Problems of Control*, vol. 93, Vilnius: Institute of Mathematics and Cybernetics Press, pp. 167-181, 1990 (in Russian).

[115] M. Skurichina and R.P.W. Duin, Stabilizing Classifiers for Very Small Sample Sizes, in *Proceedings of the 13th International Conference on Pattern Recognition*, vol. 2, Track B: Pattern Recognition and Signal Analysis, Los Alamitos: IEEE Computer Society Press, pp. 891-896, 1996.

[116] M. Skurichina and R.P.W. Duin, Scale Dependence of Noise Injection in Perceptron Training, in *Proceedings of the 1st International Workshop Statistical Techniques in Pattern Recognition (SPR'97)*, P. Pudil, J. Novovicova, J. Grim, Eds., Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague, pp. 153-158, 1997.

[117] M. Skurichina and R.P.W. Duin, Bagging for Linear Classifiers, *Pattern Recognition*, vol. 31, no. 7, pp. 909-930, 1998.

[118] M. Skurichina and R.P.W. Duin, Regularization by Adding Redundant Features, in *Advances in Pattern Recognition*, *Proceedings of Joint International Workshop SSPR'98 and SPR'98*, A. Amin, D. Dori, P. Pudil, H. Freeman, Eds., Lecture Notes in Computer Science, vol. 1451, Berlin: Springer-Verlag, pp. 564-572, 1998.

[119] M. Skurichina and R.P.W. Duin, Regularization of Linear Classifiers by Adding Redundant Features, *Pattern Analysis and Applications*, vol. 2, no. 1, pp. 44-52, 1999.

[120] M. Skurichina, Š. Raudys and R.P.W. Duin, K-Nearest Neighbours Directed Noise Injection in Multilayer Perceptron Training, *IEEE Transactions on Neural Networks*, vol. 11, no. 2, pp. 504-511, 2000.

[121] M. Skurichina, R.P.W. Duin, Boosting in Linear Discriminant Analysis, in *Multiple Classifier Systems, Proceedings of the 1st International Workshop on Multiple Combining Systems (MCS'2000)*, J. Kittler, F. Roli, Eds., Lecture Notes in Computer Science, vol. 1857, Berlin: Springer-Verlag, pp. 190-199, 2000.

[122] M. Skurichina and R.P.W. Duin, The Role of Combining Rules in Bagging and Boosting, in *Advances in Pattern Recognition*, *Proceedings of the Joint International Workshops SSPR'2000 and SPR'2000*, F.J. Ferri, J.M. Inesta, A. Amin and P. Pudil, Eds., Lecture Notes in Computer Science, vol. 1876, Berlin: Springer-Verlag, pp. 631-640, 2000.

[123] M. Skurichina, A. Ypma and R.P.W. Duin, The Role of Subclasses in Machine Diagnostics, in *Proceedings of the 15th International Conference on Pattern Recognition (ICPR'2000)*, vol. 2: Pattern Recognition and Neural Networks, Los Alamitos: IEEE Computer Society Press, pp. 668-671, 2000.

[124] M. Skurichina and R.P.W. Duin, Bagging and the Random Subspace Method for Redundant Feature Spaces, in *Multiple Classifier Systems, Proceedings of the 2nd International Workshop on Multiple Combining Systems (MCS'2001)*, J. Kittler, F. Roli, Eds., Lecture Notes in Computer Science, vol. 2096, Berlin: Springer-Verlag, pp. 1-10, 2001.

[125] M. Skurichina and R.P.W. Duin, Bagging, Boosting and the Random Subspace Method for Linear Classifiers, submitted to *FMC Special Issue of Pattern Analysis and Applications*, 2001.

[126] R. Tibshirani, Bias, Variance and Prediction Error for Classification Rules, Technical Report, University of Toronto, Canada, 1996.

[127] P. Turney, The Management of Context-Sensitive Features: a Review of Strategies, in *Proceedings of the Workshop on Learning in Context-Sensitive Domains* (at the ICML-96), Bari, Italy, pp. 60-66, 1996.

[128] V.N. Vapnik, *Estimation of Dependencies based on Empirical Data*. New York: Springer-Verlag, 1982.

[129] V.N. Vapnik, *The Nature of Statistical Learning Theory*. Berlin: Springer-Verlag, 1995.

[130] V. Vyöniauskas, Searching for Minimum in Neural Networks, *Informatica*, vol. 5, no. 1-2, Vilnius: MII Press, pp. 241-255, 1994.

[131] A.S. Weigend, D.E. Rumelhart and B.A. Huberman, Generalization by Weight-Elimination with Application to Forecasting, in *Advances in Neural Information Processing*, vol. 3, R. Lippmann, J. Moody, and D. Touretzky, Eds., San Mateo: Morgan Kaufmann, pp. 875-882, 1991.

[132] S.M. Weiss and C.A. Kulikowski, *Computer Systems That Learn*. San Matteo: Morgan Kaufmann, 1991.

[133] N. Weymaere and J.P. Martens, On the Initialization and Optimization of Multilayer Perceptrons, *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 738-751, 1994.

[134] B. Widrow and M.E. Hoff, Adaptive Switching Circuits, *IRE WESCON Convention Record*, part 4, pp. 96-104, 1960.

[135] D.H. Wolpert and W.G. Macready, An Efficient Method to Estimate Bagging's Generalization Error, Technical Report, Santa Fe Institute, Santa Fe, 1996.

[136] F. Wyman, D.M. Young and D.W. Turner, A Comparison of Asymptotic Error Rate Expansion for the Sample Linear Discriminant Function, *Pattern Recognition*, vol. 23, pp. 775-783, 1990.

[137] A. Ypma, R. Ligteringen, E.E.E. Frietman and R.P.W. Duin, Recognition of Bearing Failures using Wavelets and Neural Networks, in *Proceedings of the 2nd UK Symposium on Applications of Time-Freqency and Time-Scale Methods*, University of Warwick, Coventry, UK, pp. 69-72, 1997.

[138] N.G. Zagoruiko, V.N. Elkina and V.S. Temirkaev, ZET - an Algorithm of Filling Gaps in Experimental Data Tables, *Computing Systems*, vol. 67, Novosibirsk: Nauka, pp. 3-28, 1976 (in Russian).

Summary

Stabilizing Weak Classifiers

Regularization and Combining Techniques in Discriminant Analysis

Marina Skurichina

In discriminant analysis one often has to face the *small training sample size problem*. This problem arises when the data feature space dimensionality is large compared with the number of available training objects. In order to construct a classifier in the data feature space, one has to estimate the classifier parameters (coefficients) from the training set. This becomes difficult, when the number of training objects is smaller than the number of the classifier parameters. Some statistical classifiers require the inverse of the covariance matrix. When the training sample size is small, the sample estimate of the covariance matrix is close to a singular matrix. The inversion of this matrix is principally impossible when the data dimensionality exceeds the number of training objects. Other classifiers, that do not require the inverse of the covariance matrix, may have difficulties as well, when the training data set is small. Often small training sets misrepresent the entire data set and are located in a subspace of the data feature space. The statistical parameters estimated on such small training sets are usually biased and may have a large variance. As a result one may obtain quite dissimilar discriminant functions when constructing the classifier on different training sample sets of the same size. Thus classifiers obtained on small training sets are unstable and are weak, i.e. having a high classification error.

One of the ways to overcome the small sample size problem is to modify a classifier in one way or another in order to avoid the inverse of an ill-conditioned covariance matrix. However, modified classifiers may also have problems related to the training sample size. For instance, the pseudo Fisher linear discriminant, which uses the pseudo inverse of the covariance matrix instead of the direct inverse, has a very bad performance for the training sample sizes that are equal to the dimensionality of the data feature space.

Another way is to stabilize classifiers mathematically, using different *regularization* techniques applied to parameters in classification rules. The examples of the regularization performed on parameters are, for instance, a *ridge estimate* of the covariance matrix in linear discriminant analysis or *weight decay* in perceptron training. However, for some classifiers to perform the analytical regularization on parameters might be impossible.

A more global approach consists in increasing the number of training objects, either by using larger training sets, or, if it is not possible, by bootstrapping the training set or by artificially generating additional objects (*noise injection*). The latter case can be considered as regularization performed on data objects. Usually, in order to regularize the training data set, Gaussian spherical noise is added to each training object.

One more way to avoid the small sample size problem consists in removing data features by some feature selection or extraction method. By this, the number of training objects is relatively increased. However, this subject is not studied in this thesis.

A completely different approach to improve the performance of weak classifiers is to use a combined decision of many weak classifiers instead of stabilizing a single weak classifier.

Both regularization approaches, applied to parameters of the classifier and to the training data set, are known as *stabilizing* techniques. *Combining techniques* (*bagging*, *boosting* and the *random subspace method*) are also often considered as stabilizing techniques. However, it is not completely clear whether the regularization techniques and the combining techniques really stabilize the classifier or just improve the performance of the classification rule. It is also of interest to understand how different regularization techniques, applied to the data and to the parameters, are related to each other, how the regularization and combining techniques perform, what controls their performance and when they are the most effective.

The main goal of this thesis is to study different regularization and combining techniques in linear discriminant analysis, their relationship and effectiveness. As the performance and the stability of linear classifiers depend on the training sample size, we consider the performance of the regularization and combining techniques in close relation to the training sample size and the small sample size properties of linear classifiers.

Our study reveals that different regularization techniques, applied to parameters of the classifier and to the training data set, have much in common. The ridge estimate of the covariance matrix in linear discriminant analysis is equivalent to weight decay in the training of linear single layer perceptrons, and they both are asymptotically equivalent to Gaussian noise injection to the training objects. We prove that an optimal value of the regularization parameter exists. This value depends on the training sample size, on the dimensionality of the data feature space and on the data distribution. It tends to zero with an increase of the training sample size. We show that the effectiveness of Gaussian noise injection depends on the variance of the noise, on the number of noise injections and on the directions in which noise is added. Based on this conclusion, we suggest using the *k nearest neighbours directed noise injection* instead of Gaussian spherical noise for the data with a low intrinsic dimensionality.

In order to avoid a poor performance of the pseudo Fisher linear discriminant when the training sample size is equal to the data dimensionality, one may add training objects, remove features and remove objects. We suggest another possibility - to regularize the pseudo Fisher linear discriminant by noise injection into the feature space by *adding redundant noise features*.

It is shown that this approach is somewhat similar (however, not equivalent) to using the ridge estimate of the covariance matrix and to noise injection to the training objects.

When studying the effectiveness and the stabilizing effect of different regularization and combining techniques, we conclude that regularization techniques are indeed stabilizing, while the combining techniques do not always stabilize the classifier, although they help to improve the performance of weak classifiers. It appears, that the performance and the effectiveness of the regularization and combining techniques are strongly affected by the choice of a linear classifier, by its instability and the behaviour of its learning curve (the dependency of its generalization error, as a function, on the training sample size), by the data distribution and by the training sample size. Therefore, different regularization and combining techniques are useful in different situations.

When applying regularization and combining techniques in order to improve the performance of weak classifiers, one has to take into account the training sample size, the small sample size properties of the classifier (its learning curve) as well as the data distribution, and the level of redundancy in the data feature set (the intrinsic dimensionality). Without this, one can hardly benefit from using regularization or combining techniques.

Samenvatting

Stabiliseren van Zwakke Classificatoren

Regularisatie- en Stabilisatietechnieken in Discriminantanalyse

Marina Skurichina

Bij het gebruik van discriminantanalyse heeft men vaak te weinig trainingsdata ter beschikking (*het probleem van beperkte steekproefgrootte van datasets*). Dit probleem treedt op wanneer de dimensionaliteit van de kenmerken groter is dan het aantal beschikbare trainingsobjecten. Om een classificator in de kenmerkruimte te construeren, moet men de parameters (coëfficiënten) van een classificator met trainingsdata schatten. Dit wordt moeilijk wanneer het aantal trainingsobjecten kleiner is dan het aantal parameters van de classificator. Sommige statistische classificatoren vereisen de inverse van de covariantiematrix. Wanneer de trainingssteekproef klein is, wordt de schatting van de covariantiematrix een singuliere matrix. Het is in principe onmogelijk deze matrix te inverteren wanneer de dimensionaliteit van de data het aantal trainingsobjecten overschrijdt. Andere classificatoren, die niet de inverse van de covariantiematrix nodig hebben, zijn ook niet zonder problemen te gebruiken wanneer er weinig trainingsdata zijn. Kleine trainingssets representeren de gehele dataset vaak slecht en liggen in een subruimte van de kenmerkruimte. De statistische parameters geschat met deze kleine trainingssets zijn gewoonlijk niet zuiver en kunnen een grote variantie hebben. Hierdoor kan men zeer verschillende discriminantfuncties verkrijgen wanneer de classificator is geconstrueerd met verschillende trainingssets van dezelfde grootte. Classificatoren verkregen uit kleine trainingssets zijn daarom onstabiel en zwak, dat wil zeggen, ze hebben een grote classificatiefout.

Eén van de manieren om het probleem van de kleine steekproefgrootte op te lossen, is de classificator op een of andere manier te modificeren om de inversie van een slecht-geconditioneerde covariantiematrix te omzeilen. Echter, de gemodificeerde classificatoren kunnen nog steeds problemen opleveren. De pseudo-Fisher lineaire discriminant bijvoorbeeld, die de pseudo-inverse van de covariantiematrix gebruikt in plaats van de directe inverse, presteert erg slecht in situaties waarbij er evenveel trainingsobjecten als datadimensies zijn.

Een andere manier is het mathematisch stabiliseren van classificatoren door verschillende *regularisatie*technieken te gebruiken voor de parameters in de classificatieregels. Voorbeelden van regularisatie toegepast op de parameters zijn een "*ridge estimate*" van de covariantiematrix voor lineaire discriminantanalyse of "*weight decay*" voor

perceptrontraining. Voor sommige classificatoren is het echter onmogelijk een analytische regularisatie van de parameters uit te voeren.

Een meer globale aanpak bestaat uit het vergroten van het aantal trainingsobjecten, door grotere trainingssets te gebruiken, of wanneer dat niet mogelijk is, door het "bootstrappen" van de trainingsset of door kunstmatig extra objecten te genereren ("*noise injection*"). Het laatste geval kan beschouwd worden als een regularisatie toegepast op de dataobjecten. Om de trainingsdata te regulariseren, wordt er gewoonlijk sferische Gaussische ruis aan ieder trainingsobject toegevoegd.

Een andere manier om het problem van kleine steekproefgrootte te omzeilen bestaat uit het verwijderen van datakenmerken, met behulp van kenmerkselectie of -extractie. Hierdoor wordt het aantal trainingsobjecten relatief groter. Dit onderwerp is echter niet in dit proefschrift onderzocht.

Een geheel andere aanpak om de prestaties van zwakke classificatoren te verbeteren, bestaat uit het combineren van de beslissingen van een groot aantal zwakke classificatoren, in plaats van het stabiliseren van één zwakke classificator.

Beide regularisatiebenaderingen, toegepast op de parameters van de classificator en op de trainingsset, staan bekend als *stabilisatietechnieken*. *Combinatietechnieken* ("*bagging*", "*boosting*" en de "*random subspace*" *methode*) worden ook vaak tot de stabilisatietechnieken gerekend. Het is echter niet helemaal duidelijk of de regularisatietechnieken en de combinatietechnieken echt de classificator stabiliseren, of dat ze enkel de prestaties van de classificatieregel verbeteren. Het is ook interessant te begrijpen hoe verschillende regularisatietechnieken, toegepast op de data en de parameters, aan elkaar gerelateerd zijn, hoe de regularisatie- en de combinatietechnieken presteren, wat hun prestaties beïnvloedt en wanneer ze het meest effectief zijn.

Het voornaamste doel van dit proefschrift is het bestuderen van verschillende regularisatie- en combinatietechnieken in lineaire discriminantanalyse, hun verband en hun effectiviteit. Omdat de prestatie en de stabiliteit van lineaire classificatieregels afhangen van de grootte van de trainingsset, beschouwen we de prestatie van de regularisatie- en de combinatietechnieken als functie van de grootte van de trainingsset en de kleine-steekproef-eigenschappen van lineaire classificatoren.

Onze studie laat zien dat verschillende regularisatietechnieken, toegepast op de parameters van de classificator en op de trainingsset, veel gemeen hebben. De "ridge estimate" van de covariantiematrix voor lineaire discriminantanalyse is equivalent aan "weight decay" voor de training van een enkele lineaire perceptron, en ze zijn beide asymptotisch equivalent met Gaussische-ruis-injectie van de verzameling trainingsobjecten. We hebben bewezen dat er een optimale waarde voor de regularisatieparameter bestaat. Deze waarde hangt af van de grootte van de trainingsset, en van de dimensionaliteit en de verdeling van de data. Ze daalt naar 0 voor oplopende groottes van de trainingsset. We laten zien dat de effectiviteit van de

Gaussische-ruis-injectie afhangt van de variantie van de ruis, van het aantal injecties en van de richtingen waarin de ruis is toegevoegd. Op grond van deze conclusies, suggereren we om de *k-naaste-nabuur-gerichte ruis-injectie* te gebruiken in plaats van Gaussische sferische ruis voor data met een lage intrinsieke dimensionaliteit.

Om slechte prestaties van de pseudo-Fisher lineaire discriminant te voorkomen wanneer de grootte van de trainingsset gelijk is aan de datadimensionaliteit, kan men trainingsobjecten toevoegen, kenmerken verwijderen of objecten verwijderen. Wij hebben nog een andere mogelijkheid onderzocht: regularisatie van de pseudo-Fisher lineaire disciminant door het injecteren van ruis in de kenmerkruimte via het *toevoegen van redundante ruis-kenmerken*. Er is aangetoond dat deze benadering gelijkenissen vertoont (hoewel hij niet equivalent is) met het gebruik van de "ridge estimate" van de covariantiematrix en met de ruis-injectie bij de trainingsobjecten.

Bij de bestudering van de effectiviteit en het stabiliserend effect van de verschillende regularisatie- en combinatietechnieken, concluderen we dat de regularisatietechnieken inderdaad stabiliseren, terwijl de combinatietechnieken de classificatoren niet altijd stabiliseren, hoewel ze wel de prestaties van de zwakke classificatoren verbeteren. Het blijkt dat de prestaties en effectiviteit van de regularisatie- en combinatietechnieken sterk worden beïnvloed door de keuze van de lineaire classificator, door zijn instabiliteit en het gedrag van zijn leercurve (de afhankelijkheid van zijn generalisatiefout van de grootte van de trainingsset), door de verdeling van de data en door de grootte van de trainingsset. Hierdoor zijn verschillende regularisatie- en combinatietechnieken nuttig in verschillende situaties.

Wanneer regularisatie- en combinatietechnieken worden toegepast om de prestaties van zwakke classificatoren te verbeteren, moet men rekening houden met zowel de grootte van de trainingsset en de kleine-steekproef-eigenschappen van de classificator (zijn leercurve) als met de verdeling van de data en de redundantie in de kenmerkset (de intrinsieke dimensionaliteit). Zonder deze voorzorgen verdwijnt vrijwel het voordeel van het gebruik van regularisatie- of combinatietechnieken.

Curriculum Vitae

Marina Skurichina was born in Vilnius, Lithuania on the 21st of May, 1967. She graduated from the 6th Vilnius Secondary School in 1984. Then she went to study Applied Mathematics at Vilnius State University where she obtained her Masters Degree in mathematics (with honour) in 1989. She made her Master thesis on differential equations under the supervision of Prof.dr. B. Kvedaras.

In 1989 she joined the Data Analysis Department at the Institute of Mathematics and Cybernetics of the Lithuanian Academy of Sciences as a postgraduate researcher. Her research involved regularization techniques for linear classifiers and neural networks. In 1992 she became a Ph.D. student at the same department and continued her work on regularization techniques under the supervision of Prof.dr. Š. Raudys. In 1995 Marina Skurichina came at the Pattern Recognition Group of Delft University of Technology as a guest research fellow. After a short brake, in 1997 she again joined the Pattern Recognition Group as a research fellow. She was involved in the STW project “Machine diagnostics by neural networks”. She was studying regularization and combining techniques in discriminant analysis under the supervision of Dr.ir. R.P.W. Duin. In 2001, she continued her work as a postdoc in the STW project “Improving endoscopic detection of lung cancer using autofluorescence spectroscopy analysed by a neural network“.