Depth- and Potential-Based Supervised Learning

Inauguraldissertation

zur

Erlangung des Doktorgrades

der

Wirtschafts- und Sozialwissenschaftlichen Fakultät

der

Universität zu Köln

2016

vorgelegt von

Oleksii Pokotylo

aus

Kiew, Ukraine

Referent:	Prof. Dr. Karl Mosler
Korreferent:	Prof. Dr. Jörg Breitung
Tag der Promotion:	17. Oktober 2016

To the memories of my Grandmother
Nina Zamkova
(1921 – 2014)

Acknowledgements

I would like to acknowledge many people for helping and supporting me during my studies.

I would like to express my sincere gratitude to my supervisor Prof. Karl Mosler for the continuous support, for his patience, motivation, inspiration, and expertise. His guidance helped me all the time to cope with technical issues of my research and his valuable comments helped to improve the presentation of my papers and this thesis. I could not imagine having a better PhD-supervisor. I show heartfelt gratitude to Prof. Tatjana Lange, who introduced me the field of pattern recognition during my internship at the Hochschule Merseburg. She taught me a lot and inspired me to continue my research in this field. Further she continuously supported me through my studies. This thesis would not have been possible without these people. I appreciate the Cologne Graduate School in Management, Economics and Social Sciences for funding my research, and personally Dr. Dagmar Weiler for taking care of all CGS students.

I am extremely grateful to my close friend and colleague Dr. Pavlo Mozharovskyi for his substantial support from my very first days in Germany and for numerous discussions and valuable comments on my papers. Together with Prof. Rainer Dyckerhoff he is also the co-author of the R-package **ddalpha** and the corresponding paper. I thank Dr. Ondrej Vencalek from the Palacky University in Olomouc, Czech Republic, for fruitful cooperation, which led to a joint paper, and for his rich feedback on the **ddalpha** package. Daniel Fischer, Dr. Jyrki Möttönen, Dr. Klaus Nordhausen and Dr. Daniel Vogel are acknowledged for the R-package **OjaNP**, and Tommi Ronkainen for his implementation of the exact algorithm for the computation of the Oja median, that I used to implement the exact bounded algorithm.

I thank all participants of the Research Seminar of the Institute of Statistics and Econometrics for active discussions, especially Prof. Jörg Breitung, Prof. Roman Liesenfeld, Prof. Oleg Badunenko and Dr. Pavel Bazovkin. I appreciate the maintainers of the CHEOPS HPC Cluster, that was intensely used for simulations in all my projects, the maintainers of The Comprehensive R Archive Network (CRAN) and also the users of the **ddalpha** package who submitted their feedback.

Finally, I express my gratitude to my parents Natalia and Oleksandr, my sister Marina and my wife Katja for providing me with unfailing support and continuous encouragement throughout my years of study. Thank you from the bottom of my heart.

Contents

Chapter 1

Introduction

Classification problems arise in many fields of application like economics, biology, medicine. Naturally, the human brain tends to classify the objects that surround us according to their properties. There exist numerous classification schemes that separate objects in quite different ways, depending on the application field and the importance that is assigned to each property. People teach each other to distinguish one class from another by showing objects from that classes and indicating the differences between them. The class membership of newly observed objects may be then determined by their properties using the previously gained knowledge. In the last decades the increased computing power allowed to automatize this process. A new field — machine learning — emerged, that *inter alia* develops algorithms for supervised learning.

The task of supervised learning is to define a data-based rule by which the new objects are assigned to one of the classes. For this a training data set is used that contains objects with known class membership. Formally we regard the objects as points in a multivariate space that is formed by their properties. It is important to determine the properties that are most relevant for the particular classification problem. Analyzing the training set, a classifier generates a separating function that determines the class relationship of newly observed objects.

Classification procedures have to determine how close an object is situated with respect to a class and how typical it is for that class. This is done by studying location, scale, and shape of the underlying distribution of the classes.

1.1 Measuring closeness to a class

The very basic notions of center regarding univariate data are the *mean*, that is the mass center, and the *median*, that is the 0.5-quantile. The median is a most robust statistic, having a breakdown point of 50%, and hence is often preferred to the mean. Closeness to a class may be defined as a measure of distance to a properly defined center, or as a measure of correspondence to the whole class, like a data depth or a density estimate. The quantile function can also be used for this purpose.

The simplest way to describe the shape of the data is to assume that it follows some known family of probability distributions with a fixed set of parameters. If this assumption is

made then it is easy to calculate the probability of future observations after the parameters have been estimated. Often such a distribution family is not known. Then one may recur to nonparametric tests of equality of probability distributions like the Kolmogorov-Smirnov test or the χ^2-test. However, parametric approaches are limited to known families of distribution functions, and thus may be not applicable to real data in general.

With multidimensional data, things become even more complicated. The median, as well as the quantile function are not directly generalisable to higher dimensions. Therefore semi- and non-parametric methods were introduced that provide distribution-freeness and are applicable to multidimensional data of any form.

The *kernel density estimator* is probably the most well known nonparametric estimator of density. Consider a data cloud $\boldsymbol{X}$ of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, and assume that the cloud is generated as an independent sample from some probability density f. Let the kernel be $K_{\boldsymbol{H}}(\boldsymbol{x}, \boldsymbol{x}_i) = |\det \boldsymbol{H}|^{-1/2} K\left(\boldsymbol{H}^{-1/2}(\boldsymbol{x} - \boldsymbol{x}_i)\right)$, $\boldsymbol{H}$ be a symmetric and positive definite bandwidth matrix, and $K : \mathbb{R}^d \to [0, \infty[$ be a spherical probability density function, $K(\boldsymbol{z}) = r(\boldsymbol{z}^T\boldsymbol{z})$, with $r : [0, \infty[\to [0, \infty[$ non-increasing and bounded. Then

$$\hat{f}_{\boldsymbol{X}}(\boldsymbol{x}) \quad = \quad \frac{1}{n}\sum_{i=1}^{n} K_{\boldsymbol{H}}(\boldsymbol{x}, \boldsymbol{x}_i) \tag{1.1}$$

is a kernel estimator of the density at a point $\boldsymbol{x} \in \mathbb{R}^d$ with respect to the data cloud $\boldsymbol{X}$. In particular, the Gaussian function $K(\boldsymbol{z}) = (2\pi)^{-d/2} \exp\left(-\frac{1}{2}\boldsymbol{z}^T\boldsymbol{z}\right)$ is widely employed for K. The method is tuned with the bandwidth matrix $\boldsymbol{H}$. The kernel density estimator is applied for classification in Chapter 2.

In 1975 John W. Tukey, in his work on mathematics and the picturing of data, proposed a novel way of data description, which evolved into a measure of multivariate centrality named *data depth*. For a data sample, this statistical function determines centrality, or representativeness of an arbitrary point in the data, and thus allows for multivariate ordering of data regarding their centrality. More formally, given a data cloud $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ in $\mathbb{R}^d$, for a point $\boldsymbol{z}$ of the same space, a depth function $D(\boldsymbol{z}|\boldsymbol{X})$ measures how *close* $\boldsymbol{z}$ is located to some (implicitly defined) *center* of $\boldsymbol{X}$. Different concepts of closeness between a point $\boldsymbol{z}$ and a data cloud $\boldsymbol{X}$ suggest a diversity of possibilities to define such a function and a center as its maximizer. Naturally, each depth notion concentrates on a certain aspect of $\boldsymbol{X}$, and thus possesses various theoretical and computational properties. The concept of a depth function can be formalized by stating postulates it should satisfy. Following Dyckerhoff (2004) and Mosler (2013), a *depth function* is a function $D(\boldsymbol{z}|\boldsymbol{X}) : \mathbb{R}^d \mapsto [0, 1]$ that is *affine invariant*, *zero at infinity*, *monotone on rays* from the deepest point $\boldsymbol{z}^*$ and *upper semicontinuous*. The properties ensure that the upper level sets $D_\alpha(\boldsymbol{X}) = \{\boldsymbol{z} \in \mathbb{R}^d : D(\boldsymbol{z}|\boldsymbol{X}) \geq \alpha\}$ are bounded, closed and star-shaped around $\boldsymbol{z}^*$.

Data depth is reversely related to outlyingness. In a natural way, it involves a notion of center that is any point attaining the highest depth value in $\boldsymbol{X}$; the center is not necessarily unique. It provides a center-outward ordering of the data, which also allows to define multi-

variate quantiles as the upper level sets of the depth function, called depth-trimmed regions. Being intrinsically nonparametric, a depth function captures the geometrical features of given data in an affine-invariant way. By that, it appears to be useful for description of data's location, scatter, and shape, allowing for multivariate inference, detection of outliers, ordering of multivariate distributions, and in particular classification, that recently became an important and rapidly developing application of the depth machinery. While the parameter-free nature of data depth ensures attractive theoretical properties of classifiers, its ability to reflect data topology provides promising predicting results on finite samples. Many depth notions have arisen during the last decades differing in properties and being suitable for various applications. Mahalanobis (Mahalanobis, 1936), halfspace (Tukey, 1975), simplicial volume (Oja, 1983), simplicial (Liu, 1990), zonoid (Koshevoy and Mosler, 1997), projection (Zuo and Serfling, 2000), spatial (Vardi and Zhang, 2000) depths can be seen as well developed and most widely employed notions of depth function. Comprehensive surveys of depth functions can be e.g. found in Zuo and Serfling (2000) and Mosler (2013). Chapter 3 reviews the concept of data depth and its fundamental properties, and gives the definitions of seven depth functions in their empirical versions.

Several notions of multivariate medians have been proposed in the literature. Like the univariate median most of the multivariate medians can be regarded as maximizers of depth functions or minimizers of outlyingness functions. Generally, a depth median is not unique but forms a convex set. Multivariate medians are surveyed by Small (1997) and Oja (2013).

Due to the fact that depth functions are related to one center, they only follow the shape of unimodal distributions. However, multimodal distributions are widely used in practice. Therefore, different concepts of local depth (Agostinelli and Romanazzi, 2011, Paindaveine and Van Bever, 2013) were introduced that generalize data depth to reveal local features of the distribution. An overview of local depths is found in Appendix at the end of the thesis.

The practical applications require efficient algorithms and fast implementations of depth functions and their approximations. Chapter 3 is devoted to the R-package **ddalpha** (Pokotylo et al., 2016) that provides an implementation for exact and approximate computation of seven most reasonable and widely applied depth notions: Mahalanobis, halfspace, zonoid, projection, spatial, simplicial and simplicial volume.

To be applicable to realistic problems, a median must be computable for dimensions $d > 2$ and at least medium sized data sets. In Chapter 5, we develop an algorithm (Mosler and Pokotylo, 2015) to calculate the exact value of the Oja median (Oja, 1983). This algorithm is faster and has lower complexity than the existing ones by Niinimaa et al. (1992) and Ronkainen et al. (2003). Our main idea is to introduce bounding hyperplanes that iteratively restrict the area where the median is searched.

1.2 Supervised learning

The task of the supervised learning is to analyze the training data with known class membership, and to infer a separating function, which can be used to assign new objects to the classes. Each object in the training set is described by a set of properties (explanatory variables) and a class label (dependent variable).

Consider the following setting for supervised classification: Given a training sample consisting of q classes $\boldsymbol{X}_1, ..., \boldsymbol{X}_q$, each containing n_i, $i = 1, ..., q$, observations in $\mathbb{R}^d$, their densities are denoted by f_i and prior probabilities by π_i. For a new observation $\boldsymbol{x}_0$, a class shall be determined to which it most probably belongs.

The Bayes classifier minimizes the probability of misclassification and has the following form:

$$class_B(\boldsymbol{x}) = \underset{i}{\operatorname{argmax}}\, \pi_i f_i(\boldsymbol{x}). \tag{1.2}$$

In practice the densities have to be estimated. A classical nonparametric approach to solve this task is by *kernel density estimates* (KDE); see e.g. Silverman (1986). In KDE classification, the density f_i is replaced by a proper kernel estimate $\hat{f}_i$ as in (1.1) and a new object is assigned to a class i at which its *estimated potential*,

$$\hat{\phi}_i(\boldsymbol{x}) = \pi_i \hat{f}_i(\boldsymbol{x}) = \frac{1}{\sum_{k=1}^{q} n_k} \sum_{j=1}^{n_i} K_{\boldsymbol{H}_i}(\boldsymbol{x}, \boldsymbol{x}_{ij}) \,, \tag{1.3}$$

is maximal. It is well known that KDE is Bayes consistent, that means, its expected error rate converges to the error rate of the Bayes rule for any generating densities. As a practical procedure, KDE depends largely on the choice of the multivariate kernel and, particularly, its bandwidth matrix. Wand and Jones (1993) demonstrate that the choice of bandwidth parameters strongly influences the finite sample behavior of the KDE-classifier. With higher-dimensional data, it is computationally infeasible to optimize a full bandwidth matrix. In Chapter 2 we extend the KDE classifier and discuss the selection of the bandwidth matrix.

The Bayes classifier is a useful benchmark in statistical classification. Usually the classifiers are designed to minimize the empirical error over a certain family of rules and are compared by their misclassification rate, i.e. the part of errors they make in the test sample. The parameters of the classifiers are also tuned by means of cross-validation, minimizing the error rate.

The requirement of correct classification of "typical" points, however, might not be met when using the Bayes classifier, especially in the case of imbalanced data, when a point that is central w.r.t. one class and rather peripheral to another may still be assigned to the larger one. In such situations the Bayes classifier (1.2) may be additionally weighted to achieve the desired misclassification rate for the minor class, but in this case its outliers are also overweighted, which leads to misclassification of the major class in their neighbourhood.

Points that are close to the center of a class are considered to be more "typical" for this class than more outlying ones. Data depth generalizes the concept of centrality and outlyingness for multivariate distributions. In Chapter 4 we suggest to weight the classification errors using

data depth, so that the misclassification of points close to the center of the data cloud is seen as a more serious mistake than the misclassification of outlying points, see Vencalek and Pokotylo (2016). This criterion can also be used to measure the performance of other classifiers and to tune their parameters by cross-validation.

The *k-Nearest Neighbors* (k-NN) algorithm is one of the most simple and widely applied classifiers. For a new object, the classifier finds k nearest neighbors and assigns the object to the class that is common for most of them. This method strongly depends on a measure of distance and the scales of the parameters, e.g. measurement units. A natural way to make the k-NN classifier affine-invariant is to standardize data points with the sample covariance matrix. A more sophisticated depth-based version of the affine-invariant k-NN was proposed in Paindaveine and Van Bever (2015). They symmetrize the data around the new object $\boldsymbol{z}$ and use a depth of the original objects in this symmetrized set to define a $\boldsymbol{z}$-outward ordering, which allows to identify the k nearest of them.

Many classification problems consider a huge set of properties. The quality of these properties is not known in general and some properties may introduce more noise than useful information to the classification rule. Then separation in the whole space may become complicated and unstable and, thus, poorly classify new observations due to overfitting. This problem is referred to as the 'curse of dimensionality'. Vapnik and Chervonenkis (1974) state that the probability of misclassification of new data is reduced either by enormously increasing the training sample, or by simplifying the separation, or by reducing the number of properties. The first two variants provide more stable classifiers, although it is hard to get a big data set in practice. By reducing the dimension of the space we focus on the most relevant properties

The α-procedure (Vasil'ev and Lange, 1998, Vasil'ev, 2003) is an iterative procedure that finds a linear solution in the given space. If no good linear solution exists in the original space it is extended with extra properties, e.g., using polynomial extension. The linear solution in the extended space leads then to a non-linear solution in the original one. The procedure iteratively synthesizes the space of features, choosing those minimizing two-dimensional empirical risk in each step. The α-procedure is more widely described in the Appendix to Chapter 3.

1.3 Dimension reducing plots

Depth-based classification started with the *maximum depth classifier* (Ghosh and Chaudhuri, 2005b) that assigns an observation $\boldsymbol{x}$ to the class, in which it has maximal depth.

Liu et al. (1999) proposed the DD-(depth versus depth) plot as a graphical tool for comparing two given samples by mapping them into a two-dimensional depth space. Later Li et al. (2012) suggested to perform classification in the DD-plot by selecting a polynomial that minimizes empirical risk. Finding such an optimal polynomial numerically is a very challenging and computationally involved task, with a solution that in practice can be unstable. In addition, the polynomial training phase should be done twice, rotating the DD-plot. Nevertheless, the scheme itself allows to construct optimal classifiers for wider classes of distributions than the

elliptical family. Further, Vencalek (2011) proposed to use k-NN in the DD-plot and Lange et al. (2014b) proposed the $DD\alpha$-classifier. The DD-plot also proved to be useful in the functional setting (Mosler and Mozharovskyi, 2015, Cuesta-Albertos et al., 2016).

Analogously to the DD-plot we define the *potential-potential (pot-pot) plot* in Chapter 2. The potential of a class is defined as a kernel density estimate multiplied by the class's prior probability (1.3). For each pair of classes, the original data are mapped to a two-dimensional pot-pot plot and classified there. The pot-pot plot allows for more sophisticated classifiers than KDE, that corresponds to separating the classes by drawing the diagonal line in the pot-pot plot. To separate the training classes, we may apply any known classification rule to their representatives in the pot-pot plot. Such a separating approach, being not restricted to lines of equal potential, is able to provide better adapted classifiers. Specifically, we propose to use either the k-NN-classifier or the α-procedure on the plot.

1.4 The structure of the thesis

The thesis contains four main chapters. The second chapter named *Classification with the pot-pot plot* introduces a procedure for supervised classification, that is based on potential functions. The potential of a class is defined as a kernel density estimate multiplied by the class's prior probability. The method transforms the data to a potential-potential (pot-pot) plot, where each data point is mapped to a vector of potentials, similarly to the DD-plot. Separation of the classes, as well as classification of new data points, is performed on this plot. For this, either the α-procedure or the k-nearest neighbors classifier is employed. Thus the bias in kernel density estimates due to insufficiently adapted multivariate kernels is compensated by a flexible classifier on the pot-pot plot. The potentials depend on the kernel and its bandwidth used in the density estimate. We investigate several variants of bandwidth selection, including joint and separate pre-scaling and a bandwidth regression approach. The new method is applied to benchmark data from the literature, including simulated data sets as well as 50 sets of real data. It compares favorably to known classification methods such as LDA, QDA, maximal kernel density estimates, k-NN, and DD-plot classification. This chapter is based on a joint paper with Prof. Karl Mosler. The proposed method has been implemented in the R-package **ddalpha**. The paper has been published in the journal *Statistical Papers*.

In the third chapter named *Depth and depth-based classification with R-package* **ddalpha** we describe our package **ddalpha** that provides an implementation for exact and approximate computation of seven most reasonable and widely applied depth notions: Mahalanobis, halfspace, zonoid, projection, spatial, simplicial and simplicial volume. The main feature of the proposed methodology on the DD-plot is the $DD\alpha$-classifier, which is an adaptation of the α-procedure to the depth space. Except for its efficient and fast implementation, **ddalpha** suggests other classification techniques that can be employed in the DD-plot: the original polynomial separator and the depth-based k-NN-classifier. Unlike other packages, **ddalpha** implements various depth functions and classifiers for multivariate and functional data under one roof. The func-

tional data are transformed into a finite dimensional basis and classified there. **ddalpha** is the only package that implements zonoid depth and efficient exact halfspace depth. All depths in the package are implemented for any dimension $d \geq 2$. Except for the projection depth all implemented algorithms are exact, and supplemented by their approximating versions to deal with the increasing computational burden for large samples and higher dimensions. The package is expandable with user-defined custom depth methods and separators. Insights into data geometry as well as assessing the pattern recognition quality are feasible by functions for depth visualization and by built-in benchmark procedures. This chapter carries on joint work with Pavlo Mozharovskyi and Prof. Rainer Dyckerhoff. The paper has been submitted to the *Journal of Statistical Software*.

The fourth chapter named *Depth-weighted Bayes classification* introduces two procedures for supervised classification that focus on the centers of the classes and are based on data depth. The classifiers add either a depth or a depth rank term to the objective function of the Bayes classifier. The cost of misclassification of a point depends not only on its belongingness to a class but also on its centrality in this class. Classification of more central points is enforced while outliers are underweighted. The proposed objective function may also be used to evaluate the performance of other classifiers instead of the usual average misclassification rate. The usage of the depth function increases the robustness of the new procedures against big inclusions of contaminated data, which impede the Bayes classifier. At the same time smaller contaminations distort the outer depth contours only slightly and thus cause only small changes in the classification procedure. This chapter is a result of a cooperation with Ondrej Vencalek from Palacky University Olomouc.

The fifth chapter named *Computation of the Oja median by bounded search* suggests a new algorithm for the exact calculation of the Oja median. It modifies the algorithm of Ronkainen et al. (2003) by employing bounded regions which contain the median. The regions are built using the centered rank function. The new algorithm is faster and has lower complexity than the previous one and is able to calculate data sets of the same size and dimension. It is mainly restricted by the amount of RAM, as it needs to store all $\binom{n}{k}$ hyperplanes. It can also be used for an even faster approximative calculation, although it still needs the same amount of RAM and is slower than existing approximating algorithms. The new algorithm was implemented as a part of the R-package **OjaNP**. The chapter is partially based on a paper with Prof. Karl Mosler that has been published in the book *Modern Nonparametric, Robust and Multivariate Methods: Festschrift in Honour of Hannu Oja*. Some material of the chapter is taken from our joint paper with Daniel Fischer, Jyrki Möttönen, Klaus Nordhausen and Daniel Vogel, submitted to the *Journal of Statistical Software*.

Chapter 2

Classification with the pot-pot plot

2.1 Introduction

Statistical classification procedures belong to the most useful and widely applied parts of statistical methodology. Problems of classification arise in many fields of application like economics, biology, medicine. In these problems objects are considered that belong to $q \geq 2$ classes. Each object has d attributes and is represented by a point in d-space. A finite number of objects is observed together with their class membership, forming q training classes. Then, objects are observed whose membership is not known. The task of supervised classification consists in finding a rule by which any object with unknown membership is assigned to one of the classes.

A classical nonparametric approach to solve this task is by *comparing kernel density estimates* (KDE); see e.g. Silverman (1986). The *Bayes rule* indicates the class of an object $\boldsymbol{x}$ as $\text{argmax}_j\,(p_j f_j(\boldsymbol{x}))$, where p_j is the prior probability of class j and f_j its generating density. In KDE classification, the density f_j is replaced by a proper kernel estimate $\hat{f}_j$ and a new object is assigned to a class j at which its *estimated potential*,

$$\hat{\phi}_j(\boldsymbol{x}) = p_j \hat{f}_j(\boldsymbol{x})\,, \tag{2.1}$$

is maximum. It is well known (e.g. Devroye et al. (1996)) that KDE is *Bayes consistent*, that means, its expected error rate converges to the error rate of the Bayes rule for any generating densities.

As a practical procedure, KDE depends largely on the way by which the density estimates $\hat{f}_j$ and the priors p_j are obtained. Many variants exist, differing in the choice of the multivariate kernel and, particularly, its bandwidth matrix. Wand and Jones (1993) demonstrate that the choice of bandwidth parameters strongly influences the finite sample behavior of the KDE -classifier. With higher-dimensional data, it is computationally infeasible to optimize a full bandwidth matrix. Instead, one has to restrict on rather few bandwidth parameters.

In this chapter we modify the KDE approach by introducing a more flexible assignment rule in place of the maximum potential rule. We transform the data to a low-dimensional

space, in which the classification is performed. Each data point $\boldsymbol{x}$ is mapped to the vector[1] $(\phi_1(\boldsymbol{x}), \ldots, \phi_q(\boldsymbol{x}))^T$ in $\mathbb{R}^q_+$. The *potential-potential plot*, shortly *pot-pot plot*, consists of the transformed data of all q training classes and the transforms of any possible new data to be classified. With KDE, according to the maximum potential rule, this plot is separated into q parts,

$$\{\boldsymbol{x} \in \mathbb{R}^q_+ : j = \underset{i}{\operatorname{argmax}} \, (\hat{\phi}_i(\boldsymbol{x}))\} \, . \tag{2.2}$$

If only two classes are considered, the pot-pot plot is a subset of $\mathbb{R}^2_+$, where the coordinates correspond to potentials regarding the two classes. Then, KDE corresponds to separating the classes by drawing the diagonal line in the pot-pot plot.

However, the pot-pot plot allows for more sophisticated classifiers. In representing the data, it reflects their proximity in terms of differences in potentials. To separate the training classes, we may apply any known classification rule to their representatives in the pot-pot plot. Such a separating approach, being not restricted to lines of equal potential, is able to provide better adapted classifiers. Specifically, we propose to use either the k-NN-classifier or the α-procedure to be used on the plot. By the pot-pot plot procedure – once the transformation and the separator have been established – any classification step is performed in q-dimensional space.

To construct a practical classifier, we first have to determine proper kernel estimates of the potential functions for each class. In doing this, the choice of a kernel, in particular of its bandwidth parameters, is a nontrivial task. It requires the analysis and comparison of many possibilities. We evaluate them by means of the classification error they produce, using the following cross-validation procedure: Given a bandwidth matrix, one or more points are continuously excluded from the training data. The classifier is trained on the restricted data using the selected bandwidth parameter; then its performance is checked with the excluded points. The average portion of misclassified objects serves as an estimate of the classification error. Notice that a kernel bandwidth minimizing this criterion may yield estimated densities that differ significantly from the actual generating densities of the classes.

Then we search for the optimal separation in q-dimensional space of the pot-pot plot. This, in turn, allows us to keep the number of bandwidth parameters reasonably low, as well as the number of their values to be checked.

The principal achievements of this approach are:

- The possibly high dimension of original data is reduced by the pot-pot transformation so that the classification can be done on a low-dimensional space, whose dimension equals the number of classes.

- The bias in kernel density estimates due to insufficiently adapted multivariate kernels is compensated by a flexible classifier on the pot-pot plot.

[1] $\boldsymbol{z}^T$ denotes the transpose of $\boldsymbol{z}$.

- In case of two classes, the proposed procedures are either always strongly consistent (if the final classifier is k-NN) or strongly consistent under a slight restriction (if the final classifier is the α-classifier).
- The two procedures, as well as a variant that scales the classes separately, compare favourably with known procedures such as linear and quadratic discrimination and DD-classification based on different depths, particularly for a large choice of real data.

The chapter is structured as follows. Section 2.2 presents density-based classifiers and the Kernel Discriminant Method. Section 2.3 treats the problem of selecting a kernel bandwidth. The pot-pot plot is discussed in Section 2.4, and the consistency of the new procedures is established in Section 2.5. Section 2.6 presents a variant of pre-scaling the data, namely separate scaling. Experiments with simulated as well as real data are reported in Section 2.7. Section 2.8 concludes.

2.2 Classification by maximum potential estimate

Comparing kernel estimates of the densities or potentials is a widely applied approach in classification. Consider a data cloud $\boldsymbol{X}$ of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and assume that the cloud is generated as an independent sample from some probability density f. The potential of a given point $\boldsymbol{x} \in \mathbb{R}^d$ regarding the data cloud $\boldsymbol{X}$ is estimated by a kernel estimator.

Let $K_{\boldsymbol{H}}(\boldsymbol{x}, \boldsymbol{x}_i) = |\det \boldsymbol{H}|^{-1/2} K\left(\boldsymbol{H}^{-1/2}(\boldsymbol{x} - \boldsymbol{x}_i)\right)$, $\boldsymbol{H}$ be a symmetric and positive definite bandwidth matrix, $\boldsymbol{H}^{-1/2}$ be a square root of its inverse, and $K : \mathbb{R}^d \to [0, \infty[$ be a spherical probability density function, $K(\boldsymbol{z}) = r(\boldsymbol{z}^T\boldsymbol{z})$, with $r : [0, \infty[\to [0, \infty[$ non-increasing and bounded. Then

$$\begin{aligned} \hat{f}_{\boldsymbol{X}}(\boldsymbol{x}) &= \frac{1}{n}\sum_{i=1}^{n} K_{\boldsymbol{H}}(\boldsymbol{x}, \boldsymbol{x}_i) \\ &= \frac{1}{n}\sum_{i=1}^{n} |\det \boldsymbol{H}|^{-\frac{1}{2}} K\big(\boldsymbol{H}^{-\frac{1}{2}}(\boldsymbol{x} - \boldsymbol{x}_i)\big) \end{aligned} \tag{2.3}$$

is a kernel estimator of the density of $\boldsymbol{x}$ with respect to $\boldsymbol{X}$. In particular, the Gaussian function $K(\boldsymbol{z}) = (2\pi)^{-d/2} \exp\left(-\frac{1}{2}\boldsymbol{z}^T\boldsymbol{z}\right)$ will be employed below.

Let $\boldsymbol{H}_{\boldsymbol{X}}$ be an affine invariant estimate of the dispersion of $\boldsymbol{X}$, that is,

$$\boldsymbol{H}_{\boldsymbol{AX}+\boldsymbol{b}} = \boldsymbol{A}\boldsymbol{H}_{\boldsymbol{X}}\boldsymbol{A}^T \quad \text{for any } \boldsymbol{A} \text{ of full rank and } \boldsymbol{b} \in \mathbb{R}^d. \tag{2.4}$$

Then $\boldsymbol{H}_{\boldsymbol{AX}+\boldsymbol{b}}^{-\frac{1}{2}} = (\boldsymbol{A}\boldsymbol{H}_{\boldsymbol{X}}\boldsymbol{A}^T)^{-\frac{1}{2}} = (\boldsymbol{A}\boldsymbol{H}_{\boldsymbol{X}}^{\frac{1}{2}})^{-1} = \boldsymbol{H}_{\boldsymbol{X}}^{-\frac{1}{2}}\boldsymbol{A}^{-1}$ and

$$\begin{aligned} \hat{f}_{\boldsymbol{AX}+\boldsymbol{b}}(\boldsymbol{A}\boldsymbol{y} + \boldsymbol{b}) &= \frac{1}{n}\sum_{i=1}^{n} |\det \boldsymbol{A}|^{-1} |\det \boldsymbol{H}_{\boldsymbol{X}}|^{-\frac{1}{2}} K\big(\boldsymbol{H}_{\boldsymbol{X}}^{-\frac{1}{2}}\boldsymbol{A}^{-1}(\boldsymbol{A}\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_i)\big) \\ &= |\det \boldsymbol{A}|^{-1} \hat{f}_{\boldsymbol{X}}(\boldsymbol{y}) \end{aligned}$$

Hence, in this case, the potential is affine invariant, besides a constant factor $|\det \boldsymbol{A}|^{-1}$.

Examples

- If $\boldsymbol{H_X} = h^2\hat{\Sigma}_{\boldsymbol{X}}$, (2.4) is satisfied; the potential is affine invariant (besides a factor).
- If $\boldsymbol{H_X} = h^2\boldsymbol{I}$ and $\boldsymbol{A}$ is orthogonal, we obtain $\boldsymbol{H_{AX+b}} = h^2\boldsymbol{I} = h^2\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{A}\boldsymbol{H_X}\boldsymbol{A}^T$, hence (2.4); the potential is *orthogonal invariant*.
- If $\boldsymbol{H_X} = h^2 diag(\hat{\sigma}_1^2, \ldots, \hat{\sigma}_d^2)$ and $\boldsymbol{A} = diag(a_1, \ldots, a_d)$, then $\boldsymbol{H_{AX+b}} = h^2 diag(a_1^2\hat{\sigma}_1^2, \ldots, a_d^2\hat{\sigma}_d^2) = \boldsymbol{A}\boldsymbol{H_X}\boldsymbol{A}^T$; the potential is *invariant regarding componentwise scaling*.

The selection of the bandwidth matrices $\boldsymbol{H}$ is further discussed in Section 2.3.

Now, consider a classification problem with q training classes $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_q$, generated by densities $f_1, \ldots, f_q$, respectively. Let the class $\boldsymbol{X}_j$ consist of points $\boldsymbol{x}_{j1}, \ldots, \boldsymbol{x}_{jn_j}$. The potential of a point $\boldsymbol{x}$ with respect to $\boldsymbol{X}_j$ is estimated by

$$\hat{\phi}_j(\boldsymbol{x}) = p_j \hat{f}_j(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n_j} K_{\boldsymbol{H}_j}(\boldsymbol{x}, \boldsymbol{x}_{ji}), \tag{2.5}$$

$j = 1, \ldots, q$. Figure 2.1 exhibits the potentials of two classes $A_1 = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$ and $A_2 = \{\boldsymbol{x}_4, \boldsymbol{x}_5\}$.

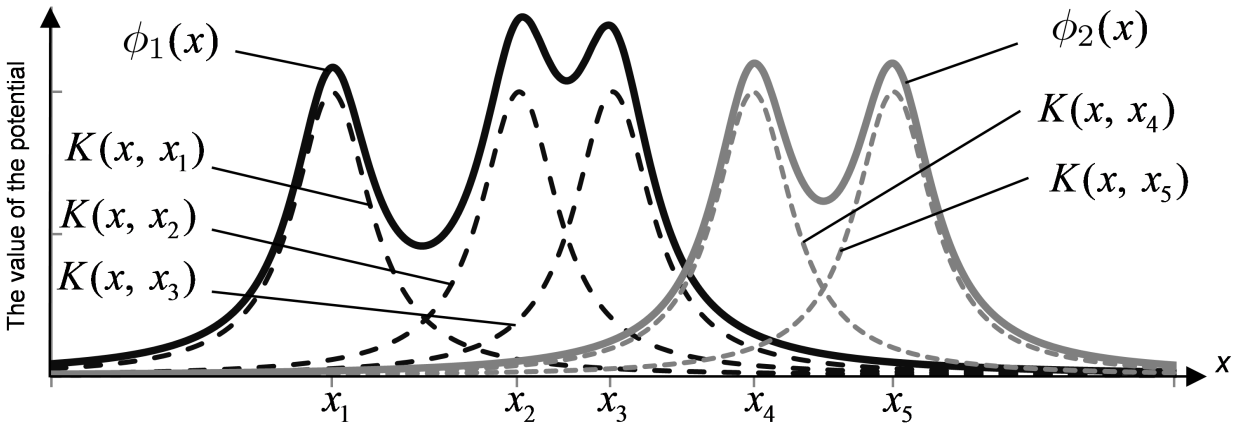


Figure 2.1: Potentials of two classes, $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$ and $\{\boldsymbol{x}_4, \boldsymbol{x}_5\}$.

The Bayes rule yields the class index of an object $\boldsymbol{x}$ as $\operatorname{argmax}_j (p_j f_j(\boldsymbol{x}))$, where p_j is the prior probability of class j. The *potential discriminant rule* mimics the Bayes rule: Estimating the prior probabilities by $p_j = n_j / \sum_{k=1}^{q} n_k$ it yields

$$\operatorname*{argmax}_j (p_j \hat{f}_j(\boldsymbol{x})) = \operatorname*{argmax}_j \sum_{i=1}^{n_j} K_{\boldsymbol{H}_j}(\boldsymbol{x}, \boldsymbol{x}_{ji}). \tag{2.6}$$

2.3 Multivariate bandwidth

The kernel bandwidth controls the range on which the potentials change with a new observation. Aizerman et al. (1970), among others, use a kernel with bandwidth matrix $\boldsymbol{H}_j = h^2\boldsymbol{I}$

for both classes and apply this kernel to the data as given. This means that the kernels are spherical and treat the neighborhood of each point equally in all directions. However, the distributions of the data are often not close to spherical, thus with a single-parameter spherical kernel the estimated potential differs from the real one more in some directions than in the others (Figure 2.2.a). In order to fit the kernel to the data a proper bandwidth matrix $\boldsymbol{H}$ is selected (Figure 2.2.b). This matrix $\boldsymbol{H}$ can be decomposed into two parts, one of which follows the shape of the data, and the other the width of the kernel. Then the first part may be used to transform the data, while the second is employed as a parameter of the kernel and tuned to achieve the best separation (Figure 2.2.b,c).

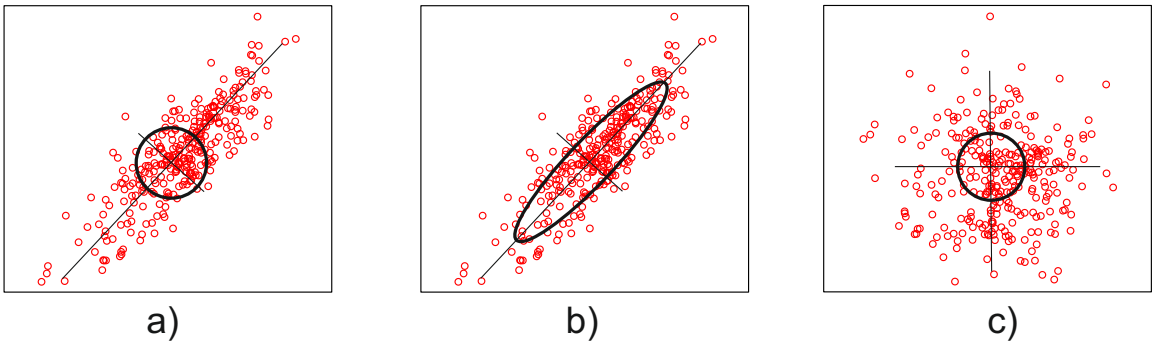


Figure 2.2: Different ways of fitting a kernel to the data: a) applying a spherical kernel to the original data; b) applying an elliptical kernel to the original data; c) applying a spherical kernel to the transformed data.

Wand and Jones (1993) distinguish several types of bandwidth matrices to be used in (2.3) and (2.5): a spherically symmetric kernel bandwidth matrix $\boldsymbol{H}_1 = h^2\boldsymbol{I}$ with one parameter; a matrix with d parameters $\boldsymbol{H}_2 = diag(h_1^2, h_2^2, ..., h_d^2)$, yielding kernels that are elliptical along the coordinate axes; and an unrestricted symmetric and positive definite matrix $\boldsymbol{H}_3$ having $\frac{d(d+1)}{2}$ parameters, that produces elliptical kernels with an arbitrary orientation. With more parameters the kernels are more flexible, but require more costly tuning procedures. The data may also be transformed beforehand using their mean $\bar{\boldsymbol{x}}$ and either the marginal variances $\hat{\sigma}_i^2$ or the full empirical covariance matrix $\hat{\boldsymbol{\Sigma}}$. These approaches are referred to as *scaling* and *sphering*. They employ the matrices $\boldsymbol{C}_2 = h^2\hat{\boldsymbol{D}}$ and $\boldsymbol{C}_3 = h^2\hat{\boldsymbol{\Sigma}}$, respectively, where $\hat{\boldsymbol{D}} = diag(\hat{\sigma}_1^2, \ldots, \hat{\sigma}_d^2)$. The matrix $\boldsymbol{C}_2$ is a special case of $\boldsymbol{H}_2$, and the matrix $\boldsymbol{C}_3$ is a special case of $\boldsymbol{H}_3$. Each has only one tuning parameter h^2 and thus the same tuning complexity as $\boldsymbol{H}_1$, but fits the data much better. Clearly, the bandwidth matrix $\boldsymbol{C}_3 = h^2\hat{\boldsymbol{\Sigma}}$ is equivalent to the bandwidth matrix $\boldsymbol{H}_1 = h^2\boldsymbol{I}$ applied to the pre-scaled data $\boldsymbol{x}' = \hat{\boldsymbol{\Sigma}}^{-\frac{1}{2}}(\boldsymbol{x} - \bar{\boldsymbol{x}})$.

Wand and Jones (1993) show by experiments that sphering with one tuning parameter h^2 shows poor results compared to the use of $\boldsymbol{H}_2$ or $\boldsymbol{H}_3$ matrices. Duong (2007) suggests to employ at least a diagonal bandwidth matrix $\boldsymbol{H}_2$ together with a scaling transformation, $\boldsymbol{H} = \hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{H}_2\hat{\boldsymbol{\Sigma}}^{1/2}$. But, even in this simplified procedure the training time grows exponentially with the number of tuned parameters, that is the dimension of the data.

In density estimation the diagonal bandwidth matrix $\boldsymbol{H}_2$ is often chosen by a rule of thumb (Härdle et al., 2004),

$$h_j^2 = \left(\frac{4}{d+2}\right)^{2/(d+4)} n^{-2/(d+4)} \hat{\sigma}_j^2, \tag{2.7}$$

which is based on an approximative normality assumption, and for the univariate case coincides with that of Silverman (1986). As the first factor in (2.7) is almost equal to one, the rule is further simplified to Scott's rule (Scott, 1992), $h_j^2 = n^{-2/(d+4)} \hat{\sigma}_j^2$. If the covariance structure is not negligible, the generalized Scott's rule may be used, having matrix

$$\boldsymbol{H}_s = n^{-2/(d+4)} \hat{\boldsymbol{\Sigma}} \,. \tag{2.8}$$

Observe that the matrix $\boldsymbol{H}_s$ is of type $\boldsymbol{C}_3$. Equivalently, after sphering the data with $\hat{\boldsymbol{\Sigma}}$, a bandwidth matrix of type $\boldsymbol{H}_1$ is applied with $h^2 = n^{-2/(d+4)}$.

Here we propose procedures that employ one-parameter bandwidths combined with sphering transformations of the data. While this yields rather rough density estimates, the imprecision of the potentials is counterbalanced by a sophisticated non-linear classification procedure on the pot-pot plot. The parameters tuning procedure works as follows: The bandwidth parameter is systematically varied over some range, and a value is selected that gives smallest classification error.

2.4 Pot-pot plot classification

In KDE classification a new object is assigned to the class that grants it the largest potential. A pot-pot plot allows for more sophisticated solutions. By this plot, the original d-dimensional data is transformed to q-dimensional objects. Thus the classification is performed in q-dimensional space.

E.g. for $q = 2$, denote the two training classes as $\boldsymbol{X}_1 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and $\boldsymbol{X}_2 = \{\boldsymbol{x}_{n+1}, \ldots, \boldsymbol{x}_{n+m}\}$. Each observed item corresponds to a point in $\mathbb{R}^d$. The pot-pot plot Z consists of the potential values of all data w.r.t. the two classes.

$$Z = \{\boldsymbol{z}_i = (z_{i1}, z_{i2}) : z_{i1} = \phi_1(\boldsymbol{x}_i), z_{i2} = \phi_2(\boldsymbol{x}_i), i = 1, ..., n+m\} \,.$$

Obviously, the maximum-potential rule results in a diagonal line separating the classes in the pot-pot plot.

However, any classifier can be used instead for a more subtle separation of the classes in the pot-pot plot. Special approaches to separate the data in the pot-pot plot are: using k-nearest neighbors (k-NN) or linear discriminant analysis (LDA), regressing a polynomial line, or employing the *α-procedure*. The α-procedure is a fast heuristic that yields a polynomial separator; see Lange et al. (2014b). Besides k-NN, which classifies directly to $q \geq 2$ classes in the pot-pot plot, the other procedures classify to $q = 2$ classes only. If $q > 2$, several binary

classifications have to be performed, either q 'one against all' or $q(q-1)/2$ 'one against one', and be aggregated by a proper majority rule.

Recall that our choice of the kernel needs a cross-validation of the single bandwidth parameter h. For each particular pot-pot plot an optimal separation is found by selecting the appropriate number of neighbors for the k-NN-classifier, or the degree of the α-classifier. For the α-classifier a selection is performed in the constructed pot-pot plot, by dividing its points into several subsets, sequentially excluding one of them, training the pot-pot plot classifier using the others and estimating the classification error in the excluded subset. For the k-NN-classifier an optimization procedure is used that calculates the distances from each point to the others, sorts the distances and estimates the classification error for each value of k. The flexibility of the final classifier compensates for the relative rigidity of the kernel choice.

Our procedure bears an analogy to *DD-classification*, as it was introduced by Li et al. (2012). There, for each pair of classes, the original data are mapped to a two-dimensional depth-depth (DD) plot and classified there. A function $\boldsymbol{x} \mapsto D^d(\boldsymbol{x}|\boldsymbol{X})$ is used that indicates how central a point $\boldsymbol{x}$ is situated in a set $\boldsymbol{X}$ of data or, more general, in the probability distribution of a random vector $\boldsymbol{X}$ in $\mathbb{R}^d$. The upper level sets of $D^d(\cdot|\boldsymbol{X})$ are regarded as 'central regions' of the distribution. $D^d(\cdot|\boldsymbol{X})$ is called a *depth function* if its *level sets* are

- *closed* and *bounded*,
- *affine equivariant*, that is, if $\boldsymbol{X}$ is transformed to $\boldsymbol{A}\boldsymbol{X} + \boldsymbol{b}$ with some regular matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{b} \in \mathbb{R}^d$, then the level sets are transformed in the same way.

Clearly, a depth function is *affine invariant*; $D^d(\boldsymbol{x}|\boldsymbol{X})$ does not change if both $\boldsymbol{x}$ and $\boldsymbol{X}$ are subjected to the same affine transformation. For surveys on depth functions and their properties, see e.g. Zuo and Serfling (2000), Serfling (2006), and Mosler (2013).

More generally, in *DD-classification*, the depth of all data points is determined with respect to each of the q classes, and a data point is represented in a q-variate DD-plot by the vector of its depths. Classification is done on the DD-plot, where different separators can be employed. In Li et al. (2012) a polynomial line is constructed, while Lange et al. (2014b) use the α-procedure and Vencalek (2014) suggests to apply k-NN in the depth space. Similar to the polynomial separator of Li et al. (2012), the α-procedure results in a polynomial separation, but is much faster and produces more stable results. Therefore we focus on the α-classifier in this chapter.

Note that Fraiman and Meloche (1999) mention a density estimate $\hat{f}_{\boldsymbol{X}}(\boldsymbol{x})$ as a *'likelihood depth'*. Of course this 'depth' does not satisfy the usual depth postulates. Principally, a depth relates the data to a 'center' or 'median', where it is maximal; a 'local depth' does the same regarding several centers. Paindaveine and Van Bever (2013) provide a local depth concept that bears a connection with local centers. Different from this, a density estimate measures at a given point how much mass is located around it; it is of a local nature, but not related to any local centers. This fundamental difference has consequences in the use of these notions as descriptive tools as well as in their statistical properties, e.g. regarding consistency of the resulting classifiers; see also Paindaveine and Van Bever (2015). Appendix at the end of the thesis contains an overview of local depths.

Maximum-depth classification with the 'likelihood depth' (being weighted with prior probabilities) is the same as KDE. Cuevas et al. (2007) propose an extension of this notion to functional data, the h-depth that is calculated as $\hat{D}^d(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} K\left(\frac{m(\boldsymbol{x},\boldsymbol{x}_i)}{h}\right)$, where m is a distance. The h-depth is used in Cuesta-Albertos et al. (2016), among several genuine depth approaches, in a generalized DD-plot to classify functional data. However, the DD^G classifier with h-depth applies equal spherical kernels to both classes, with the same parameter h. The authors also do not discuss about the selection of h, while Cuevas et al. (2007) proposed keeping it constant for the functional setup. Our contribution differs in many respects from the latter one: (1) We use Gaussian kernels with data dependent covariance structure and optimize their bandwidth parameters. (2) The kernel is selected either simultaneously or separately for each class. (3) When $q = 2$, in case of separate sphering, a regression between the two bandwidths is proposed, that allows to restrict the optimization to just one bandwidth parameter (see sec. 2.6). (4) Strong consistency of the procedure is demonstrated (see the next Section). (5) The procedure is compared with known classification procedures on a large number of real data sets (see Section 2.7).

2.5 Bayes consistency

We advocate the pot-pot procedure as a data-analytic tool to classify data of unknown origin, generally being non-normal, asymmetric and multi-modal. Nevertheless, it is of more than theoretical interest, how the pot-pot procedure behaves when the sample size goes to infinity. Regarded as a statistical regression approach, Bayes consistency is a desirable property of the procedure.

We consider the case $q = 2$. The data are seen as realizations of a random vector $(\boldsymbol{X}, Y) \in \mathbb{R}^d \times \{1, 2\}$, that has probability distribution P. A classifier is any function $g : \mathbb{R}^d \to \{1, 2\}$. Notate $p_j(\boldsymbol{x}) = P(Y = j|\boldsymbol{X} = \boldsymbol{x})$. The *Bayes classifier* g^* is given by $g^*(\boldsymbol{x}) = 2$ if $p_2(\boldsymbol{x}) > p_1(\boldsymbol{x})$ and $g^*(\boldsymbol{x}) = 1$ otherwise. Its probability of misclassification, $P(g^*(\boldsymbol{X}) \neq Y)$ is the best achievable risk, which is named the *Bayes risk*.

We assume that the distributions of $(\boldsymbol{X}, 1)$ and $(\boldsymbol{X}, 2)$ are continuous. Let the potentials be estimated by a continuous regular kernel (see Definition 10.1 in Devroye et al. (1996)), like a Gaussian kernel, and let (h_n) be a sequence of univariate bandwidths satisfying

$$h_n \to 0 \quad \text{and} \quad nh_n^d \to \infty\,. \tag{2.9}$$

It is well-known (see Theorem 10.1 in Devroye et al. (1996)) that then the maximum potential rule is *strongly Bayes-consistent*, that is, its error probability almost surely approaches the Bayes risk for any continuous distribution of the data. The question remains, whether the proposed procedures operating on the pot-pot plot attain this risk asymptotically.

We present two theorems about the Bayes consistency of the two variants of the pot-pot classifier.

Theorem 2.1 (k-NN) *Assume that $(\boldsymbol{X}, 1)$ and $(\boldsymbol{X}, 2)$ have continuous distributions. Then the pot-pot procedure is strongly Bayes consistent if the separation on the pot-pot plot is performed by k-nearest neighbor classification with $k_n \to \infty$ and $k_n/n \to 0$.*

Proof: Let us first define a sequence of pot-pot classifiers that satisfies (2.9). We start with two training classes of sizes n_1^* and n_2^*, set $n^* = n_1^* + n_2^*$, and determine a proper bandwidth h^* by cross-validation as described above. Then, let $n_1 \to \infty$ and $n_2 \to \infty$, $n = n_1 + n_2$. For $n > n^*$ we restrict the search for h_n to the interval

$$\left[h^* \cdot n^{\frac{\epsilon-1}{d}}, h^* \cdot n^{\delta-1}\right],$$

with some $0 < \epsilon \leq \delta < 1$. It follows that $h_n \to 0$ and $nh_n^d \to \infty$ as n goes to infinity, which yields the a.s. strong Bayes consistency of the maximum potential rule. The maximum potential rule corresponds to the diagonal of the pot-pot plot. This separator almost surely asymptotically attains the Bayes risk.

We have still to demonstrate that the k-nearest neighbor procedure applied to the transformed data on the pot-pot plot yields the same asymptotic risk. Under $k_n \to \infty$ and $k_n/n \to 0$, the k-NN procedure on the pot-pot plot is strongly Bayes consistent if either $\phi_1(\boldsymbol{X})$ or $\phi_2(\boldsymbol{X})$ is continuously distributed in $\mathbb{R}^2$; see Theorem 11.1 and page 190 in Devroye et al. (1996). But the latter follows from the continuity of the regular kernel. Obviously, the Bayes risk of classifying the transformed data is the same as that of classifying the original data. It follows that for any distribution of the original data the pot-pot procedure achieves the Bayes risk almost surely asymptotically. □

Theorem 2.2 (α-procedure) *Assume that $(\boldsymbol{X}, 1)$ and $(\boldsymbol{X}, 2)$ have continuous distributions and that*

$$P(p_1(\boldsymbol{X}) = p_2(\boldsymbol{X})) = 0\,. \tag{2.10}$$

Then the pot-pot procedure is strongly Bayes consistent if the separation on the pot-pot plot is performed by the α-procedure.

Proof: As in the preceding proof, the maximum potential rule corresponds to the diagonal of the pot-pot plot, and this separator almost surely asymptotically attains the Bayes risk. Consider the sign of the difference between the two (estimated) potentials. If the sample size goes to infinity the number of 'wrong' signs goes to zero. By assumption (2.10) also the number of ties (corresponding to points on the diagonal) goes to zero. By definition, the α-procedure in its first step considers all pairs of features, the original z_1 and z_2 and possibly polynomials of them up to some pre-given degree. Then for each pair a separating line is determined that minimizes the empirical risk; see Lange et al. (2014b). Thus, once the differences of the potentials have the correct sign, the α-procedure will produce the diagonal of the (z_1, z_2)-plane (or a line that separates the same points) in its very first step. □

Compared to these results, the Bayes consistency of depth-depth (DD) plot procedures is rather limited. It has been established only if both classes follow unimodal elliptically

symmetric distributions; see Li et al. (2012) and Lange et al. (2014b). The reason is that depth functions fit primarily to unimodal distributions. Under unimodal ellipticity, since a depth function is affine invariant, its level sets are ellipsoids that correspond to density level sets, and the depth is a monotone function of the density.

Note that the distributions of the two training classes, after having been transformed by the 'sphering transformation', can be still far away from being spherical. This happens particularly often with real data. It is well known that with a single parameter the multidimensional potentials are often poorly estimated by their kernel estimates. Then the best separator may differ considerably from the diagonal of the pot-pot plot as our results will demonstrate, see Figure 2.3. The final classification on the plot compensates the insufficient estimate by searching for the best separator on the plot.

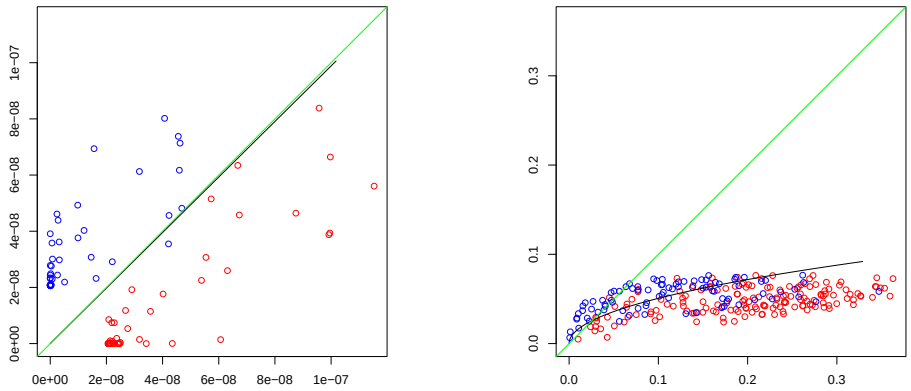


Figure 2.3: Examples of α-separation in the pot-pot plot. In the left panel (data set 'tennis') the α-classifier coincides with the diagonal, while in the right panel (data set 'baby') it provides a completely different separation. Bandwidth parameters are selected according to best performance of the α-classifier.

2.6 Scaling the data

In Section 2.3 we have shown that it is convenient to divide the bandwidth matrix into two parts, one of which is used to scale the data, and the other one to tune the width of a spherical kernel. The two classes may be scaled jointly or separately, before proper bandwidth parameters are tuned. Note that Aizerman et al. (1970) do not scale the data and use the same kernel for both classes.

KDE naturally estimates the densities individually for each class and tunes the bandwidth matrices separately. On the other hand, SVM scales the classes jointly (Chang and Lin, 2011), either dividing each attribute by its standard deviation, or scaling it to $[0; 1]$.

In what follows we consider two approaches: joint and separate scaling. With *joint scaling* the data is sphered using a proper estimate $\hat{\boldsymbol{\Sigma}}$ of the covariance matrix of the merged classes; then a spherical kernel of type $\boldsymbol{H}_1$ is applied. This results in potentials

$$\phi_j(\boldsymbol{x}) = p_j \hat{f}_j(\boldsymbol{x}) = p_j \frac{1}{n_j} \sum_{i=1}^{n_j} K_{h^2\hat{\boldsymbol{\Sigma}}}(\boldsymbol{x} - \boldsymbol{x}_{ji}) \,,$$

where an estimate $\hat{\boldsymbol{\Sigma}}$ of the covariance matrix has to be calculated and one scalar parameter h^2 has to be tuned. (Note that $\hat{\boldsymbol{\Sigma}}$ is not necessarily the empirical covariance matrix; in particular, some more robust estimate may be used.) The scaling procedure and the obtained pot-pot plot are illustrated in Fig. 2.4. Obviously, as the classes differ, the result of joint scaling is far away from being spherical and the spherical kernel does not fit the two distributions well. However, these kernels work well when the classes' overlap is small; in this case the separation is no big task.

An alternative is *separate scaling*. It results in potentials

$$\phi_j(\boldsymbol{x}) = p_j \hat{f}_j(\boldsymbol{x}) = p_j \frac{1}{n_j} \sum_{i=1}^{n_j} K_{h_j^2\hat{\boldsymbol{\Sigma}}_j}(\boldsymbol{x} - \boldsymbol{x}_{ji}) \,.$$

With separate scaling the two kernels are built with different bandwidth matrices $\boldsymbol{H}_j = \boldsymbol{C}_3 = h_j^2\hat{\boldsymbol{\Sigma}}_j$ to fit the form of each class. We need estimates of two covariance matrices and have two parameters, h_1^2 and h_2^2, to tune. Figure 2.5 illustrates the approach. It is clearly seen that many points receive much less potential with respect to the opposite class.

In case of heavy-tailed data, robustness is achieved by applying the Minimum Covariance Determinant (MCD) or the Minimum Volume Ellipsoid (MVE) estimates to transform the data. Note, that in some cases the problem may occur, that the covariance matrix is singular, e.g. if the dimension is higher than the number of points. In this case one may use a proper pseudoinverse.

As tuning the parameters comes at high computational costs, we try to simplify the tuning in the case of separate scaling. Either we use the same parameter, $h_1^2 = h_2^2$ for both classes or we establish some relationship between the two parameters, $h_2^2 = g(h_1^2)$ where g is a function of the first bandwidth. After a proper function g is found only one parameter must be tuned.

In our experiments (sec. 7), we observe a relationship between the bandwidth parameters that provide the smallest error rates. For the real data sets we see that, with separate scaling, close to smallest classification errors are usually achieved on a straight line in the $(\log_{10} h_1^2, \log_{10} h_2^2)$-plane (see Fig. 2.7 and the description there). We profit from this observation and spare the effort of separately tuning the two parameters. Note that the line is not always the main diagonal. We propose to regress one parameter on the other, evaluating the error at a few pairs $(\log_{10} h_1^2, \log_{10} h_2^2)$ and using them as sampling points. Specifically, we calculate the error at five sets of five points, with the five-point sets being taken orthogonally to the main diagonal of the plot; cf. Figures 2.6 and 2.7. Then the minimal error is found in each

set and a linear or non-linear regression of $\log_{10} h_2^2$ on $\log_{10} h_1^2$ is used to find a proper relation between the bandwidths. Consequently, we combine separate scaling with a linear bandwidth regression function, g, which simplifies the procedure enormously. Specifically, we use g to determine h_2^2 for every h_1^2, cross-validated at 60 possible values, while the full tuning would involve cross-validation of (h_1^2, h_2^2) at 3600 points. Clearly, separate scaling with bandwidth regression yields the same computational complexity as joint scaling.

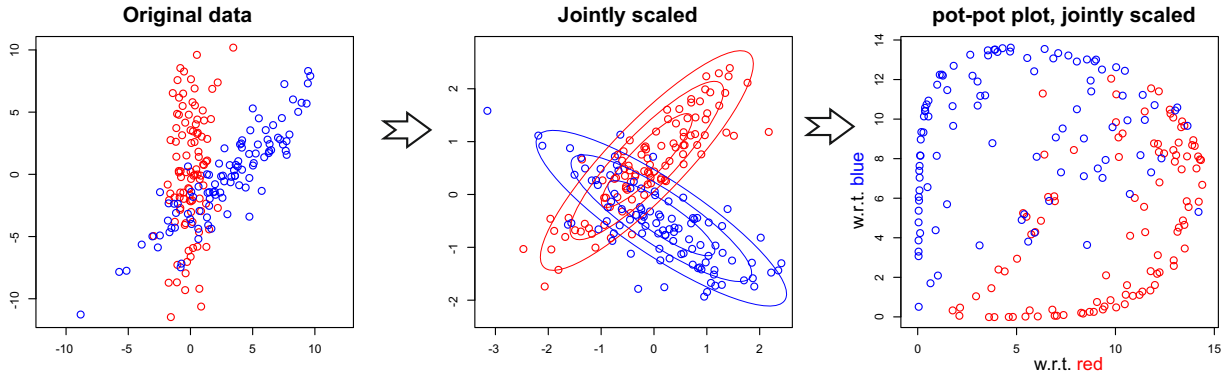


Figure 2.4: The data is jointly scaled. The plots show the original data, the scaled data with their lines of equal potential, and the corresponding pot-pot plot.

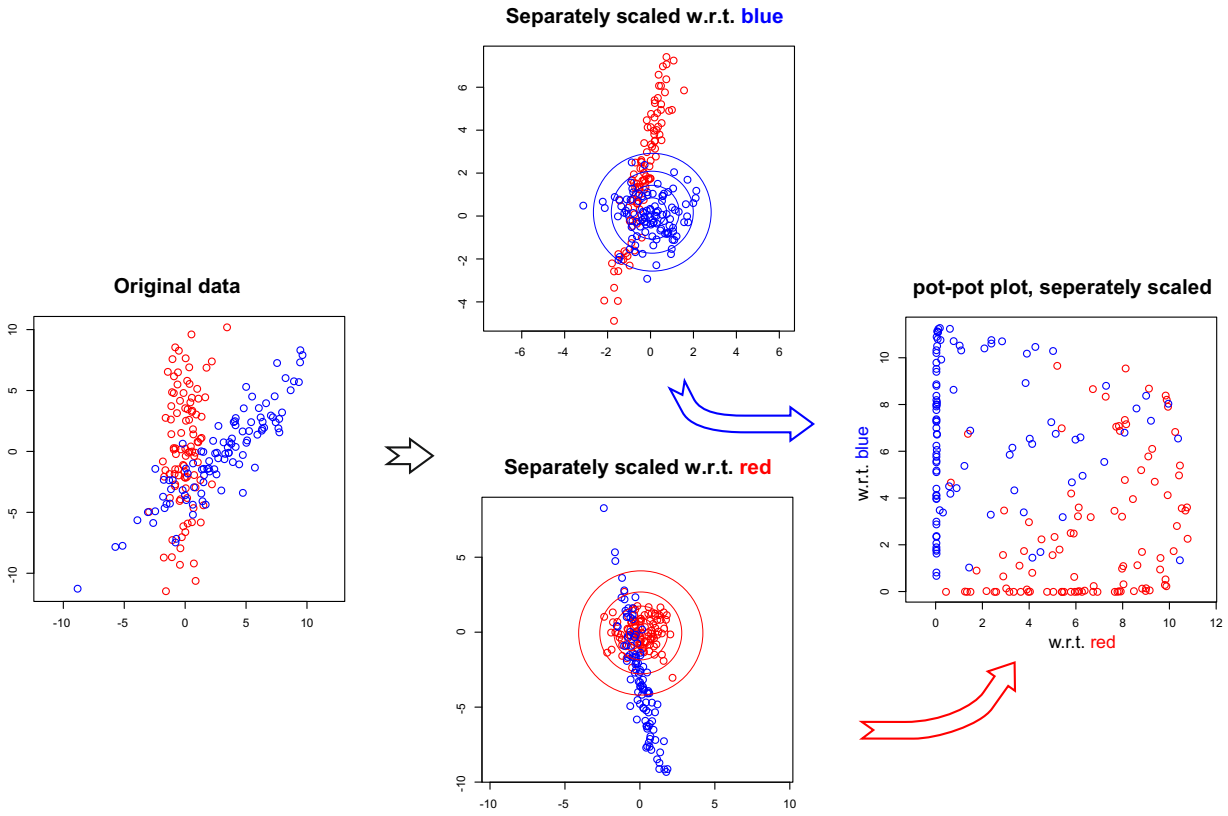


Figure 2.5: The data is separately scaled. The plots show the original data, the scaled data with their lines of equal potential, and the corresponding pot-pot plot.

2.7 Experiments

We have conducted an experimental survey to check the theoretical implications and to compare the performance of the proposed method with several traditional classifiers and *DD*-classification using popular global depths.

In the experiments we consider two classes. We compare joint and separate scaling of the classes and examine the variants of bandwidth selection. As Figures 2.6 and 2.7 illustrate, the error functions are multimodal and erratic, and can hardly be minimized in a way other than iterative search. In our experiments we select the bandwidth within a wide range and use logarithmic steps.

2.7.1 The data

Simulated as well as real data are considered in the experiments. The first two simulated series consist of two-dimensional data sets of two normally distributed classes. The classes are located at different distances, and they are scaled and rotated in different ways:

1. Location: $C_1 \sim N\left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1 & 0\\0 & 1\end{smallmatrix}\right]\right)$, $C_2 \sim N\left(\left[\begin{smallmatrix}l\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1 & 0\\0 & 1\end{smallmatrix}\right]\right)$, $l = 1, 2, 3, 4$;

2. Scale: $C_1 \sim N\left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1 & 0\\0 & 1\end{smallmatrix}\right]\right)$, $C_2 \sim N\left(\left[\begin{smallmatrix}3\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1 & 0\\0 & s\end{smallmatrix}\right]\right)$, $s = 1, 2, 3, 4, 5$;

3. Scale*: $C_1 \sim N\left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1 & 0\\0 & 1\end{smallmatrix}\right]\right)$, $C_2 \sim N\left(\left[\begin{smallmatrix}3\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}s & 0\\0 & 1\end{smallmatrix}\right]\right)$, $s = 2, 3, 4, 5$;

4. Rotation: $C_1 \sim N\left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1 & 0\\0 & 5\end{smallmatrix}\right]\right)$, $C_2 \sim N\left(\left[\begin{smallmatrix}3\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1 & 0\\0 & 5\end{smallmatrix}\right]\right)$.
   Firstly, C_1 and C_2 are generated. After that C_2 is rotated around $\mu_2 = (3, 0)$ by $\alpha \in [0, \pi/2]$ in 5 steps, then C_2 stays rotated by $\alpha = \pi/2$ and C_1 is rotated around $\mu_1 = (0, 0)$ by the same angles, giving 9 data sets in total.

The training sequence of the first series contains 100 points in each class, while the second is more asymmetric and contains 1000 resp. 300 points in the two classes. The testing sequence contains 300 points in each class of the first series, and 1000 resp. 300 points in two classes of the second series. We use the following acronyms for the data sets: the number of series (1 for equally, 2 for unequally sized classes), the name of transformation, the number of the transformation. E.g., `2scale2` means a data set with 1000 and 300 points in the classes, transformed by 'scale' transformation with $s = 2$.

The third simulated series (Dutta et al., 2012) is generated with uniform distributions on nested disks. In this case the classes cannot be separated by a simple line or curve. The two classes are distributed as $C_1 \sim U_d(0, 1) + U_d(2, 3)$ and $C_2 \sim U_d(1, 2) + U_d(3, 4)$, with $U_d(r_1, r_2)$ being the uniform distribution on $\{\boldsymbol{x} \in \mathbb{R}^d : r_1 < ||\boldsymbol{x}|| < r_2\}$. We generate the data sets in two ways: The first ones, as proposed by Dutta et al. (2012), have an equal number of points in both classes ($n_1 = n_2 = 100$ resp. $n_1 = n_2 = 400$). Note that in this case one class is less densely populated than the other. The second ones are generated so that the points of both

classes are equally dense, with $n_1 = 80$, $n_2 = 120$ resp. $n_1 = 300$, $n_2 = 500$. The naming of these data sets reflects the number of points in the classes.

For the simulated data sets the classification errors are estimated using training and testing sequences drawn from the same distributions. The procedure is replicated 40 times and the mean values are taken. The standard deviations of the error rates are mostly around five to ten times smaller than their values, and this ratio is smaller than two only in one percent of all cases.

We also use 50 real multivariate binary classification problems, collected and described by Mozharovskyi et al. (2015). The data sets are available at `http://www.wisostat.uni-koeln.de/de/forschung/software-und-daten/data-for-classification/` and in the R-package **ddalpha** (Pokotylo et al., 2016). The data have up to 1 000 points in up to 15 dimensions; they include asymmetries, fat tails and outliers. As the real data sets have no separate testing sequence, the cross-validation procedure described in the introduction is used to estimate the classification errors. We exclude one or more points from the data set, train the classifiers on the remaining points and check them using the excluded points. The number of excluded points is chosen to limit the number of such iterations to 200.

2.7.2 Comparison with depth approaches and traditional classifiers

We compare the classifiers with the Bayes classifier for the simulated data and with LDA for the real data. Note that in this case LDA gave the best results among the traditional classifiers: *linear discriminant analysis* (LDA), *quadratic discriminate analysis* (QDA), and *k-nearest neighbors* (k-NN). We introduce an efficiency index as the ratio of the error rates of the chosen classifier and the referenced one (the Bayes classifier or the LDA, resp.): $I_{classifier} = \epsilon_{classifier}/\epsilon_{reference}$. The index measures the relative efficiency of a classifier compared to the referenced one for each particular data set. We study the distribution of the efficiency index for each method over all data sets using box plots, which allows to compare the efficiency of different methods visually.

We compare our method with *kernel density estimation* (KDE) and *DD-plot classification*. To construct the DD-plots, we apply five most popular global depth functions, that can be calculated in reasonable time: zonoid, halfspace, Mahalanobis, projection and spatial ($=L_1$) depth. We do not use simplicial or simplicial volume depth, as they need much more calculation time, which is not feasible in higher dimensions. For details on these depth notions, see Zuo and Serfling (2000), Serfling (2006), and Mosler (2013). Then we use α-classification (Lange et al., 2014b), polynomial (Li et al., 2012) or k-NN-classification on the DD-plot, or take its diagonal as the separation line. The diagonal separation on the DD-plot corresponds to the maximum-depth approach. In contrast to this, the α-classifier as well as the polynomial separator and the k-NN-method produce a non-linear separation of the DD-plot.

The zonoid and halfspace depths vanish outside the convex support of the training data and the points lying there have zero depth. If a point has zero depth w.r.t. both classes it is called an *outsider*; it cannot be classified on the DD-plot. Lange et al. (2014b) propose a separate

outsiders treatment procedure, which classifies outsiders in the original space. However, if an outsider treatment procedure is used, the obtained error rate is a mixture of that of the DD-classifier and that of the outsider treatment procedure. In our study we use QDA as an outsider treatment procedure for the simulated data and LDA for the real data. Note that the number of outsiders is much larger in the real data sets than in the simulated data. It is higher than 50% for 2/3 of the real data sets.

We also add the depth-based k-NN of Paindaveine and Van Bever (2015) for comparison, as it is close in spirit to depth-based classification and to k-NN. Here we apply halfspace and Mahalanobis depths, and choose k by cross-validation.

On the pot-pot plot we proceed in a similar way. We separate the classes by either the diagonal line or the α-classifier or k-NN. Clearly, using the diagonal as a separator corresponds to KDE applied to the original data. This is combined with different bandwidth approaches: (1) joint scaling and optimizing a single h^2, (2) separate scaling and optimizing both h_1^2 and h_2^2, (3) separate scaling, regressing h_2^2 on h_1^2 (where h_1^2 belongs to the larger class), and optimizing the regressor only; see Section 2.7.3.

Tables 2.1 and 2.3 show errors from using the different methods. The methods are grouped by type, and the best values within each group are marked black. The best classifiers for the particular data set are underlined. For the DD- and pot-pot classifiers we report the errors of the α-classifier, averaged over the replications resp. cross-validation runs. For the pot-pot classifiers we use the errors, obtained with the best bandwidth parameters, and report them for the joint, separate and regressive separate approaches. The error rate estimating procedure is stable as the standard errors are very small (not reported in the tables). Figure 2.8 exhibits box plots of the efficiency index of each of these methods. Here we use the α-classifier to display the efficiency of different methods. The tables and figures for other separating methods (diagonal, polynomial and k-NN) are transferred to an Online Appendix to Pokotylo and Mosler (2016).

The experimental results show that the proposed method outperforms all compared methods on the real data and shows competitive results on the simulated data. We also observe that separate scaling (*pot-pot separate*) is more efficient than joint scaling (*pot-pot joint*). Under the name *pot-pot regressive separate* we show results that are obtained with separate scaling and bandwidth regression.

Tables 2.2 and 2.4 compare the minimal errors obtained with pot-pot classifiers that use the three separating procedures on the pot-pot plot: diagonal, α, and k-NN. Also additional bandwidth approaches in estimating the potentials (ROT, mM) are included; see Section 2.7.3 below. Figure 2.9 illustrates the corresponding boxplots of the efficiency index. For the real data sets, using either α- or k-NN-classifiers on the pot-pot plot shows better classification results than classical KDE does. The α-classifier usually has a lower error rate than k-NN, but sometimes k-NN produces unexpectedly good results, as for example in the 'cloud' and some of the 'crab' data sets. The error of the maximum-depth classifier slightly outperforms that of the α- and k-NN-classifiers for the simulated data, but is more dispersed in the case of separate scaling.

The efficiency of the DD-plot classifiers is also compared for each of the five depths. $DD\alpha$-, polynomial and k-NN-classifiers are more efficient than the maximum-depth classifier, and $DD\alpha$ shows best results in most of the cases (see Fig. 2.10). We also observe that $DD\alpha$ shows almost the same results as the polynomial classifier, and slightly outperforms it for the real data.

The pot-pot approach performs much better than the other classifiers on real data. For the first two sets of simulated data, as they are generated by normal distributions, QDA and KDE are best classifiers under separate scaling (see *pot-pot separate diagonal*).

To illustrate the performance of the pot-pot classifiers in the higher dimensions we simulated multidimensional hyperspheres similar to the third simulated series in Section 2.7.1. The points were generated uniformly in $d \in \{2, 3, 4, 5, 10\}$ and sample length $n \in \{50, 100, 250, 500, 1000\}$. With the growth of dimension the volume of the outer spheres increases faster than of the inner ones. In this case the classes become unbalanced and the probability of the first class is $\{0.38, 0.31, 0.26, 0.21, 0.06\}$, respectively. To make the classes equal we inverted the labeling in one half of the hypersphere. We observe that with the growth of d the error rate of the diagonal separation (=KDE) and the $DD\alpha$ grow much faster than the error rate of k-NN, see Figure 2.13. As shown before, in dimension 10 the hypersphere is divided into two halves, each containing mostly one class, and therefore the error rates of all classifiers become smaller. Nevertheless it is still much lower for k-NN, which means that the pot-pot plot allows to improve the separation even in higher dimensions. As for the real examples, the advantage of the pot-pot classifiers is bigger in small dimensions, while in the dimensions higher than eight they mostly perform as good as the diagonal separation, see Table 2.4.

2.7.3 Selection of the optimal bandwidth

We vary the kernel bandwidth parameters over a wide range using logarithmic steps. The bandwidth selection process is shown on the bandwidths-to-errors plots, which illustrate the dependencies between the selected kernel bandwidths and the classification errors for particular data sets using diverse DD-classifiers under joint and separate scaling. See Figures 2.6 and 2.7 and the explanations there. In the *joint scaling* case the abscissa represents the logarithm of the bandwidth parameter $\log_{10} h^2$, and the ordinate the error rate. For the *separate scaling* case the axes present the $\log_{10} h_i^2$ bandwidth of the first and the second classes kernels, respectively. The colors correspond to the classification errors achieved with these bandwidth combinations, where *red* indicates the highest error rate, *violet* the lowest, and the colors in between are in the rainbow order.

Experimentally we have found higher and lower bounds for the kernel bandwidth parameters search. The lower bound of $h^2 = 10^{-3}$ is explained with computational limitations, as the potential induced by such narrow kernels is too small to be represented with the machine type 'double'. Thus, most points cannot be classified on the pot-pot plot, as they obtain zero potential. Such points are *outsiders* in DD-classification, where they are classified separately; see Lange et al. (2014b). When the bandwidth reaches the level of $h^2 = 10^3$, separation does

not improve any more, since the kernels become too flat. As the classification error stabilizes at this level, we take it as an upper bound.

It is also observed that extremely wide or narrow kernels give fairly good separation errors. This feature may be used for a fast 'draft' estimation of the classification error which could be reached with the classifier, and possibly for reducing the search intervals of the bandwidth parameters.

In selecting the kernels' bandwidths we compare the performance of the generalized Scott's rule of thumb (column ROT) and the extreme bandwidths (column mM). The latter means that we use the bound (either lower or upper) of h^2 that gives the smaller error. The rule of thumb works better for the simulated data sets than for the real ones.

For any pair of normally distributed classes (shifted, scaled, and/or rotated) the optimal bandwidths are equal for both classes $h_1^2 = h_2^2$; they lie around $h^2 = 1$ if the classes have about the same size. This also holds for the $DD\alpha$- and k-NN-classifiers if the classes have different sizes, while for the maximum-depth classifier a shifted line is observed. The results obtained using the rule of thumb are close to the optimal ones.

We compared the results of the full-bandwidth tuning and the regression approach, described in the end of section 2.6. The linear regression is the most reasonable, as higher order regressions, described under Figure 2.6 did not improve the efficiency of our procedure relative to simple linear regression, which is illustrated in Figure 2.11. This bandwidth regression approach is abbreviated as *pot-pot regressive separate* in the tables and figures. Observe that the results are close to the minimum obtained by full bandwidth search using separate scaling. They also outperform the ones provided by joint scaling, as Figures 2.8, 2.9 and 2.11 demonstrate. This approach works much better on the real data settings, while on simulated data the difference between joint and separate scaling is not really large. Note that in this approach the α-classifier shows best results for both simulated and real data.

2.7.4 Comparison of the classification speed

As the experiments have shown, the proposed method has a very good relative performance, but at the cost of a reduced training speed. The *training time* of a pot-pot (or DD-) classifier consists of the time to calculate the potential (resp. the depth) of each point w.r.t. each class and of the time needed to train the separator; the latter does virtually not depend on the choice of the space transformation function. If $q > 2$ and the aggregating procedures are involved, multiple separators may be trained on the pot-pot plot. The *classification time* contains only the time for calculation of the potentials (resp. depths) of a given point w.r.t. each class.

We compare the computation times of potentials and various depth notions by graphics in Figure 2.12. On the logarithmic time scale, the lines represent the time (in seconds) needed to compute depth or potential of a single point, averaged over 50 points w.r.t. 60 samples, varying dimension $d \in \{2, 3, 4, 5\}$ and sample length $n \in \{50, 100, 250, 500, 1000\}$. Due to the fact that computation times of the algorithms do not depend on the particular shape of the data, the data has been drawn from the standard normal distribution. Some of the graphics

are incomplete due to excessive time. Projection and halfspace depths have been approximated using 1 000 random projections and simplicial depth (if $d > 2$) has been approximated using 5% of simplices. The other depths have been computed exactly as well as simplicial depth for $d = 2$. The calculation of simplicial depth dramatically slows down with the growth of n and d, therefore we do not include it in the comparison of Section 2.7.2.

We observe that it takes from 0.5ms to 1ms to calculate the potential of one point. The calculation speed does not depend on the dimension of the data and slightly depends on the number of points. In big data sets it is only outperformed by Mahalanobis and spatial depths.

This advantage is nevertheless suppressed by the tuning of the bandwidth parameters. Having k_p bandwidth parameters the error rate obtained with each of them is estimated by cross-validation that repeats the training and classification procedures for k_e times. A usual choice of k_e is 10, but in this chapter we set it to 200 to obtain precise estimates. The number k_p of possible values of the bandwidth parameter is set to 60 in the *joint scaling* case, 3600 in the *separate scaling* case, and 25+60 in the *regressive separate* approach. This means that with the regressive separate approach we have trained the pot-pot classifier $85 * 200 = 17\,000$ times, and the DD-classifiers only 200 times to estimate the error rate. In practice these numbers may be seriously reduced by iterating less bandwidth parameters and shortening their range, and applying less iterations during cross-validation. Note that the classical KDE classification needs the same number of iterations as the pot-pot classifiers to tune the bandwidth parameters, given that the same bandwidth matrices are taken in both approaches.

2.8 Conclusion

A new method is proposed that combines ideas of kernel discriminant analysis and depth-depth-plot classification. Potentials of data points, which amount to weighted kernel-estimated densities, are used for classification on a potential-potential (pot-pot) plot. Compared with classical approaches the method shows a very good relative performance, especially on real data settings.

The two most important aspects of the method of potentials are: Firstly, compared to classification methods based on depths the method reflects local properties of the distributions and, thus, gives better results for multimodal and non-elliptical distributions. Secondly, the use of the pot-pot plot allows for more sophisticated separations than the simple comparison of estimated densities.

Consequently, the bandwidth parametrization and the selection of bandwidth parameter values can be kept simple. Instead of scaling the kernels we scale the data separately and apply a simple spherical kernel to these sphered data. Then the kernel bandwidth itself is tuned using just one parameter. Joint and separate scaling of the classes are compared. Under separate scaling the kernels provide a better fit to the classes' distributions, which clearly improves the classification.

As our experiments demonstrate, the classification error can be a multimodal and very erratic function of bandwidth parameters, which makes it difficult to minimize. We search for a global minimum by iterating over a proper set of possible parameter values (see Fig. 2.6 and 2.7). Our experiments show further that a roughly linear relation between the logarithms of two bandwidth parameters can be established under which a separation close to the best one is achieved. This allows us to restrict on tuning a single bandwidth parameter of one class h_1^2, using a linear regression to determine the bandwidth parameter of the second one h_2^2. The bandwidth parameter is selected from the diapason $h^2 \in [10^{-3}; 10^3]$ with a logarithmic scale.

The naive KDE approach estimates the potential of a point regarding the classes and assigns the point to the class of maximum estimated potential. Asymptotically the naive KDE approach reaches the optimal Bayes risk. However, with finite samples in higher dimensions these estimates are highly biased (Friedman, 1997) since the kernel bandwidth can only be roughly adjusted to the dependency structure of the distributions. In contrast our procedure is asymptotically Bayes-optimal as well, but reduces the influence of the finite-sample bias by additionally optimizing the line separating the potentials.

If two jointly scaled classes are considered, strong Bayes consistency is shown in general for pot-pot separation by k-NN, and under a slight restriction for separation by the α-procedure. Further consistency results may be derived for variants of this along the same lines. E.g. the data dependent kernels arising with separate scaling and h_2-h_1-regression are still continuous and regular; for consistency it is sufficient to warrant that the constant ρ appearing in Theorem 10.1 of Devroye et al. (1996) satisfies $\rho = o(n)$.

The new method has been implemented as part of the R-package **ddalpha**, which was also used for the experimental study. The package is described in the next Chapter.

Appendix
Experimental results

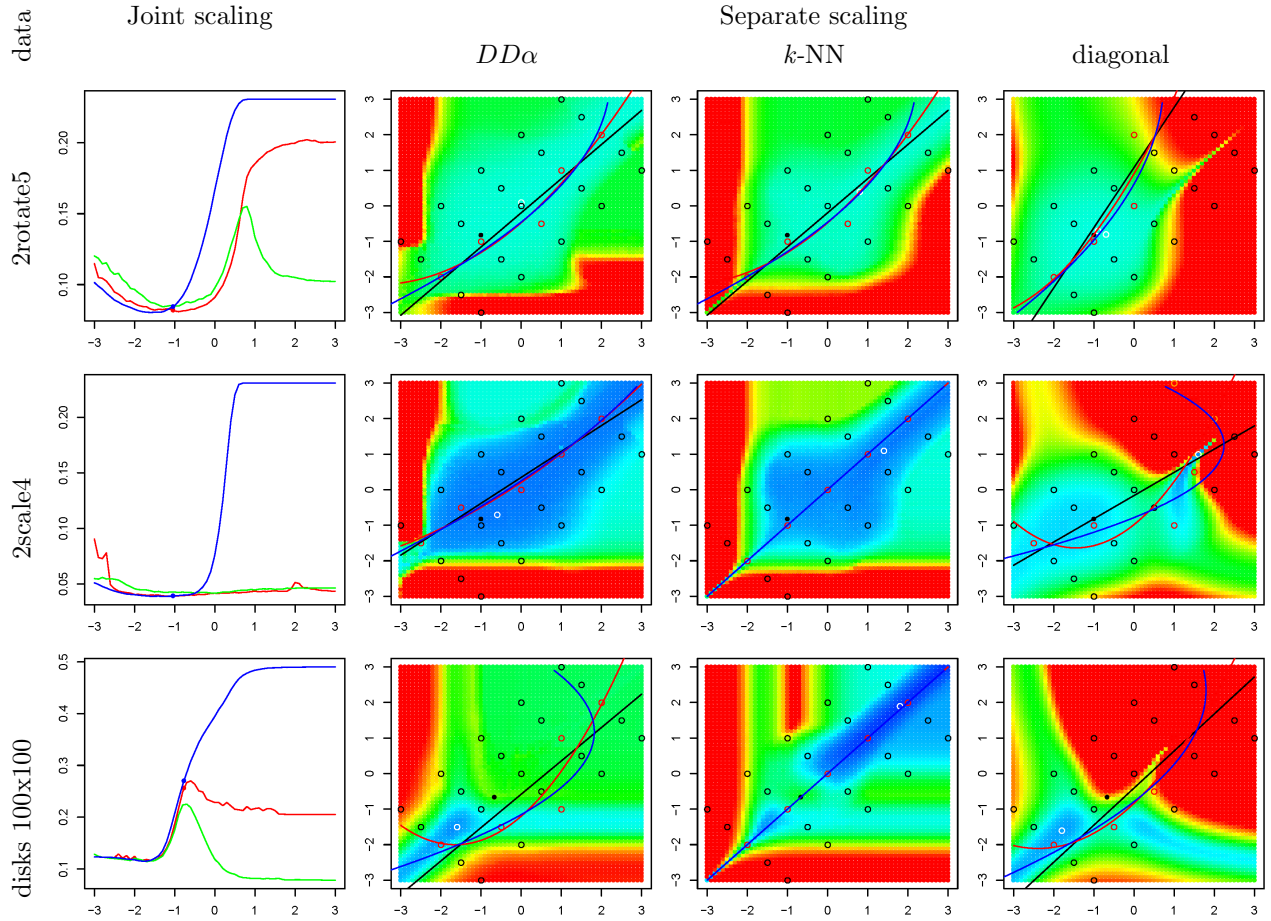


Figure 2.6: Examples of bandwidths-to-error plots (simulated data).

Left column of panels (joint scaling):
abscissa $\widehat{=} \log_{10} h^2$; ordinate $\widehat{=}$ classification error rate;
classifiers: diagonal (*blue*), k-NN (*green*), $DD\alpha$ (*red*).

Other panels (separate scaling):
abscissa $\widehat{=} \log_{10} h_1^2$; ordinate $\widehat{=} \log_{10} h_1^2$;
colors: classification error rates from *violet* to *red*;
points: *black* – sample points (grouped orthogonally to the main diagonal);
minima in each test group (*red points*); global minima (*white points*);
regressions (h_1^2 is the bandwidth of the larger class):

Linear	— $\log_{10} h_2^2 = a + b \log_{10} h_1^2$;
Quadratic	— $\log_{10} h_2^2 = a + b \log_{10} h_1^2 + c(\log_{10} h_1^2)^2$;
Quadratic inverted	— $\log_{10} h_1^2 = a + b \log_{10} h_2^2 + c(\log_{10} h_2^2)^2$.

See detailed description under Figure 2.7.

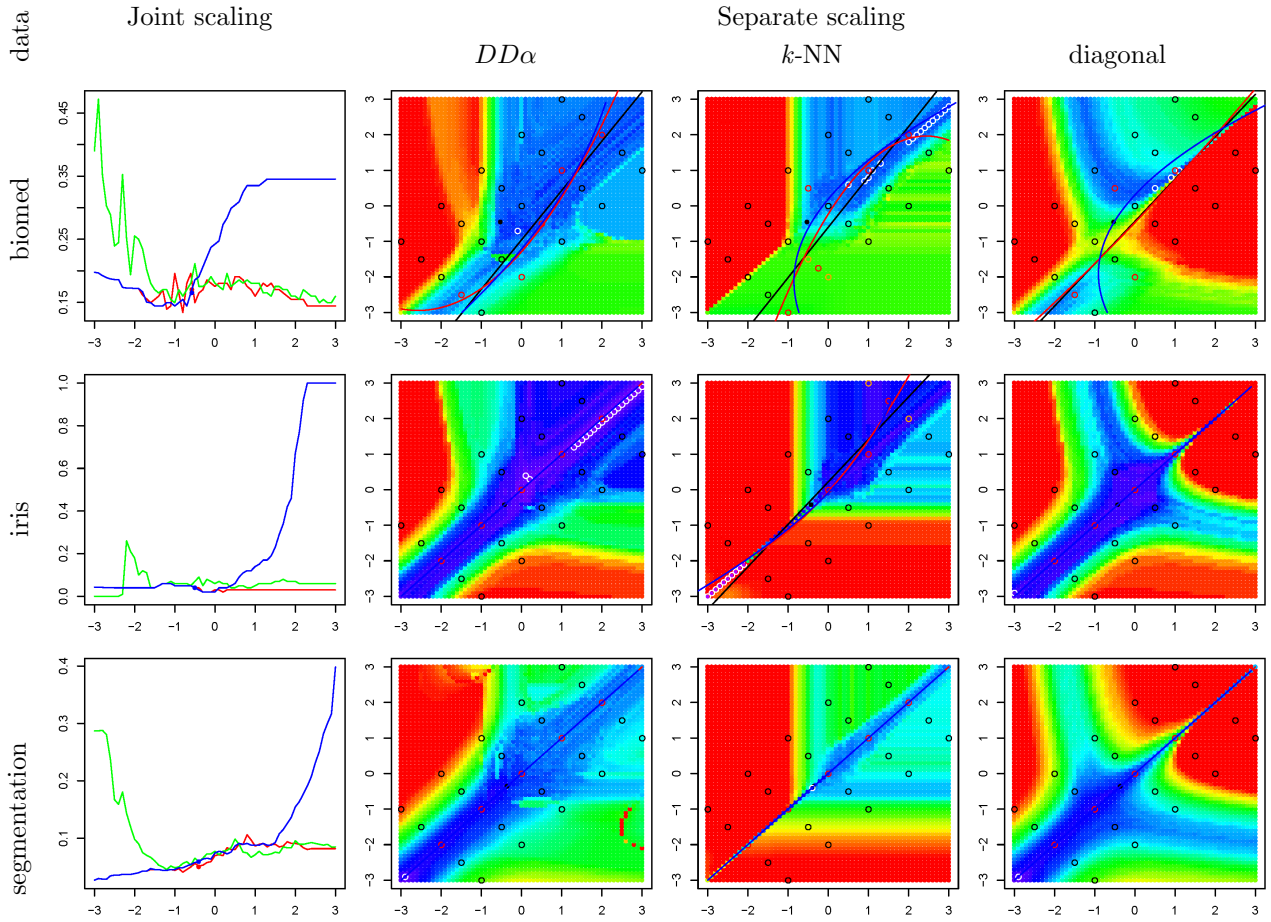


Figure 2.7: Examples of bandwidths-to-errors plots (real data).

See the legend under Figure 2.6.

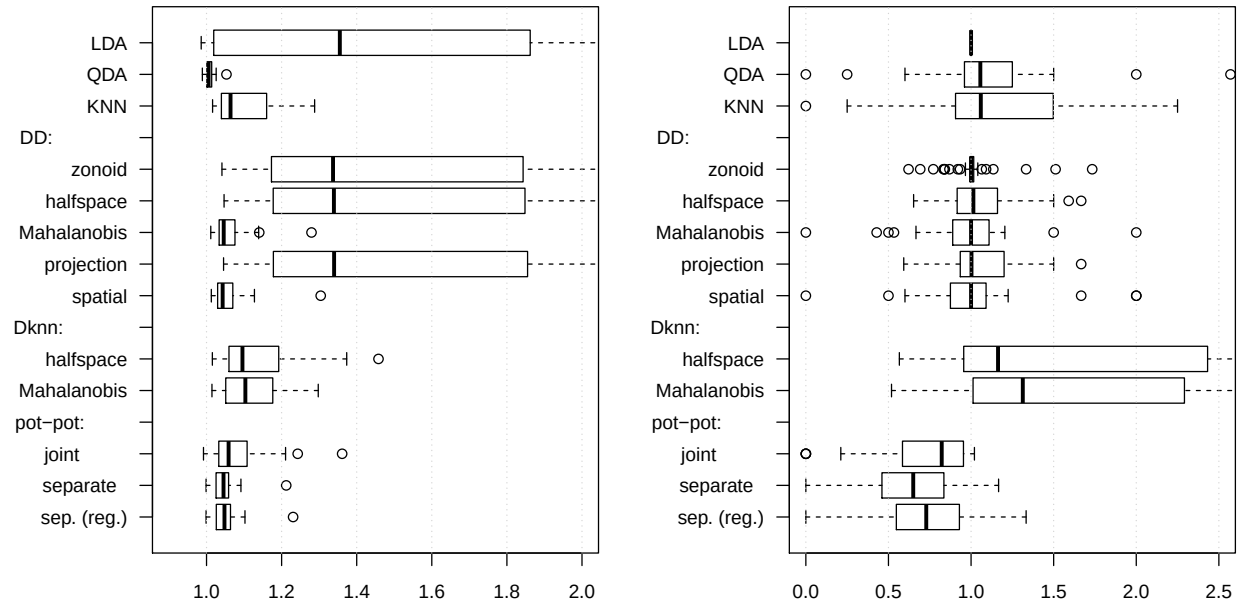
The bandwidths-to-errors plots illustrate the dependencies between the selected kernel bandwidths and the classification errors for particular data sets using diverse DD-classifiers using joint (left column of panels) and separate (other panels) scaling.

In the ***joint scaling*** case the abscissa represents the logarithm of the bandwidth parameter $\log_{10} h^2$, and the ordinate the error rate. The rule of thumb (ROT) errors are shown in bold.

For the ***separate scaling*** case the axes present the $\log_{10} h_i^2$ bandwidth of the first and the second classes kernels, respectively. The colors correspond to the classification errors achieved with these bandwidth combinations, where *red* corresponds to the highest error rate, *violet* to the lowest, and the colors in between are in the rainbow order. The *black points* represent the rule of thumb (ROT) bandwidths.

We search for the relationship between the bandwidth parameters using regressions. At first we calculate the classification error rate at 25 bandwidth points divided into five sets orthogonal to the main diagonal. Then the minimum is found over each set and a regression is computed to find the proper relation between the bandwidths. We use the found relationship to estimate error rates along the regression line, iterating one bandwidth parameter and calculating the other. For comparing the performance of this approach (*pot-pot regressive separate*), to joint and separate scaling; see Fig. 2.9.

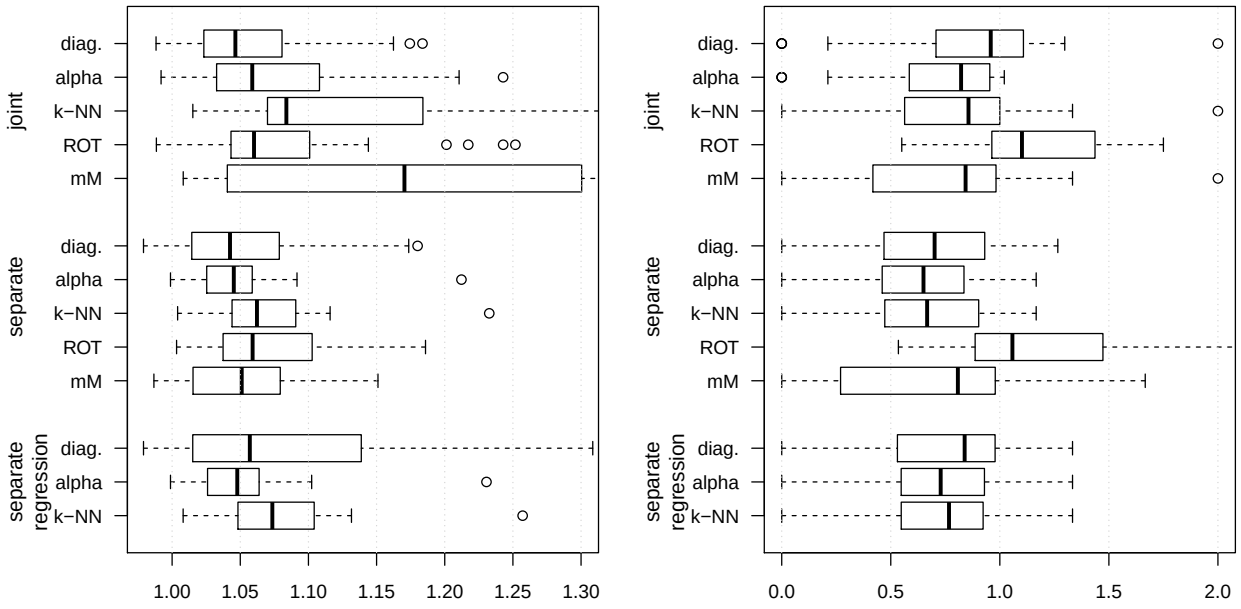
The minimum errors are found in Table 2.2 for simulated and in Table 2.4 for real data.



a) simulated elliptical data b) real data

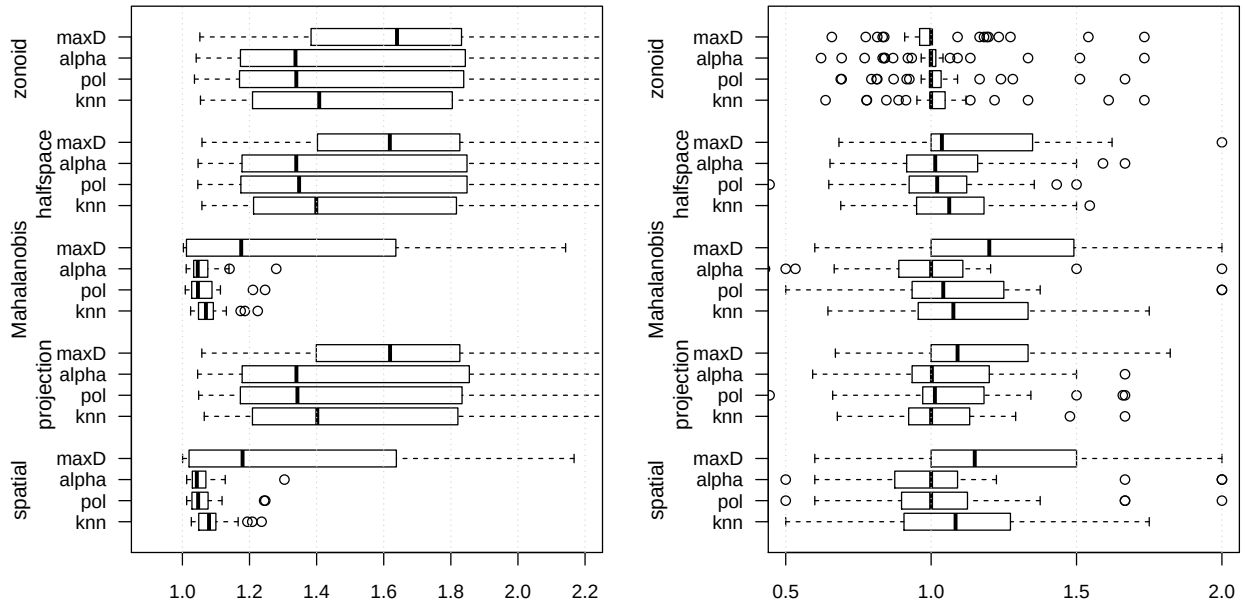
Figure 2.8: Efficiency of the methods. For the *DD*- and pot-pot classifiers the errors of the α-classifier are given.

The index is the relation of the error rates of the chosen classifier and the reference classifier. Here and in the following figures we take the Bayes risk as the reference for the simulated data and LDA – for the real data. The index measures the relative efficiency of a classifier compared to the reference for a particular data set (the more efficient classifier has smaller index). For each classifier a boxplot is built that illustrates the distribution of the efficiency index over all data sets.



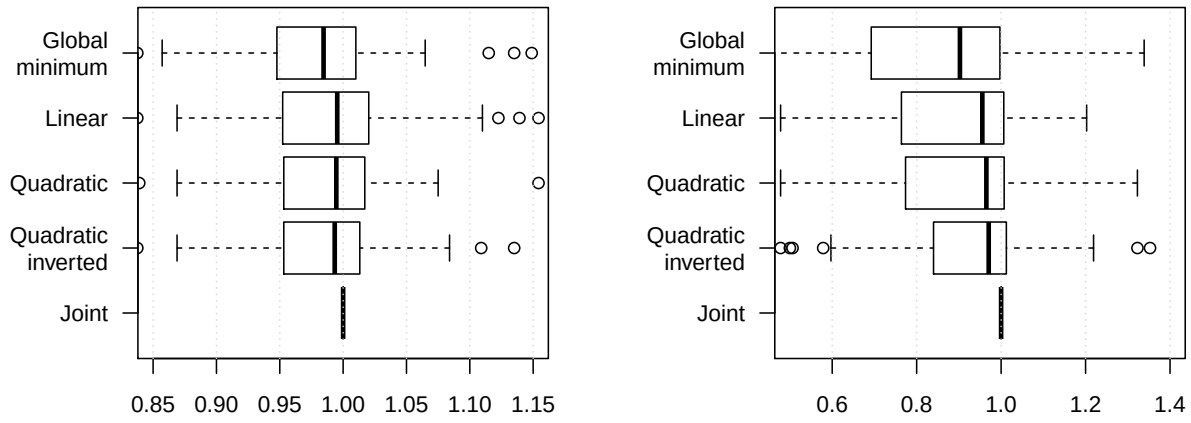
a) simulated elliptical data b) real data

Figure 2.9: Efficiency of the pot-pot classifiers.



a) simulated elliptical data b) real data

Figure 2.10: Efficiency of DD-classifiers.



a) simulated elliptical data b) real data

Figure 2.11: Efficiency of separate scaling with different bandwidth regressions over joint scaling.

The boxplots show minimal errors using the same bandwidth regressions as in the description of Figure 2.6. The first and the last rows show the global minima for resp. *separate* and *joint* scaling. The classifier using joint scaling is taken as the reference.

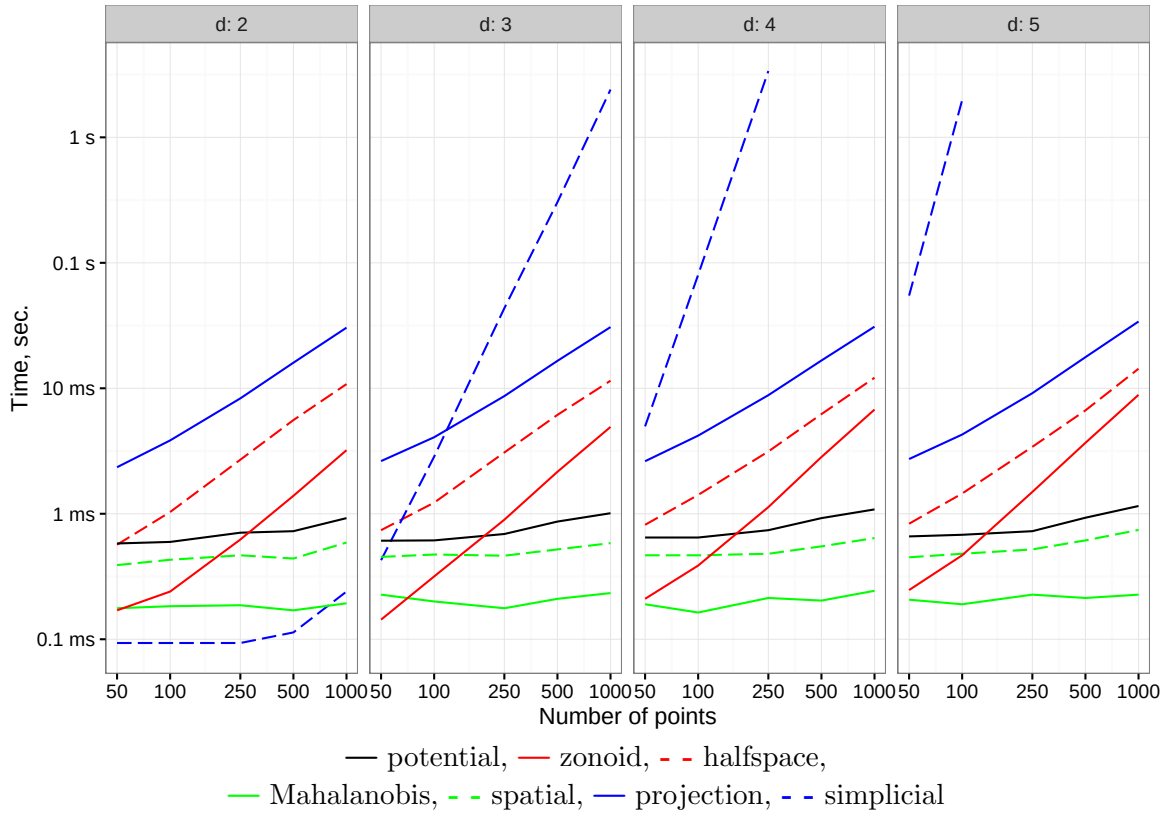


Figure 2.12: Calculation time of a single data point for various depth functions and potential, on the logarithmic time scale. Here the approximative versions are used, for the exact versions see Fig. 3.4.

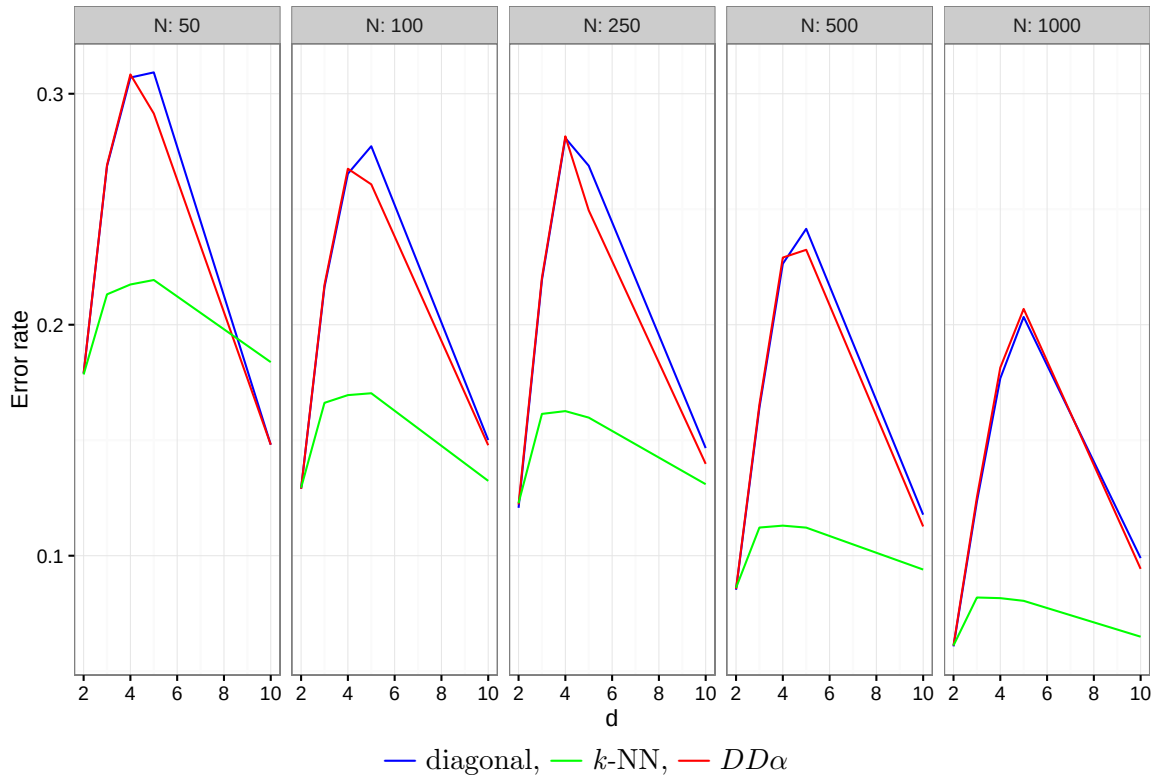


Figure 2.13: Performance of the KDE and the pot-pot classifiers in the multidimensional space.

Tables 2.1 and 2.3 show errors from using the different methods. The methods are grouped by type, and the best values within each group are marked black. The best classifiers for the particular data set are underlined. In the tables we use the following abbreviations for the data depths: HS for halfspace, Mah for Mahalanobis, Proj for projection and Spat for spatial. Dknn abbreviates the depth-based k-NN of Paindaveine and Van Bever (2015).

Table 2.1: Error rates (in %) of different classifiers for simulated data sets.

Columns (6) – (10) and (13) – (15): α-classifier in the DD- and pot-pot plots.

(1) dataset	(2) Bayes	(3) LDA	(4) QDA	(5) KNN	(6) Zonoid	(7) HS	(8) Mah	(9) Proj	(10) Spat	(11) Dknn HS	(12) Dknn Mah	(13) pot-pot joint	(14) pot-pot separate	(15) pot-pot regress. separate
1dist1	30.9	**31.1**	31.3	31.8	35.8	35.8	**31.9**	35.7	**31.9**	32.4	32.4	32.2	**31.8**	**31.8**
1dist2	15.8	**15.8**	16.0	16.6	21.1	20.9	**16.6**	20.9	16.7	16.8	16.8	16.7	**16.3**	16.6
1dist3	6.7	6.9	**6.8**	7.4	12.5	12.6	**7.1**	12.6	7.2	7.3	7.4	7.1	**6.9**	**6.9**
1dist4	2.0	**2.1**	2.2	2.5	8.5	8.4	**2.6**	8.4	2.7	2.3	2.4	2.8	**2.5**	**2.5**
1rotate1	6.7	6.9	**6.8**	8.7	12.5	12.6	**7.1**	12.5	7.2	7.3	7.4	7.1	**6.9**	**6.9**
1rotate2	12.5	14.6	**12.6**	14.8	18.5	18.4	13.6	18.2	**13.4**	14.2	14.3	13.6	**13.2**	**13.2**
1rotate3	13.0	24.3	**13.1**	15.3	20.1	20.0	**13.6**	20.0	13.7	15.3	15.2	13.9	**13.2**	**13.2**
1rotate4	11.7	27.5	**11.8**	13.3	18.8	18.6	12.4	18.6	**12.1**	13.7	13.4	12.8	**12.1**	**12.1**
1rotate5	11.0	25.2	**11.0**	13.2	18.4	18.5	**11.6**	18.4	**11.6**	13.5	13.2	12.2	**11.5**	11.6
1rotate6	12.0	25.9	**12.1**	14.0	19.3	19.1	**12.7**	19.1	12.9	14.2	14.4	13.3	**12.1**	12.2
1rotate7	15.3	28.5	**15.4**	17.6	21.8	22.2	**15.7**	22.0	15.9	17.4	17.4	16.7	**15.3**	15.4
1rotate8	23.4	33.8	**23.6**	28.7	29.7	29.7	24.3	29.9	**24.1**	26.4	26.6	25.2	**24.5**	**24.5**
1rotate9	38.0	**38.3**	38.4	39.4	42.5	42.5	**38.9**	42.5	39.0	39.9	39.8	**39.2**	39.7	39.7
1scale1	6.7	6.9	**6.8**	7.5	12.5	12.6	**7.1**	12.6	7.2	7.4	7.4	7.1	**6.9**	**6.9**
1scale2	5.8	6.8	**5.9**	6.5	11.8	11.7	**6.4**	11.6	6.5	7.0	6.6	6.4	**6.1**	**6.1**
1scale3	4.9	6.8	**5.0**	5.7	10.9	10.7	5.6	10.7	**5.5**	6.3	5.8	5.5	**5.1**	**5.1**
1scale4	4.3	6.8	**4.2**	4.9	10.3	10.3	4.8	10.3	**4.7**	5.7	5.2	4.9	**4.5**	4.6
1scale5	3.7	6.8	**3.8**	4.5	9.9	9.9	4.2	9.9	**4.0**	5.4	4.8	4.5	**4.0**	**4.0**
1scale*1	6.6	**6.7**	6.8	7.1	12.2	12.3	7.3	12.3	**7.1**	7.0	7.1	7.1	**7.0**	**7.0**
1scale*2	14.4	14.7	**14.6**	15.2	19.8	19.8	15.4	19.8	**15.3**	15.6	15.4	**14.9**	15.2	15.2
1scale*3	17.0	19.0	**17.4**	19.0	22.6	22.8	18.2	22.8	**18.1**	20.5	20.0	19.2	**17.6**	17.8
1scale*4	16.6	21.8	**17.0**	18.3	22.2	22.2	17.8	22.2	**17.7**	22.0	21.0	19.7	**17.3**	**17.3**
1scale*5	15.2	24.3	**15.3**	16.7	20.9	20.9	16.4	21.0	**16.2**	20.9	19.8	18.9	**15.8**	**15.8**
2dist1	20.7	**20.8**	**20.8**	21.3	21.9	22.0	**21.1**	22.0	21.4	21.6	21.5	**21.0**	21.2	21.2
2dist2	12.1	**12.1**	**12.1**	12.4	13.2	13.2	**12.4**	13.2	**12.4**	12.8	12.7	**12.4**	12.6	12.6
2dist3	5.2	**5.2**	**5.2**	5.5	6.5	6.5	**5.4**	6.5	**5.4**	5.9	5.7	**5.4**	5.7	5.7
2dist4	1.9	**1.9**	**1.9**	2.0	3.1	3.1	**2.0**	3.1	**2.0**	2.3	2.1	**1.9**	2.0	2.0
2rotate1	5.2	**5.2**	**5.2**	5.9	6.5	6.5	**5.4**	6.5	**5.4**	5.9	5.7	**5.4**	5.7	5.7
2rotate2	8.7	9.6	**8.8**	9.3	10.1	10.1	**9.1**	10.1	**9.1**	9.3	9.3	**9.2**	**9.2**	**9.2**
2rotate3	9.0	15.0	**9.1**	9.6	10.6	10.6	9.4	10.6	**9.3**	9.6	9.5	**9.4**	**9.4**	**9.4**
2rotate4	8.0	21.3	**8.0**	8.5	9.7	9.7	**8.4**	9.7	**8.4**	8.5	8.4	**8.4**	**8.4**	**8.4**
2rotate5	7.7	25.5	**7.7**	8.1	9.3	9.4	8.0	9.4	**7.9**	8.1	8.0	**8.1**	8.2	8.2
2rotate6	8.3	25.5	**8.4**	8.7	9.9	9.9	**8.6**	10.0	**8.6**	8.8	8.7	**8.7**	**8.7**	8.8
2rotate7	10.5	25.3	**10.5**	11.1	12.1	12.1	10.9	12.1	**10.8**	11.1	11.0	10.8	**10.7**	**10.7**
2rotate8	16.2	24.6	**16.2**	17.1	17.6	17.6	16.5	17.6	**16.4**	16.9	16.8	**16.0**	16.1	16.1
2rotate9	22.9	**22.9**	**22.9**	23.3	23.8	24.0	**23.2**	23.9	**23.2**	23.3	23.2	23.1	**23.0**	**23.0**
2scale1	5.2	**5.2**	**5.2**	5.5	6.5	6.6	**5.4**	6.5	**5.4**	5.9	5.7	**5.4**	5.7	5.7
2scale2	4.6	5.2	**4.7**	4.9	6.0	6.0	**4.8**	5.9	**4.8**	4.9	4.8	**4.8**	5.0	5.0
2scale3	4.1	5.2	**4.1**	4.2	5.4	5.4	**4.2**	5.4	**4.2**	4.1	4.2	**4.3**	**4.3**	**4.3**
2scale4	3.7	5.2	**3.7**	3.8	5.1	5.0	**3.8**	5.0	**3.8**	3.8	3.9	3.9	**3.8**	**3.8**
2scale5	3.3	5.2	**3.4**	**3.4**	4.7	4.7	3.5	4.7	**3.4**	3.5	3.6	3.6	**3.4**	3.5
2scale*1	5.2	**5.2**	5.3	5.5	6.5	6.5	**5.5**	6.5	**5.5**	5.8	5.6	**5.4**	5.6	5.6
2scale*2	15.0	15.9	**15.0**	15.5	16.0	16.1	**15.4**	16.1	**15.4**	15.9	15.8	**15.3**	15.5	15.5
2scale*3	19.1	26.6	**19.1**	19.5	20.2	20.4	**19.4**	20.4	**19.4**	20.8	20.8	20.2	**19.3**	**19.3**
2scale*4	18.8	27.2	**18.9**	19.6	20.0	20.1	**19.1**	20.1	19.2	20.3	20.7	20.0	**19.1**	**19.1**
2scale*5	17.4	25.6	**17.5**	17.9	18.8	18.7	17.9	18.8	**17.8**	19.1	19.2	18.7	**17.7**	**17.7**
disks_100x100	0.0	49.3	35.5	**13.0**	24.2	24.2	21.1	24.2	**20.5**	15.3	16.0	11.6	**11.3**	13.0
disks_300x500	0.0	37.5	29.3	**5.7**	16.3	16.3	14.1	16.3	**12.7**	6.6	7.1	5.2	**5.0**	5.2
disks_400x400	0.0	49.9	30.8	**6.1**	18.5	18.4	17.5	18.4	**16.7**	7.0	7.4	5.7	**5.5**	**5.5**
disks_80x120	0.0	36.7	26.2	**11.8**	24.1	24.1	16.5	24.1	**16.4**	14.8	15.8	10.9	**10.8**	11.6

Table 2.2: Error rates (in %) of the pot-pot classifiers for simulated data sets.

dataset	Bayes	Joint					Separate					Regressive separate		
		diag.	α	k-NN	ROT	mM	diag.	α	k-NN	ROT	mM	diag.	α	k-NN
1dist1	30.9	**31.4**	32.2	33.3	32.7	**31.4**	**31.2**	31.8	33.2	32.3	**31.2**	**31.2**	31.8	33.2
1dist2	15.8	**16.2**	16.7	16.9	16.7	**16.2**	**16.0**	16.3	16.5	16.4	**16.0**	**16.0**	16.6	16.5
1dist3	6.7	**6.8**	7.1	7.2	7.1	**6.8**	**6.7**	6.9	7.0	7.0	6.8	**6.8**	6.9	7.1
1dist4	2.0	**2.4**	2.8	2.7	2.5	2.5	**2.3**	2.5	2.5	2.4	2.4	**2.3**	2.5	2.6
1rotate1	6.7	**6.8**	7.1	7.2	7.1	**6.8**	**6.7**	6.9	7.0	7.0	6.8	**6.8**	6.9	7.1
1rotate2	12.5	**13.4**	13.6	14.3	13.5	14.4	**13.0**	13.2	13.9	13.6	**13.0**	**13.0**	13.2	14.0
1rotate3	13.0	**13.8**	13.9	15.3	14.0	16.6	**13.1**	13.2	13.8	13.4	**13.1**	**13.1**	13.2	13.8
1rotate4	11.7	**12.5**	12.8	14.1	12.8	15.3	**11.6**	12.1	12.6	11.9	**11.6**	**11.6**	12.1	12.7
1rotate5	11.0	**12.2**	**12.2**	13.7	12.3	14.8	**11.2**	11.5	11.8	11.5	**11.2**	**11.2**	11.6	11.9
1rotate6	12.0	**13.0**	13.3	14.5	13.2	16.0	**11.9**	12.1	12.8	12.3	12.0	**11.9**	12.2	12.8
1rotate7	15.3	**16.2**	16.7	18.1	**16.2**	19.9	**15.0**	15.3	16.0	15.3	15.1	**15.0**	15.4	16.2
1rotate8	23.4	**25.0**	25.2	26.7	**25.0**	30.2	**23.8**	24.5	25.4	24.3	23.9	**23.8**	24.5	25.4
1rotate9	38.0	**38.6**	39.2	40.8	40.3	38.7	**39.5**	39.7	42.1	40.7	**39.5**	**39.5**	39.7	42.1
1scale1	6.7	**6.8**	7.1	7.2	7.1	**6.8**	**6.7**	6.9	7.0	7.0	6.8	**6.8**	6.9	7.1
1scale2	5.8	**6.2**	6.4	6.5	**6.2**	6.7	**6.0**	6.1	6.1	6.4	6.2	6.2	**6.1**	**6.1**
1scale3	4.9	**5.3**	5.5	5.5	5.4	5.8	**5.0**	5.1	5.1	5.4	5.2	5.6	**5.1**	**5.1**
1scale4	4.3	**4.7**	4.9	4.9	4.9	5.2	**4.2**	4.5	4.4	4.8	4.6	5.4	4.6	**4.4**
1scale5	3.7	**4.3**	4.5	4.5	4.5	4.8	**3.8**	4.0	4.1	4.4	4.0	5.1	**4.0**	4.2
1scale*1	6.6	**6.9**	7.1	7.2	7.0	**6.9**	**6.7**	7.0	7.1	7.3	**6.7**	**6.7**	7.0	7.1
1scale*2	14.4	**14.9**	**14.9**	15.4	**14.9**	15.0	**15.0**	15.2	15.3	16.0	15.3	16.4	**15.2**	15.3
1scale*3	17.0	**19.0**	19.2	20.1	19.1	19.5	17.8	**17.6**	18.2	18.7	17.9	**17.8**	**17.8**	18.2
1scale*4	16.6	**18.9**	19.7	20.1	19.9	21.9	**17.3**	**17.3**	18.1	19.0	17.4	17.9	**17.3**	18.1
1scale*5	15.2	**17.9**	18.9	18.7	19.1	21.6	**15.7**	15.8	16.6	17.2	16.0	16.7	**15.8**	16.7
2dist1	20.7	21.1	**21.0**	21.7	21.2	**21.0**	21.4	**21.2**	21.7	21.5	21.3	21.5	**21.2**	21.7
2dist2	12.1	**12.4**	**12.4**	12.6	12.5	12.5	**12.5**	12.6	12.6	**12.5**	12.7	**12.5**	12.6	12.6
2dist3	5.2	**5.4**	**5.4**	5.6	**5.4**	**5.4**	**5.6**	5.7	5.7	**5.6**	5.7	**5.6**	5.7	5.7
2dist4	1.9	**1.9**	**1.9**	2.0	2.0	2.0	**2.0**	**2.0**	**2.0**	2.1	2.1	**2.0**	**2.0**	2.2
2rotate1	5.2	**5.4**	**5.4**	5.6	**5.4**	**5.4**	**5.6**	5.7	5.7	**5.6**	5.7	**5.6**	5.7	5.7
2rotate2	8.7	**9.0**	9.2	9.3	9.1	9.6	**9.1**	9.2	9.4	**9.1**	9.3	**9.2**	**9.2**	9.4
2rotate3	9.0	**9.2**	9.4	9.7	9.4	12.0	**9.3**	9.4	9.5	9.4	9.5	**9.4**	**9.4**	9.6
2rotate4	8.0	**8.3**	8.4	8.7	8.4	10.5	**8.4**	**8.4**	8.5	**8.4**	**8.4**	**8.4**	**8.4**	8.6
2rotate5	7.7	**8.0**	8.1	8.4	8.2	10.1	**8.1**	8.2	8.2	**8.1**	8.2	**8.1**	8.2	8.2
2rotate6	8.3	**8.5**	8.7	8.9	8.7	10.7	**8.7**	**8.7**	8.8	**8.7**	8.8	**8.7**	8.8	8.9
2rotate7	10.5	**10.6**	10.8	11.0	10.8	13.4	**10.7**	**10.7**	10.9	10.8	10.8	10.8	**10.7**	10.9
2rotate8	16.2	**16.0**	**16.0**	16.5	**16.0**	20.0	16.2	**16.1**	16.2	16.2	16.2	16.2	**16.1**	16.3
2rotate9	22.9	**23.0**	23.1	23.2	23.2	23.1	**22.9**	23.0	23.1	23.2	23.1	**23.0**	**23.0**	23.1
2scale1	5.2	**5.4**	**5.4**	5.6	**5.4**	**5.4**	**5.6**	5.7	5.7	**5.6**	5.7	**5.6**	5.7	5.7
2scale2	4.6	**4.8**	**4.8**	5.0	**4.8**	5.0	5.4	**5.0**	5.1	5.1	5.1	5.5	**5.0**	5.2
2scale3	4.1	**4.2**	4.3	4.5	4.3	4.5	4.7	**4.3**	4.4	4.4	**4.3**	5.1	**4.3**	4.6
2scale4	3.7	**3.9**	**3.9**	4.2	**3.9**	4.4	4.1	**3.8**	**3.8**	3.9	3.9	4.3	**3.8**	4.0
2scale5	3.3	**3.6**	**3.6**	3.9	**3.6**	4.3	3.8	**3.4**	3.5	3.5	3.5	4.3	**3.5**	3.7
2scale*1	5.2	5.5	**5.4**	5.6	5.5	**5.4**	**5.6**	**5.6**	5.7	**5.6**	5.7	5.7	**5.6**	5.7
2scale*2	15.0	15.4	**15.3**	15.8	15.4	15.4	17.7	**15.5**	15.7	15.8	15.7	19.0	**15.5**	15.7
2scale*3	19.1	**20.2**	**20.2**	20.6	**20.2**	21.0	20.4	**19.3**	19.9	19.6	**19.3**	21.6	**19.3**	20.0
2scale*4	18.8	20.1	**20.0**	20.2	20.5	21.1	19.8	**19.1**	19.5	19.3	19.2	20.6	**19.1**	19.6
2scale*5	17.4	**18.6**	18.7	19.1	19.0	20.8	18.5	**17.7**	18.1	18.0	**17.7**	18.8	**17.7**	18.1
disks_100x100	0.0	11.5	11.6	**7.8**	22.9	**7.8**	11.8	11.3	**7.9**	22.3	8.1	12.4	13.0	**8.1**
disks_300x500	0.0	5.6	5.2	**2.9**	9.4	**2.9**	4.8	5.0	**3.4**	9.3	3.7	5.0	5.2	**3.5**
disks_400x400	0.0	5.7	5.7	**4.0**	10.8	**4.0**	5.7	5.5	**3.6**	11.8	3.8	5.8	5.5	**3.8**
disks_80x120	0.0	10.9	10.9	**6.7**	18.6	**6.7**	10.4	10.8	**7.0**	17.7	7.2	11.5	11.6	**7.2**

Table 2.3: Error rates (in %) of different classifiers for real data sets.

Columns (7) – (11) and (14) – (16): α-classifier in the DD- and pot-pot plots.

(1) dataset	(2) N	(3) d	(4) LDA	(5) QDA	(6) KNN	(7) Zonoid	(8) HS	(9) Mah	(10) Proj	(11) Spat	(12) Dknn HS	(13) Dknn Mah	(14) pot-pot joint	(15) pot-pot separate	(16) pot-pot regress. separate
baby	247	5	22.3	22.3	**21.5**	22.7	22.7	24.7	**21.5**	25.5	29.1	32.8	**20.2**	23.1	24.3
banknoten	200	6	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**	1.0	**0.5**	1.0	2.0	5.0	0.5	**0.0**	**0.0**
biomed	194	4	16.0	**12.4**	**12.4**	13.4	**11.3**	12.4	12.9	13.4	26.8	17.5	13.4	**9.3**	9.8
bloodtransfusion	748	3	23.0	22.3	**21.3**	23.1	24.3	**20.5**	23.5	21.8	21.3	20.7	19.9	**19.0**	19.7
breast_cancer_wisconsin	699	9	4.0	4.9	**3.1**	19.3	14.9	**3.6**	15.7	**3.7**	27.6	31.8	0.9	**0.7**	**0.7**
bupa	345	6	**30.1**	40.6	30.7	30.7	30.4	29.9	**27.8**	29.9	30.7	31.3	29.9	**29.0**	29.9
chemdiab_1vs2	112	5	**3.6**	7.1	9.8	**3.6**	**3.6**	**3.6**	**3.6**	**3.6**	11.6	8.0	**1.8**	2.4	3.3
chemdiab_1vs3	69	5	10.1	**8.7**	**8.7**	10.1	8.7	10.1	**7.2**	**7.2**	8.7	11.6	8.7	**5.8**	7.2
chemdiab_2vs3	109	5	3.7	**0.9**	**0.9**	3.7	3.7	**1.8**	3.7	**1.8**	6.4	11.0	0.9	**0.0**	**0.0**
cloud	108	7	54.6	**47.2**	54.6	54.6	50.9	**46.3**	52.8	48.1	40.7	50.0	39.8	**32.4**	39.8
crabB_MvsF	200	5	**9.0**	10.0	14.0	9.0	8.0	**6.0**	8.0	**6.0**	9.0	16.0	**5.0**	**5.0**	6.0
crabF_BvsO	200	5	**0.0**	1.0	5.0	**0.0**	**0.0**	1.0	**0.0**	1.0	2.0	1.0	**0.0**	**0.0**	**0.0**
crabM_BvsO	100	5	**0.0**	**0.0**	5.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2.0	3.0	**0.0**	**0.0**	**0.0**
crabO_MvsF	100	5	3.0	**2.0**	8.0	3.0	3.0	**2.0**	3.0	**2.0**	4.0	5.0	2.0	**1.0**	2.0
crab_BvsO	100	5	**0.0**	**0.0**	3.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.5	1.0	**0.0**	**0.0**	**0.0**
crab_MvsF	100	5	**4.0**	5.0	9.0	4.0	3.5	4.5	**3.0**	3.5	6.0	7.0	4.0	**3.5**	4.0
cricket_CvsP	156	4	68.6	64.1	**63.5**	**59.6**	62.8	64.7	64.1	60.9	56.4	55.1	56.4	**47.4**	**47.4**
diabetes	768	8	**22.4**	26.6	25.1	33.9	30.1	**24.6**	29.4	**24.7**	28.6	29.3	**22.1**	22.4	24.2
ecoli_cpvsim	220	5	**1.4**	1.8	1.8	**1.4**	2.3	**1.4**	2.3	**1.4**	14.5	9.5	**0.9**	**0.9**	1.8
ecoli_cpvspp	195	5	**3.1**	4.1	3.6	**4.1**	4.6	4.6	4.6	5.1	14.4	9.2	**3.1**	3.6	3.6
ecoli_imvspp	129	5	5.4	**3.9**	6.2	5.4	5.4	**2.3**	5.4	3.9	4.7	6.2	2.3	**1.0**	**1.1**
gemsen_MvsF	1349	6	19.1	14.2	**14.1**	14.8	16.4	14.8	16.5	**14.0**	10.8	12.5	10.9	**10.3**	10.7
glass	146	9	27.4	39.7	**18.5**	**27.4**	30.8	30.1	33.6	28.1	33.6	30.8	23.4	**17.8**	23.3
groessen_MvsF	230	3	10.9	**10.4**	15.7	**10.0**	12.6	10.9	12.2	10.9	12.6	12.2	10.4	**7.8**	9.6
haberman	306	3	25.2	**24.5**	**24.5**	26.8	28.1	27.1	26.5	**25.5**	28.8	29.7	**23.9**	**23.9**	25.2
heart	270	13	**16.3**	16.7	35.2	**16.3**	25.9	19.6	24.4	19.3	30.4	30.0	3.4	**0.0**	**0.0**
hemophilia	75	2	**14.7**	16.0	18.7	16.0	**13.3**	17.3	**13.3**	16.0	16.0	13.3	**12.0**	**12.0**	**12.0**
indian_liver_patient_1vs2	579	10	**29.7**	44.6	32.0	29.4	30.4	30.7	30.1	**28.5**	28.2	28.3	27.9	**25.0**	26.5
indian_liver_patient_FvsM	579	9	**24.5**	63.0	25.4	24.9	24.9	**24.7**	26.1	25.7	24.7	24.2	24.4	**22.7**	**22.7**
iris_setosavsversicolor	100	4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
iris_setosavsvirginica	100	4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
iris_versicolorvsvirginica	100	4	**3.0**	4.0	6.0	**3.0**	**3.0**	**3.0**	**3.0**	6.0	3.0	7.0	**2.0**	**2.0**	3.0
irish_ed_MvsF	500	5	45.0	**43.4**	47.0	42.0	**40.4**	45.2	40.6	45.6	43.0	44.2	39.8	**37.4**	37.6
kidney	76	5	**28.9**	**28.9**	32.9	**28.9**	30.3	30.3	30.3	31.6	30.3	38.2	23.7	**15.8**	17.1
pima	200	7	**24.5**	27.5	29.5	**25.5**	27.5	28.5	26.0	30.0	30.5	35.5	25.0	**23.0**	23.5
plasma_retinol_MvsF	315	13	14.3	14.0	**13.7**	**14.3**	15.9	14.6	17.1	**14.3**	13.7	12.7	12.0	**9.1**	9.2
segmentation	660	10	8.2	9.4	**4.5**	6.8	6.1	9.2	**4.8**	8.8	5.9	4.2	**2.7**	**2.7**	**2.7**
socmob_IvsNI	1156	5	33.2	34.3	**32.5**	**27.9**	33.6	31.6	33.3	30.4	35.3	46.6	33.6	**28.8**	29.5
socmob_WvsB	1156	5	28.1	29.2	**18.9**	**17.5**	18.3	19.9	18.4	19.5	17.3	31.1	28.3	**16.4**	16.7
tae	151	5	**17.2**	19.9	24.5	**11.9**	13.2	17.2	17.2	17.2	25.8	18.5	**13.2**	13.9	14.6
tennis_MvsF	87	15	**41.4**	44.8	43.7	41.4	44.8	**36.8**	46.0	36.8	48.3	46.0	31.2	**20.0**	**20.0**
tips_DvsN	244	6	6.1	**3.7**	7.8	10.7	8.2	**3.3**	9.0	3.7	9.4	8.6	3.7	**2.9**	3.3
tips_MvsF	244	6	36.5	38.5	**32.4**	41.4	44.3	36.5	43.0	**38.1**	34.8	34.4	32.8	**28.7**	32.4
uscrime_SvsN	47	13	17.0	19.1	**10.6**	**17.0**	**17.0**	19.1	**17.0**	19.1	17.0	27.7	**0.0**	**0.0**	**0.0**
vertebral_column	310	6	**15.5**	17.4	16.5	15.8	15.8	**14.5**	17.4	15.8	15.8	16.5	15.2	**13.5**	14.5
veteran_lung_cancer	137	7	64.2	51.8	**50.4**	62.0	57.7	**47.4**	57.7	50.4	48.9	48.2	40.1	**29.2**	39.4
vowel_MvsF	990	13	0.1	0.7	**0.0**	**0.1**	24.0	0.4	24.8	0.5	0.3	0.2	**0.0**	**0.0**	**0.0**
wine_1vs2	130	13	**0.0**	0.8	5.4	**0.0**	3.1	1.5	2.3	1.5	6.9	38.5	**0.0**	**0.0**	**0.0**
wine_1vs3	107	13	**0.0**	**0.0**	13.1	**0.0**	0.9	**0.0**	1.9	**0.0**	4.7	19.6	**0.0**	**0.0**	**0.0**
wine_2vs3	119	13	0.8	**0.0**	23.5	0.8	4.2	**0.0**	4.2	**0.0**	10.1	11.8	**0.0**	**0.0**	**0.0**

Table 2.4: Error rates (in %) of the pot-pot classifiers for real data sets.

dataset	N	d	Joint					Separate					Regressive separate		
			diag.	α	k-NN	ROT	mM	diag.	α	k-NN	ROT	mM	diag.	α	k-NN
baby	247	5	25.9	**20.2**	24.3	31.6	21.5	24.3	**23.1**	**23.1**	33.2	23.9	25.1	24.3	**23.1**
banknoten	200	6	0.5	0.5	**0.0**	2.0	**0.0**	**0.0**	**0.0**	**0.0**	1.0	**0.0**	**0.0**	**0.0**	**0.0**
biomed	194	4	14.4	**13.4**	14.9	16.5	14.4	10.8	**9.3**	10.8	11.9	12.4	13.4	**9.8**	10.8
bloodtransfusion	748	3	21.3	**19.9**	20.5	20.7	21.4	20.1	**19.0**	20.5	20.5	21.5	20.1	**19.7**	20.9
breast_cancer_wisconsin	699	9	**0.9**	**0.9**	3.6	4.1	**0.9**	**0.7**	**0.7**	3.0	8.0	2.5	**0.7**	**0.7**	3.9
bupa	345	6	30.1	**29.9**	30.4	31.3	32.2	30.1	**29.0**	30.1	31.9	31.0	32.8	**29.9**	30.1
chemdiab_1vs2	112	5	3.9	**1.8**	2.7	8.0	3.9	**2.4**	**2.4**	3.6	7.1	3.3	**2.5**	3.3	4.5
chemdiab_1vs3	69	5	11.6	**8.7**	**8.7**	13.0	**8.7**	5.8	5.8	**4.3**	13.0	7.2	**7.2**	**7.2**	**7.2**
chemdiab_2vs3	109	5	4.8	**0.9**	3.7	6.4	**0.9**	**0.0**	**0.0**	**0.0**	8.3	**0.0**	**0.0**	**0.0**	**0.0**
cloud	108	7	40.7	39.8	**0.0**	42.6	**0.0**	38.9	32.4	**0.0**	38.0	**0.0**	38.9	39.8	**17.1**
crabB_MvsF	200	5	10.0	5.0	**0.0**	12.0	**0.0**	6.0	5.0	**0.0**	10.0	**0.0**	9.0	6.0	**0.0**
crabF_BvsO	200	5	**0.0**	**0.0**	**0.0**	2.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
crabM_BvsO	100	5	**0.0**	**0.0**	**0.0**	2.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
crabO_MvsF	100	5	3.0	2.0	**0.0**	4.0	**0.0**	1.0	1.0	**0.0**	3.0	**0.0**	2.0	2.0	**0.0**
crab_BvsO	100	5	0.6	**0.0**	**0.0**	1.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
crab_MvsF	100	5	5.0	4.0	**0.0**	6.5	**0.0**	4.0	3.5	**0.0**	6.5	**0.0**	5.0	4.0	**0.0**
cricket_CvsP	156	4	67.9	56.4	**34.8**	73.7	**34.8**	48.1	47.4	**34.8**	74.4	**34.8**	48.7	**47.4**	54.5
diabetes	768	8	26.2	**22.1**	23.0	27.6	22.7	24.3	**22.4**	23.8	28.5	24.9	25.5	**24.2**	24.7
ecoli_cpvsim	220	5	2.7	**0.9**	1.8	3.2	**0.9**	1.4	**0.9**	**0.9**	2.7	2.3	**1.8**	**1.8**	**1.8**
ecoli_cpvspp	195	5	3.6	**3.1**	3.6	3.6	4.1	**3.6**	**3.6**	**3.6**	4.6	4.1	**3.6**	**3.6**	**3.6**
ecoli_imvspp	129	5	2.9	**2.3**	3.1	3.9	2.9	**1.0**	**1.0**	3.1	3.1	1.1	**1.1**	**1.1**	3.1
gemsen_MvsF	1349	6	10.6	10.9	**10.5**	**10.5**	13.3	**10.2**	10.3	10.5	**10.2**	13.4	11.0	10.7	**10.5**
glass	146	9	24.1	23.4	**21.5**	24.1	27.1	22.4	**17.8**	25.3	26.9	27.8	26.2	**23.3**	25.3
groessen_MvsF	230	3	10.0	10.4	**9.6**	10.9	10.4	8.3	7.8	**7.4**	11.7	10.0	**9.1**	9.6	9.6
haberman	306	3	25.8	**23.9**	**23.9**	27.1	25.5	24.5	**23.9**	**23.9**	25.5	26.1	25.2	25.2	**24.8**
heart	270	13	**3.4**	**3.4**	17.4	23.7	5.3	**0.0**	**0.0**	12.5	23.7	**0.0**	**0.0**	**0.0**	12.5
hemophilia	75	2	12.0	12.0	**9.3**	12.0	14.7	13.3	12.0	**9.3**	13.3	16.0	13.3	12.0	**10.7**
indian_liver_patient_1vs2	579	10	28.4	27.9	**27.8**	29.1	28.5	26.0	**25.0**	25.9	26.8	28.5	**26.3**	26.5	**26.3**
indian_liver_patient_FvsM	579	9	**24.2**	24.4	**24.2**	24.4	**24.2**	**22.7**	**22.7**	22.8	24.3	24.0	**22.7**	**22.7**	23.5
iris_setosavsversicolor	100	4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
iris_setosavsvirginica	100	4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
iris_versicolorvsvirginica	100	4	2.0	2.0	**0.0**	4.0	**0.0**	3.8	2.0	**0.0**	5.0	**0.0**	3.8	3.0	**0.0**
irish_ed_MvsF	500	5	42.8	39.8	**38.5**	42.6	**38.5**	39.2	**37.4**	38.4	40.0	40.0	41.0	**37.6**	39.0
kidney	76	5	26.0	23.7	**14.9**	23.7	**14.9**	**13.2**	15.8	13.8	25.0	13.8	**13.2**	17.1	13.8
pima	200	7	25.5	25.0	**20.7**	28.5	**20.7**	25.5	23.0	**21.4**	30.5	22.0	25.5	23.5	**21.4**
plasma_retinol_MvsF	315	13	12.0	12.0	**8.3**	16.2	12.7	**9.1**	**9.1**	**9.1**	14.3	13.3	**9.2**	**9.2**	**9.2**
segmentation	660	10	**2.7**	**2.7**	4.5	5.0	**2.7**	**2.7**	**2.7**	5.0	5.2	2.8	**2.7**	**2.7**	5.0
socmob_IvsNI	1156	5	40.3	33.6	**32.4**	45.9	33.7	31.1	**28.8**	30.6	30.4	31.7	31.2	**29.5**	30.6
socmob_WvsB	1156	5	30.5	**28.3**	28.4	29.5	28.7	17.1	**16.4**	17.0	17.0	20.8	17.7	**16.7**	17.0
tae	151	5	16.6	**13.2**	13.9	15.9	14.4	14.6	13.9	**12.6**	15.2	14.5	14.6	14.6	**13.2**
tennis_MvsF	87	15	**31.2**	**31.2**	32.2	36.8	**31.2**	**20.0**	**20.0**	26.3	34.5	30.0	**20.0**	**20.0**	26.3
tips_DvsN	244	6	4.1	**3.7**	4.5	6.6	**3.7**	3.7	**2.9**	**2.9**	5.3	3.3	4.1	**3.3**	4.5
tips_MvsF	244	6	35.7	**32.8**	33.8	43.0	33.8	33.2	28.7	**32.0**	39.8	33.6	33.6	**32.4**	32.8
uscrime_SvsN	47	13	**0.0**	**0.0**	**0.0**	17.0	12.8	**0.0**	**0.0**	**0.0**	19.1	17.0	**0.0**	**0.0**	**0.0**
vertebral_column	10	6	17.4	**15.2**	16.5	17.4	**15.2**	16.5	**13.5**	14.2	17.1	15.2	19.7	14.5	**14.2**
veteran_lung_cancer	37	7	43.0	**40.1**	40.9	43.1	44.6	42.1	**29.2**	33.6	43.1	44.5	42.4	39.4	**35.0**
vowel_MvsF	990	13	**0.0**	**0.0**	0.1	0.1	**0.0**	**0.0**	**0.0**	0.1	0.2	**0.0**	**0.0**	**0.0**	0.1
wine_1vs2	130	13	**0.0**	**0.0**	0.8	6.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1.5	**0.0**	**0.0**	**0.0**
wine_1vs3	107	13	**0.0**	**0.0**	**0.0**	2.8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
wine_2vs3	119	13	**0.0**	**0.0**	1.7	5.0	1.7	**0.0**	**0.0**	**0.0**	1.7	**0.0**	**0.0**	**0.0**	**0.0**

Chapter 3

Depth and depth-based classification with R-package ddalpha

3.1 Introduction

Consider the following setting for supervised classification: Given a training sample consisting of q classes $\boldsymbol{X}_1, ..., \boldsymbol{X}_q$, each containing n_i, $i = 1, ..., q$, observations in $\mathbb{R}^d$. For a new observation $\boldsymbol{x}_0$, a class should be determined, to which it most probably belongs. Depth-based learning started with plug-in type classifiers. Ghosh and Chaudhuri (2005b) construct a depth-based classifier, which, in its naïve form, assigns the observation $\boldsymbol{x}_0$ to the class in which it has maximal depth. They suggest an extension of the classifier, that is consistent w.r.t. Bayes risk for classes stemming from elliptically symmetric distributions. Further Dutta and Ghosh (2011, 2012) suggest a robust classifier and a classifier for L_p-symmetric distributions, see also Cui et al. (2008), Mosler and Hoberg (2006), and additionally Jörnsten (2004) for unsupervised classification.

A novel way to perform depth-based classification has been suggested by Li et al. (2012): first map a pair of training classes into a two-dimensional depth space, which is called the DD-plot, and then perform classification by selecting a polynomial that minimizes empirical risk. Finding such an optimal polynomial numerically is a very challenging and — when done appropriately — computationally involved task, with a solution that in practice can be unstable (see Mozharovskyi, 2015, Section 1.2.2 for examples). In addition, the DD-plot should be rotated and the polynomial training phase should be done twice. Nevertheless, the scheme itself allows to construct optimal classifiers for wider classes of distributions than the elliptical family. Being further developed and applied by Vencalek (2011), Lange et al. (2014b), Mozharovskyi et al. (2015) it proved to be useful in practice, also in the functional setting (Mosler and Mozharovskyi, 2015, Cuesta-Albertos et al., 2016).

The general depth-based supervised classification framework implemented in the R-package **ddalpha** can be described as follows. In the first part of the training phase, each point of the training sample is mapped into the q-variate space of its depth values with respect to each of the classes $\boldsymbol{x}_i \mapsto (D(\boldsymbol{x}_i|\boldsymbol{X}_1), ..., D(\boldsymbol{x}_i|\boldsymbol{X}_q))$. In the second part of the training phase, a low-

dimensional classifier, flexible enough to account for the change in data topology due to the depth transform, is employed in the depth space. We suggest to use the α-procedure, which is a nonparametric, robust, and computationally efficient separator. When classifying an unknown point $\boldsymbol{x}_0$, the first part is the same as in the training phase, $(\boldsymbol{x}_0 \mapsto (D(\boldsymbol{x}_0|\boldsymbol{X}_1), ..., D(\boldsymbol{x}_0|\boldsymbol{X}_q)))$, and in the second part the trained q-variate separator assigns the depth-transformed point to one of the classes. Depth notions best reflecting data geometry share the common feature to attain value zero immediately beyond the convex hull of the data cloud. Thus, if such a data depth is used in the first phase, it may happen that $\boldsymbol{x}_0$ is mapped to the origin of the depth space, and thus cannot be readily classified. We call such a point an *outsider* and suggest to apply a special treatment to assign it. If the data is of functional nature, a finitization step based on the *location-slope (LS-) transform* precedes the above described process. Depth transform, α-procedure, outsider treatment, and the preceding LS-transform constitute the $DD\alpha$-classifier. This together with the depth-calculating machinery constitutes the heart of the R-package **ddalpha**.

3.1.1 The R-package ddalpha

The R-package **ddalpha** is a software directed to fuse experience of the applicant with recent theoretical and computational achievements in the area of data depth and depth-based classification. It provides an implementation for exact and approximate computation of seven most reasonable and widely applied depth notions: Mahalanobis, halfspace, zonoid, projection, spatial, simplicial and simplicial volume depths. The variety of depth-calculating procedures includes functions for computation of data depth of one or more points w.r.t. a data set, construction of the classification-ready q-dimensional depth space, visualization of the bivariate depth function for a sample in the form of upper-level contours and of a 3D-surface.

The main feature of the proposed methodology on the DD-plot is the $DD\alpha$-classifier, which is an adaptation of the α-procedure to the depth space. Except for its efficient and fast implementation, **ddalpha** suggests other classification techniques that can be employed in the DD-plot: the original polynomial separator by Li et al. (2012) and the depth-based k-NN-classifier proposed by Vencalek (2011).

Halfspace, zonoid and simplicial depths vanish beyond the convex hull of the sample, and thus cause outsiders during classification. For this case, **ddalpha** offers a number of outsider treatments and a mechanism for their management.

If it is decided to employ the DD-classifier, its constituents are to be chosen: data depth, classification technique in the depth space, and, if needed, outsider treatment and aggregation scheme for multi-class classification. Their parameters, such as type and subset size of the variance-covariance estimator for Mahalanobis and spatial depth, number of approximating directions for halfspace and projection depth or fraction of simplices for approximating simplicial and simplicial volume depths, degree of polynomial extension for the α-procedure or the polynomial classifier, number of nearest neighbors in the depth space or for an outsider treatment, *etc.* must be set. Rich built-in benchmark procedures allow to estimate the empir-

ical risk and error rates of the DD-classifier and the portion of outsiders help in making the decision concerning the settings.

ddalpha possesses tools for immediate classification of functional data in which the measurements are first brought onto a finite dimensional basis, and then fed to the depth-classifier. In addition, the componentwise classification technique by Delaigle et al. (2012) is implemented.

Unlike other packages, **ddalpha** implements various depth functions and classifiers for multivariate and functional data under one roof. **ddalpha** is the only package that implements zonoid depth and efficient exact halfspace depth. All depths in the package are implemented for any dimension $d \geq 2$; except for the projection depth all implemented algorithms are exact, and supplemented by their approximating versions to deal with the increasing computational burden for large samples and higher dimensions. It also supports user-friendly definitions of depths and classifiers. In addition, the package contains 50 multivariate and 4 functional ready-to-use classification problems and data generators for a palette of distributions.

Most of the functions of the package are programmed in C++, in order to be fast and efficient. The package has a module structure, which makes it expandable and allows user-defined custom depth methods and separators. **ddalpha** employs **boost** (package **BH** Eddelbuettel, Emerson and Kane (2016)), a well known fast and widely applied library, and resorts to **Rcpp** (Eddelbuettel, Francois, Allaire, Ushey, Kou, Bates and Chambers, 2016) allowing for calls of R functions from C++.

3.1.2 Comparison to existing implementations

Having proved to be useful in many areas, data depth and its applications find implementation in a number of R-packages: **aplpack** (Wolf, 2014), **depth** (Genest et al., 2012), **localdepth** (Agostinelli et al., 2013), **fda.usc** (Febrero-Bande and Oviedo de la Fuente, 2012), **rsdepth** (Mustafa et al., 2014), **depthTools** (Lopez-Pintado and Torrente, 2013), **MFHD** (Hubert and Vakili, 2013), **depth.plot** (Mahalanobish and Karmakar, 2015), **DepthProc** (Kosiorowski et al., 2016), **WMTregions** (Bazovkin, 2013), **modQR** (Šiman and Boček, 2016), **OjaNP** (Fischer et al., 2016), and MATLAB-packages: **CompPD** (Liu and Zuo, 2015) and **modQR** (Boček and Šiman, 2016), which suggest substantial possibilities. Out of this diversity, we concentrate on the two main aspects to which the package **ddalpha** is devoted, namely computation of the multivariate data depth function and depth-based supervised classification.

Regarding the depth calculation, the wide range of the possibilities of the package **ddalpha** can be better seen in comparison with the existing functionality on calculation of data depth. Being a monotone transformation of the Mahalanobis distance, Mahalanobis depth can be programmed in a few script lines, and due to its wide spread is implemented in numerous software packages, among others, *e.g.*, **DepthProc**, **localdepth**, **fda.usc** from the above list. The R-package **ddalpha** adds a possibility to compute robust Mahalanobis depth using MCD estimates for mean and covariance matrix. Spatial depth can be computed using the R-package **depth.plot** that also provides spatial ranks and constructs corresponding DD-plots; different to it **ddalpha** allows to compute affine-invariant spatial depth, while the affine invariance can

be accounted for in a robust way. Simplicial depth can be calculated exactly by the R-packages **depth** and **fda.usc** for bivariate data sets, and by the R-package **localdepth** in higher dimensions, while **depth** also provides an implementation for exact simplicial volume depth in any dimension. To avoid the enormous burden of exact computation, **ddalpha** additionally suggests a possibility to approximate both depths, either keeping computation time constant (in n, given d) or maintaining calculation precision on the same level. Projection depth and associated estimators can be computed exactly using the MATLAB-package **CompPD**. While exact computation of the projection depth even for moderate data sets is infeasible, one can make use of its approximation by minimizing over univariate projections on random directions, implemented in the R-packages **DepthProc** and **fda.usc**. **ddalpha** only approximates the projection depth, but does it not only by random projections but using a fast local optimization algorithm as well.

Most distinctive is the computation of the halfspace and zonoid depths. For $d \leq 3$, exact halfspace depth can be calculated by the R-package **depth**. Pioneering for $d > 3$, Liu and Zuo (2014a) construct an exact algorithm which regards all necessary halfspaces exploiting the idea of the cone segmentation of the Euclidean space, whose MATLAB-implementation can be obtained upon request from the authors. Regarding these halfspaces in a combinatorial order, Dyckerhoff and Mozharovskyi (2016) propose an entire family of algorithms, which are sizeably more efficient (*e.g.*, 16.1 seconds on Intel Core i7-2600 3.4 GHz against 10 hours on Intel Pentium Duo 2.0 GHz when computing depth of a single point w.r.t. a sample of $n = 160$ and $d = 5$) and do not require data to be in general position. Three most important cases of this family are implemented in **ddalpha**. **ddalpha** is the only software providing an (efficient) exact implementation for zonoid depth, by means of linear programming (Dyckerhoff et al., 1996).

ddalpha not only contains a comprehensive implementation of seven most popular depth notions, but also provides a possibility to define a new (or extend an existing) depth function corresponding exactly to the user's needs, and integrates this in a generic way in further procedures implemented in the package. Thus for any depth function, **ddalpha** plots depth contours and the surface of the depth function for bivariate data set, but also provides the entire implemented classification machinery. It is worth to note here that computation of depth contours for $d \geq 3$, as well as their visualization, are implemented for the weighted-mean trimmed regions (R-package **WMTregions**) and zonoid depth as their particular case, multiple-output regression quantiles (MATLAB- and R-packages **modQR** and halfspace depth as their particular case and projection depth (MATLAB-package **CompPD**). Further, **ddalpha** does not include median-search algorithms (*i.e.*, finding the deepest location(s)) implemented, *e.g.*, in **OjaNP** for simlplicial volume depth, **rsdepth** for the ray shooting depth, **MFHD** for bivariate functional halfspace depth. Depth notions and accompanying statistics developed for functional data can be calculated in such R-packages as **DepthProc**, **fda.usc**, **depthTools**, **MFHD**, and do not constitute the content of the current thesis.

The main feature of **ddalpha** is the unified DD-plot based framework for depth-based classification, which allows for choosing the data depth to construct the DD-plot, the multivariate classifier to employ there, the treatment for points not handled by the depth if this is the case, and the discretization scheme for projection of functional data onto a finite-dimensional basis and the aggregation scheme for multi-class classification if needed. Together with a variety of depths, multivariate separators, and outsider treatments, **ddalpha** contains tools for visualization and validation of classification results, and has by that no analogs. DD-plot-based techniques employing functional depths and suited for supervised classification of functional data are implemented in the R-package **fda.usc**.

3.1.3 Outline of the chapter

To facilitate understanding and keep the presentation solid, the functionality of the R-package **ddalpha** is illustrated through the chapter on the same functional data set "ECG Five Days" from Chen et al. (2015), which is a long ECG time series constituting two classes. The data set originally contains 890 objects. We took a subset consisting of 70 objects only (35 from each of the days) which best demonstrates the general and complete aspects of the proposed procedures (*e.g.*, existence of outliers in its bivariate projection or necessity of three features in the α-procedure).

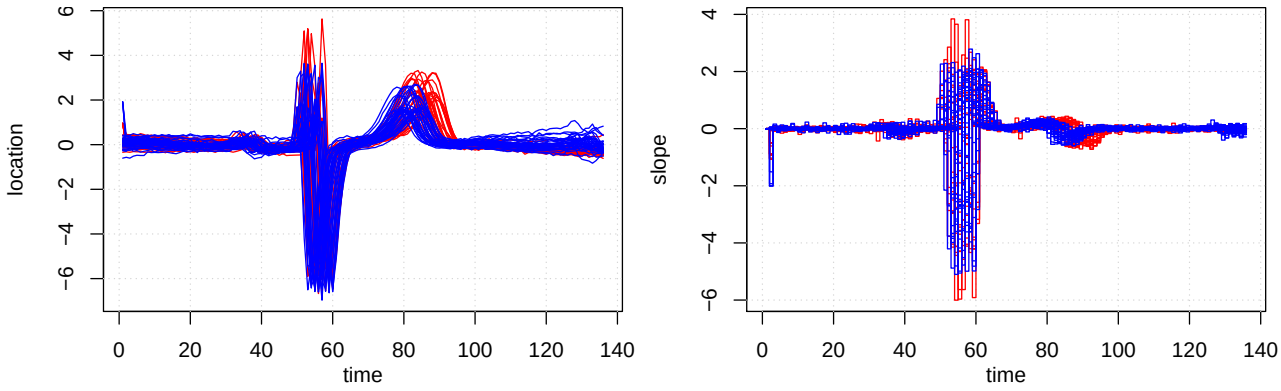


Figure 3.1: ECG Five Days data (left) and their derivatives (right).

In Section 3.5 functional data are transformed into a finite dimensional space using the LS-transform, as it is shown in Figure 3.1 presenting the functions and their derivatives. In this example we choose $L = S = 1$, and thus the LS-transform produces the two-dimensional discrete space, where each function is described by the area under the function and under its derivative as it is shown in Figure 3.6, left. Then in Section 3.2, the depth is calculated in this two-dimensional space (Figure 3.6, middle) and the DD-plot is constructed in Section 3.3 (Figure 3.6, right). The classification is performed by the $DD\alpha$-separator in the DD-plot. The steps of the α-procedure are illustrated in Figure 3.7.

Section 3.2 presents a theoretical description of the data depth and the depth notions implemented in the package. In addition, it compares their computation time and performance

when employed in the maximum depth classifier. Section 3.3 includes a comprehensive algorithmic description of the $DD\alpha$-classifier with a real-data illustration. Further, it discusses other classification techniques that can be employed in the DD-plot. The questions whether one should choose a depth that avoids outsiders or should allow for outsiders and classify them separately, and in which way, are considered in Section 3.4. Section 3.5 addresses the classification of functional data. In Section 3.6, the basic structure and concepts of the R-package user interface are presented, along with a discussion of their usage for configuring the classifier and examples for calling its functions.

3.2 Data depth

This section regards depth functions. First (Section 3.2.1), we briefly review the concept of data depth and its fundamental properties. Then (Section 3.2.2), we give the definitions in their empirical versions for several depth notions: Mahalanobis, projection, spatial, halfspace, simplicial, simplicial volume, zonoid depths. For each notion, we shortly discuss relevant computational aspects, leaving motivations, ideas, and details to the corresponding literature and the software manual. We do not touch the question of computation of depth-trimmed regions for the following reasons: first, for a number of depth notions there exist no algorithms; then, for some depth notions these can be computed using different R-packages, *e.g.*, **WMTregions** for the family of weighted-mean regions including zonoid depth (Bazovkin and Mosler, 2012) or **modQR** for multiple-output quantile regression including halfspace depth as a particular case; finally, this is not required in classification. After having introduced depth notions, we compare the speed of the implemented exact algorithms by means of simulated data (Section 3.2.3). The section is concluded (Section 3.2.4) by a comparison of error rates of the naïve maximum depth classifier, paving a bridge to the more developed DD-plot classification which is covered in the following sections.

3.2.1 The concept

Consider a point $\boldsymbol{z} \in \mathbb{R}^d$ and a data sample $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)^\top$ in the d-dimensional Euclidean space, with $\boldsymbol{X}$ being a $(n \times d)$-matrix and $^\top$ being the transposition operation. A data depth is a function $D(\boldsymbol{z}|\boldsymbol{X}) : \mathbb{R}^d \mapsto [0, 1]$ that describes how deep, or central, the observation $\boldsymbol{z}$ is located w.r.t. $\boldsymbol{X}$. In a natural way, it involves some notion of center. This is any point of the space attaining the highest depth value in $\boldsymbol{X}$, and not necessarily a single one. In this view, depth can be seen as a center-outward ordering, *i.e.*, points closer to the center have a higher depth, and those more outlying a smaller one.

The concept of a depth function can be formalized by stating postulates (requirements) it should satisfy. Following Dyckerhoff (2004) and Mosler (2013), a *depth function* is a function $D(\boldsymbol{z}|\boldsymbol{X}) : \mathbb{R}^d \mapsto [0, 1]$ that is:

(*D1*) *translation invariant*: $D(\boldsymbol{z}+\boldsymbol{b}|\boldsymbol{X}+\mathbf{1}_n\boldsymbol{b}^\top) = D(\boldsymbol{z}|\boldsymbol{X})$ for all $\boldsymbol{b} \in \mathbb{R}^d$ (here $\mathbf{1}_n = (1, ..., 1)^\top$),

(*D2*) *linear invariant*: $D(\boldsymbol{A}\boldsymbol{z}|\boldsymbol{X}\boldsymbol{A}^\top) = D(\boldsymbol{z}|\boldsymbol{X})$ for every nonsingular $d \times d$ matrix $\boldsymbol{A}$,

(*D3*) *zero at infinity*: $\lim_{\|\boldsymbol{z}\|\to\infty} D(\boldsymbol{z}|\boldsymbol{X}) = 0$,

(*D4*) *monotone on rays*: Let $\boldsymbol{z}^* = \operatorname{argmax}_{\boldsymbol{z}\in\mathbb{R}^d} D(\boldsymbol{z}|\boldsymbol{X})$, then for all $\boldsymbol{r} \in S^{d-1}$ the function $\beta \mapsto D(\boldsymbol{z}^* + \beta\boldsymbol{r}|\boldsymbol{X})$ decreases in the weak sense, for $\beta > 0$,

(*D5*) *upper semicontinuous*: the upper level sets $D_\alpha(\boldsymbol{X}) = \{\boldsymbol{z} \in \mathbb{R}^d : D(\boldsymbol{z}|\boldsymbol{X}) \geq \alpha\}$ are closed for all α.

For slightly different postulates see Liu (1992) and Zuo and Serfling (2000).

The first two properties state that $D(\cdot|\boldsymbol{X})$ is *affine invariant*. $\boldsymbol{A}$ in (D2) can be weakened to isometric linear transformations, which yields an *orthogonal invariant* depth. Taking instead of $\boldsymbol{A}$ some constant $\lambda > 0$ gives a *scale invariant* depth function. (D3) ensures that the upper level sets D_α, $\alpha > 0$, are bounded. According to (D4), the upper level sets are starshaped around $\boldsymbol{z}^*$, and $D_{\max_{\boldsymbol{z}\in\mathbb{R}^d} D(\boldsymbol{z}|\boldsymbol{X})}(\boldsymbol{X})$ is convex. (D4) can be strengthened by requiring $D(\cdot|\boldsymbol{X})$ to be a *quasiconcave* function. In this case, the upper level sets are convex for all $\alpha > 0$. (D5) is a useful technical restriction.

Upper level sets $D_\alpha(\boldsymbol{X}) = \{\boldsymbol{x} \in \mathbb{R}^d : D(\boldsymbol{x}|\boldsymbol{X}) \geq \alpha\}$ of a depth function are also called *depth-trimmed* or *central regions*. They describe the distribution's location, dispersion, and shape. For given $\boldsymbol{X}$, the sets $D_\alpha(\boldsymbol{X})$ constitute a nested family of trimming regions. Note that due to (D1) and (D2) the central regions are affine equivariant, due to (D3) bounded, due to (D5) closed, and due to (D4) star-shaped (respectively convex, if quasiconcaveness of $D(\cdot|\boldsymbol{X})$ is additionally required).

3.2.2 Implemented notions

The R-package **ddalpha** implements a number of depths. Below we consider their empirical versions. For each implemented notion of data depth, the depth surface (left) and depth contours (right) are plotted in Figures 3.2 and 3.3 for bivariate data used in Section 3.5.

Mahalanobis depth is based on an outlyingness measure, *viz.* the Mahalanobis distance (Mahalanobis, 1936) between $\boldsymbol{z}$ and a center of $\boldsymbol{X}$, $\boldsymbol{\mu}(\boldsymbol{X})$ say:

$$d^2_{Mah}\big(\boldsymbol{z}; \boldsymbol{\mu}(\boldsymbol{X}), \boldsymbol{\Sigma}(\boldsymbol{X})\big) = \big(\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{X})\big)^\top \boldsymbol{\Sigma}(\boldsymbol{X})^{-1}\big(\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{X})\big).$$

The depth of a point $\boldsymbol{z}$ w.r.t. $\boldsymbol{X}$ is then defined as (Liu, 1992)

$$D_{Mah}(\boldsymbol{z}|\boldsymbol{X}) = \frac{1}{1 + d^2_{Mah}\big(\boldsymbol{z}; \boldsymbol{\mu}(\boldsymbol{X}), \boldsymbol{\Sigma}(\boldsymbol{X})\big)}, \tag{3.1}$$

where $\boldsymbol{\mu}(\boldsymbol{X})$ and $\boldsymbol{\Sigma}(\boldsymbol{X})$ are appropriate estimates of mean and covariance of $\boldsymbol{X}$. This depth function obviously satisfies all the above postulates and is quasi-concave, too. It can be regarded as a *parametric depth* as it is defined by a finite number of parameters (namely $\frac{d(d+1)}{2}$). Based on the two first moments, its depth contours are always ellipsoids centered at $\boldsymbol{\mu}(\boldsymbol{X})$, and

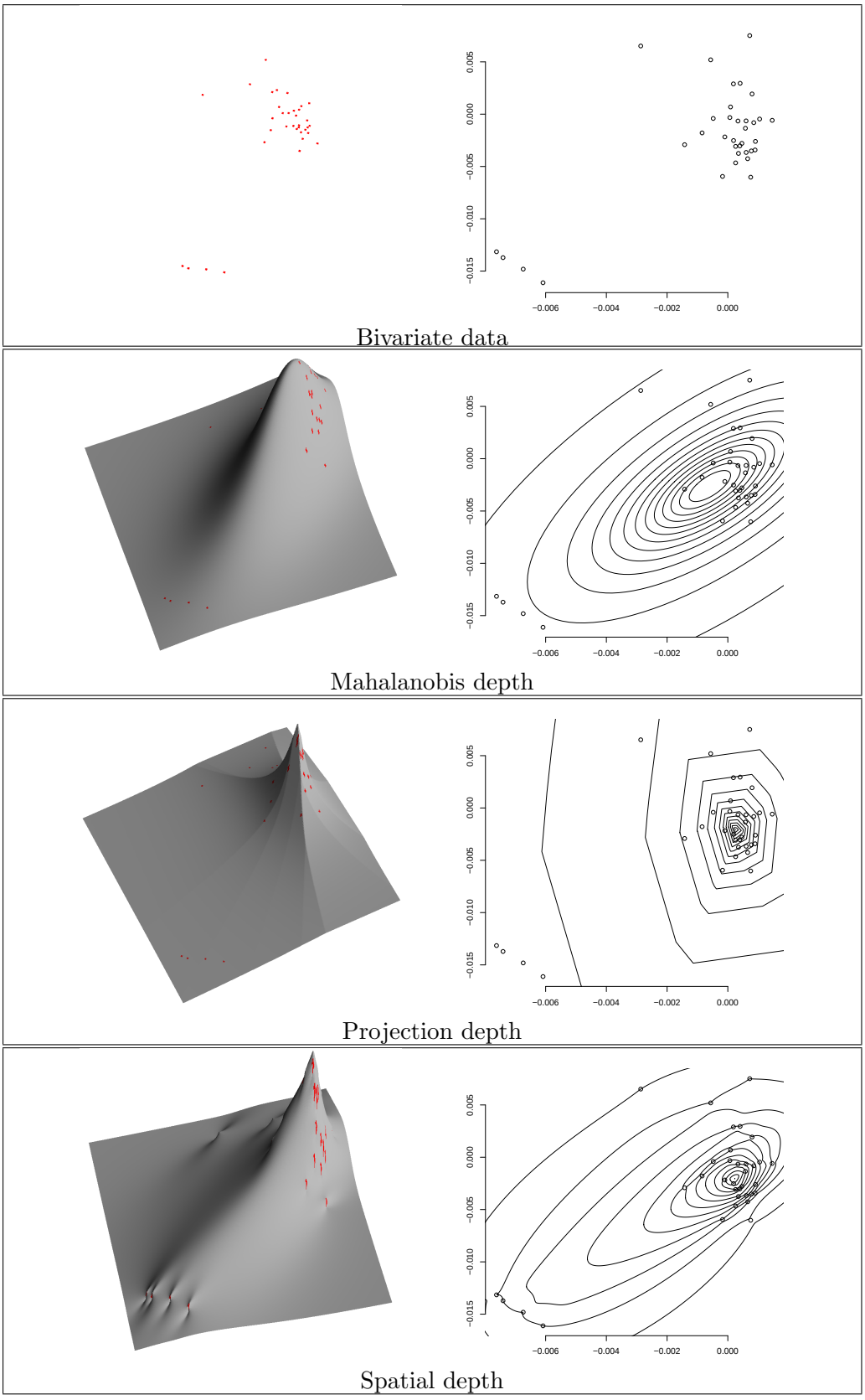


Figure 3.2: Depth plots and contours of bivariate data.

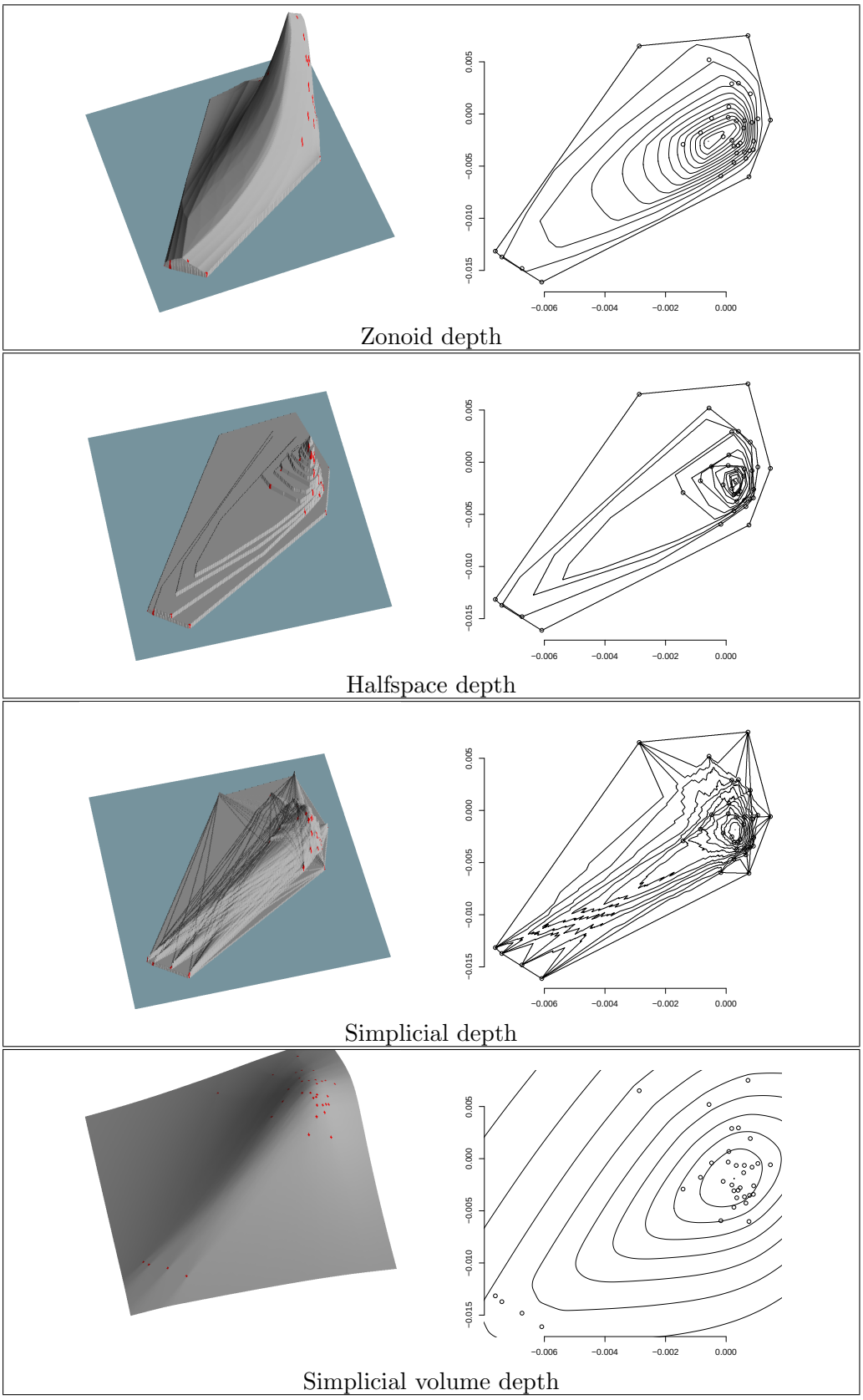


Figure 3.3: Depth plots and contours of bivariate data.

thus independent of the shape of $\boldsymbol{X}$. If $\boldsymbol{\mu}(\boldsymbol{X})$ and $\boldsymbol{\Sigma}(\boldsymbol{X})$ are chosen to be moment estimates, *i.e.*, $\boldsymbol{\mu}(\boldsymbol{X}) = \frac{1}{n}\boldsymbol{X}^\top \mathbf{1}_n$ being the traditional *average* and $\boldsymbol{\Sigma}(\boldsymbol{X}) = \frac{1}{n-1}(\boldsymbol{X} - \mathbf{1}_n\boldsymbol{\mu}(\boldsymbol{X})^\top)^\top(\boldsymbol{X} - \mathbf{1}_n\boldsymbol{\mu}(\boldsymbol{X})^\top)$ being the *empirical covariance matrix*, the corresponding depth may be sensitive to outliers. A more robust depth is obtained with the *minimum covariance determinant* (MCD) estimator, see Rousseeuw and Leroy (1987).

Calculation of the Mahalanobis depth consists in estimation of the center vector $\boldsymbol{\mu}(\boldsymbol{X})$ and the inverse of the scatter matrix $\boldsymbol{\Sigma}(\boldsymbol{X})$. In the simplest case of traditional moment estimates the time complexity amounts to $O(nd^2 + d^3)$ only. Rousseeuw and Van Driessen (1999) develop an efficient algorithm for computing robust MCD estimates.

Projection depth, similar to Mahalanobis depth, is based on a measure of outlyingness. See Stahel (1981), Donoho (1982), and also Liu (1992), Zuo and Serfling (2000). The worst case outlyingness is obtained by maximizing an outlyingness measure over all univariate projections:

$$o_{prj}(\boldsymbol{z}|\boldsymbol{X}) = \sup_{\boldsymbol{u}\in S^{d-1}} \frac{|\boldsymbol{z}^\top\boldsymbol{u} - m(\boldsymbol{X}^\top\boldsymbol{u})|}{\sigma(\boldsymbol{X}^\top\boldsymbol{u})},$$

with $m(\boldsymbol{y})$ and $\sigma(\boldsymbol{y})$ being any location and scatter estimates of a univariate sample $\boldsymbol{y}$. Taking $m(\boldsymbol{y})$ as the mean and $\sigma(\boldsymbol{y})$ as the standard deviation one gets the Mahalanobis outlyingness, due to the projection property (Dyckerhoff, 2004). In the literature and in practice most often *median*, $med(\boldsymbol{y}) = \boldsymbol{y}_{(\frac{\lfloor n+1\rfloor}{2})}$, and *median absolute deviation from the median*, $MAD(\boldsymbol{y}) = med(|\boldsymbol{y} - med(\boldsymbol{y})\mathbf{1}_n|)$, are used, as they are robust. Projection depth is then obtained as

$$D_{prj}(\boldsymbol{z}|\boldsymbol{X}) = \frac{1}{1 + o_{prj}(\boldsymbol{z}|\boldsymbol{X})}. \tag{3.2}$$

This depth satisfies all the above postulates and quasiconcavity. By involving the symmetric scale factor MAD its contours are centrally symmetric and thus are not well suited for describing skewed data.

Exact computation of the projection depth is a nontrivial task, which fast becomes intractable for large n and d. Liu and Zuo (2014b) suggest an algorithm (and a MATLAB implementation, see Liu and Zuo, 2015). In practice one may approximate the projection depth from above by minimizing it over projections on k random lines, which has time complexity $O(knd)$. It can be shown that finding the exact value is a zero-probability event though.

Spatial depth (also L_1-depth) is a distance-based depth formulated by Vardi and Zhang (2000) and Serfling (2002), exploiting the idea of spatial quantiles of Chaudhuri (1996) and Koltchinskii (1997). For a point $\boldsymbol{z} \in \mathbb{R}^d$, it is defined as one minus the length of the average direction from $\boldsymbol{X}$ to $\boldsymbol{z}$:

$$D_{spt}(\boldsymbol{z}|\boldsymbol{X}) = 1 - \Big\|\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{v}\big(\boldsymbol{\Sigma}^{-\frac{1}{2}}(\boldsymbol{X})(\boldsymbol{z} - \boldsymbol{x}_i)\big)\Big\|, \tag{3.3}$$

with $\boldsymbol{v}(\boldsymbol{y}) = \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|}$ if $\boldsymbol{y} \neq \mathbf{0}$, and $\boldsymbol{v}(\mathbf{0}) = \mathbf{0}$. The scatter matrix $\boldsymbol{\Sigma}(\boldsymbol{X})$ provides the affine invariance.

Affine invariant spatial depth satisfies all the above postulates, but is not quasiconcave. Its maximum is referred to as the *spatial median*. In the one-dimensional case it coincides with the halfspace depth, defined below.

Spatial depth can be efficiently computed even for large samples amounting in the simplest case to time complexity $O(nd^2 + d^3)$; for calculation of $\boldsymbol{\Sigma}^{-\frac{1}{2}}(\boldsymbol{X})$ see the above discussion of the Mahalanobis depth.

Halfspace depth follows the idea of Tukey (1975), see also Donoho and Gasko (1992). The Tukey (=halfspace, location) depth of $\boldsymbol{z}$ w.r.t. $\boldsymbol{X}$ is determined as:

$$D_{hs}(\boldsymbol{z}|\boldsymbol{X}) = \min_{\boldsymbol{u}\in S^{d-1}} \frac{1}{n}\#\{i : \boldsymbol{x}_i^\top \boldsymbol{u} \leq \boldsymbol{z}^\top \boldsymbol{u}; i = 1, ..., n\}. \tag{3.4}$$

Halfspace depth satisfies all the postulates of a depth function. In addition, it is quasiconcave, and equals zero outside the convex hull of the support of $\boldsymbol{X}$. For any $\boldsymbol{X}$, there exists at least one point having depth not smaller than $\frac{1}{1+d}$ (Mizera, 2002). For empirical distributions, halfspace depth is a discrete function of $\boldsymbol{z}$, and the set of depth-maximizing locations — the *halfspace median* — can consist of more than one point (to obtain a unique median, an average of this deepest trimmed region can be calculated). Halfspace depth determines the empirical distribution uniquely (Struyf and Rousseeuw, 1999, Koshevoy, 2002).

Dyckerhoff and Mozharovskyi (2016) develop a family of algorithms (for each $d > 1$) possessing time complexity $O(n^{d-1} \log n)$ and $O(n^d)$ (the last has proven to be computationally more efficient for larger d and small n). These algorithms are applicable for moderate n and d. For large n or d and (or) if the depth has to be computed many times, approximation by minimizing over projections on random lines can be performed (Dyckerhoff, 2004, Cuesta-Albertos and Nieto-Reyes, 2008). By that, $D_{hs}(\boldsymbol{z}|\boldsymbol{X})$ is approximated from above with time complexity $O(knd)$, and $D_{hs}(\boldsymbol{X}|\boldsymbol{X})$ with time complexity $O\big(kn(d + \log n)\big)$, using k random directions (see also Mozharovskyi et al., 2015).

Simplicial depth (Liu, 1990) is defined as the portion of simplices having vertices from $\boldsymbol{X}$ which contain $\boldsymbol{z}$:

$$D_{sim}(\boldsymbol{z}|\boldsymbol{X}) = \frac{1}{\binom{n}{d+1}} \sum_{1\leq i_1<i_2<\ldots<i_{d+1}\leq n} I\big(\boldsymbol{z} \in \mathrm{conv}(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, ..., \boldsymbol{x}_{i_{d+1}})\big) \tag{3.5}$$

with $\mathrm{conv}(\mathcal{Y})$ being the convex hull of $\mathcal{Y}$ and $I(\mathcal{Y})$ standing for the indicator function, which equals 1 if $\mathcal{Y}$ is true and 0 otherwise.

It satisfies postulates (D1), (D2), (D3), and (D5). The set of depth-maximizing locations is not a singleton, but, different to the halfspace depth, it is not convex (in fact it is not even necessarily connected) and thus simplicial depths fails to satisfy (D4). It characterizes the empirical measure if the data, *i.e.* the rows of $\boldsymbol{X}$, are in general position, and is, as well as the halfspace depth, due to its nature rather insensitive to outliers, but vanishes beyond the convex hull of the data $\mathrm{conv}(\boldsymbol{X})$.

Exact computation of the simplicial depth has time complexity of $O(n^{d+1}d^3)$. Approximations accounting for a part of simplices can lead to time complexity $O(kd^3)$ only when drawing k random $(d+1)$-tuples from $\boldsymbol{X}$, or reduce real computational burden with the same time complexity, but keeping precision when drawing a constant portion of $\binom{n}{d+1}$. For $\mathbb{R}^2$, Rousseeuw and Ruts (1996) proposed an exact efficient algorithm with time complexity $O(n \log n)$.

Simplicial volume depth (Oja, 1983) is defined via the average volume of the simplex with d vertices from $\boldsymbol{X}$ and one being $\boldsymbol{z}$:

$$D_{simv}(\boldsymbol{z}|\boldsymbol{X}) = \frac{1}{1 + \frac{1}{\binom{n}{d}\sqrt{\det\big(\boldsymbol{\Sigma}(\boldsymbol{X})\big)}} \sum_{1\le i_1<i_2<\ldots<i_d\le n} \mathrm{vol}\big(\mathrm{conv}(\boldsymbol{z}, \boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, ..., \boldsymbol{x}_{i_d})\big)} \tag{3.6}$$

with $\mathrm{vol}(\mathcal{Y})$ being the Lebesgue measure of $\mathcal{Y}$.

It satisfies all above postulates, is quasiconcave, determines $\boldsymbol{X}$ uniquely (Koshevoy, 2003), and has a nonunique median.

Time complexity of the exact computation of the simplicial volume depth amounts to $O(n^d d^3)$, and thus approximations similar to the simplicial depth may be necessary.

Zonoid depth has been first introduced by Koshevoy and Mosler (1997), see also Mosler (2002) for a discussion in detail. The zonoid depth function is most simply defined by means of depth contours — the zonoid trimmed regions. The zonoid α-trimmed region of an empirical distribution is defined as follows: For $\alpha \in \left[\frac{k}{n}, \frac{k+1}{n}\right]$, $k = 1, ..., n-1$ the zonoid region is defined as

$$Z_\alpha(\boldsymbol{X}) = \mathrm{conv}\Big\{\frac{1}{\alpha n}\sum_{j=1}^{k} \boldsymbol{x}_{i_j} + \Big(1 - \frac{k}{\alpha n}\Big)\boldsymbol{x}_{i_{k+1}} : \{i_1, ..., i_{k+1}\} \subset \{1, ..., n\}\Big\},$$

and for $\alpha \in \left[0, \frac{1}{n}\right)$

$$Z_\alpha(\boldsymbol{X}) = \mathrm{conv}(\boldsymbol{X}).$$

Thus, *e.g.*, $Z_{\frac{3}{n}}(\boldsymbol{X})$ is the convex hull of the set of all possible averages involving three points of $\boldsymbol{X}$, and $Z_0(\boldsymbol{X})$ is just the convex hull of $\boldsymbol{X}$.

The zonoid depth of a point $\boldsymbol{z}$ w.r.t. $\boldsymbol{X}$ is then defined as the largest $\alpha \in [0, 1]$ such that $Z_\alpha(\boldsymbol{X})$ contains $\boldsymbol{z}$ if $\boldsymbol{z} \in \mathrm{conv}(\boldsymbol{X})$ and 0 otherwise:

$$D_{zon}(\boldsymbol{z}|\boldsymbol{X}) = \sup\{\alpha \in [0, 1] : \boldsymbol{z} \in Z_\alpha(\boldsymbol{X})\}, \tag{3.7}$$

where sup of $\emptyset$ is defined to be 0.

The zonoid depth belongs to the class of weighted-mean depths, see Dyckerhoff and Mosler (2011). It satisfies all the above postulates and is quasiconcave. As well as halfspace and simplicial depth, zonoid depth vanishes beyond the convex hull of $\boldsymbol{X}$. Its maximum (always equaling 1) is located at the mean of the data, thus this depth is not robust.

Its exact computation with the algorithm of Dyckerhoff et al. (1996), based on linear programming and exploiting the idea of Danzig-Wolf decomposition, appears to be fast enough for large n and d, not to need approximation.

A common property of the considered above depth notions is that they concentrate on global features of the data ignoring local specifics of sample geometry. Thus they are unable to reflect multimodality of the underlying distribution. Several depths have been proposed in the literature to overcome this difficulty. Two of them were introduced in the classification context, localized extension of the spatial depth (Dutta and Ghosh, 2015) and the data potential (Pokotylo and Mosler, 2016). They are also implemented in the R-package **ddalpha**. The performance of these depths and of the classifiers exploiting them depends on the type of the kernel and its bandwidth. While the behaviour of these two notions substantially differs from the seven depth notions mentioned above, we leave them beyond the scope of this chapter and relegate to the corresponding literature for theoretical and experimental results.

3.2.3 Computation time

To give insights into the speed of exactly calculating various depth notions we indicate computation times by graphics in Figure 3.4. On the logarithmic time scale, the lines represent the time (in seconds) needed to compute the depth of a single point, averaged over 50 points w.r.t. 60 samples, varying dimension $d \in \{2, 3, 4, 5\}$ and sample length $n \in \{50, 100, 250, 500, 1000\}$. Due to the fact that computation times of the algorithms do not depend on the particular shape of the data, the data has been drawn from the standard normal distribution. Some of the graphics are incomplete due to excessive time. Projection depth has been approximated using $10\,000\,000/n$ random projections, all other depths have been computed exactly. Here we used one kernel of the Intel Core i7-4770 (3.4 GHz) processor having enough physical memory.

One can see that, for all considered depths and $n \leq 1\,000$, computation of the two-dimensional depth never oversteps one second. For halfspace and simplicial depth this can be explained by the fact that in the bivariate case both depths depend only on the angles between the lines connecting $\boldsymbol{z}$ with the data points $\boldsymbol{x}_i$ and the abscissa. Computing these angles and sorting them has a complexity of $O(n \log n)$ which determines the complexity of the bivariate algorithms. As expected, halfspace, simplicial, and simplicial volume depths, being of combinatorial nature, have exponential time growth in (n, d). Somewhat surprising, zonoid depth being computed by linear programming, seems to be way less sensitive to dimension. One can conclude that in applications with restricted computational resources, halfspace, projection, simplicial and simplicial volume depths may be rather approximated in higher dimensions, while exact algorithms can still be used in the low-dimensional framework, *e.g.* when computing time cuts of multivariate functional depths, or to assess the performance of approximation algorithms.

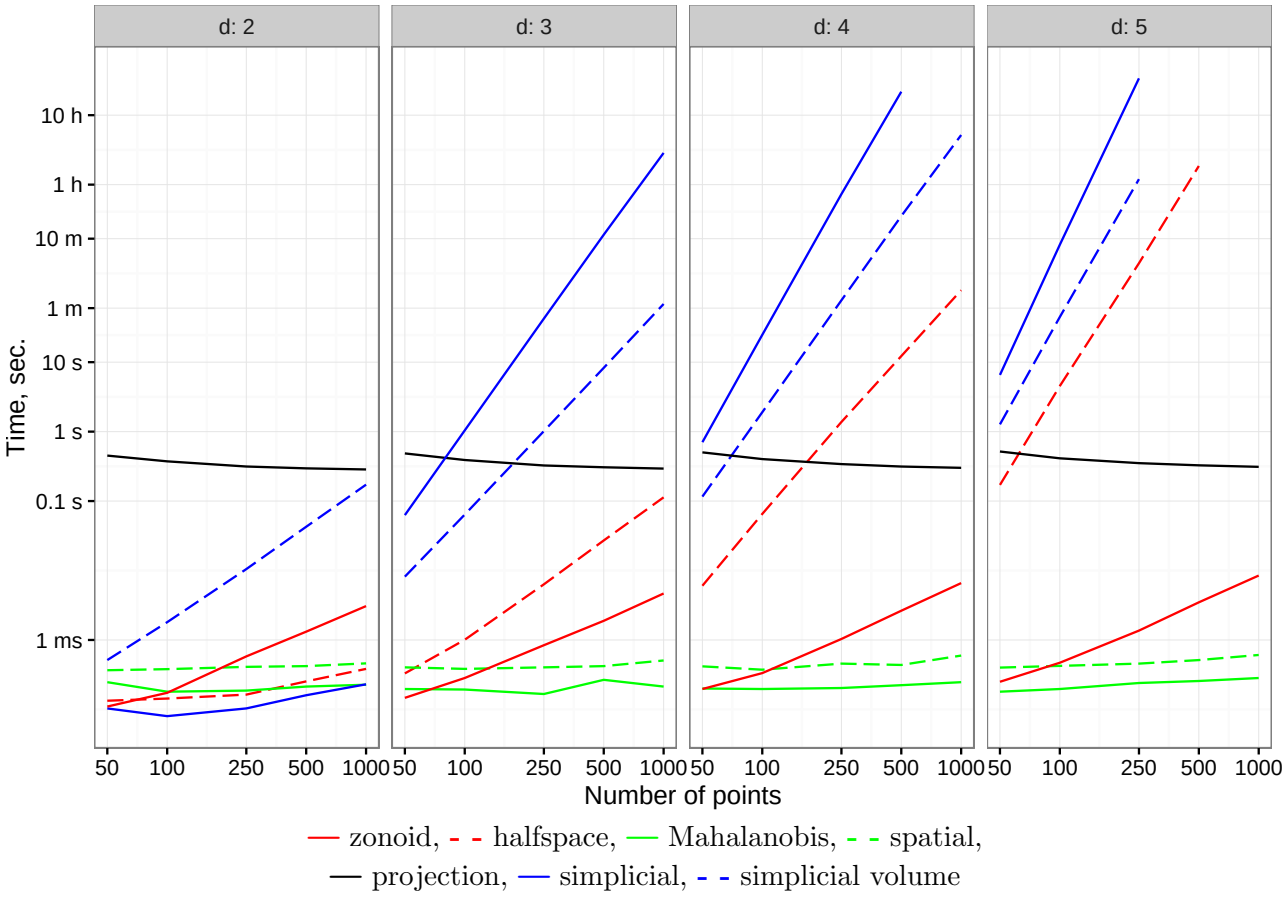


Figure 3.4: Calculation time of various depth functions, on the logarithmic time scale. For the approximative versions see Section 2.7.4 and Figure 2.12.

3.2.4 Maximum depth classifier

To demonstrate the differing finite-sample behavior of the above depth notions and to construct a bridge to supervised classification, in this section we compare the depths in the frame of the maximum depth classifier. This is obtained by simply choosing the class in which $\boldsymbol{x}_0$ has the highest depth (breaking ties at random):

$$class(\boldsymbol{x}_0) = \underset{i \in \{1,\dots,q\}}{\operatorname{argmax}} \; D(\boldsymbol{x}_0 | \boldsymbol{X_i}). \tag{3.8}$$

Ghosh and Chaudhuri (2005b) have proven that its misclassification rate converges to the optimal Bayes risk if each $\boldsymbol{X}_i$, $i = 1, \dots, q$, is sampled from a unimodal elliptically symmetric distribution having a common nonincreasing density function, a prior probability $\frac{1}{q}$, and differing in location parameter only (location-shift model), for halfspace, simplicial, and projection depths, and under additional assumptions for spatial and simplicial volume depths. Setting $q = 2$, and $n = 24, 50, 100, 250, 500, 1000$, $n_i = n/2$, $i = 1, 2$, we sample $\boldsymbol{X}_i$ from a Student-t distribution with location parameters $\mu_1 = [0, 0]$, $\mu_2 = [1, 1]$ and common scale parameter $\Sigma = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 4 \end{smallmatrix}\right]$, setting the degrees of freedom to $t = 1, 5, 10, \infty$. Average error rates over 250 samples each checked on 1000 observations are indicated in Figure 3.5. The testing observations

were sampled inside the convex hull of the training set. The problem of outsiders is addressed in Section 3.4. For $n = 1000$, experiments have not been conducted with the simplicial depth due to high computation time.

As expected, with increasing n and t classification error and difference between various depths decrease. As the classes stem from elliptical family, depths accounting explicitly for ellipticity (Mahalanobis and spatial due to covariance matrix), symmetry of the data (projection), and also volume, form the error frontier. On the other hand, except for the projection depth, they are nonrobust and perform poorly for Cauchy distribution. While projection depth, even being approximated, behaves excellent in all the experiments, it may perform poorly if distributions of $\boldsymbol{X}_i$ retain asymmetry due to inability to reflect this.

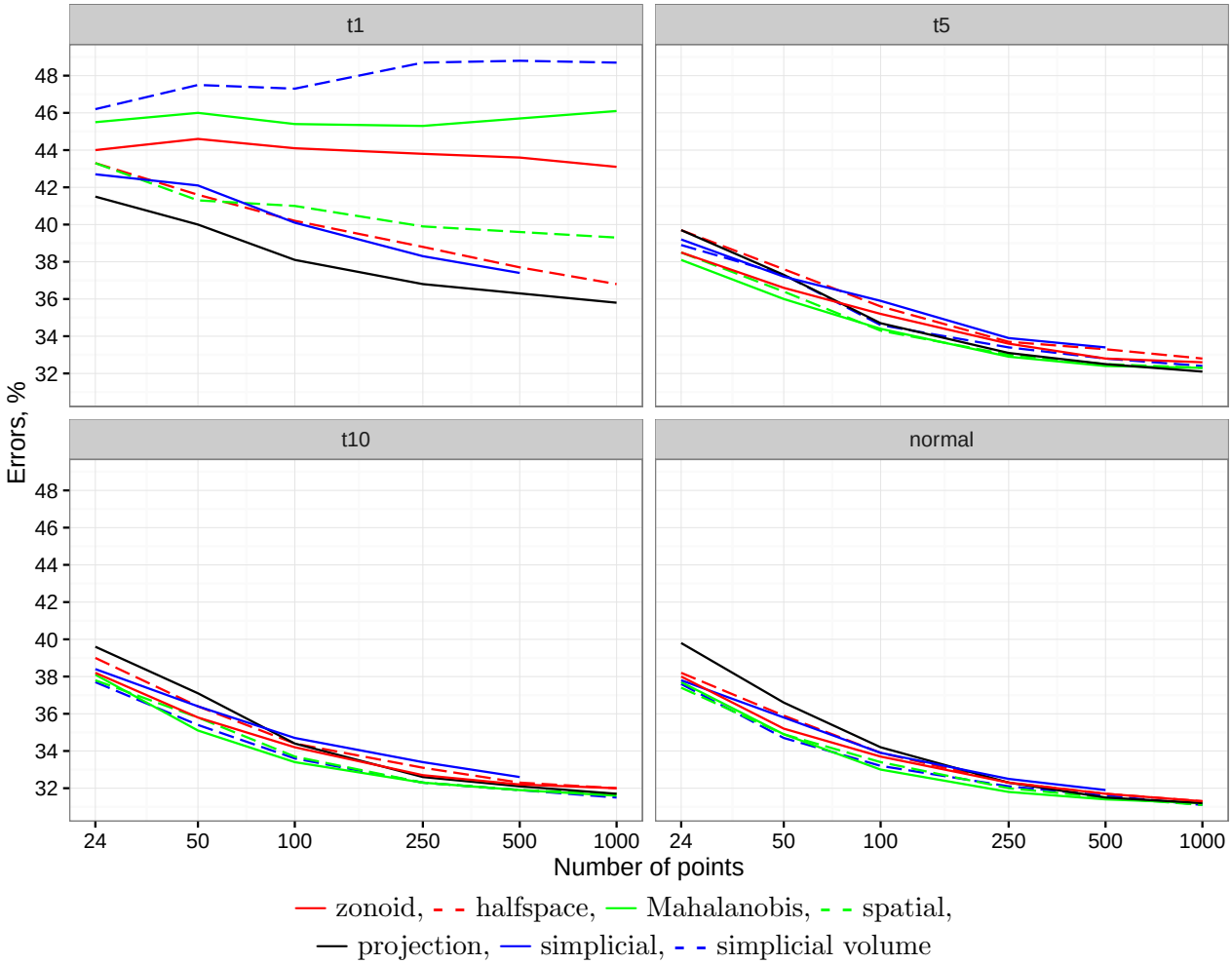


Figure 3.5: Average error rates of the maximum depth classifier with different data depths. The samples are simulated from the Student-t distribution possessing 1, 5, 10, and ∞ degrees of freedom.

3.3 Classification in the DD-plot

In Section 3.2.4, we have already considered the naive way of depth-based classification — the maximum depth classifier. Its extension beyond the equal-prior location-shift model, *e.g.*, to account for differing shape matrices of the two classes, or unequal prior probabilities, is somewhat cumbersome, *cf.* Ghosh and Chaudhuri (2005a), Cui et al. (2008). A simpler way, namely to use the DD-plot (or, more general, a q-dimensional depth space), has been proposed by Li et al. (2012). For a training sample consisting of $\boldsymbol{X}_1, ..., \boldsymbol{X}_q$, the depth space is constructed by applying the mapping $\mathbb{R}^d \to [0,1]^q : \boldsymbol{x} \mapsto \big(D(\boldsymbol{x}|\boldsymbol{X}_1), ..., D(\boldsymbol{x}|\boldsymbol{X}_q)\big)$ to each of the observations. Then the classification is performed in this low-dimensional space of depth-extracted information, which, *e.g.*, for $q = 2$ is just a unit square. The core idea of the $DD\alpha$-classifier is the $DD\alpha$-separator, a fast heuristic for the DD-plot. This is presented in Section 3.3.1, where we slightly abuse the notation introduced before. This is done in an intuitive way for the sake of understandability and closeness to the implementation. Further, in Section 3.3.2 we discuss application of alternative techniques in the depth space.

3.3.1 The $DD\alpha$-separator

The $DD\alpha$-separator (Lange and Mozharovskyi, 2014) is an extension of the α-procedure to the depth space, see Vasil'ev (2003), Vasil'ev and Lange (1998) and Appendix to Chapter 3. It iteratively synthesizes the space of features, coordinate axes of the depth space or their (polynomial) extensions, choosing features minimizing a two-dimensional empirical risk in each step. The process of space enlargement stops when adding features does not further reduce the empirical risk. Here we give its comprehensive description.

Regard the two-class sample illustrated on Figure 3.6, left, representing discretizations of the electrocardiogram curves. Explanation of the data is given in Section 3.1.3, we postpone the explanation of the discretization scheme till Section 3.5 and consider a binary classification in the DD-plot for the moment. Figure 3.6, middle, represents the depth contours of each class computed using the spatial depth. The DD-plot is obtained as a depth mapping $(\boldsymbol{X}_1, \boldsymbol{X}_2) \mapsto \boldsymbol{Z} = \{\boldsymbol{z}_i = (D_{i,1}, D_{i,2}),\ i = 1, ..., n_1 + n_2\}$, when the first class is indexed by $i = 1, ..., n_1$ and the second by $i = n_1 + 1, ..., n_2$, and writing $D_{spt}(\boldsymbol{x}_i|\boldsymbol{X}_1)$ (respectively $D_{spt}(\boldsymbol{x}_i|\boldsymbol{X}_2)$) by $D_{i,1}$ (respectively $D_{i,2}$) for shortness. Further, to enable for nonlinear separation in the depth space, but to employ linear discrimination in the synthesized subspaces, the kernel trick is applied. As the $DD\alpha$-separator explicitly works with the dimensions (space axis), a finite-dimensional resulting space is required. We choose the space extension degree by means of a fast cross-validation, which is performed over a small range and in the depth space only. The high computation speed of the $DD\alpha$-separator allows for this.

We use polynomial extension of degree p, which results in $r = \binom{p+q}{q} - 1$ dimensions (by default, we choose p among $\{1, 2, 3\}$ using 10-fold cross-validation); truncated series or another finitized basis of general reproducing kernel Hilbert spaces can be used alternatively. This extended depth space serves as the input to the $DD\alpha$-separator. For $q = 2$,

and taking $p = 3$, one gets the extended depth space $\boldsymbol{Z}^{(p)}$ consisting of observations $\boldsymbol{z}_i^{(p)} = (D_{i,1},\ D_{i,2},\ D_{i,1}^2,\ D_{i,1} \times D_{i,2},\ D_{i,2}^2,\ D_{i,1}^3,\ D_{i,1}^2 \times D_{i,2},\ D_{i,1} \times D_{i,2}^2,\ D_{i,2}^3) \in \mathbb{R}^r$.

After initializations, on the *1st step*, the $DD\alpha$-separator starts with choosing the pair of extended properties minimizing the empirical risk. For this, it searches through all coordinate subspaces $\boldsymbol{Z}^{(k,l)} = \{\boldsymbol{z}_i^{(k,l)} \,|\, \boldsymbol{z}_i^{(k,l)} = (\boldsymbol{z}_{ik}^{(p)}, \boldsymbol{z}_{il}^{(p)}),\ i = 1, ..., n_1 + n_2\}$ for all $1 \le k < l \le r$, *i.e.* all pairs of coordinate axis of $\boldsymbol{Z}^{(p)}$. For each of them, the angle $\alpha_1^{(k,l)}$ minimizing the empirical risk is found

$$\alpha_1^{(k,l)} \in \underset{\alpha \in [0;2\pi)}{\operatorname{argmin}} \Delta^{(k,l)}(\alpha) \tag{3.9}$$

with

$$\Delta^{(k,l)}(\alpha) = \sum_{i=1}^{n_1} I(\boldsymbol{z}_{ik}^{(p)} \cos\alpha - \boldsymbol{z}_{il}^{(p)} \sin\alpha < 0) + \sum_{i=n_1+1}^{n_1+n_2} I(\boldsymbol{z}_{ik}^{(p)} \cos\alpha - \boldsymbol{z}_{il}^{(p)} \sin\alpha > 0). \tag{3.10}$$

For the regarded example, this is demonstrated in Figure 3.7 by the upper triangle of the considered subspaces. Computationally, it is reasonable to check only those α corresponding to (radial) intervals between points and to choose $\alpha_1^{(k,l)}$ as an average angle between two points from $\boldsymbol{Z}^{(k,l)}$ in case there is a choice, as it is implemented in procedure *GetMinError*. Computational demand is further reduced by skipping uninformative pairs, *e.g.*, if one feature is a power of another one and, therefore, the bivariate plot is collapsed to a line, as shown in Figure 3.7. Finally, a triplet is chosen:

$$(\alpha_1^{(k^*,l^*)}, k^*, l^*) \in \underset{1 \le k < l \le r,\ \alpha \in [0;2\pi)}{\operatorname{argmin}} \Delta^{(k,l)}(\alpha), \tag{3.11}$$

i.e. a two-dimensional coordinate subspace $\boldsymbol{Z}^{(k^*,l^*)}$ in which the minimal empirical risk over all such subspaces is achieved, and the corresponding angle $\alpha_1^{(k^*,l^*)}$ minimizing this. Among all the minimizing triplets (there may be several as empirical risk is discrete) it is reasonable to choose k^* and l^* with the smallest polynomial degree, the simplest model. Using $\alpha_1^{(k^*,l^*)}$, $\boldsymbol{Z}^{(k^*,l^*)}$ is convoluted to a real line

$$\boldsymbol{z}^{(1^*)} = \{z_i \,|\, z_i = \boldsymbol{z}_{ik^*}^{(p)} \cos\alpha_1^{(k^*,l^*)} - \boldsymbol{z}_{il^*}^{(p)} \sin\alpha_1^{(k^*,l^*)},\ i = 1, ..., n_1 + n_2\}, \tag{3.12}$$

— first feature of the synthesized space.

On each following *s-step* ($s \ge 2$), the $DD\alpha$-separator proceeds as follows. The feature, obtained by the convolution on the previous $(s-1)$-*step*, is coupled with each of the extended properties of the depth space, such that a space $\boldsymbol{Z}^{((s-1)^*,k)} = \{\boldsymbol{z}_i^{((s-1)^*,k)} \,|\, \boldsymbol{z}_i^{((s-1)^*,k)} = (\boldsymbol{z}_i^{((s-1)^*)}, \boldsymbol{z}_{ik}^{(p)}),\ i = 1, ..., n_1 + n_2\}$ is regarded, for all k used in no convolution before. For each $\boldsymbol{Z}^{((s-1)^*,k)}$, $\Delta^{((s-1)^*,k)}(\alpha_s^{(k)})$ and the corresponding empirical-risk-minimizing angle $\alpha_s^{(k)}$ are obtained using (3.9) and (3.10). Out of all considered k, the one minimizing $\Delta^{((s-1)^*,k)}(\alpha_s^{(k)})$ is chosen, as in (3.11), and the corresponding $\boldsymbol{Z}^{((s-1)^*,k)}$ is convoluted to $\boldsymbol{z}^{(s^*)}$, as in (3.12). The second part of Figure 3.7 illustrates a possible second step of the algorithm.

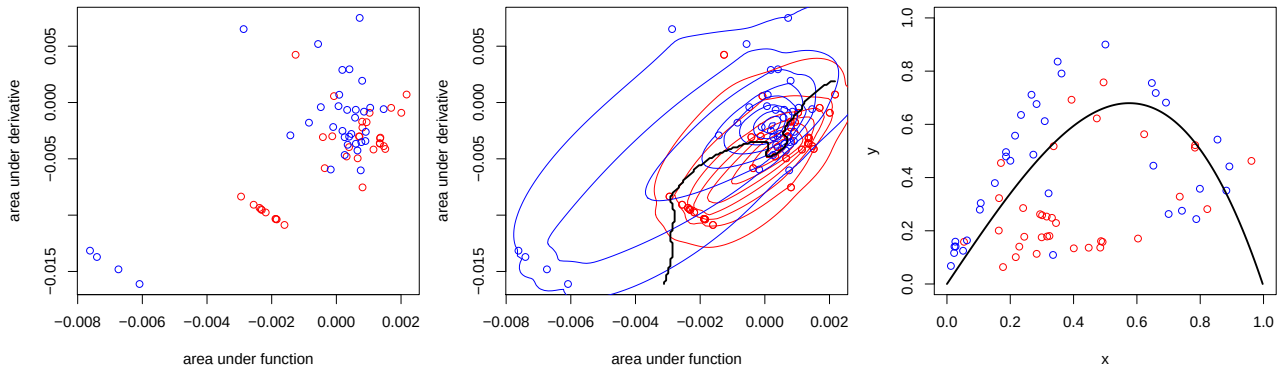


Figure 3.6: The discretized space (left), the depth contours with the separating rule (middle) and the *DD*-plot with the separating line in it (right), using spatial depth. Here we denote the depth of a point w.r.t. red and blue classes by x and y, respectively.

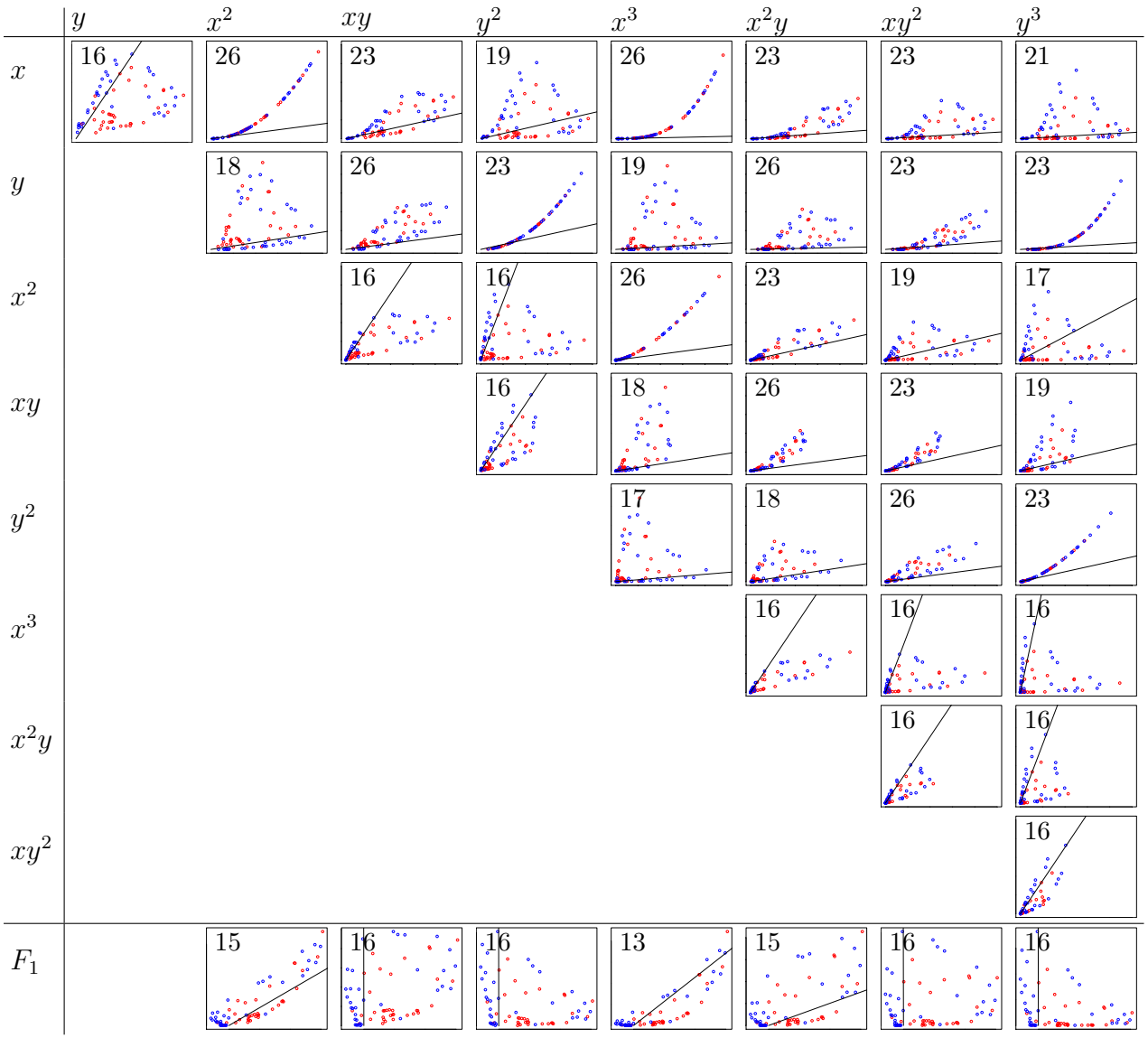


Figure 3.7: The steps of the α-procedure. The number of errors is shown in the left top corner of each plot. The two-dimensional spaces are shown for each pair of properties. On the first step all pairs of properties are considered, on the second step the remaining features are taken together with the first feature F_1. In this example properties x and y are selected on the first step and x^3 on the second.

Here we present the algorithm of the $DD\alpha$-separator:

The main procedure

Input: $\tilde{\boldsymbol{X}} = \{\tilde{\boldsymbol{x}}_1, ..., \tilde{\boldsymbol{x}}_n\}$, $\tilde{\boldsymbol{x}}_i \in \mathbb{R}^d$,
$\{y_1, ..., y_n\}$, $y_i \in \{-1, 1\}$ for all $i = 1, ..., m = m_{-1} + m_{+1}$.

1. $\boldsymbol{X} = \tilde{\boldsymbol{X}}^T = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_d\}$, $\boldsymbol{x}_i \in \mathbb{R}^n$.

2. Initialize arrays:
   (a) array of available properties $\boldsymbol{P} \leftarrow \{1..d\}$;
   (b) array of constructed features $\boldsymbol{F} \leftarrow \emptyset$;
   (c) for a feature $f \in \boldsymbol{F}$ denote $f.p$ and $f.\alpha$ the number of the used property and the optimal angle.

3. *1st step:* Find the first features:
   (a) select optimal starting features considering all pairs from $\boldsymbol{P}$:
   $(opt_1, opt_2, e_{min}, \alpha) = \arg\min_{g \in \boldsymbol{G}} g.e$ with
   $\boldsymbol{G} = \{(p_1, p_2, e, \alpha) : (e, \alpha) = \mathit{GetMinError}(\boldsymbol{x}_{p_1}, \boldsymbol{x}_{p_2}), p_1, p_2 \in \boldsymbol{P}, p_1 < p_2\}$
   (b) $\boldsymbol{F} \leftarrow \boldsymbol{F} \cup \{(opt_1, 0), (opt_2, \alpha)\}$
   (c) $\boldsymbol{P} \leftarrow \boldsymbol{P} \setminus \{opt_1, opt_2\}$
   (d) set current feature $f' = \boldsymbol{x}_{opt_1} \times \cos(\alpha) + \boldsymbol{x}_{opt_2} \times \sin(\alpha)$

4. *Following steps:* Search an optimal feature space while empirical error rate decreases
   while $e_{min} \neq 0$ and $\boldsymbol{P} \neq \emptyset$ **do**
   (a) select next optimal feature considering all properties from $\boldsymbol{P}$:
   $(opt, \tilde{e}_{min}, \alpha) = \arg\min_{g \in \boldsymbol{G}} g.e$ with
   $\boldsymbol{G} = \{(p, e, \alpha) : (e, \alpha) = \mathit{GetMinError}(f', \boldsymbol{x}_p), p \in \boldsymbol{P}\}$
   (b) Check if the new feature improves the separation:
   if $\tilde{e}_{min} < e_{min}$ **then**
   $e_{min} = \tilde{e}_{min}$
   $\boldsymbol{F} \leftarrow \boldsymbol{F} \cup (opt, \alpha)$
   $\boldsymbol{P} \leftarrow \boldsymbol{P} \setminus opt$
   update current feature $f' = f' \times \cos(\alpha) + \boldsymbol{x}_{opt} \times \sin(\alpha)$
   else break

5. Get the normal vector of the separating hyperplane:
   (a) Declare a vector $\boldsymbol{r} \in \mathbb{R}^d$, $\boldsymbol{r}_i = 0$ for all $i = 1, ..., d$. Set $a = 1$.
   (b) Calculate the vector components as $\boldsymbol{r}_{\boldsymbol{F}_i.p} = \prod_{j=i+1}^{\sharp \boldsymbol{F}} \Big(\cos(\boldsymbol{F}_j.\alpha)\Big) \sin(\boldsymbol{F}_i.\alpha)$:
   for all $i \in \{\sharp \boldsymbol{F}..2\}$ **do**
   $\boldsymbol{r}_{\boldsymbol{F}_i.p} = a \times \sin(\boldsymbol{F}_i.\alpha)$
   $a = a \times \cos(\boldsymbol{F}_i.\alpha)$
   $\boldsymbol{r}_{\boldsymbol{F}_1.p} = a$
   (c) Project the points on the ray: $\boldsymbol{p}_i.y = y_i$, $\boldsymbol{p}_i.x = \boldsymbol{r} \cdot \tilde{\boldsymbol{x}}_i$
   (d) Sort $\boldsymbol{p}$ w.r.t. $\boldsymbol{p}_{\cdot}.x$ in ascending order.

(e) Count the cardinalities before the separation plane
$m_{l-} = \sharp\{i : \boldsymbol{p}_i.y = -1, \boldsymbol{p}_i.x \le 0\}$,
$m_{l+} = \sharp\{i : \boldsymbol{p}_i.y = +1, \boldsymbol{p}_i.x \le 0\}$

(f) Count the errors
$e_- = m_{l+} + m_- - m_{l-}$,
$e_+ = m_{l-} + m_+ - m_{l+}$

(g) **if** $e_- > e_+$ **then**
$\boldsymbol{r} \leftarrow -\boldsymbol{r}$

Output: the normal vector of the separating hyperplane r.

Procedure *GetMinError*

Input: current feature $f \in \mathbb{R}^n$, property $x \in \mathbb{R}^n$.

1. Obtain angles:
   (a) Calculate $\alpha_i = \arctan \frac{x_i}{f_j}$, $i = 1, ..., n$, with $\arctan \frac{0}{0} = 0$.
   (b) Aggregate angles into set $\mathcal{A}$. Denote $\mathcal{A}_i.\alpha = \alpha_i$ and $\mathcal{A}_i.y = y_i$ the angle and the pattern of the corresponding point. Set $\mathcal{A}_i.y$ to 0 for the points having both $x_i = 0$ and $f_i = 0$.
   (c) Sort $\mathcal{A}$ w.r.t. $\mathcal{A}..\alpha$ in ascending order.
2. Look for the optimal threshold:
   (a) Define $i_{opt} = \arg\max_i \left(|\sum_1^i \mathcal{A}_i.y| + |\sum_{i+1}^n \mathcal{A}_i.y| \right)$ as the place of the optimal threshold and $e_{min} = n - \max_i \left(|\sum_1^i \mathcal{A}_i.y| + |\sum_{i+1}^n \mathcal{A}_i.y| \right)$ as the minimal number of incorrectly classified points
   (b) Define the optimal angle $\alpha_{opt} = \frac{1}{2}(\mathcal{A}_{i_{opt}+1}.\alpha + \mathcal{A}_{i_{opt}+2}.\alpha) - \frac{\pi}{2}$.

Output: min error e_{min}, optimal angle α_{opt}

From the practical point of view, the routine $DD\alpha$-separator has high computation speed as in each plane it has the complexity of the quick-sort procedure: $O\big(\sum_{i=1}^q n_i \log(\sum_{i=1}^q n_i)\big)$.

While minimizing empirical risk in two-dimensional coordinate subspaces and due to the choice of efficient for classification features, the $DD\alpha$-separator tends to be *close to* the optimal *risk-minimizing* hyperplane in the extended space. To a large extent, this explains the performance of the $DD\alpha$-procedure on finite samples.

The robustness of the procedure is twofold: First, *regarding points*, as the depth-space is compact, the outlyingness of the points in it is restricted, and the $DD\alpha$-separator is robust due to its risk-minimizing nature, *i.e.* by the discrete (zero-or-one) loss function. And second, *regarding features*, the separator is not entirely driven by the exact points' location, but accounts for importance of features of the (extended) depth space. By that, the model complexity is kept low; in practice a few features are selected only, see, *e.g.*, Section 5.2 of Mozharovskyi et al. (2015).

For theoretical results on the $DD\alpha$-procedure the reader is referred to Section 4 of Lange et al. (2014b). Mozharovskyi et al. (2015) provide an extensive comparative empirical study

of its performance with a variety of data sets and for different depth notions and outsider treatments, while Lange et al. (2014a) conduct a simulation study on asymmetric and heavy-tailed distributions.

3.3.2 Alternative separators in the DD-plot

Besides the $DD\alpha$-separator, the package **ddalpha** allows for two alternative separators in the depth space: a polynomial rule and the k-nearest-neighbor (k-NN) procedure.

When Li et al. (2012) introduce the DD-classifier, they suggest to use a polynomial of certain degree passing through the origin of the DD-plot to separate the two training classes. Based on the fact that by choosing the polynomial order appropriately the empirical risk can be approximated arbitrarily well, they prove the consistency of the DD-classifier for a wide range of distributions including some important cases of the elliptically symmetric family. In practice, the minimal error is searched by smoothing the empirical loss with a logistic function and then optimizing the parameter of this function. This strategy has sources of instability such as choice of the smoothing constant and multimodality of the loss function. The authors (partially) solve the last issue by varying the starting point for optimization and multiply running the entire procedure, which increases computation time. For theoretical derivations and implementation details see Sections 4 and 5 of Li et al. (2012). For a simulation comparison of the polynomial rule in the DD-plot and the $DD\alpha$-separator see Section 5 of Lange et al. (2014b).

In his PhD-thesis, Vencalek (2011) suggests to perform the k-NN classification in the depth space, and proves its consistency for elliptically distributed classes with identical radial densities. For theoretical details and a simulation study see Sections 3.4.3 and 3.7 of Vencalek (2011), respectively. It is worth to notice that the k-NN-separator has another advantage — it is directly extendable to more than two classes.

3.4 Outsiders

For a number of depth notions like halfspace, zonoid, or simplicial depth, the depth of a point vanishes beyond the convex hull of the data. This leads to the problem that new points (to be classified) lying beyond the convex hull of each of the training classes have depth zero w.r.t. all of them. By that, they are depth-mapped to the origin of the DD-plot, and thus cannot be readily classified. We call these points *outsiders* (Lange et al., 2014b).

Regard Figure 3.8, where three green points are to be classified. Point "1" has positive depth in both classes, and based on its location in the DD-plot will be assigned to the less scattered "red" class. Point "2" has zero depth in the "red" class, but a positive one in the more scattered "blue" class, to which it will be assigned based on the classification rule in the DD-plot. Point "3" on the other hand has zero depth w.r.t. both training classes, and thus classification rule in the DD-plot is helpless. Nevertheless, visually it clearly belongs to the "blue" class, and most probably would be correctly classified by a very simple classifier, say a

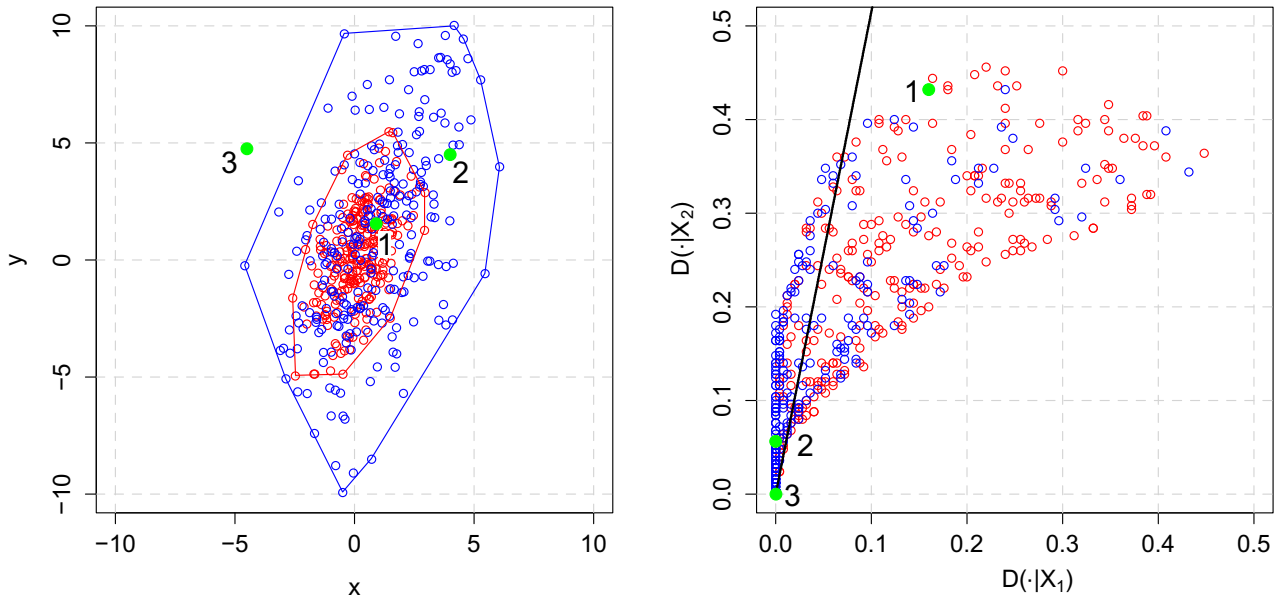


Figure 3.8: Points to be classified (green) in the original (left) and depth (right) space.

poorly tuned k-NN (*e.g.* 1NN). The suggestion thus is to apply an additional fast classifier to the outsiders.

The R-package **ddalpha** implements a number of outsider treatments: linear (LDA) and quadratic (QDA) discriminant analysis, k-NN, maximum depth classifier based on Mahalanobis depth; and additionally random classification or identification of outsiders for statistical analysis or passing to another procedure. For the same experimental setting as in Section 3.2.4, we contrast these treatments in Figure 3.9, comparing classification errors on outsiders only. One can see that for the heavy-tailed Cauchy distribution, where classes may be rather mixed, no outsider treatment performs significantly better than random assignment. The situation improves with increasing number of degrees of freedom of the Student-t distribution, with LDA forming the classification error frontier, as the classes differ in location only. On the other hand, with increasing n_i, difference between the treatment becomes negligible. For an extensive comparative study of different outsider treatments the reader is referred to Mozharovskyi et al. (2015).

If outsiders pose a serious problem, one can go for a nowhere-vanishing depth. But in general, the property of generating outsiders should not necessarily be seen as a shortfall, as it allows for additional information when assessing the configured classifier or a data point to be classified. If too many points are identified as outsiders (what can be checked by a validation procedure), this may point onto inappropriate tuning. On the other hand, if outsiders appear extremely rarely in the classification phase (or, *e.g.*, during online learning), an outsider may be an atypical observation not fitting to the data topology in which case one may not want to classify it at all but rather label indicatively.

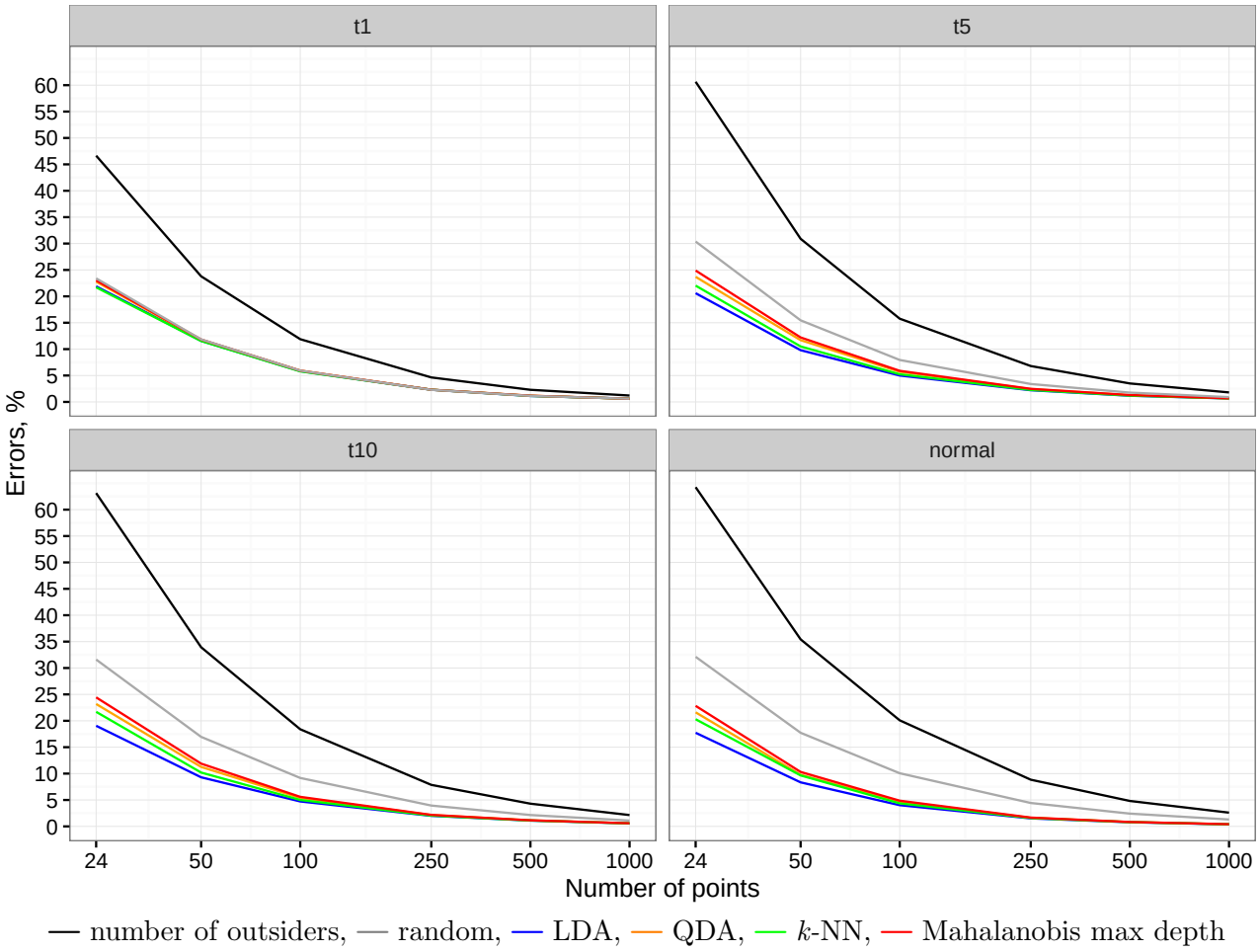


Figure 3.9: Error rates of various outsiders treatment. Only outsiders are classified.

3.5 An extension to functional data

Similar to Section 3.3, consider a binary classification problem in the space of real valued functions defined on a compact interval, which are continuous and smooth everywhere except for a finite number of points, *i.e.* given two classes of functions: $\boldsymbol{\mathcal{F}}_1 = \{f_1, ..., f_{n_1}\}$ and $\boldsymbol{\mathcal{F}}_2 = \{f_1, ..., f_{n_2}\}$, again indexing observations by $i = 1, ..., n_1, n_1 + 1, ..., n_1 + n_2$ for convenience. (An aggregation scheme extends this binary classification to the multiple one.) The natural extension of the depth-based classification to the functional setting consists in defining a proper depth transform $(\boldsymbol{\mathcal{F}}_1, \boldsymbol{\mathcal{F}}_2) \mapsto \boldsymbol{Z} = \{\boldsymbol{z}_i = (D(f_i|\boldsymbol{\mathcal{F}}_1), D(f_i|\boldsymbol{\mathcal{F}}_2)),\ i = 1, ..., n_1+n_2\}$ similar to that in Section 3.3. For this, a proper functional depth should be employed (see Mosler and Polyakova, 2012, Nieto-Reyes and Battey, 2016, and references therein for an overview), followed by the suitable classification technique in the (finite dimensional) depth space. As the functional data depth reduces space dimensionality from infinity to one, the final performance is sensitive to the choice of the depth representation and of the finite-dimensional separator, and thus both constituents should be chosen very carefully. Potentially, this lacks quantitative flexibility because of the finite set of existing components. Nevertheless, in many cases this solution provides satisfactory results; see a comprehensive discussion by Cuesta-Albertos

et al. (2016) with experimental comparisons involving a number of functional depth notions and q-dimensional classifiers, as well as their implementation in the R-package **fda.usc**. Corresponding functional depth procedures can also be used with R-package **ddalpha**, see Section 3.6 for a detailed explanation.

ddalpha suggests two implementations of the strategy of immediate functional data projection onto a finite-dimensional space with further application of a multivariate depth-based classifier: componentwise classification by Delaigle et al. (2012) and LS-transform proposed by Mosler and Mozharovskyi (2015). Both methodologies allow to control for the quality of classification in a quantitative way (*i.e.* by tuning parameters) when constructing the multivariate space, which in addition enables consistency derivations. For the first one the reader is referred to the literature; the second one we present right below.

In application, functional data is usually given in a form of discretely observed paths $\tilde{\boldsymbol{f}}_i = [f_i(t_{i1}), f_i(t_{i2}), ..., f_i(t_{iN_i})]$, which are the measurements at ordered (time) points $t_{i1} < t_{i2} < ... < t_{iN_i}$, $i = 1, ..., n_1 + n_2$, not necessarily equidistant nor same for all i. Fitting these to a basis is avoided as the choice of such a basis turns out to be crucial for classification and thus should better not be independently selected prior to it. Instead, a simple scheme is suggested based on integrating linearly extrapolated data and their derivatives over a chosen number of intervals. Let $\min_i t_{i1} = 0$ and let $T = \max_i t_{iN_i}$, then one obtains the following finite-dimensional transform:

$$\hat{f}_i \mapsto \boldsymbol{x}_i = \left[\int_0^{T/L} \hat{f}_i(t)dt, \ldots, \int_{T(L-1)/L}^{T} \hat{f}_i(t)dt, \int_0^{T/S} \hat{f}_i'(t)dt, \ldots, \int_{T(S-1)/S}^{T} \hat{f}_i'(t)dt\right], \quad (3.13)$$

with $\hat{f}_i(t)$ being the function obtained by connecting the points $(t_{ij},\ f_i(t_{ij}))$, $j = 1, \ldots, N_i$ with line segments and setting $\hat{f}_i(t) = f_i(t_{i1})$ when $0 \le t \le t_{i1}$ and $\hat{f}_i(t) = f_i(t_{ik_i})$ when $t_{iN_i} \le t \le T$, $\hat{f}_i'(t)$ being its derivative, and $L, S \ge 0$, $L + S \ge 2$ being integers. L and S are the numbers of intervals of equivalent length to integrate over the location and the slope of the function, and have to be tuned. One can use intervals of different length or take into account higher-order derivatives (constructed as differences, say), but the suggested way appears to be simple and flexible enough. Moreover it does not introduce any spurious information. The set of considered LS-pairs can be chosen on the basis of some prior knowledge about the nature of the functions or just by properly restricting the dimension of the constructed space by $d_{min} \le L + S \le d_{max}$. Cross-validation is then used to choose the best LS-pair. **ddalpha** suggests to reduce the set of cross-validated LS-pairs by employing the Vapnik-Chervonenkis bound. The idea behind is that, while being conservative, the bound can still provide insightful ordering of the LS-pairs, especially in the case when the empirical risk and the bound have the same order of magnitude.

Given a set of considerable pairs $\mathcal{S} = \{(l_i, s_i) | i = 1, ..., N_{ls}\}$, for each its element calculate the Vapnik-Chervonenkis bound (see Mosler and Mozharovskyi, 2015, for this particular derivation)

$$b_i^{VC} = \epsilon\left(\boldsymbol{c}, \hat{\boldsymbol{\mathcal{F}}}_1^{(l_i,s_i)}, \hat{\boldsymbol{\mathcal{F}}}_2^{(l_i,s_i)}\right) + \sqrt{\frac{\ln 2 \sum_{k=0}^{l_i+s_i-1} \binom{n_1+n_2-1}{k} - \ln \eta}{2(n_1+n_2)}}, \quad (3.14)$$

where $\epsilon\left(\boldsymbol{c}, \hat{\boldsymbol{\mathcal{F}}}_1^{(l_i,s_i)}, \hat{\boldsymbol{\mathcal{F}}}_2^{(l_i,s_i)}\right)$ is the empirical risk achieved by a linear classifier $\boldsymbol{c}$ on the data transformed according to (3.13) with $L = l_i$, $S = s_i$ and $1-\eta$ is the chosen reliability level. In **ddalpha** we set $\eta = \frac{1}{n_1+n_2}$, and choose $\boldsymbol{c}$ to be the LDA for its simplicity and speed. Then a subset $\mathcal{S}^{CV} \subset \mathcal{S}$ is chosen possessing the smallest values of b_i^{VC}: $\left((l_j, s_j) \in \mathcal{S}^{CV}, (l_k, s_k) \in \mathcal{S} \setminus \mathcal{S}^{CV}\right) \Rightarrow (b_j^{VC} < b_k^{VC})$, and cross-validation is performed over all $(l, s) \in \mathcal{S}^{CV}$. For the subsample referenced in introduction, the functions' levels and slopes are shown in Figure 3.1; the LS-representation is selected by reduced cross-validation due to (3.13) having $(L, S) = (1, 1)$, and is depicted in Figure 3.6, left.

3.6 Usage of the package

The package **ddalpha** is a structured solution that provides computational machinery for a number of depth functions and classifiers for multivariate and functional data. It also allows for user-defined depth functions and separators in the DD-plot (further DD-separators). The structure of the package is presented in Figure 3.10.

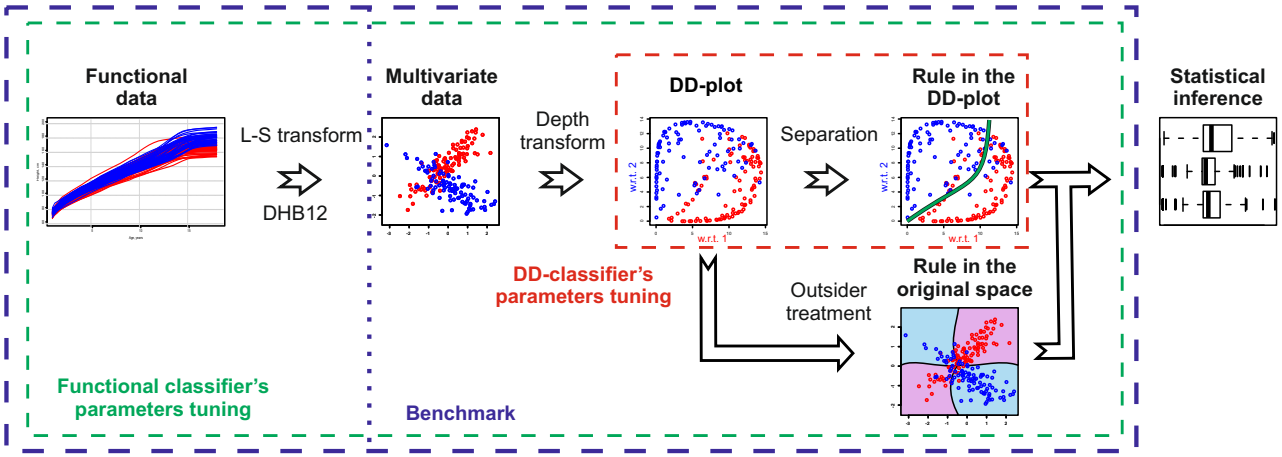


Figure 3.10: The structure of the package.

3.6.1 Basic functionality

Primary aims of the package are calculation of data depth and depth-classification.

Data depth is calculated by calling

```
R> depths <- depth.(x, data, notion = <depthName>, ...)
R> depths <- depth.<depthName>(x, data, ...)
```

Parameter `notion` specifies the used depth (`<depthName>`), `data` is a matrix with each row being a d-variate point, and `x` is a matrix of objects whose depth is to be calculated. Additional arguments (`...`) differ between depth notions. The output of the function is a vector of depths of points from `x`. Most of the depth functions possess both exact and approximative versions

Table 3.1: Implemented depth algorithms.

Depth	Exact	Approximative	Parameter
Mahalanobis	`moment`		`mah.estimate`
	`MCD`		
spatial	`moment`		`mah.estimate`
	`MCD`		
	`none`		
projection		`random`	`method`
		`linearize`	
halfspace	`recursive`	`Sunif.1D`	`method`
	`plane`		
	`line`		
simplicial	+	+	`exact`
simplicial volume	+	+	`exact`
zonoid	+		

that are toggled with parameters `exact` and `method`, see Table 3.1. The exact algorithms of Mahalanobis, spatial, and zonoid depths are very fast and thus exclude the need of approximation. Mahalanobis and spatial depths use either traditional moment or MCD estimates of mean and covariance matrix. Methods `random` for projection depth and `Sunif.1D` for halfspace depth approximate the depth as the minimum univariate depth of the data projected on `num.directions` directions uniformly distributed on S^{d-1}. The exact algorithms for the halfspace depth implement the framework described in Section 3.2.2, where the dimensionality k of the combinatorial space is specified as follows: $k = 1$ for method `recursive`, $k = d - 2$ for `plane` and $k = d - 1$ for `line`, see additionally Dyckerhoff and Mozharovskyi (2016). The second approximating algorithm for projection depth is `linearize` — the Nelder-Mead method for function minimization, taken from Nelder and Mead (1965) and originally implemented in R by Subhajit Dutta. For simplicial and simplicial volume depths, parameter `k` specifies the number (if `k` > 1) or portion (if $0 <$ `k` < 1) of simplices chosen randomly among all possible simplices for approximation.

In addition, calculation of the entire DD-plot at once is possible by

```
R> dspace <- depth.space.(data, cardinalities, notion = <depthName>, ...)
R> dspace <- depth.space.<depthName>(data, cardinalities, ...)
```

The matrix `data` consists of q stacked training classes, and `cardinalities` is a vector containing numbers of objects in each class. The method returns a matrix with q columns representing the depths of each point w.r.t. each class.

Classification can be performed either in two steps — training the classifier with the function `ddalpha.train` and using it for classification in `ddalpha.classify` or `predict`, or in one step — by function `ddalpha.test(learn, test, ...)` that trains the classifier with `learn` sample and checks it on the `test` one. Other parameters are the same as for function `ddalpha.train` and are described right below.

Function `ddalpha.train` is the main function of the package. Its structure is shown on the right part of Figure 3.10.

```
R> ddalpha <- ddalpha.train(formula, data, subset,
+                           depth = "halfspace", separator = "alpha",
+                           outsider.methods = "LDA", outsider.settings = NULL,
+                           aggregation.method = "majority",
+                           use.convex = FALSE,
+                           seed = 0, ...)
R> classes <- ddalpha.classify(ddalpha, objects,
+                           outsider.method, use.convex)
R> classes <- predict(ddalpha, objects,
+                           outsider.method, use.convex, ...)
```

The training set is passed either through `data` in a form of a matrix or a data set with each row being a d-variate point and the last column being the class label, or using `formula`. In the latter case the variables from the formula are found either in `data` or in the environment. The resulting set of columns is printed in the output. The used part of the observations may be additionally specified with `subset`. The notion of the depth function and the DD-separator are specified with the parameters `depth` and `separator`, respectively. Parameter `aggregation.method` determines the method applied to aggregate outcomes of binary classifiers during multiclass classification. When `"majority"`, $q(q-1)/2$ binary one-against-one classifiers are trained, and for `"sequent"`, q binary one-against-all classifiers are taught. During classification, the results are aggregated using the majority voting, where classes with larger proportions in the training sample are preferred when tied (by that implementing both aggregating schemes at once). Additional parameters of the chosen depth function and DD-separator are passed using the dots, and are described in the help sections of the corresponding R-functions. Also, the function allows to use a pre-calculated DD-plot by choosing `depth = "ddplot"`. For each depth function and depth-separator, a validator is implemented — a special R-function that specifies the default values and checks the received parameters allowing by that definition of custom depths and separators; see Section 3.6.2 for details.

Outsider treatment is a supplementary classifier for data that lie outside the convex hulls of all q training classes. It is only needed during classification when the used data depth produces outsiders or obtains zero values in the neighborhood of the data. Parameter `use.convex` of `ddalpha.train` indicates whether outsiders should be determined as the points not contained in any of the convex hulls of the classes from the training sample (`TRUE`) or those having zero depth w.r.t. each class from the training sample (`FALSE`); the difference is explained by the depth approximation error. The following methods are available: `"LDA"`, `"QDA"` and `"kNN"`; affine-invariant k-NN (`"kNNAff"`), *i.e.* k-NN with Euclidean distance normalized by the pooled covariance matrix, suited only for binary classification and using aggregation with multiple classes and not accounting for ties, but very fast; maximum Mahalanobis depth classifier (`"depth.Mahalanobis"`); equal and proportional randomization (`"RandEqual"` and

`"RandProp"`) and ignoring (`"Ignore"`) — a string "Ignored" is returned for the outsiders. Outsider treatment is set by means of parameters `outsider.methods` and `outsider.settings` in `ddalpha.train`. Multiple methods may be trained and then the particular method is selected in `ddalpha.classify` by passing its name to parameter `outsider.method`. Parameter `outsider.methods` of `ddalpha.train` accepts a vector of names of basic outsider methods that are applied with the default settings. Parameter `outsider.settings` allows to train a list of outsider treatments, whose elements specify the names of the methods (used in `ddalpha.classify` later) and their parameters.

Functional classification is performed with functions `ddalphaf.train` implementing LS-transform (Mosler and Mozharovskyi, 2015) and `compclassf.train` implementing componentwise classification (Delaigle et al., 2012).

```
R> ddalphaf <- ddalphaf.train(dataf, labels,
+           adc.args = list(instance = "avr",
+                           numFcn = -1,
+                           numDer = -1),
+           classifier.type = c("ddalpha", "maxdepth",
+                               "knnaff", "lda", "qda"),
+           cv.complete = FALSE,
+           maxNumIntervals = min(25, ceiling(length(dataf[[1]]$args)/2)),
+           seed = 0, ...)
R> classes <- ddalphaf.classify(ddalphaf, objectsf, ...)
R> classes <- predict(ddalphaf, objectsf, ...)

R> compclassf <- compclassf.train(dataf, labels,
+           to.equalize = TRUE, to.reduce = TRUE,
+           classifier.type = c("ddalpha", "maxdepth",
+                               "knnaff", "lda", "qda"),
+           ...)
R> classes <- compclassf.classify(compclassf, objectsf, ...)
R> classes <- predict(compclassf, objectsf, ...)
```

In both functions, `dataf` is a list of functional observations, each having two vectors: `"args"` for arguments sorted in ascending order and `"vals"` for the corresponding functional evaluations; `labels` is a list of class labels of the functional observations; `classifier.type` selects the classifier that separates the finitized data, and additional parameters are passed to this selected classifier with dots. In the componentwise classification, `to.equalize` specifies whether the data is adjusted to have equal (the largest) argument interval, and `to.reduce` indicates whether the data has to be projected onto a low-dimensional space via the principal components analysis (PCA) in case their affine dimension after finitization is lower than expected. (Both parameters are recommended to be set true.)

The LS-transform converts functional data into multidimensional ones by averaging over intervals or evaluating values on equally-spaced grid for each function and its derivative on L (respectively S) equal nonoverlapping covering intervals. The dimension of the multivariate space then equals $L + S$. Parameter `adc.args` is a list that specifies: `instance` — the type of discretization of the functions having values `"avr"` for averaging over intervals of the same length and `"val"` for taking values on equally-spaced grid; `numFcn` (L) is the number of function intervals, and `numDer` (S) is the number of first-derivative intervals.

The parameters L and S may be set explicitly or may be automatically cross-validated. The cross-validation is turned on by setting `numFcn = -1` and `numDer = -1`, or by passing a list of *adc.args* objects to `adc.args` — the range of (L, S)-pairs to be checked. In the first case all possible pairs of L and S are considered up to the maximal dimension that is set in `maxNumIntervals`, while in the latter case only the pairs from the list are considered. The parameter `cv.complete` toggles the complete cross-validation; if `cv.complete` is set to false the Vapnik-Chervonenkis bound is applied, which enormously accelerates the cross-validation, as described in Mosler and Mozharovskyi (2015) in detail. The optimal values of L and S are stored in the *ddalphaf* object, that is returned from `ddalphaf.train`.

3.6.2 Custom depths and separators

As mentioned above, the user can amplify the existing variety by defining his own depth functions and separators. Custom depth functions and separators are defined by implementing three functions: parameters validator, learning, and calculating functions, see Tables 3.2 and 3.3. Usage examples are found in the manual of the package **ddalpha**.

Validator is a nonmandatory function that validates the input parameters and checks if the depth calculating procedure is applicable to the data. All the parameters of a user-defined depth or separator must be returned by a validator as a named list, otherwise they will not be saved in the `ddalpha` object.

Definition of a custom depth function is done as follows: The *depth-training function* `.<name>_learn(ddalpha)` calculates any data-based statistics that the depth function needs (*e.g.*, mean and covariance matrix for Mahalanobis depth) and then calculates the depths of the training classes, *e.g.*, by calling for each pattern i the *depth-calculating function* `.<name>_depths(ddalpha, objects = ddalpha$patterns[[i]]$points)` that calculates the depth of each point in `objects` w.r.t. each pattern in `ddalpha` and returns a matrix with q columns. The learning function returns a `ddalpha` object, where the calculated statistics and parameters are stored. All stored objects, including the parameters returned by the validator, are accessible through the `ddalpha` object, on each stage. After having defined these functions, the user only has to specify `depth = "<name>"` in `ddalpha.train` and pass the required parameters there. (The functions are then linked via the `match.fun` method.)

Definition of a custom separator is similar. Recall that there exist binary separators applicable to two classes, and multiclass ones that separate more than two classes at once. In case if the custom method is binary, the package takes care of the voting procedures, and

Table 3.2: Definition of a custom depth function.

`.<name>_validate`	
validates parameters passed to `ddalpha.train` and passes them to the `ddalpha` object.	
IN:	
`ddalpha`	the ddalpha object, containing the data and settings
`<custom params>`	parameters that are passed to the user-defined method
`...`	other parameters (mandatory)
OUT:	
`list()`	list of output parameters, after the validation is finished these parameters are stored in the `ddalpha` object
`.<name>_learn`	
trains the depth	
IN:	
`ddalpha`	the ddalpha object containing the data and settings
MODIFIES:	
`ddalpha`	store the calculated statistics in the `ddalpha` object

depths	calculate the depths of each pattern, e.g.
	`R> for (i in 1:ddalpha$numPatterns)`
	`+ ddalpha$patterns[[i]]$depths =`
	`+ .<name>_depths(ddalpha, ddalpha$patterns[[i]]$points)`

OUT:	
`ddalpha`	the updated `ddalpha` object
`.<name>_depths`	
calculates the depths	
IN:	
`ddalpha`	the ddalpha object containing the data and settings
`objects`	the objects for which the depths are calculated
OUT:	
`depths`	the calculated depths for each object (rows), with respect to each class (columns)
Usage:	`ddalpha.train(data, depth = "<name>", <custom params>, ...)`

the user only has to implement a method that separates two classes. The training method for a *binary separator* `.<name>_learn(ddalpha, index1, index2, depths1, depths2)` accepts the depths of the objects w.r.t. two classes and returns a trained classifier. A *multiclass separator* has to implement another interface: `.<name>_learn(ddalpha)`, accessing the depths of the different classes via `ddalpha$patterns[[i]]$depths`. The binary classifier can utilize the whole depth space (*i.e.* depths w.r.t. other classes than the two currently under consideration) to get more information like the α-separator does, or restrict to the DD-plot w.r.t. the two given classes like the polynomial separator, by accessing `depths1` and `depths2` matrices. The *classifying function* `.<name>_classify(ddalpha, classifier, objects)` accepts the previously

Table 3.3: Definition of a custom separator.

`.<name>_validate`	
validates parameters passed to `ddalpha.train` and passes them to the `ddalpha` object	
IN:	
`ddalpha`	the ddalpha object containing the data and settings
`<custom params>`	parameters that are passed to the user-defined method
`...`	other parameters (mandatory)
OUT:	
`list()`	list of output parameters, after the validation is finished, these parameters are stored in the `ddalpha` object.
	`methodSeparatorBinary = F` in case of a multiclass classifier
`.<name>_learn`	
trains the classifier. Is different for binary and multiclass classifiers.	
IN:	
`ddalpha`	the ddalpha object, containing the data and settings
`index1`	(only for binary) index of the first class
`index2`	(only for binary) index of the second class
`depths1`	(only for binary) depths of the first class w.r.t. all classes
`depths2`	(only for binary) depths of the second class w.r.t. all classes
	depths w.r.t. only given classes are received by `depths1[,c(index1, index2)]`
	for multiclass separator the depths are accessible via `ddalpha$patterns[[i]]$depths`
OUT:	
`classifier`	the trained `classifier` object
`.<name>_classify`	
classifies the objects	
IN:	
`ddalpha`	the ddalpha object, containing the data and global settings
`classifier`	the previously trained classifier
`objects`	the objects (depths) that are classified
OUT:	
`result`	a vector with classification results: positive values for class `"classifier$index1"` (binary) or the indices of a pattern in `ddalpha` (multiclass)
Usage:	
binary	`R> ddalpha <- ddalpha.train(data, separator = "<name>",` `+  aggregation.method = <any>, <custom params>, ...)`
multiclass	`R> ddalpha <- ddalpha.train(data, separator = "<name>",` `+  aggregation.method = "none", <custom params>, ...)`

trained `classifier` and the depths of the objects that are classified. For a binary classifier, the indices of the currently classified patterns are accessible as `classifier$index1` and `classifier$index2`. A binary classifier shall return a vector with positive values for the objects from the first class, and the multiclass classifier shall assign to each object to be classified the index of the corresponding pattern in `ddalpha`. Similarly to the depth function, the defined separator is accessible by `ddalpha.train` by specifying `separator = "<name>"`. If a nonbinary method is used, it is important to set `aggregation.method = "none"` or (preferred but more complicated) to return `ddalpha$methodSeparatorBinary = F` from the validator, otherwise the method will be treated as a binary one, as by default `aggregation.method = "majority"`.

3.6.3 Additional features

A number of additional functions are implemented in the package to facilitate assessing quality and time of classification, handle multimodally distributed classes, and visualize depth statistics.

Benchmark procedures implemented in the package allow for estimating expected error rate and training time:

```
R> ddalpha.test(learn, test, ...)
R> ddalpha.getErrorRateCV(data, numchunks = 10,  ...)
R> ddalpha.getErrorRatePart(data, size = 0.3, times = 10,  ...)
```

The first function trains the classifier on the `learn` sample, checks it on the `test` one, and reports the error rate, the training time and other related values such as the numbers of correctly and incorrectly classified points, number of ignored outsiders, *etc.* The second function performs a cross-validation procedure over the given data. On each step, every `numchunks`th observation is removed from the data, the classifier is trained on these data and tested on the removed observations. The procedure is performed until all points are used for testing. Setting `numchunks` to n leads to the leave-one-out cross-validation (=jackknife) that is a consistent estimate of the expected error rate. The procedure returns the error rate, *i.e.* the total number of incorrectly classified objects divided by the total number of objects. The third function performs a benchmark procedure by partitioning the given data. On each of `times` steps, randomly picked `size` observations are removed from the data, the classifier is trained on these data and tested on the removed observations. The outputs of this function are the vector of errors, their mean and standard deviation. Additionally, both functions report mean training time and its standard deviation. In all three functions, dots denote the additional parameters passed to `ddalpha.train`. Benchmark procedures may be used to *tune the classifier* by setting different values and assessing the error rate. The function `ddalpha.test` is more appropriate for simulated data, while the two others are more suitable for subsampling learning with real data and testing sequences from it. Analogs of these procedures for a functional setting are present in the package as well:

```
R> ddalphaf.test(learn, learnlabels, test, testlabels, disc.type, ...)
```

```
R> ddalphaf.getErrorRateCV(dataf, labels, numchunks, disc.type, ...)
R> ddalphaf.getErrorRatePart(dataf, labels, size, times, disc.type, ...)
```

The discretization scheme is chosen with parameter `disc.type` setting it to `"LS"` or `"comp"`. Note that these procedures are made to assess the error rates and the learning time for a single set of parameters. If the LS-transform is used, the parameters L and S shall be explicitly set with `adc.args` rather then cross-validated.

Several approaches reflecting multimodality of the underlying distribution are implemented in the package. These methods appear to be useful if the data substantially deviate from elliptical symmetry (*e.g.* having nonconvex or nonconnected support) and the classification based on a global depth fails to achieve close to optimal error rates. The methods need more complicated and fine parameter tuning, whose detailed description we leave to the corresponding articles.

The potential-potential (pot-pot) plot (Pokotylo and Mosler, 2016) bears the analogy to the DD-plot and thus can be directly used in DD-classification as well. The potential of a class j is defined as a kernel density estimate multiplied by the class's prior probability and is used in the same way as a depth

$$\hat{\phi}_j(\boldsymbol{x}) = p_j \hat{f}_j(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n_j} K_{\boldsymbol{H}_j}(\boldsymbol{x}, \boldsymbol{x}_{ji}),$$

with a Gaussian kernel $K_{\boldsymbol{H}}(\boldsymbol{x})$ and bandwidth matrix $\boldsymbol{H} = h^2 \hat{\boldsymbol{\Sigma}}(\boldsymbol{X})$. The bandwidth parameter h (called `kernel.bandwidth` in the package) is separately tuned for each class. The parameters have to be properly tuned, using the following benchmark procedures:

```
R> min_error = list(a = NA, error = 1)
R> for (h in list(c(h_11, h_21), ... , c(h_1k, h_2k)))
+  {
+    error = ddalpha.getErrorRateCV(data, numchunks = <nc>,
+                separator = <sep>, depth = "potential", kernel.bandwidth = h,
+                pretransform = "NMahMom")
+    if(error < min_error$error)
+      min_error = list(a = a, error = error)
+  }
```

Localized spatial depth and a classifier based on it, proposed by Dutta and Ghosh (2015), can be seen as a DD-classifier. The global spatial depth calculates the average of the unit vectors pointing from the points from $\boldsymbol{X}$ in direction $\boldsymbol{z}$. We rewrite (3.3) denoting $\boldsymbol{t}_i = \boldsymbol{\Sigma}^{-\frac{1}{2}}(\boldsymbol{X})(\boldsymbol{z} - \boldsymbol{x}_i)$

$$D_{spt}(\boldsymbol{z}|\boldsymbol{X}) = 1 - \left\| \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{v}(\boldsymbol{t}_i) \right\|.$$

The local version is obtained by kernelizing the distances

$$D_{Lspt}(\boldsymbol{z}|\boldsymbol{X}) = \left\| \frac{1}{n} \sum_{i=1}^{n} K_h(\boldsymbol{t}_i) \right\| - \left\| \frac{1}{n} \sum_{i=1}^{n} K_h(\boldsymbol{t}_i) \boldsymbol{v}(\boldsymbol{t}_i) \right\|,$$

with the Gaussian kernel function $K_h(\boldsymbol{x})$. The bandwidth parameter h defines the localization rate. (If $h > 1$, the depth is multiplied by h^d.)

The depth-based k-*NN* (Paindaveine and Van Bever, 2015) is an affine-invariant version of the k-nearest-neighbor procedure. This method is different, in the sense that it is not using the DD-plot. It is accessible through functions `dknn.train`, `dknn.classify` and `dknn.classify.trained`. For each point $\boldsymbol{x}_0$ to be classified, data points are appended by their reflection w.r.t. $\boldsymbol{x}_0$, which results in the extended centrally symmetric data set of size $2n$. Then the depth of each data point is calculated in this extended data cloud, and $\boldsymbol{x}_0$ is assigned to the most representable class among k points with the highest depth value, breaking ties randomly. Each depth notion may be inserted. Training the classifier constitutes in its tuning by the leave-one-out cross-validation. The method is integrated into the benchmark procedures, accessible there by setting `separator = "Dknn"`.

Depth visualization functions applicable to the two-dimensional data are also implemented in the package. To visualize a depth function as a three-dimensional landscape, use

```
R> depth.graph(data, depth_f,
+        main, xlim, ylim, zlim, xnum, ynum, theta, phi, bold = F, ...)
```

The function accepts additional parameters: plot-limiting parameters `xlim`, `ylim`, `zlim` are calculated automatically, parameters `xnum`, `ynum` control the resolution of the plot, parameters `theta` and `phi` rotate the plot, and with parameter `bold` equal to `TRUE` the data points are drawn in bold face.

Depth contours are pictured by the following functions:

```
R> depth.contours(data, depth, main, xlab, ylab, drawplot = T,
+        frequency=100, levels = 10, col, ...)
R> depth.contours.ddalpha(ddalpha, main, xlab, ylab, drawplot = T,
+        frequency=100, levels = 10, drawsep = TRUE, ...)
```

Function `depth.contours` calculates and draws the depth contours D_α for given `data`. Parameter `frequency` controls the resolution of the plot, and parameter `levels` controls the vector of depth values of α for which the contours are drawn. Note that a single value set as `levels` defines either the depth of a single contour ($0 <$ `levels` ≤ 1) or the number (as its ceiling) of contours that are equally gridded between zero and maximal depth value (`levels` > 1). To combine the contours of several data sets or several different depth notions in one plot, parameter `drawplot` should be set to `FALSE` for all but the first plot and the color should be set individually through `col`. It is also possible to draw depth contours for a previously trained `ddalpha` classifier. In this case classes will differ in colors and the separation will be drawn.

Figures 3.2 and 3.3 show depth surface (left) and depth contours (right) for each of the implemented depth notions. The two plots, *e.g.* for Mahalanobis depth, correspond (without additional parameters that orientate the plot) to the calls `depth.graph(data, "Mahalanobis")` and `depth.contours(data, "Mahalanobis")`.

Another useful function draws the DD-plot either from the trained $DD\alpha$-classifier or from the depth space, additionally indicating the separation between the classes:

```
R> draw.ddplot(ddalpha, depth.space, cardinalities,
+       main = "DD plot", xlab = "C1", ylab = "C2",
+       classes = c(1, 2), colors = c("red", "blue", "green"), drawsep = T)
```

To facilitate saving the default parameters for the plots and resetting them, which may become annoying when done often, function `par(resetPar())` can be used.

Multivariate and functional data sets and data generators have been included in the package **ddalpha** to make the empirical comparison of different classifiers and data depths easier. 50 real multivariate binary classification problems were gathered and described by Mozharovskyi et al. (2015) and are also available at http://www.wisostat.uni-koeln.de/de/forschung/software-und-daten/data-for-classification/. The data can be loaded to a separate variable with function `variable = getdata("<name>")`. Class labels are in the last column of each data set. Functional data sets are accessible through functions `dataf.<name>()` and contain four functional data sets and two generators from Cuevas et al. (2007). A functional data object contains a list of functional observations, each characterized by two vectors of coordinates, the arguments vector `args` and the values vector `vals`, and a list of class labels. Although this format is clear, visualization of such data can be a nontrivial task, which is solved by function `plotf`.

3.6.4 Tuning the classifier

Classification performance depends on many aspects: chosen depth function, separator, outsider treatment, and their parameters.

When selecting a depth function, such properties as ability to reflect asymmetry and shape of the data, robustness, vanishing beyond the convex hull of the data, and computational burden have to be considered.

Depth contours of Mahalanobis depth are elliptically symmetric and those of projection depth are centrally symmetric, thus both are not well suited for skewed data. Contours of spatial depth are also rounded, but fit substantially closer to the data, which can also be said about simplicial volume depth. Being intrinsically nonparametric, halfspace, simplicial, and zonoid depths fit closest to the geometry of the data cloud, but vanish beyond its convex hull, and thus produce outsiders during classification. All these depths are global and not able to reflect localities possibly present in the data. Local spatial depth as well as potentials compensate for this by fitting multimodal distributions well, which is bought at the price of computational burden for tuning a parameter due to an application specific criteria.

Halfspace, simplicial, and projection depths are robust, while outlier sensitivity of Mahalanobis and spatial depths depends on the underlying estimate of the covariance matrix. To obtain their robust versions, the MCD estimator is applied in package **ddalpha**. Parameter `mah.parMcd` used with Mahalanobis and spatial depths corresponds to the portion of the data for which the covariance determinant is minimized. Simplicial volume and zonoid depths, being based on volume and mean, fail to be robust in general as well.

Halfspace, zonoid, and simplicial depths produce outsiders; their depth contours are also not smooth, and the contours of the simplicial depth are even star-shaped. These depths must not be considered if a substantial portion of points lies on the convex hull of the data cloud; in some cases, especially in high dimensions, this may reach 100%, see also Mozharovskyi et al. (2015).

Most quickly computable are Mahalanobis, spatial, and zonoid depths. Their calculation speed depends minorly on data dimension and moderately on the size of the data set, while computation time for simplicial, simplicial volume, and exact halfspace depths dramatically increases with the number of points and dimension of the data. Approximating algorithms balance between calculation speed and precision depending on their parameters. Random halfspace and projection depths are driven by parameter `num.directions`, *i.e.* the number of directions used in the approximation. The approximations of simplicial and simplicial volume depths depend on the number of simplices picked, which is set with parameter `k`. If a fixed number of simplices $\texttt{k} > 1$ is given the algorithmic complexity is polynomial in d but is independent of n, given `k`. If a proportion of simplices is given ($0 < \texttt{k} < 1$), then the corresponding portion of all simplices is used and the algorithmic complexity is exponential in n, but one can assume that the approximation precision is kept on the same level when n changes. Note that in $\mathbb{R}^2$, the exact efficient algorithm of Rousseeuw and Ruts (1996) is used to calculate simplicial depth.

Based on the empirical study using real data (Pokotylo and Mosler, 2016), the classifiers' error rates grow in the following order: $DD\alpha$, polynomial classifier, k-NN; although $DD\alpha$ and the polynomial classifier provide similar polynomial solutions and k-NN sometimes delivers good results when the other two fail. The degree of the $DD\alpha$ and the polynomial classifier and the number of nearest neighbors are automatically cross-validated, but maximal values may be set manually. To gain more insights, depth-transformed data may be plotted (using `draw.ddplot`).

The outsider treatment should not be regarded as the one that gives the best separation of the classes in the original space, but rather be seen as a computationally cheap solution for points right beyond their convex hulls.

In functional classification, parameters L and S can be set by the experience-guided applicant or determined automatically by means of cross-validation. The ranges for cross-validation can be based on previous knowledge of the area or conservatively calculated.

Benchmark procedures that we included in the package may be used for empirical parameters' tuning, by iterating the parameters values and estimating the error rates. For example,

the following code fragment searches for the separator, depth, and some other parameters, which deliver best classification:

```
R> min_error = list(error = 1, par = NULL)
R> for (par in list(par_set_1, ... , par_set_k))
+  {
+    error = ddalpha.getErrorRateCV(data, numchunks = <nc>,
+                separator = par$sep, depth = par$depth,
+                other_par = par$other_par )
+    if(error < min_error$error)
+      min_error = list(error = error, par = par)
+  }
```

Appendix
The α-procedure

The α-procedure Vasil'ev and Lange (1998), Vasil'ev (2003) is an iterative procedure that finds a linear solution in the given properties space. If no good linear solution exists in the original space it be extended with extra properties, e.g., using polynomial extension. The linear solution in the extended space leads then to a non-linear solution in the original one. The procedure iteratively synthesizes the space of features, choosing those minimizing two-dimensional empirical risk in each step as it is illustrated on Figure 3.11.

On the first step all pairs of properties are considered and the one leading to the best linear separation in its two-dimensional space is taken. This first solution is characterized by *properties* $p_{(1)}$ and $p_{(2)}$ and the *angle* α_1 of the *normal vector* of the separating line. The points are then projected to the normal vector and form a *feature* f_1 (Figure 3.11.a). On the following s-steps $s \geq 2$, the feature f_{s-1}, obtained on the previous $(s-1)$-step, is coupled with each of the properties that were not included to the solution. Again we select the property $p_{(s+1)}$ leading to the best linear separation in the two-dimensional space with f_{s-1}, and find the angle α_s and the feature f_s (Figure 3.11.b). The procedure is performed as long as the new features improve the separation by reducing the empirical risk. The final *separating hyperplane* is orthogonal to the last feature f_s (Figure 3.11.c), and is characterized by the *set of properties* $\{p_{(j)}\}, j = 1, ..., s+1$, the *direction vector* $\boldsymbol{r}$ and the *distance* from the space origin to the separating hyperplane t (Figure 3.11.d).

The direction vector $\boldsymbol{r}$ of the separating hyperplane is decomposed along the properties $p_{(j)}$ and is found as

$$\boldsymbol{r} = \left(\prod_{i=1}^{s} cos(\alpha_i); \quad sin(\alpha_1) \prod_{i=2}^{s} cos(\alpha_i); \quad ... \quad ; \quad sin(\alpha_q) \prod_{i=q+1}^{s} cos(\alpha_i); \quad ... \quad ; \quad sin(\alpha_k) \right),$$

where the length of $\boldsymbol{r}$ equals to one, each coordinate of $\boldsymbol{r}$ is measured along the corresponding property and α_i is the angle obtained on step i as illustrated on Figure 3.12. This procedure is most reasonably done iteratively as in step 5 of the algorithm of the $DD\alpha$-separator (see Section 3.3.1).

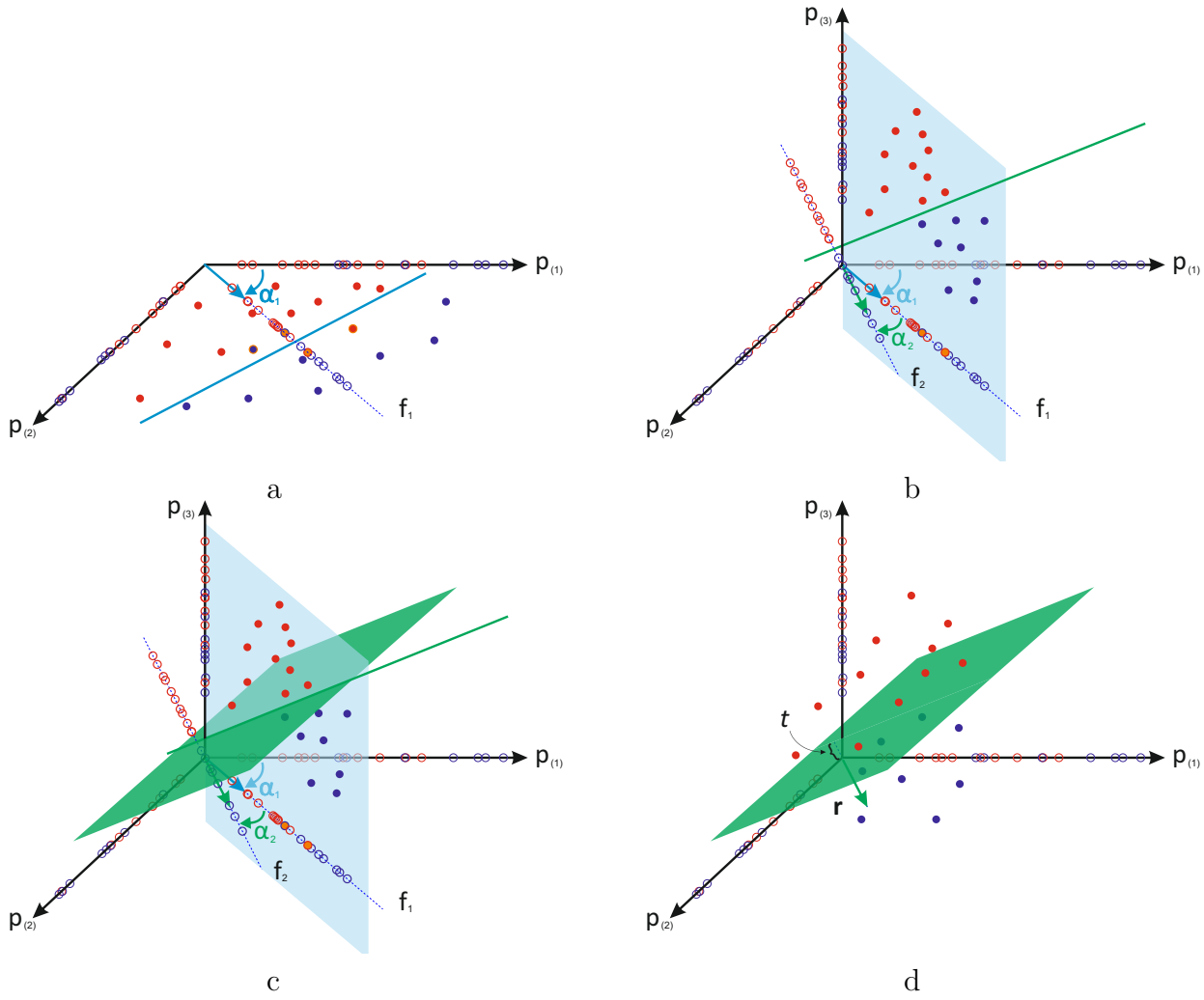


Figure 3.11: Steps of the α procedure. The solution is first found in a space of properties $p_{(1)}$ and $p_{(2)}$ and a feature f_1 is formed (a), then in a space spanned by the feature f_1 and $p_{(3)}$ forming f_2 (b). The final separating hyperplane is orthogonal to the last feature f_2 (c), and is characterized by the set of properties $p_{(j)}$, the direction vector $\boldsymbol{r}$ and the distance t (d).

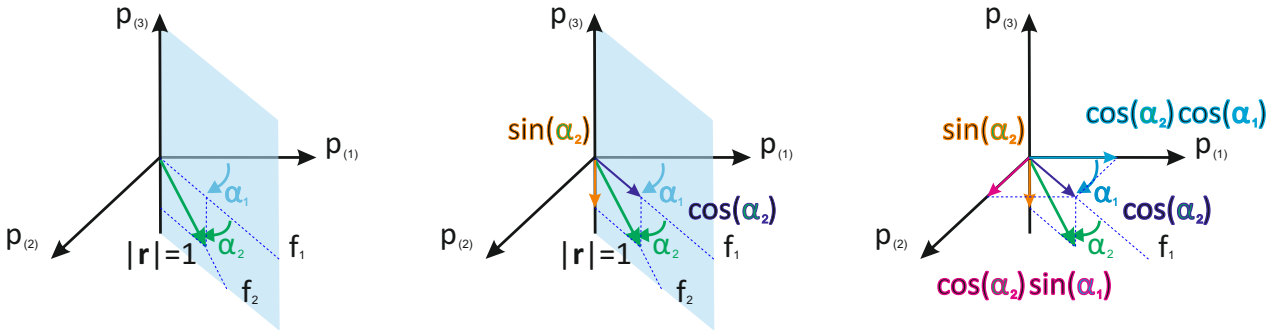


Figure 3.12: Retrieving components of the direction vector.

Chapter 4

Depth-weighted Bayes classification

4.1 Introduction

Let us consider a classification problem which consists in creating a rule for assigning new observations to one of two (or more) distributions. For this moment let us assume that the distributions are known and that their supports are overlapping. Then there does not exist any rule with zero probability of misclassification. A very natural request is to classify at least the "typical" points correctly. For example, we seamlessly accept wrong classification of the point $x = 4.417$ which comes from $P_1 = N(0, 1)$, as $P(|x| > 4.417|x \sim P_1) \approx 10^{-5}$. On the other hand the points "close to zero" should be assigned to this distribution.

The requirement of correct classification of "typical" points might not be met when using the (Bayes) classifier which guarantees the minimal probability of misclassification, especially in case of imbalanced data. For example, if $P_1 = N(0, 1)$, $P_2 = N(1, 1)$ and the prior probabilities of these distributions are $\pi_1 = 0.7, \pi_2 = 0.3$, then the point $x = 1$ will be assigned to P_1 by the Bayes classifier although it is the center (mode=med=mean) of the distribution P_2, as it is more likely that the point $x = 1$ comes from P_1 than from P_2. In such situations the Bayes classifier may be additionally weighted to achieve the desired misclassification rate for the minor class, but in this case its outliers are also overweighted which leads to misclassification of the major class in their neighbourhood.

What do we mean by the term "typical"? In the previous example nobody doubts that the point $x = 1$ plays a central role for the distribution $N(1, 1)$. Hence, typical points are those that are close to the center whereas outliers represent rather atypical cases. But still, the term center should be discussed in detail. The term is clear for symmetric distributions where the center is the point of symmetry. Note that the point of symmetry defines the median for univariate variables. The median can be considered to be the center of distribution even if the distribution is not symmetric. On the other hand the notion of quantile can be used to define outlyingness. The whole concept might be generalized for multivariate distributions using the notion of data depth, see e.g. Mosler (2013). Depth function provides a measure of centrality. The point with the highest depth is a multivariate analogy to the median, while points far from the center have small depth.

It is important to become aware of similarities as well as differences between the Bayes classifier which uses the density function and any other approach based on a depth function. In a simple, but fundamental, case of a unimodal elliptically symmetric distribution the level sets of a depth function correspond to those of the density function. However the correspondence disappears as soon as the assumption of unimodality or symmetry is not fulfilled. One can argue that the assumption of unimodality is justified in the context of classification (a class which is a location mixture might be decomposed into several unimodal subclasses), however there is no justification for the assumption of symmetry. And the difference between depth and density might be substantial when the distribution is skewed. Let us consider a simple example of a lognormal distribution which logarithm is standard N(0,1), see Figure 4.1. Let $q_{0.05}$ be 5% quantile of $N(0,1)$. Note that $-q_{0.05}$ is 95% quantile of $N(0,1)$. Consider points $x_1 = \exp(q_{0.05})$ and $x_2 = \exp(-q_{0.05})$. Both x_1 and x_2 determine areas of 5% of extreme values (low in case of x_1 and high in case of x_2), so that they have exactly the same (non)-central location. This is manifested in the fact that their depth[1] is equal. However there is an immense difference in density in these points: $f(x_1) = 0.53, f(x_2) = 0.02$, indeed $f(x_1)$ is almost 26 times greater than $f(x_2)$. So the correct classification of the point x_1 is much more important than the correct classification of the point x_2 from the classical point of view, but it is equally important from the point of view based on centrality of the points.

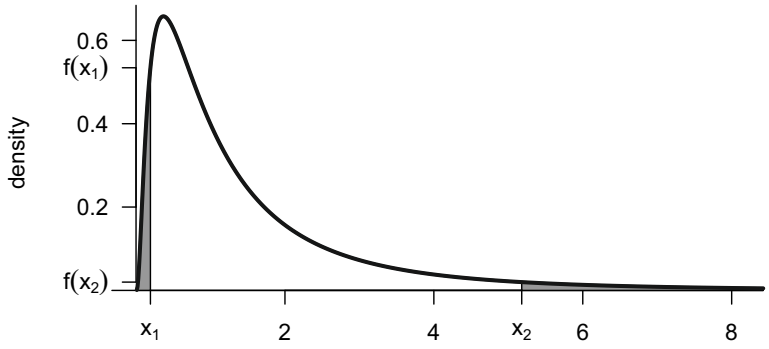


Figure 4.1: The lognormal distribution. Points x_1 and x_2 determine areas of 5% of extreme values and thus have the same outlyingness, although the density is much higher in x_1.

In recent years several classifiers using different notions of data depth were proposed. The maximum-depth classifier and its improvement (Ghosh and Chaudhuri, 2005b), the DD-plot classifier (Liu et al., 1999), the k-NN in the DD-plot (Vencalek, 2014) and the DD-alpha procedure (Lange et al., 2014b) are examples of such classifiers. Although the concept of these classifiers is different from that of the Bayes classifier, they were traditionally compared to it. In some cases the average misclassification rate (an empirical version of the probability of misclassification) of the depth-based classifiers have been proven to be asymptotically approaching the error rate of the Bayes classifier. But these are rather special cases, e.g. when the considered distributions are elliptically symmetric (Ghosh and Chaudhuri, 2005b). Depth-based classifiers are not primarily constructed to minimize total probability of misclassification

[1] more precisely: halfspace depth

(or the average misclassification rate). In the current chapter we discuss another measure of performance that can be used for the evaluation of the depth-based (as well as any other) classifier.

The discussion about alternative measures of classifiers' performance leads directly to introduction of the depth-weighted and the depth-rank weighted classifiers. Instead of the global weighting of the classes we propose the depth weights of the misclassification cost, such that the outlying points get less weight than the central ones. Analysis of the properties of the newly proposed classifiers is the objective of this chapter.

The chapter is structured as follows. Section 4.2 presents the Bayes classifier and introduces the depth-weighted classifier. Section 4.3 discusses the relationship between the newly proposed classifier and the Bayes classifier, their possible coincidence as well as their maximal possible difference. Discussion about the choice of the depth function included in Section 4.4 leads to the introduction of the rank-weighted classifier. A broad simulation study conducted to explore behaviour of newly proposed classifiers is reported in Section 4.5. The robustness of the method is inspected in Section 4.6. Section 4.7 concludes.

4.2 Bayes classifier, its optimality and a new approach

In this section we briefly recall the Bayes classifier and the notion of cost function in context of classification. Later we introduce a new criterion of optimality and the classifier which is optimal for this criterion.

Let $\boldsymbol{X}$ be a d-dimensional random variable which follows one of the $K \geq 2$ absolutely continuous distributions $P_i, i = 1, \ldots, K$, defined on $\mathbb{R}^d$. Their densities are denoted by f_i and prior probabilities by π_i. A classifier divides the space $\mathbb{R}^d$ into K disjoint parts $A_i, i = 1, \ldots, K$, $\bigcup_{i=1}^{K} A_i = \mathbb{R}^d$ such that any $\boldsymbol{x} \in \mathbb{R}^d$ is assigned to P_i iff $\boldsymbol{x} \in A_i$. Equivalently we can write $class(\boldsymbol{x}) = i \Leftrightarrow \boldsymbol{x} \in A_i$.

4.2.1 Bayes classifier and the notion of cost function

The Bayes classifier has the following form:

$$class_B(\boldsymbol{x}) = \underset{i}{\operatorname{argmax}}\, f_i(\boldsymbol{x})\pi_i. \tag{4.1}$$

It can be easily shown that the Bayes classifier minimizes the probability of misclassification (Devroye et al., 1996). However, it can be also viewed as the classifier minimizing the expected cost for a particular cost function

$$c_{ij}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } j \neq i, \\ 0 & \text{if } j = i, \end{cases}$$

where $c_{ij}(\boldsymbol{x})$ denotes cost of classifying object $\boldsymbol{x}$ to the distribution P_j while it comes from P_i.

The overall expected cost L can be expressed in terms of conditional expected costs L_i and prior probabilities π_i:

$$L = \sum_{i=1}^{K} L_i \pi_i = \sum_{i=1}^{K} \sum_{j=1}^{K} \int_{A_j} c_{ij}(\boldsymbol{x}) f_i(\boldsymbol{x}) \pi_i d\boldsymbol{x}. \tag{4.2}$$

Minimization of the overall expected cost simplifies substantially when the cost function c_{ij} is zero when the classification is correct ($j = i$) and does not depend on j otherwise, so that it has the following form:

$$c_{ij}(\boldsymbol{x}) = \begin{cases} c_i(\boldsymbol{x}) & \text{if } j \neq i, \\ 0 & \text{if } j = i. \end{cases}$$

In this case the equation (4.2) can be rewritten as

$$\begin{aligned} L &= \sum_{i=1}^{K} \sum_{j \neq i} \int_{A_j} c_i(\boldsymbol{x}) f_i(\boldsymbol{x}) \pi_i d\boldsymbol{x} \\ &= \sum_{i=1}^{K} \sum_{j=1}^{K} \int_{A_j} c_i(\boldsymbol{x}) f_i(\boldsymbol{x}) \pi_i d\boldsymbol{x} - \sum_{i=1}^{K} \int_{A_i} c_i(\boldsymbol{x}) f_i(\boldsymbol{x}) \pi_i d\boldsymbol{x} \\ &= \sum_{i=1}^{K} \int_{\mathbb{R}^d} c_i(\boldsymbol{x}) f_i(\boldsymbol{x}) \pi_i d\boldsymbol{x} - \sum_{i=1}^{K} \int_{A_i} c_i(\boldsymbol{x}) f_i(\boldsymbol{x}) \pi_i d\boldsymbol{x}. \end{aligned}$$

Since the first term does not depend on the classifier, the minimization of overall expected cost L is equivalent to the maximization of the second term $\sum_{i=1}^{K} \int_{A_i} c_i(\boldsymbol{x}) f_i(\boldsymbol{x}) \pi_i d\boldsymbol{x}$.

The classifier minimizing overall expected loss is then

$$class(\boldsymbol{x}) = \operatorname*{argmax}_i c_i(\boldsymbol{x}) f_i(\boldsymbol{x}) \pi_i.$$

4.2.2 Depth-weighted classifier

We are suggesting to use the depth of a point $\boldsymbol{x}$ with respect to the distribution from which it comes as a cost of misclassification, that is

$$c_{ij}(\boldsymbol{x}) = \begin{cases} D_i(\boldsymbol{x}) & \text{if } j \neq i, \\ 0 & \text{if } j = i, \end{cases} \tag{4.3}$$

where $D_i(\boldsymbol{x})$ is the depth of $\boldsymbol{x}$ w.r.t. P_i. Any depth function can be used here. Instead of D_i itself one can also use more general weight $w_i D_i$, where the parameter w_i may be tuned for the imbalanced data to achieve the desired misclassification rate in one of the classes. In what follows we assume $w_i = 1$ and drop this term.

It is important to realize that the misclassification cost is not the same for all points of a certain group, but is specific for each point and depends on its position with respect to the

distribution from which it is sampled. Here, the misclassification of the points close to the center of the data cloud is seen as a more serious mistake than the misclassification of the outlying points. The main idea is thus to weight the errors using data depth.

The classifier that minimizes the total expected cost assigns a new observation $\boldsymbol{x}$ to the group, where the product of its depth, density and prior probability is maximal:

$$class_D(\boldsymbol{x}) = \underset{i}{\operatorname{argmax}}\, D_i(\boldsymbol{x}) f_i(\boldsymbol{x}) \pi_i. \tag{4.4}$$

In what follows we call the classifier (4.4) the *depth-weighted classifier* and abbreviate it in tables and figures as *DW*. We will demonstrate practical consequences of this approach by several examples in Section 4.2.3.

4.2.3 Examples

Let us now illustrate differences between the Bayes classifier and the newly proposed depth-weighted classifier on two simple examples. For simplicity, both examples deal with univariate random variables. We use halfspace depth here. The difference of the two considered approaches is illustrated in Figure 4.2.

1. Let us consider two uniform distributions with partly overlapping supports $P_1 = Unif[0, 100]$, $P_2 = Unif[50, 250]$ and equal priors $\pi_1 = \pi_2 = 0.5$. The Bayes classifier will classify all points from the overlapping part to P_1, since $f_1(x)\pi_1 > f_2(x)\pi_2$ for all $x \in [50, 100]$. The depth-weighted classifier differs from the Bayes classifier on an interval $(90, 100]$, which is classified to P_2, as the interval is closer to its center, corresponding to the 20-25% quantiles of P_2 and only to the 90% and higher quantiles of P_1.

2. Let us consider two normal distributions differing in location and priors, but with the same scale parameters: $P_1 = N(0, 50^2)$, $P_2 = N(100, 50^2)$, $\pi_1 = 0.75$, $\pi_2 = 0.25$. The Bayes classifier will classify x to P_1 iff $x < 50 + 25 \cdot \log \frac{0.75}{0.25} \approx 77.47$. If the priors were equal, the point separating the classes would be placed right in the middle between the centers of the distributions ($x = 50$). In the considered case it is shifted closer to the center of the distribution with the smaller prior probability (P_2). Thus the probability of misclassification is higher if the point is generated from P_2: $P(class(x) \neq 2 | x \in P_2) \approx 0.326$, $P(class(x) \neq 1 | x \in P_1) \approx 0.061$.

   The separating point of the depth-weighted classifier can be computed numerically. Its value is about 60.87. As before it is shifted from the middle point closer to the distribution with the smaller probability, but now the shift is smaller. The difference in misclassification rates will not be so high now: $P(class(x) \neq 2 | x \in P_2) \approx 0.217$, $P(class(x) \neq 1 | x \in P_1) \approx 0.112$.

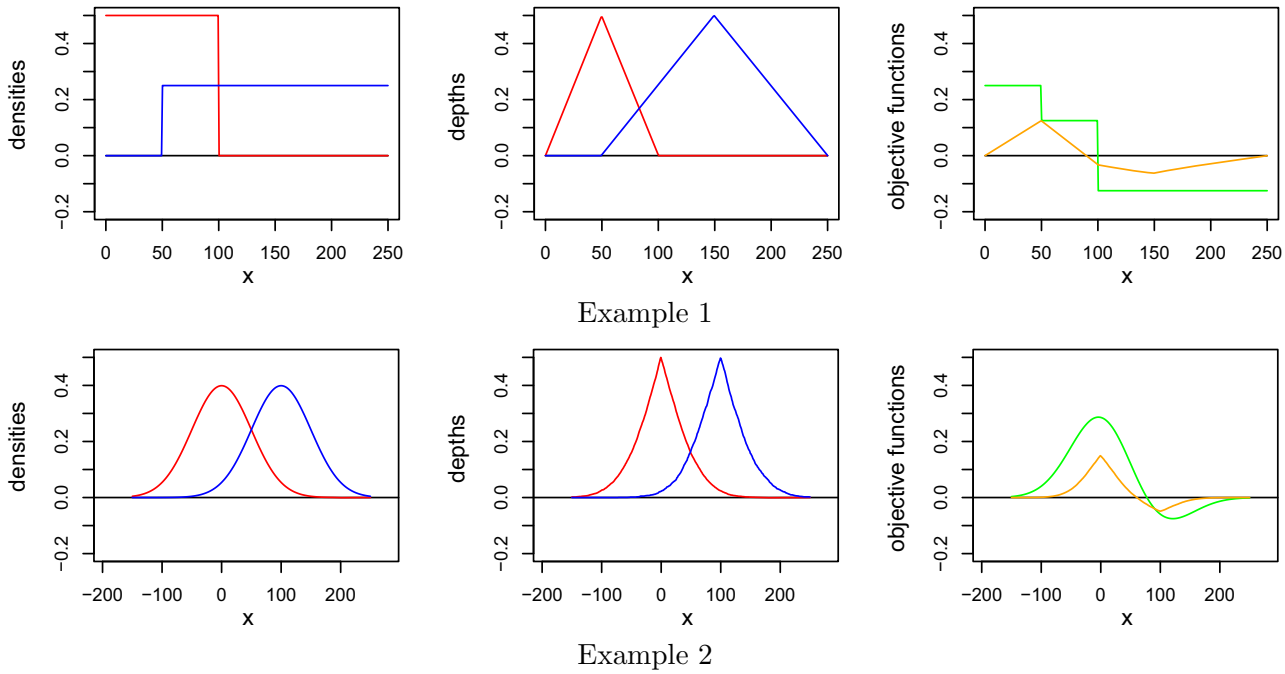


Figure 4.2: Illustration to the examples 1 and 2: densities of P_1 and P_2 (left), depths of points w.r.t P_1 and P_2 (middle) and objective functions of the Bayes classifier $f_1(\boldsymbol{x})\pi_1 - f_1(\boldsymbol{x})\pi_2$ (green) and depth-weighted classifier $D_1(\boldsymbol{x})f_1(\boldsymbol{x})\pi_1 - D_2(\boldsymbol{x})f_1(\boldsymbol{x})\pi_2$ (orange) (right).

4.3 Difference between the depth-weighted and the Bayes optimal classifiers

Let us now investigate properties of the newly defined depth-weighted classifier. We will compare it to the Bayes optimal classifier in the two-classes problem when the distributions are elliptically symmetric.

In what follows it is assumed that the distributions P_i have the following two properties:

(P1) They are elliptical of the same radial type, that is, have the same radial density up to scale. Technically this means that their densities f_i can be expressed as $f_i(\boldsymbol{x}) = k_i g(M_i(\boldsymbol{x}))$, where g is a decreasing function, $k_i > 0$ are constants, and $M_i(\boldsymbol{x}) = \left((\boldsymbol{x}-\boldsymbol{\alpha}_i)'\boldsymbol{B}_i^{-1}(\boldsymbol{x}-\boldsymbol{\alpha}_i)\right)^{\frac{1}{2}}$ with $\boldsymbol{B}_i$ positive definite, $\boldsymbol{\alpha}_i \in \mathbb{R}^d$, denotes the generalized distance of the point $\boldsymbol{x}$ from the center of the distribution P_i.

(P2) $D(\boldsymbol{x})$ is an affine invariant depth. Then, given (P1), $D_i(\boldsymbol{x}) = D(\boldsymbol{x}|P_i)$ is a fixed decreasing function of generalized distance, that can be expressed as $D_i(\boldsymbol{x}) = h(M_i(\boldsymbol{x}))$, where h is some decreasing function.

Note that the assumptions are fulfilled for example for $P_i = N(\mu_i, \boldsymbol{\Sigma}_i)$. The first assumption is fulfilled while $f_i(\boldsymbol{x}) = ((2\pi)^d|\boldsymbol{\Sigma}_i|)^{-\frac{1}{2}}\exp(-\frac{1}{2}(\boldsymbol{x}-\mu_i)'\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\mu_i)) = k_i\exp(-\frac{1}{2}M_i^2(\boldsymbol{x}))$ and the second because of affine invariance and monotonicity of the depth function.

First we define situations where the depth-weighted classifier differs from the Bayes classifier, and those where they lead to the same classification rule. We introduce the following

notation $G(\boldsymbol{x}) = \frac{k_1 g(M_1(\boldsymbol{x}))}{k_2 g(M_2(\boldsymbol{x}))}$ as likelihood ratio, $H(\boldsymbol{x}) = \frac{h(M_1(\boldsymbol{x}))}{h(M_2(\boldsymbol{x}))}$ as depth ratio and $\pi = \frac{\pi_2}{\pi_1}$ as inverse prior ratio and rewrite the classifiers in terms of these functions:

- The Bayes classifier assigns $\boldsymbol{x}$ to P_1 if $G(\boldsymbol{x}) > \pi$ (to P_2 if $G(\boldsymbol{x}) < \pi$),
- The depth-weighted classifier assigns $\boldsymbol{x}$ to P_1 if $H(\boldsymbol{x})G(\boldsymbol{x}) > \pi$ (to P_2 if $H(\boldsymbol{x})G(\boldsymbol{x}) < \pi$).

It is worthwhile to note that the relationship between $H(\boldsymbol{x})G(\boldsymbol{x})$ and $G(\boldsymbol{x})$ depends on the generalized distances of $\boldsymbol{x}$ from P_1 and P_2:

$$M_1(\boldsymbol{x}) < M_2(\boldsymbol{x}) \quad \Rightarrow \quad H(\boldsymbol{x})G(\boldsymbol{x}) > G(\boldsymbol{x}) > \frac{k_1}{k_2}, \tag{4.5}$$

$$M_1(\boldsymbol{x}) > M_2(\boldsymbol{x}) \quad \Rightarrow \quad H(\boldsymbol{x})G(\boldsymbol{x}) < G(\boldsymbol{x}) < \frac{k_1}{k_2}. \tag{4.6}$$

These inequalities together with their relation to the considered classifiers are illustrated in Figure 4.3. The classifiers differ when π is between $G(\boldsymbol{x})$ and $H(\boldsymbol{x})G(\boldsymbol{x})$. For the fixed π, the region where the classifiers differ can be expressed as

$$RD(\pi) = \left\{\boldsymbol{x} \in \mathbb{R}^d : H(\boldsymbol{x})G(\boldsymbol{x}) < \pi < G(\boldsymbol{x}) \text{ or } G(\boldsymbol{x}) < \pi < H(\boldsymbol{x})G(\boldsymbol{x})\right\}.$$

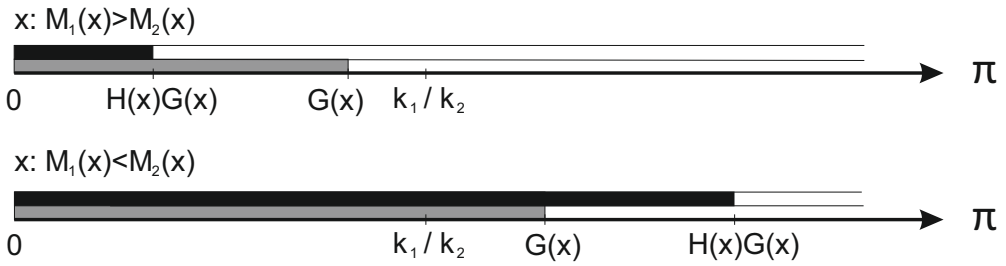


Figure 4.3: Dependence of the Bayes (gray) and the depth-weighted (black) classifiers on the prior probabilities and the position of $\boldsymbol{x}$ w.r.t. P_1 and P_2. Coloured are the regions assigned to P_1, e.g. for $\boldsymbol{x} : M_1(\boldsymbol{x}) < M_2(\boldsymbol{x})$: if $\pi < G(\boldsymbol{x})$, both rules assign to P_1; if $G(\boldsymbol{x}) < \pi < H(\boldsymbol{x})G(\boldsymbol{x})$, the Bayes classifier assigns to P_2 and the depth-weighted classifier assigns to P_1; $\pi > H(\boldsymbol{x})G(\boldsymbol{x})$, both rules assign to P_2.

Theorem 4.1 *Let (P1) and (P2) hold for P_1 and P_2. Then*

$$P(class_B(\boldsymbol{X}) \neq class_D(\boldsymbol{X})) = 0 \Leftrightarrow \pi_1 k_1 = \pi_2 k_2.$$

Roughly speaking, the theorem states necessary and sufficient condition for prior probabilities to let the Bayes and depth-weighted classifier be the same in the considered situation of two elliptically symmetric distributions.

Proof: Equation $\pi_1 k_1 = \pi_2 k_2$ can be rewritten as $\pi = \frac{k_1}{k_2}$. From (4.5) and (4.6) it follows that $P(RD(k_1/k_2)) = 0$. For any other $\pi > 0$, it holds $RD(\pi) \neq \emptyset$ and $P(RD(\pi)) > 0$. □

Let us further investigate the case of highly unequal prior probabilities ($\pi_1 \to 0$ or $\pi_1 \to 1$). We will show that the probability of different classification by Bayes and depth-weighted classifier is small (goes to zero).

Theorem 4.2 *Let (P1) and (P2) hold for P_1 and P_2. Then*

$$P(class_B(\boldsymbol{X}) \neq class_D(\boldsymbol{X})) \to 0 \text{ for } \pi_1 \to 0 \text{ or } \pi_1 \to 1.$$

Proof: The proof will be done for the case $\pi_1 \to 0$ ($\pi \to \infty$). Similar reasoning holds also for the other case, i.e for $\pi_1 \to 1$ ($\pi \to 0$).

Let us consider d-dimensional closed intervals $I_n = [-n, n] \times \ldots \times [-n, n]$. For any fixed $n \in \mathbb{N}$ we can find $\pi \in \mathbb{R}^+$ such that $\max_{\boldsymbol{x} \in I_n} H(\boldsymbol{x})G(\boldsymbol{x}) < \pi$. Thus all points from I_n will be assigned to P_2 by both Bayes and depth-weighted classifier. The theorem directly follows from the fact that $P(\boldsymbol{X} \in I_n) \to 1$ for $n \to \infty$. □

The next theorem extends our knowledge about the area where the Bayes classifier and depth-weighted classifier are equal.

Theorem 4.3 *Let (P1) and (P2) hold for P_1 and P_2. If $\pi < \frac{k_1}{k_2}$ (i.e. $\pi_1 k_1 > \pi_2 k_2$) then*

1. $M_1(\boldsymbol{x}) < M_2(\boldsymbol{x}) \Rightarrow class_B(\boldsymbol{x}) = class_D(\boldsymbol{x}) = 1$ *and*
2. $f_1(\boldsymbol{x})\pi_1 < f_2(\boldsymbol{x})\pi_2 \Rightarrow class_B(\boldsymbol{x}) = class_D(\boldsymbol{x}) = 2$.

Note that a similar proposition holds for $\pi > \frac{k_1}{k_2}$. It is enough to exchange the group labels.

Proof: $\pi < \frac{k_1}{k_2} \Rightarrow \pi_1 k_1 > \pi_2 k_2$.

From $M_1(\boldsymbol{x}) < M_2(\boldsymbol{x})$ it follows that $D_1(\boldsymbol{x}) > D_2(\boldsymbol{x})$ and $g(M_1(\boldsymbol{x})) > g(M_2(\boldsymbol{x}))$. Thus $D_1(\boldsymbol{x})g(M_1(\boldsymbol{x}))k_1\pi_1 > D_2(\boldsymbol{x})g(M_2(\boldsymbol{x}))k_2\pi_2$ and so $class_D(\boldsymbol{x}) = 1$; similarly $g(M_1(\boldsymbol{x}))k_1\pi_1 > g(M_2(\boldsymbol{x}))k_2\pi_2$ and so $class_B(\boldsymbol{x}) = 1$.

From $f_1(\boldsymbol{x})\pi_1 < f_2(\boldsymbol{x})\pi_2$ it directly follows that $class_B(\boldsymbol{x}) = 2$. When writing the inequality in terms of k_i and g we have $k_1 g(M_1(\boldsymbol{x}))\pi_1 < k_2 g(M_2(\boldsymbol{x}))\pi_2$ and thus $k_1 g(M_1(\boldsymbol{x}))/k_2 g(M_2(\boldsymbol{x})) < \pi_2/\pi_1 < k_1/k_2$, which implies $M_1(\boldsymbol{x}) > M_2(\boldsymbol{x})$ and hence $D_1(\boldsymbol{x}) < D_2(\boldsymbol{x})$. Finally we have $D_1(\boldsymbol{x})g(M_1(\boldsymbol{x}))k_1\pi_1 < D_2(\boldsymbol{x})g(M_2(\boldsymbol{x}))k_2\pi_2$ and so $class_D(\boldsymbol{x}) = 2$. □

From the second implication of the Theorem 4.3 it is clear that compared to the Bayes classifier, just one group will receive more assignments by the depth-weighted classifier, i.e. the separating line moves strictly towards one of the groups.

4.4 Choice of depth function and the rank-weighted classifier

The definition of depth-weighted classifier is general in the sense that any depth function might be used as a cost function, see formula (4.3). This universality is important because it allows to choose any of the commonly used depth functions found in e.g. Zuo and Serfling (2000) and Mosler (2013). However, use of different depth functions leads to possible differences in the classification rule. Here we discuss various aspects of the choice of the depth function in the classifier (4.4).

Next we form a list of aspects that should be taken into account when choosing a depth function: Firstly, adding the weights shall improve the robustness of the classifier. The problem of outsiders is also of big importance: if the depth function vanishes beyond the convex support of the data, the observations get zero depth there and cannot be classified. The calculation speed is also important to deal with big data sets, therefore we refer only to computationally efficient depths. The depth shall also follow the shape of non-elliptical and skewed distributions. For this reason we mainly concentrate on the spatial depth (Vardi and Zhang, 2000), as it meets all these requirements. On the contrary, Mahalanobis depth (Mahalanobis, 1936) has always elliptically symmetric level sets; halfspace (Tukey, 1975) and zonoid (Koshevoy and Mosler, 1997) depths produce outsiders; projection depth (Zuo and Serfling, 2000) has contours that are symmetric to the center and, thus, is less suited for skewed data.

A natural question is how much the final classifier depends on the depth function which is used. In the Section 4.4.1 we use one of the examples from Section 4.2.3 to show that the difference between two depth-weighted classifiers which use different depth functions might be not negligible. Dependence of the depth-weighted classifier on the used depth function might be considered as undesirable in some cases. In the Section 4.4.2 we propose a slight modification of the cost function which partly remedies the problem. Dependence of the classifier on the choice of depth is precluded by this modification in elliptically symmetric data.

4.4.1 Example illustrating differences in classification arising from different depths

Let us recall the first example from the Section 4.2.3. Two uniform distributions with partly overlapping supports $P_1 = Unif[0, 100]$, $P_2 = Unif[50, 250]$ and equal priors $\pi_1 = \pi_2 = 0.5$ are considered. It was shown that the depth-weighted classifier has the following form:

$$class(x) = \begin{cases} 1 & \text{if } x < 90, \\ 2 & \text{if } x > 90. \end{cases} \tag{4.7}$$

The separation point $x = 90$ is computed as a solution of equation $D_1(x)f_1(x)\pi_1 = D_2(x)f_2(x)\pi_2$ on the interval (50,100), where $f_1(x) = 1/100$, $f_2(x) = 1/200$, $\pi_1 = \pi_2 = 1/2$ and finally, using the halfspace depth,

$$D_1(x) = -\frac{1}{100}x + \frac{1}{2}; \qquad D_2(x) = \frac{1}{200}x - \frac{1}{4}.$$

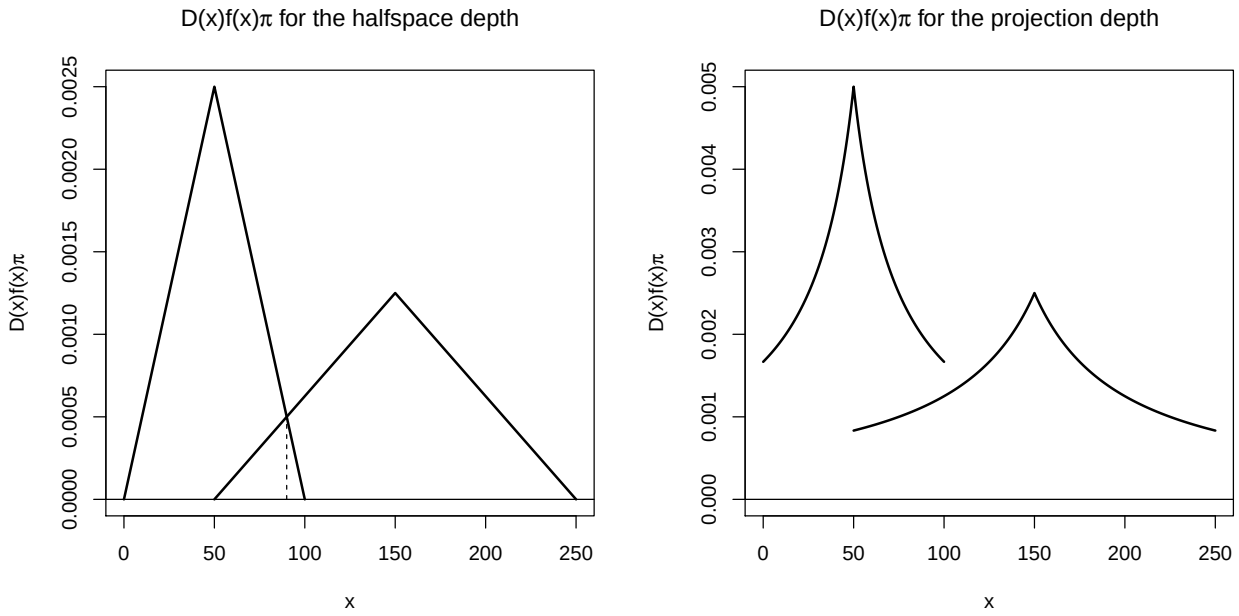
Now it can be easily shown that the depth-weighted classifier differs quite a lot if the projection depth is used instead of the halfspace depth. In this case the depth-weighted classifier has the following form:

$$class(x) = 1 \quad \text{for all } x < 100. \tag{4.8}$$

This is because the fact, that $D_1(x)f_1(x)\pi_1 > D_2(x)f_2(x)\pi_2$ for all $x \in (50, 100)$ when considering projection depth. Let us recall that the projection depth is defined (in the one dimensional case) as a fraction $\frac{1}{1+o(x)}$, where $o(x)$ denotes "outlyingness" of x, which can be computed as $\frac{|x-\text{median}(X)|}{\text{MAD}(X)}$, where $\text{MAD}(X) = \text{median}(|X - \text{median}(X)|)$ is the median absolute deviation. Thus in this example for all $x \in (50, 100)$ it holds

$$D_1(x) = \frac{1}{1 + \frac{x-50}{25}}; \qquad D_2(x) = \frac{1}{1 + \frac{150-x}{50}}.$$

The difference between the depth functions which results to the difference in classification is illustrated in Figure 4.4.



Example 1

Figure 4.4: Difference between the depth-based classifiers using halfspace and projection depths.

4.4.2 Rank-weighted classifier

The problem illustrated in the preceding section arises from the fact that different depth functions have different ranges of their values. While the halfspace depth has values between zero and one-half, the values of projection depth are always positive, approaching zero for only very distant points that do not occur in practice, with the maximal value equal to one. Such a difference might lead to different classification rules. In case of depth functions with virtually equal ranges their growth behavior may also differ, yielding different classification results. To make the procedure less dependent on choice of the depth function we suggest to use the rank function based on depth instead of the depth itself.

The main purpose of depth functions is the ordering of points in multidimensional space. What more matters is thus the rank of a point based on depth, not the value of the depth itself. Any given point might be characterized by the proportion of points having lower depth.

The empirical version of this proportion is (up to a multiplicative constant), depth-based rank of the observation.

Thus for a point $\boldsymbol{x}$ coming from the i-th distribution P_i whose depth with respect to this distribution is equal to $D_i(\boldsymbol{x})$ we suggest to use the proportion of points with lower depth than $D_i(\boldsymbol{x})$ as a cost of its eventual misclassification. More precisely, the cost function has the following form:

$$c_{ij}(\boldsymbol{x}) = \begin{cases} F_i(D_i(\boldsymbol{x})) & \text{if } j \neq i, \\ 0 & \text{if } j = i, \end{cases} \tag{4.9}$$

where $D_i(\boldsymbol{x})$ is the depth of $\boldsymbol{x}$ w.r.t. P_i and $F_i(\cdot)$ is the cumulative distribution function of $D_i(\boldsymbol{X})$, where $\boldsymbol{X} \sim P_i$. As before, any depth function can be used here.

Similar to the depth-weighted cost, the rank weighted cost of misclassification of a point which is close to the center of the distribution is high since a big part of points has lower depth.

To minimize the overall expected cost for the cost function (4.9) the new observation $\boldsymbol{x}$ is assigned according to the following rule:

$$class_R(\boldsymbol{x}) = \underset{i}{\operatorname{argmax}}\, F_i(D_i(\boldsymbol{x}))f_i(\boldsymbol{x})\pi_i. \tag{4.10}$$

The notation is the same as in (4.4) and (4.9).

We call the classifier defined by (4.10) the *rank-weighted classifier* and abbreviate it in tables and figures as *RW*.

The following theorem points out the independence of the rank-weighted classifier on the choice of the depth function in the situation described in the Section 4.3, i.e. when the considered distributions are elliptical of the same radial type and the depth functions are affine invariant.

Theorem 4.4 *Let (P1) and (P2) hold for P_1 and P_2 and any two depth functions $D(\cdot)$ and $D^*(\cdot)$. Then $class_R(\boldsymbol{x})$ is the same for both $D(\cdot)$ and $D^*(\cdot)$ with probability one.*

Proof: From (P2) there is some decreasing function $h(\cdot)$ such that $D_i(\boldsymbol{x}) = h(M_i(\boldsymbol{x}))$ and some decreasing function $h^*(\cdot)$ such that $D_i^*(\boldsymbol{x}) = h^*(M_i(\boldsymbol{x}))$ for $i = 1, 2$. Thus $F_i(D_i(\boldsymbol{x})) = P_i(D_i(\boldsymbol{X}) < D_i(\boldsymbol{x})) = P_i(M_i(\boldsymbol{X}) > M_i(\boldsymbol{x})) = F_i^*(D_i^*(\boldsymbol{x}))$ for $i = 1, 2$. The claim immediately follows. □

Note, that in non-elliptical distributions the level sets of various depth functions are very different and, therefore, the rank orders based on these depths are not equal. The other disadvantage of this method is that it produces outsiders for the depths that do not produce them originally.

4.4.3 Dealing with outsiders

As the other depth-based classifiers, the introduced method suffers from the problem of outsiders. The empirical versions of some notions of data depth like halfspace and zonoid depths

are not defined outside of the convex hull of the data. If a point lies outside the convex hulls of all classes, its depth w.r.t. each of them is zero and a point cannot be classified by the introduced method. The common problem arises when using the rank-weighted classifier. One of the possible solutions is to compare the priors-weighted densities of the outsiders w.r.t. all classes. For the points lying far from the data, however, these may also be numerically equal to zero. Nevertheless, outsider treatment procedures may be used, e.g. classification with other known methods like LDA as in Lange et al. (2014b).

4.5 Simulation study

To explore properties of the newly proposed classifiers we conducted a broad simulation study. The study has two parts with different objectives. While the first part focused on properties of the "population versions" of the new classifiers (in this part prior probabilities, densities and depth function were assumed to be known), the second part dealt with "empirical versions" of the classifiers concentrating mainly on the effect of estimation, which is needed in practice.

4.5.1 Objectives of the simulation study

The first part of the simulation compares "population versions" of the newly proposed classifiers (4.4) and (4.10) with the Bayes classifier (4.1). In practice the density function as well as the depth function are unknown and need to be estimated. Nevertheless, "population versions" of the classifiers which are defined by means of these functions shall be studied first. The density function as well as the depth function are now assumed to be known and thus the role of the simulation might be considered redundant. Use of the simulation has two reasons – absence of explicit formulae for evaluation of the depth function and difficulty of integration in nontrivial cases. From these reasons the simulation is unavoidable here. In the Section 4.3 we have shown that in some particular cases the population versions of the depth-weighted classifier and Bayes classifier coincide. We have also shown in Theorem 4.4 that the depth-weighted classifier and rank-weighted classifier coincide in some situations. Now we are questioning about the cases where the classifiers differ. Particularly we are interested in the following questions:

Q1 How much might the new classifiers differ from the Bayes classifier? How big might the corresponding difference be in the average misclassification rate?

Q2 How much might the depth-weighted classifier differ from the rank-weighted classifier?

Q3 How much depends the performance of the new classifiers on the choice of the depth function? Is the rank-weighted classifier really less dependent on the choice of the depth function? Which depth function leads to the smallest or biggest differences from the Bayes classifier?

Later on we refer to these questions in the results.

The second part of the simulation is devoted to the comparison of "empirical versions" of the newly proposed classifiers and the Bayes classifier. Since in practice prior probabilities, density and depth functions are not known, they need to be estimated from data. Here we want to study the effect of estimation on performance of considered classifiers. We hypothesize that the presence of the depth-term in the classifier may lead to a better performance of the classifier in cases where the estimation of the density is problematic. Recalling the original motivation of the new approaches we hypothesize that the misclassification rates in particular groups will be more equal for depth-based classifiers than for the Bayes classifier. The objective of the second part of the simulation study is to "test" our hypotheses in practice.

4.5.2 Simulation settings

In the simulation study we considered eight different pairs of bivariate distributions. The first four settings are elliptically symmetric distributions, and the other four are non-elliptical distributions. Most of the distributional settings were used in Lange et al. (2014b) and Li et al. (2012), the case of skewed normal distributions was used in Vencalek (2013). Details on skewed normal distributions can be found in Azzalini (2013). The examples considered in the simulation study are summarized in the Table 4.1. The variance matrix $\boldsymbol{\Sigma}_0$ used in Examples 1-4 has the following form: $\boldsymbol{\Sigma}_0 = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 4 \end{smallmatrix}\right)$. Five different prior probabilities were considered: $\pi_1 = \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Note that for location-shift models in Examples 1 and 3 the prior probability $\pi_1 = 0.1$ corresponds directly to the prior probability $\pi_1 = 0.9$ and, similarly, $\pi_1 = 0.3$ corresponds to $\pi_1 = 0.7$.

Table 4.1: Examples used in the simulation study

	Group 1		Group 2	
Ex.	Distribution	Parameters	Distribution	Parameters
1	Normal	$\mathbf{0}, \boldsymbol{\Sigma}_0$	Normal	$\mathbf{1}, \boldsymbol{\Sigma}_0$
2	Normal	$\mathbf{0}, \boldsymbol{\Sigma}_0$	Normal	$\mathbf{1}, 4\boldsymbol{\Sigma}_0$
3	Cauchy	$\mathbf{0}, \boldsymbol{\Sigma}_0$	Cauchy	$\mathbf{1}, \boldsymbol{\Sigma}_0$
4	Cauchy	$\mathbf{0}, \boldsymbol{\Sigma}_0$	Cauchy	$\mathbf{1}, 4\boldsymbol{\Sigma}_0$
5	Bivar. exponential	1, 1	Shifted bivar. expon. ($+\mathbf{1}$)	1, 1
6	Bivar. exponential	1, 1/2	Shifted bivar. expon. ($+\mathbf{1}$)	1/2, 1
7	Normal	$\mathbf{0}, \boldsymbol{I}$	Bivar. exponential	1, 1
8	Skewed normal	$\left(\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 0 \\ 0 & 7 \end{smallmatrix}\right), \left(\begin{smallmatrix} -2 \\ -5 \end{smallmatrix}\right)$	Skewed normal	$\left(\begin{smallmatrix} 0 \\ -1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 0 \\ 0 & 5 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 \\ 5 \end{smallmatrix}\right)$

In the first part of the simulation study, a training set of 5 000 points was considered for each setting to estimate the data depths, and the "true densities" were used. We afford the large number of points since we do not simulate real-life data here. The simulation is used instead of an exact enumeration, and thus the estimates need to be as precise as possible. The training set in the second part of the simulation study is smaller containing "only" 2 000 points. Although it is clear that the quality of estimation depends on the size of the training set, we considered only this sample size. The main reason was to prevent an extensive labyrinth of results. We also realized that the estimation of the density term might be very difficult even for such a large data set. In this chapter we do not discuss the density estimation and bandwidth tuning

techniques, as there is a bunch of literature on this topic; see Silverman (1986), Scott (1992), Li and Racine (2007). We estimated the densities in a standard way, viz. with the `npudens` function from the R-package **np** (Hayfield and Racine, 2008), tuning the kernel bandwidths with likelihood cross-validation (function `npudensbw`). In both parts of the simulation study we used a set of 10 000 points to evaluate the performance of the considered classifiers and their eventual differences in classification.

For a given pair of distributions (named Example further on) and given prior probabilities, 100 data sets (training sets + test sets) were generated. For a given data set the Bayes classifier, depth-weighted classifier and rank-weighted classifier were trained and subsequently tested. The latter two classifiers were fitted for three different notions of depth – halfspace depth, projection depth and the affine invariant version of spatial depth. Average misclassification rates as well as both costs considered in this chapter were recorded. We also recorded the number of points classified differently by different classifiers.

4.5.3 Results

Results are presented separately for both parts of the simulation study. We abbreviate the depth-weighted classifier as DW, the rank-weighted classifier as RW and the Bayes classifier as B in the tables and figures.

Population version of the classifiers

Percentages of points classified differently by the depth-based classifiers and the Bayes classifier are plotted in Figure 4.9. Two bars are plotted for each combination of distributional settings, prior probabilities and depth function. Their heights correspond to the percentage of points, that are classified differently by the depth-weighted classifier and the Bayes classifier (left bar) and percentage of points classified differently by the rank-weighted classifier and the Bayes classifier (right bar). Both bars are divided into two parts; in the lower part both depth-based classifiers contradict the Bayes, while in the upper part they classify differently from each other. Note, that in the upper part the other classifier agrees with the Bayes. From this graph several observations might be made:

- (Ad Q1) In most of the cases the proportion of points classified differently by the depth-based classifiers compared to the Bayes classifier is less than 15%. The biggest difference was found in the case of Cauchy distributions differing in location as well as in scales with unequal priors (Experiment 4, $\pi_1 = 0.7$), which was almost 30%.

- (Ad Q2, Q3) The rank-weighted classifier is less dependent on the choice of the depth function than the depth-weighted classifiers. It can be nicely seen for elliptically symmetric distributions (Ex. 1-4), where three bars indicate the difference of the rank-weighted classifier and the Bayes classifier for a given experiment and prior probabilities, but different depth functions yield virtually equal heights. See for example the graph dealing with Experiment 2, where $\pi_1 = 0.7$.

- (Ad Q2) In some cases the difference between the depth-weighted classifier and the rank-weighted classifier is very small (e.g. Ex. 4), but in some cases the difference is large, usually for a specific depth function (e.g. Ex. 2, $\pi_1 \geq 0.7$ for projection depth or Ex. 10, $\pi_1 \leq 0.5$ for halfspace depth), which is explained by the difference in ranges of depths.

- (Ad Q3) The depth-weighted classifier differs more from the Bayes classifier than the rank-weighted classifier with halfspace depth, while for the projection and spatial depths the opposite is true. For the spatial depth, however, the difference between the depth-based classifiers is smaller.

These results are further supported by Table 4.5. In this table, for each of the 40 classification problems (defined by eight distributional settings and five prior probabilities), we recorded several characteristics. Namely: for each depth function we recorded if the depth-based classifiers differ from the Bayes classifier and which of them differs more (according to the number of points classified differently). For each of the two depth-based classifiers we compared depth functions according to the difference of the classifiers from the Bayes classifier. The mark ✓ is used if the corresponding inequality was observed in at least 95 of the 100 simulation runs. We observed that:

- For halfspace depth both classifiers differ from the Bayes in 38 out of 40 situations. The depth-weighted classifier is more different from Bayes than the rank-weighted classifier with only one exception.

- For projection depth both classifiers differ from the Bayes in 28 out of 40 cases. In 8 out of the 12 situations, where they do not differ from the Bayes, they also do not differ for the spatial depth. The rank-weighted classifier differs more from the Bayes than the depth-weighted classifier in 23 out of 28 cases, where they differ from Bayes.

- For spatial depth both classifiers differ from the Bayes in 30 (32 respectively) out of 40 cases. The rank-weighted classifier differs from the Bayes classifier more than the depth-weighted classifier in 17 out of the 30 cases and in only two cases opposite is true.

The simulation study included also situations for which theoretical results were provided in sections 4.3 and 4.4. Assumptions of Theorem 4.1 are fulfilled in Example 1 for $\pi_1 = 0.5$. According to this theorem there should be no difference between the depth-weighted classifier and the Bayes classifier. The observed difference is really very small. Theorem 4.3 dealing with elliptically symmetric distributions can be applied in Examples 1-4. In Examples 1 and 3 the ratio $k_1/k_2 = 1$. Thus for $\pi_1 < 0.5$ it should hold $class_B(\boldsymbol{x}) = 1 \Rightarrow class_D(\boldsymbol{x}) = 1$ while for $\pi_1 > 0.5$ it should hold $class_B(\boldsymbol{x}) = 2 \Rightarrow class_D(\boldsymbol{x}) = 2$. In Examples 2 and 4 the ratio $k_1/k_2 = 4$. Thus for $\pi_1 < 0.2$ it should hold $class_B(\boldsymbol{x}) = 1 \Rightarrow class_D(\boldsymbol{x}) = 1$ while for $\pi_1 > 0.2$ it should hold $class_B(\boldsymbol{x}) = 2 \Rightarrow class_D(\boldsymbol{x}) = 2$. These implications were confirmed by simulation. The few points for which the implications were not fulfilled lied close to the separating lines. The difference was probably caused by estimation of depth.

The Table 4.5 provides practical evidence for Theorem 4.4, which deals with rank-weighted classifier for elliptically symmetric distributions (Examples 1-4). Indeed in these cases classification does not depend on used depth function – there are no ✓ marks in corresponding places in the table with only two exceptions, where the difference was very small, probably caused by depth estimation.

We also wanted to compare average misclassification rates (AMRs) of the depth-based classifiers and the Bayes classifier. Since the Bayes classifier is constructed to minimize AMR, it is clear that the depth-based classifiers can not lead to a lower AMR. Our aim here is to explore how much larger their AMRs might be. The comparison is summarized in Figure 4.10. The partial graphs of this figure are arranged in the same way as the partial graphs in Figure 4.9. Heights of the plotted bars correspond now to the AMRs. For a given distributional settings and prior probabilities the minimal possible AMR (attained by the Bayes classifier) is plotted as horizontal line dividing each of the six bars in the corresponding partial graph. Thus the difference between the minimal attainable AMR and the AMR of a given depth-based classifier can be directly seen from the graph.

Here are some observations on differences in AMRs:

- The most important observation is that the increase in AMR produced by depth-based classifiers is not dramatic and it is much smaller than the percentage of points in which the classification differ from the Bayes classifier. For example the depth-weighted classifier which uses spatial depth classifies about 10% of points differently from the Bayes classifier in Examples 4 and 8 for equal priors, but the increase in AMR is only 1%. This phenomenon is further illustrated in Figure 4.5.

- Increase in AMR higher than 5% was recorded in only 7 out of 240 settings. In all 7 situations depth-weighted classifier with halfspace depth was used. The highest difference was in Example 2: 8.2% for $\pi_1 = 0.7$, 7.2% for $\pi_1 = 0.9$ and 5.5% for $\pi_1 = 0.5$. The remaining four situations were Examples 6 and 7 for $\pi_1 = 0.3$ and $\pi_1 = 0.5$ where the difference in AMR was between 5.4 and 6.1%.

- The highest AMRs were recorded for depth-weighted classifier using halfspace depth (in 33 out of 40 situations).

- Although the difference in AMR might be quite small, the relative increase might be non-negligible. We evaluated only the situations where the Bayes misclassification rate was at least 5%. The highest relative increase in AMR (ratio of AMR for the depth-based classifier and Bayes classifier) was 2.01 for the depth-weighted classifier using halfspace-depth in Example 2 for $\pi_1 = 0.9$. The ratio was higher than 1.5 in 9 out of 216 cases.

- Some observations on proportion of points classified differently from the Bayes classifier hold also for AMRs. For example, small dependence of AMR on the choice of the depth function for rank-weighted classifier when the distributions are elliptically symmetric.

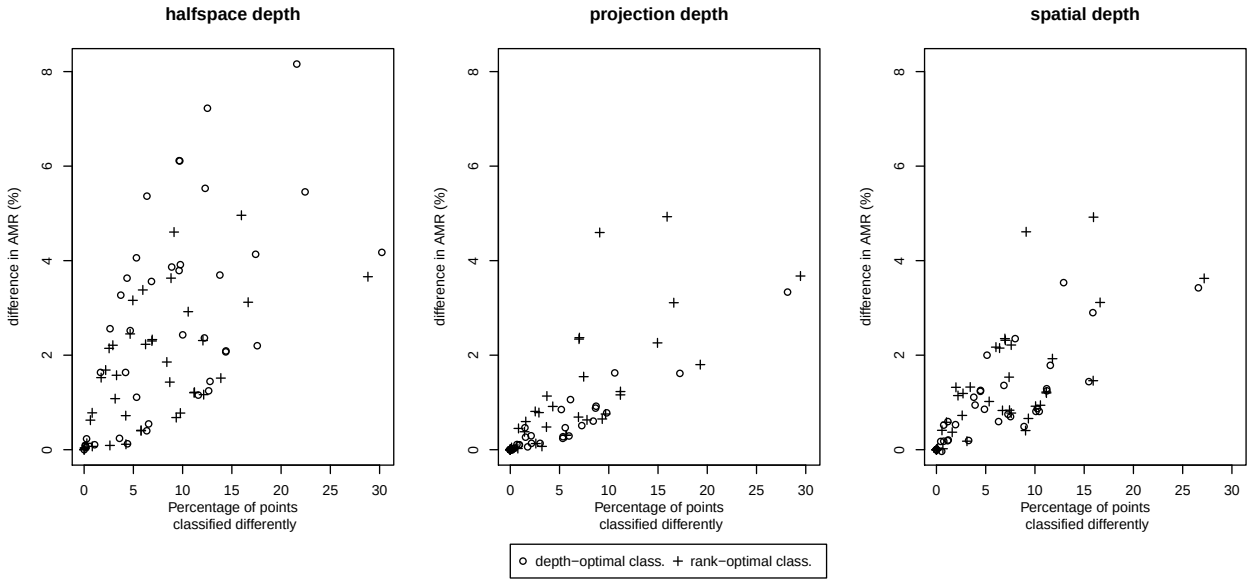


Figure 4.5: Comparison of percentage of points classified differently by depth-based classifiers than by the Bayes classifier and difference in average misclassification rates.

Empirical version of the classifiers

The difference between AMRs for theoretical and empirical versions of all three considered classifiers is visualized in Figure 4.11. The figure is plotted using spatial depth in the depth-based classifiers. We do not present the analogous figures using other depth functions since the main observations are similar. The halfspace depth provides smaller difference from the theoretical results than both projection and spatial depths for 25 out of 40 data sets and smaller than one of them in 33 cases.

The heights of particular bars now correspond to the AMRs of the empirical versions of considered classifiers and the thick lines show AMRs for the corresponding theoretical classifiers. More precisely, we are presenting medians gained from 100 simulation runs. Thus the effect of estimation which is needed in practice can be directly observed.

The main observation on effect of estimation can be summarized as follows:

- For normal distributions (Examples 1, 2) and skewed normal distributions (Ex. 8) the difference between empirical and theoretical results are negligible, while for the fat tailed distributions (Ex. 3, 4) and non-elliptical distributions (Ex. 5-7) the empirical misclassification rate is higher than the theoretical one, as the estimated densities strongly differ from the real ones.

- The most important observation is that the differences between AMRs of depth-based classifiers and the Bayes classifier for empirical classifiers are even smaller than for theoretical versions. It can even happen (as in Example 4 for $\pi_1 = 0.3$ or $\pi_1 = 0.5$) that the AMRs of the empirical depth-based classifiers are lower than the corresponding AMR of the empirical Bayes classifier.

- The effect of estimation is similar for the depth-weighted and the rank-weighted classifiers.
- A remarkable situation occurred in Example 4 for $\pi_1 = 0.7$. The AMRs for empirical depth-weighted classifiers are there smaller than their theoretical equivalents. However, this situation is rather exceptional.

Let us now concentrate on the empirical versions of the classifiers themselves. The results are similar to those of the theoretical classifiers. The observations are documented by number of cases (out of $40 = 8$ times 5 distributional settings) where the given phenomenon is observed in at least 95 out of 100 simulation runs:

- Projection depth commonly leads to the smallest AMRs, while halfspace depth yealds the highest AMRs for both depth-based classifiers. The difference among depth functions is bigger for the depth-based classifier than for the rank-weighted classifier.

  We observed that for the depth-weighted classifier the halfspace depth led to higher AMR than spatial depth in 27 cases (the opposite was true only in one case), and it led to higher AMR than projection depth in 25 cases (the opposite was not observed in any situation). Spatial depth led to higher AMR than projection depth in 14 cases (opposite was true only in 3 cases). Similarly for the rank-weighted classifier the halfspace depth led to higher AMR than spatial depth in 12 cases (opposite was true in 2 cases), and it led to higher AMR than projection depth in 10 cases (the opposite was true in 2 cases). Spatial depth led to higher AMR than projection depth in 5 cases (opposite was true in 3 cases).
- The depth-weighted classifier leads to higher AMR than the rank-weighted classifier when the halfspace depth is used; for the other two depth functions the opposite is true. The numbers of cases where the observed inequalities hold are summarized in Table 4.2. This table also provides a comparison of empirical depth-based classifiers to the empirical Bayes classifier. An interesting fact is that the empirical depth-weighted classifier for the projection depth is comparable to the empirical Bayes classifier according to AMR: it leads to the higher AMR only in 7 cases while in 6 cases opposite is true.

Table 4.2: Difference between the empirical classifiers. Number of situations (out of 40) where the observed inequality in AMR is observed in at least 95 of 100 cases.

depth	DW < B	DW > B	RW < B	RW > B	RW < DW	RW > DW
halfspace	2	29	2	22	27	1
spatial	7	18	4	20	1	16
projection	6	7	5	19	2	18

4.6 Robustness

Procedures based on data depth are usually expected to be robust, here we discuss robustness of the newly proposed depth-weighted classifier. Any statistical procedure is said to be robust if its performance is not strongly influenced by the presence of outliers. The key issue here is how to measure the performance of classification procedures. In the present chapter we stress that there are several possible measures of classifier performance, including average misclassification rate, error rate in smaller group, maximum of error rates in particular groups or total cost i.e. sum of depth-weighted errors as proposed in chapter 2.1.

Although the formula for the classifier that minimizes total probability of misclassification – the Bayes classifier (4.1) – looks simple, in practice the density must be estimated. This can be done either by a procedure which has restrictive assumptions or by some more universal procedure, which is often highly sensitive to the presence of contamination (local clusters of points) in the training set. On the contrary, most of the depth functions are robust as discussed e.g. by Donoho and Gasko (1992). Thus we hypothesize that the inclusion of a depth weight in the classification rule might improve the robustness of the classification procedure.

Let us first summarize several thoughts supporting a positive effect of the depth term in formula (4.4) on the robustness of the classifier. The depth term included in the classifier (4.4) can be viewed as a weight. In this way the outliers are underweighted – being located far from the center of the class, the outliers have much smaller misclassification cost than the central points. Moreover, with a robust depth the outliers influence only the outer depth contours. Robustness of the classification procedure in thus increased.

4.6.1 An illustrative example

We illustrate advantages gained by adding the depth term in the following (rather artificial) example.

Let us consider two triangular symmetric distributions P_1 and P_2 with disjoint supports $\text{supp}(P_1) = (0, 2z)$ and $\text{supp}(P_2) = (2z, 4z)$, where z is some fixed positive constant. In the considered situation perfect separation of the groups is possible. Now let the training sample of the distribution P_1 contain an α-part of contaminated points, where the contamination comes from the symmetric triangular distribution centered in $3z$ (hence having the same center as P_2). The situation is shown in Figure 4.6. The probability mass is divided among these three subgroups in the following way: $(1-\alpha)\pi_1$ for P_1, $\alpha\pi_1$ for the contamination of P_1 and π_2 for the noncontaminated P_2.

In an extreme case we can consider the contamination with the triangular symmetric distribution on $(2z, 4z)$, i.e. $z_0 = z$. If $\pi_2 < \alpha\pi_1$ then for the densities f_1 and f_2 it holds $f_2(x)\pi_2 < f_1(x)\pi_1$ for all $x \in (2z, 4z)$. Thus all new observations (either from P_1 or P_2) will be assigned to P_1 by the Bayes classifier. Misclassification rate in the second group is thus 100%.

It can be easily shown that the depth-weighted classifier assigns points that are smaller than $2z + \sqrt{2\pi_2}z$ to P_1 and all other points to P_2. The misclassification rate for the second

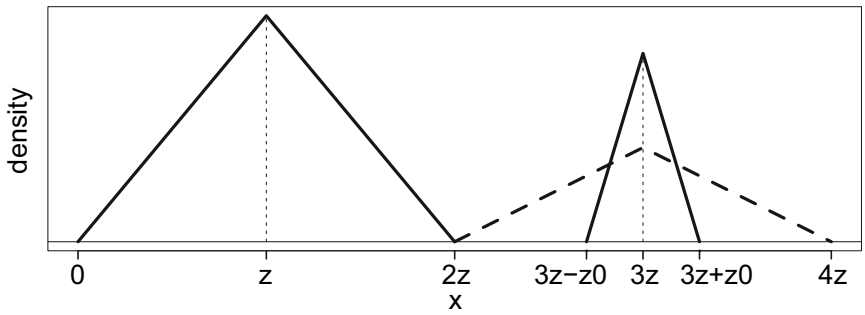


Figure 4.6: Posterior densities of the distributions P_1 and P_2

group is thus $Err_2 = P_2(X < 2z + \sqrt{2\pi_2}z) = \pi_2$. The depth-weighted classifier alleviated substantially the problem which arose from the presence of contamination in the training set – the error rate for the P_2 was decreased from 1 to π_2.

This means that the depth-weighted classifier is able to filter the big contamination inclusions, but at the same time it shifts the separation.

4.6.2 Simulation study on robustness of the depth-based classifiers

For a deeper insight into the problem of robustness of the depth-based classifiers we conducted a short simulation study in which we address a two-class problem (denoting the distributions P_1 and P_2) where the training set of P_1 contains contamination. The main objective of the simulation study is to compare depth-based classifiers with an empirical Bayes classifier from the perspective of their robustness. The depth-based classifiers use spatial depth here. For the density estimation we use one of the broadly applicable kernel procedures, as in the section 2.7.

The test set contained a total number of 2 000 points. The simulation was repeated 100 times. The distributional settings are described as follows:

- $P_1 = N((0,0)', \boldsymbol{I})$, $P_2 = N((3,0)', \boldsymbol{I})$.
- The training set of the group 1 includes 10% of contamination, which is generated from the distribution $N(\boldsymbol{\mu}_c, s\boldsymbol{I})$. The shrinkage coefficient is $s = 0.1$, and $\boldsymbol{\mu}_c$ is a location parameter.
- Four different choices of the location parameter $\boldsymbol{\mu}_c$ define four different simulation settings. The four considered values are $(3,0)'$, $(4,0)'$, $(5,0)'$ and $(9,0)'$. Since the second coordinate is always zero, we characterize the shift by its first coordinate (3, 4, 5, or 9).
- As the fifth setting we consider the situation in which 10% of points generated from P_2 are mislabeled in the training set.
- Three different priors are considered: $\pi_1 = \{0.25, 0.5, 0.75\}$.
- The AMR is calculated on a test set that is generated without contamination.

Although the newly proposed depth-weighted classifiers are designed to optimize the depth-weighted cost functions we measure their performance by AMR in this study since this is widely used. AMRs are reported as medians of the AMRs over 100 repetitions.

Results of the simulation study are presented in Figure 4.7. Different locations of the contamination are distinguished by the columns, while different prior probabilities are distinguished by rows. In each subgraph the AMRs of four different classifiers are mutually compared. The first considered classifier is the Bayes optimal classifier, which is not available in practice. It serves here as a benchmark. The remaining three classifiers are: the empirical Bayes classifier, the depth-weighted classifier DW and the rank-weighted classifier RW. For each classifier its AMR is represented by the height of the corresponding bar. The black lines show the AMR of the classifiers trained without contamination.

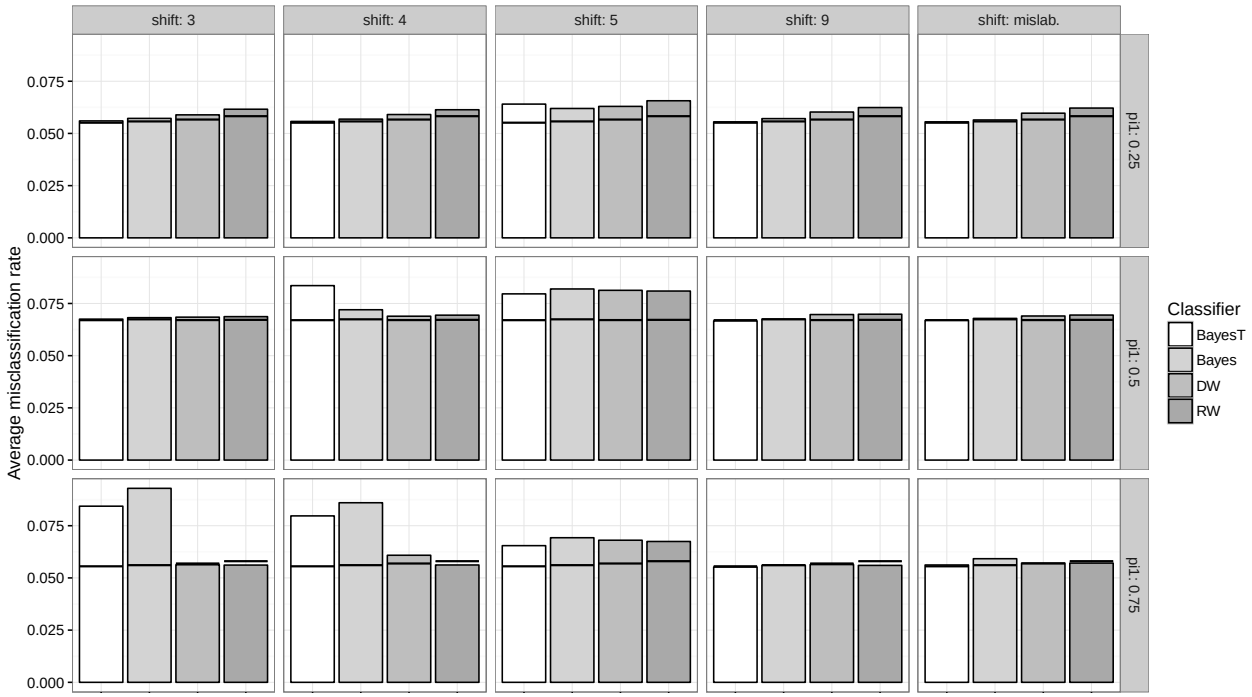
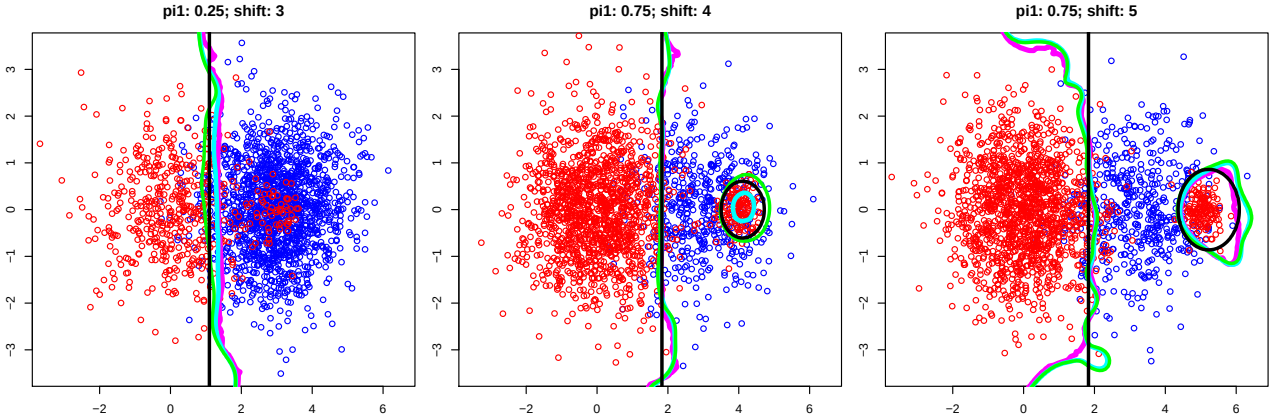


Figure 4.7: Robustness under different types of contamination. The classifiers are trained on the contaminated data and tested on clean ones. The black lines show the AMR of the classifiers trained without contamination.

We distinguish two situations based on Figure 4.7:

1. Cases in which the Bayes classifier is only slightly influenced by the contamination (increase in AMR is less than 3%), see Table 4.3. In this case the density of the first class in the contaminated area remains lower than the density of the second class, or the contamination is far from the data. On the other hand, the outliers cause deformation of the outer depth contours of the first class, and the separation of the depth-weighted classifiers is slightly shifted towards the contamination, see Figure 4.8 (left). Empirically we observe that the depth-based classifiers are slightly more influenced than the empirical Bayes classifier, nevertheless the influence is not strong – it ranges from -3 to +7%.



Separation: — Theoretical Bayes, — Empirical Bayes, — DW, — RW

The DB classifiers are shifted towards the contamination (left)

The Bayes classifier deteriorates, while DW reduces and RW completely filters the contamination (middle)

The contamination is not filtered by any of DB classifiers (right).

Figure 4.8: The three plots illustrate the behaviour of the depth-weighted classifiers.

2. Cases in which the Bayes classifier is influenced by the contamination (increase in AMR is more than 10%), see Table 4.4. In this case the density of the first class in the contaminated area becomes higher than the density of the second class. We divide these cases again into two groups:

   In the first group the depth w.r.t. the second class is high and the depth-based classifiers filter the contamination (see Figure 4.8, middle). For example, when $\pi_1 = 0.75, \text{shift} = 3$; $\pi_1 = 0.75, \text{shift} = 4$; and $\pi_1 = 0.50, \text{shift} = 4$, the contamination is big and lies in the center of P_2. In these cases AMR of the Bayes classifier increases by 52%, 43%, 25% respectively, while for the depth-based classifiers the increase is only about -2%, +3%, +3% respectively. An illustrative explanation of this phenomenon can be found in Figure 4.8 (middle). In the contaminated area the density of P_1 is (falsely) higher than that of P_2 and Bayes classifier fails in this area. On the contrary, the depth-weighted classifier substantially reduces the problematic area and the rank-weighted classifier totally overcomes the problem. The case of mislabeling with $\pi_1 = 0.75$ is also interesting since empirical Bayes deteriorates (AMR increases by more than 5%) while AMRs of rank-weighted classifier even decreases.

   In the second group the contamination lies in the peripheral of the second class and the depth w.r.t. the second class is low and cannot compensate the high difference in densities (see Figure 4.8, right). For example, with shift equal to 5 the AMR of the Bayes and the depth-weighted classifiers increases by about 15-20%.

Generally, the depth-weighted classifiers are shifted towards the outliers as the depth contours are distorted, but at the same time it is capable of filtering big groups of outliers lying inside of the second class, where the Bayes classifier deteriorates. The difference is even greater

if the contaminated class is the bigger one. This means that the depth-based methods are slightly less robust than the Bayes classifier if the outliers are well-spread and do not induce high density around them, but much more robust in case of big groups of outliers that break the Bayes rule.

Table 4.3: Results of the simulation study – changes in AMRs (in %), when the Bayes classifier is only slightly influenced.

shift	π_1	BayesT	Bayes	DW	RW
3	0.25	1.50	2.60	4.00	5.80
3	0.50	0.70	1.20	2.10	2.30
4	0.25	1.10	2.00	4.20	5.30
9	0.25	0.70	2.40	6.40	7.10
9	0.50	-0.50	0.20	4.00	4.00
9	0.75	-0.60	-0.10	-0.90	-3.40
mislab.	0.25	0.70	1.30	5.40	6.70
mislab.	0.50	-0.10	0.70	2.90	3.40

Table 4.4: Results of the simulation study – changes in AMRs (in %), when the Bayes classifier is strongly influenced.

shift	π_1	BayesT	Bayes	DW	RW
3	0.75	51.80	65.60	-1.10	-3.20
4	0.50	24.70	6.80	2.80	3.40
4	0.75	43.50	53.30	6.90	-3.10
5	0.25	16.10	11.10	11.10	12.70
5	0.50	18.80	21.60	21.30	20.60
5	0.75	17.80	23.40	19.60	16.30
mislab.	0.75	1.20	5.50	0.40	-1.60

4.7 Conclusion

In this chapter we introduced two alternative measures of classifiers' performance based on data depth and focused on the centers of the classes. The first one incorporates a depth term into the cost function of the Bayes classifier, while the second one incorporates the rank of the depth. We proposed classifiers that minimize the introduced misclassification cost functions, namely the depth-weighted and the rank-weighted classifiers. Both classifiers include weights of misclassification that depend on the location of the points. They force correct classification of the central points and underweight the outliers. This also decreases the misclassification rate in the smaller class, unless it is situated close to the center of the bigger one. The depth-weighted classifier depends on the selected depth function, while the rank-weighted one does not depend on the depth function for elliptically symmetric distributions.

The methods were theoretically and empirically compared with the Bayes classifier. For elliptically symmetric distributions we derived the regions where they mutually differ as well as conditions for their correspondence.

By construction, the average misclassification rate of the depth-based classifiers is higher than the one of the Bayes classifier. Nevertheless, the difference is rather low and is much lower than the number of differently classified points, as the depth-based classifiers change the misclassification rates in both classes.

The usage of a depth function also increases the robustness of the depth-based methods against big inclusions of contaminated data, that destroy the classification rule of the Bayes classifier. At the same time smaller contaminations distort the outer depth contours and slightly shift the separation.

Appendix

Table 4.5: Difference in classification rules of the classifiers.

The significant difference according to the number of differently classified points is marked with ✓
DW, RW: this classifier significantly differs from the Bayes classifier
DW > RW, DW < RW: one classifier differs from the Bayes classifier more than the other
h, p, s: for the halfspace, projection and spatial depths
H > P, ... : the classifier with one depth differs from the Bayes classifier more than with the other
1, 2: for DW and RW

		DW			RW			DW > RW			DW < RW			H > P		H > S		S > P		H < P		H < S		S < P	
exp	π_1	h	p	s	h	p	s	h	p	s	h	p	s	1	2	1	2	1	2	1	2	1	2	1	2
1	0.1	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓		✓		✓							
1	0.3	✓	✓	✓	✓	✓	✓	✓				✓		✓		✓		✓							
1	0.5	✓	✓	✓	✓	✓	✓	✓				✓		✓		✓									
1	0.7	✓	✓	✓	✓	✓	✓	✓				✓		✓		✓		✓							
1	0.9	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓		✓		✓							
2	0.1	✓						✓						✓		✓									
2	0.3	✓	✓	✓	✓	✓	✓	✓		✓		✓		✓		✓		✓							
2	0.5	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓		✓		✓							
2	0.7	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓		✓		✓							
2	0.9	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓		✓		✓							
3	0.1	✓			✓			✓						✓		✓									
3	0.3	✓	✓	✓	✓	✓	✓	✓																	
3	0.5	✓	✓	✓	✓	✓	✓					✓		✓				✓	✓			✓	✓		
3	0.7	✓	✓	✓	✓	✓	✓	✓																	
3	0.9				✓						✓														
4	0.1	✓						✓						✓		✓									
4	0.3	✓	✓	✓	✓	✓	✓	✓	✓					✓											
4	0.5	✓	✓	✓	✓	✓	✓	✓						✓											
4	0.7	✓	✓	✓	✓	✓	✓	✓				✓				✓									
4	0.9				✓		✓												✓						
5	0.1	✓			✓			✓						✓	✓	✓	✓								
5	0.3	✓			✓			✓						✓	✓	✓	✓								
5	0.5	✓			✓		✓	✓					✓	✓	✓	✓	✓		✓						
5	0.7	✓		✓	✓		✓	✓					✓	✓	✓	✓	✓	✓	✓						
5	0.9	✓	✓	✓	✓	✓	✓	✓				✓	✓							✓	✓	✓	✓	✓	✓
6	0.1	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓	✓							
6	0.3	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓	✓	✓						
6	0.5	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓	✓	✓						
6	0.7	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓	✓	✓						
6	0.9	✓	✓	✓	✓	✓	✓	✓				✓	✓			✓					✓		✓		✓
7	0.1	✓			✓			✓						✓	✓	✓	✓								
7	0.3	✓			✓			✓						✓	✓	✓	✓								
7	0.5	✓		✓	✓		✓	✓					✓	✓	✓	✓	✓	✓	✓						
7	0.7	✓	✓	✓	✓	✓	✓	✓		✓				✓	✓	✓	✓	✓	✓						
7	0.9	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓		✓	✓						
8	0.1	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓	✓	✓						
8	0.3	✓	✓	✓	✓	✓	✓	✓				✓		✓		✓		✓			✓				✓
8	0.5	✓	✓	✓	✓	✓	✓	✓				✓		✓	✓	✓	✓	✓	✓						
8	0.7	✓	✓	✓	✓	✓	✓	✓				✓		✓	✓	✓	✓	✓							
8	0.9	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓		✓	✓	✓							
∑	40	38	28	30	38	28	32	37	1	2	1	23	17	33	16	32	16	22	12	1	3	2	3	1	3

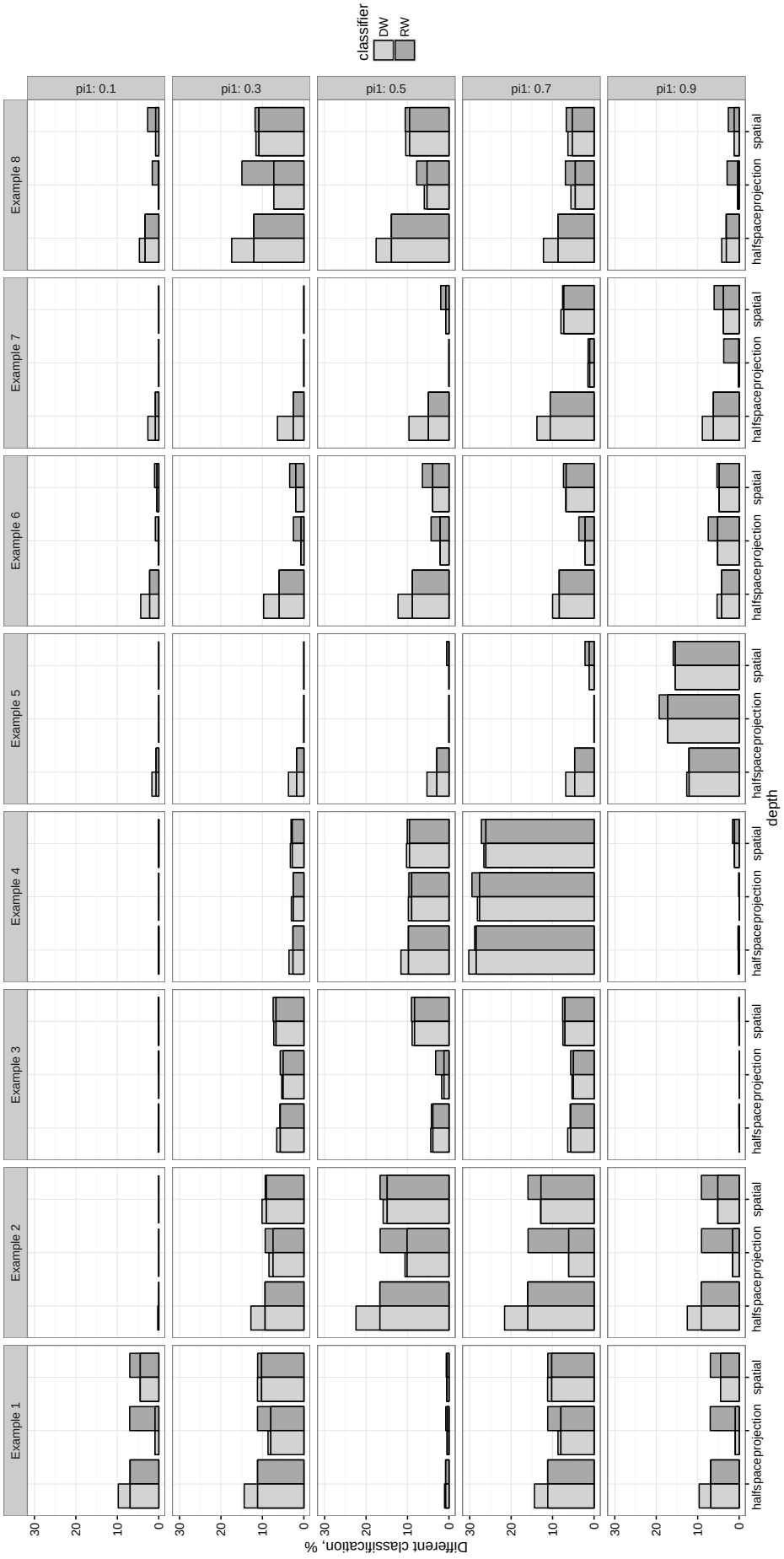


Figure 4.9: Difference in classification rules of the classifiers.

Figure 4.10: Average misclassification rate. The black line shows AMR of the Bayes classifier.

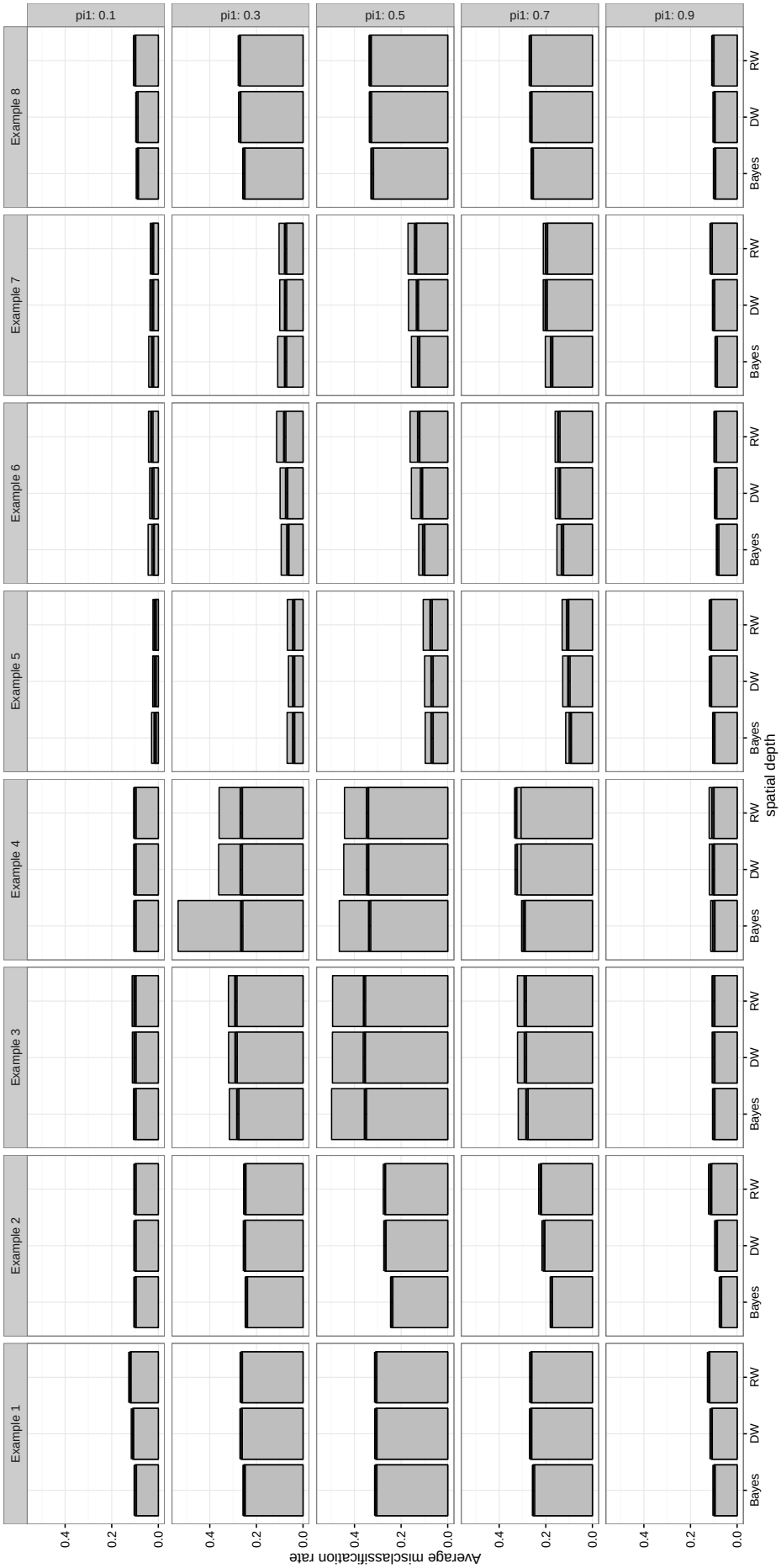


Figure 4.11: Average misclassification rate of theoretical (fat line) and empirical (thin line) classifiers for spatial depth.

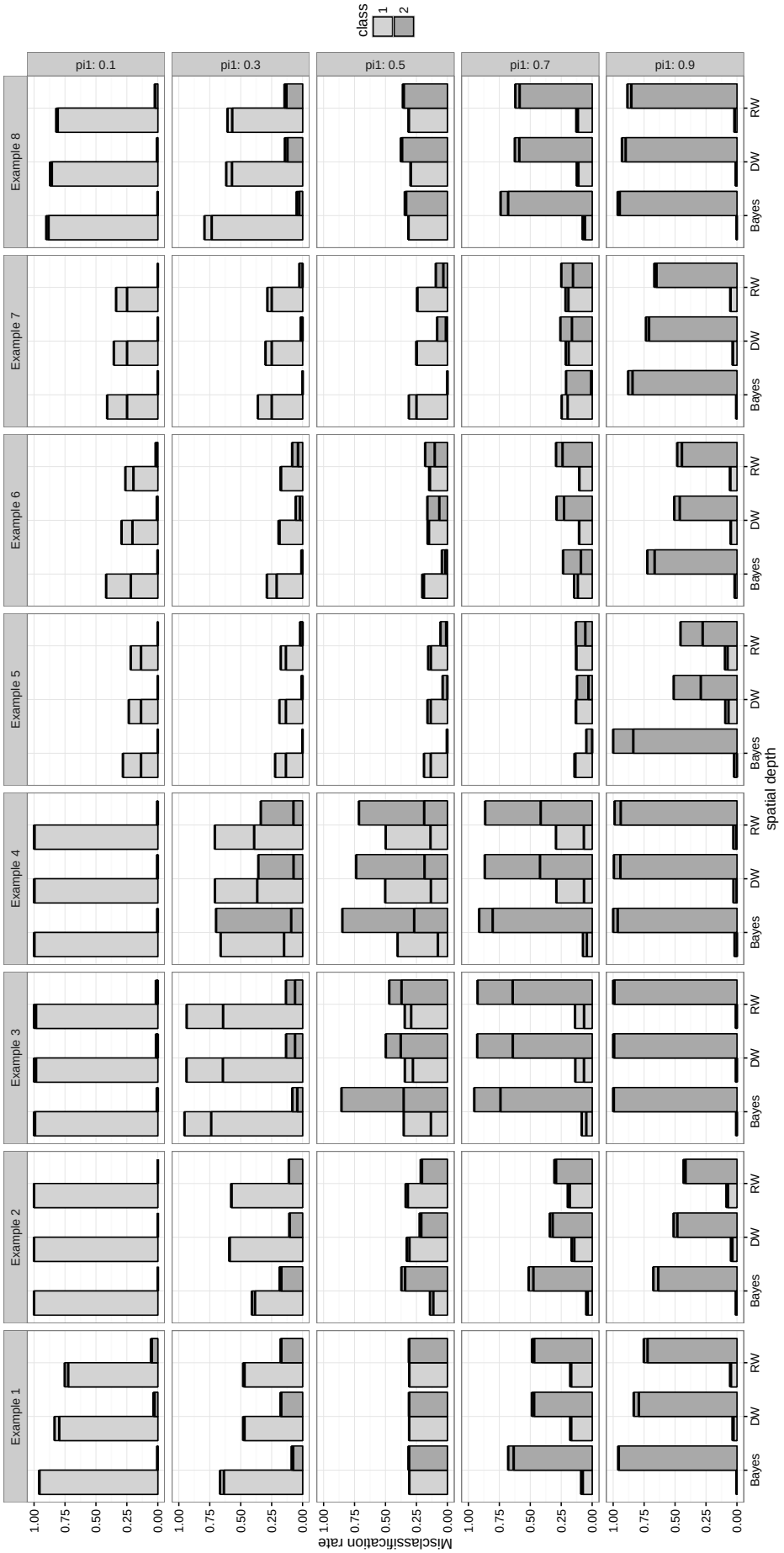


Figure 4.12: Misclassification rates by classes of theoretical (fat line) and empirical (thin line) classifiers for spatial depth.

Chapter 5

Computation of the Oja median by bounded search

5.1 Introduction

A basic task in multivariate analysis is to describe the general location of data by some point in their middle. Several notions of multivariate medians have been proposed in the literature. They extend different properties and characterizations of the usual univariate median to Euclidean k-space. Besides these defining characterizations the multivariate medians may be distinguished by their invariance properties. These include invariances against monotone transformations of the marginals (like the componentwise median), against spherical transformations (like the spatial median), against affine transformations (like the Oja median, proposed in the seminal paper (Oja, 1983)), and combinatorial invariance. The latter means that the data may be varied in their compartments without changing the median. Examples are the Tukey median (Tukey, 1975) and the simplicial median by Liu (1988). These medians are, at least in some sense, more robust against outlying data than the arithmetic mean, which is the center of gravity. Multivariate medians are surveyed by Small (1997) and Oja (2013).

Like the univariate median most of the multivariate medians can be regarded as maximizers of goal functions, so called data depths, the Tukey depth, the simplicial depth, the Oja depth, and the spatial depth, among others. See Mosler (2013) for a recent survey.

To be applicable to realistic problems, a median must be computable for dimensions $k > 2$ and at least medium sized data sets. Here we develop an algorithm to calculate the exact value of the Oja median and demonstrate that it is faster, having also less complexity, than the existing ones by Niinimaa et al. (1992) and Ronkainen et al. (2003), ROO hereafter. The exact algorithm can also serve as a benchmark for faster heuristic procedures. In principle, the computation of the Oja median involves repeated checking of all intersections of hyperplanes generated by the data. Our main idea is to introduce bounding hyperplanes that iteratively restrict the area where the median is searched.

The chapter is structured as follows: Section 2 introduces the Oja median and depth and some basic notions and properties connected with them, it also sketches the algorithm of

Ronkainen et al. (2003) for exact calculation of the Oja median. In Section 3 the ideas of the new bounding procedure are discussed, followed by a description of the algorithm in Section 4. Finally, in Section 5 numerical experience is reported regarding data in $\mathbb{R}^k$ for k up to dimension seven.

5.2 Oja median and depth

Let $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ be a data set of observations in $\mathbb{R}^k$. Each k observations $\boldsymbol{x}_{i_1}, ..., \boldsymbol{x}_{i_k}$ generate an *observation hyperplane* passing through them, which is notated by $p = (i_1, ..., i_k)$, $1 \leq i_1 < ... < i_k \leq n$. Let P denote the set of all $\binom{n}{k}$ observation hyperplanes.

k observations together with a given point $\boldsymbol{x} \in \mathbb{R}^k$ span a simplex in k-space. Its k-dimensional volume is found as

$$\begin{aligned} V_p(\boldsymbol{x}) := V(\boldsymbol{x}_{i_1}, ..., \boldsymbol{x}_{i_k}, \boldsymbol{x}) &= \frac{1}{k!} abs \left(\begin{vmatrix} 1 & ... & 1 & 1 \\ \boldsymbol{x}_{i_1} & ... & \boldsymbol{x}_{i_k} & \boldsymbol{x} \end{vmatrix} \right) \\ &= \frac{1}{k!} abs(d_{0p} + \mathbf{d}_p{}^\top \boldsymbol{x}). \end{aligned}$$

Here d_{0p} is the distance of the hyperplane p from the origin, and $\boldsymbol{d}_p$ is its normal, given by the vector of cofactors of $\boldsymbol{x}$ in the determinant. The average of all such volumes is mentioned as the *Oja outlyingness function* of $\boldsymbol{x}$,

$$\begin{aligned} O(\boldsymbol{x}|\boldsymbol{X}) &= ave_{i_1<...<i_k} \left(V(\boldsymbol{x}_{i_1}, ..., \boldsymbol{x}_{i_k}, \boldsymbol{x}) \right) \\ &= ave_{i_1<...<i_k} \left(\frac{1}{k!} abs \left(\begin{vmatrix} 1 & ... & 1 & 1 \\ \boldsymbol{x}_{i_1} & ... & \boldsymbol{x}_{i_k} & \boldsymbol{x} \end{vmatrix} \right) \right) \\ &= \frac{1}{k!} ave_{p \in P} \left(abs(d_{0p} + \boldsymbol{d}_p^\top \boldsymbol{x}) \right). \end{aligned} \tag{5.1}$$

It is clear from (5.1) that the Oja outlyingness function is piecewise linear and convex on $\boldsymbol{x}$ as well as continuous on $\boldsymbol{x}$ and the data in $\boldsymbol{X}$. The minimizer of the outlyingness function is the *Oja median*, $\mathbf{Med}(\boldsymbol{X})$. Generally, this median is not unique but forms a convex set. The Oja median is a measure of location and *affine equivariant* regarding $\boldsymbol{X}$,

$$\mathbf{Med}(\boldsymbol{Y}) = \boldsymbol{A}\mathbf{Med}(\boldsymbol{X}) + \boldsymbol{b}, \tag{5.2}$$

if $\boldsymbol{Y} = \{\boldsymbol{A}\boldsymbol{x}_1 + \boldsymbol{b}, \ldots, \boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{b}\}$ with some matrix $\boldsymbol{A}$ of full rank k and $\boldsymbol{b} \in \mathbb{R}^k$; see Oja (1983). The outlyingness function can be made affine invariant (to simultaneous transformation of $\boldsymbol{x}$ and $\boldsymbol{X}$) by multiplying it with a proper scale factor, *viz.* $(\det \boldsymbol{S}(\boldsymbol{X}))^{-1/2}$, where $\boldsymbol{S}(\boldsymbol{X})$ is a positive definite $k \times k$ matrix depending on $\boldsymbol{X}$ and measuring the dispersion of the data cloud $\boldsymbol{X}$ in an affine equivariant way, that is, with $\boldsymbol{Y}$ as above, satisfying

$$\boldsymbol{S}(\boldsymbol{Y}) = \boldsymbol{A}^\top \boldsymbol{S}(\boldsymbol{X}) \boldsymbol{A}. \tag{5.3}$$

In particular, the usual covariance matrix of $\boldsymbol{X}$ can serve as $\boldsymbol{S}(\boldsymbol{X})$. The *Oja depth function* is defined as (Zuo and Serfling, 2000)

$$depth(\boldsymbol{x}|\boldsymbol{X}) = \frac{1}{1 + O(\boldsymbol{x}|\boldsymbol{X})(\det \boldsymbol{S}(\boldsymbol{X}))^{-1/2}} \,. \tag{5.4}$$

Observe that the Oja depth function is affine invariant and continuous. It is maximal at the Oja median of $\boldsymbol{X}$ and vanishes for $||\boldsymbol{x}|| \to \infty$. Given $\boldsymbol{X}$, the depth function is a strictly decreasing transformation of the outlyingness function and, thus, the contour lines of the two functions coincide, though at different values. As the function $O(\cdot|\boldsymbol{X})$ is convex, all its contour lines are convex. Hence the level sets of the Oja depth are convex and compact sets in $\mathbb{R}^k$. Moreover, the Oja depth decreases monotonically on rays from each point in the median set.

In the case of a centrally symmetric distribution the median set includes the center of symmetry. It can be shown that the Oja depth function determines the data cloud $\boldsymbol{X}$ uniquely (Koshevoy, 2003). The usual breakdown point of the Oja depth is zero, while a slightly different notion of breakdown appears to be positive (Niinimaa et al., 1990).

Given $\boldsymbol{X}$, the *centered rank function* R is defined by

$$\boldsymbol{R}(\boldsymbol{x}) \;=\; \frac{1}{k!} ave_{p \in P} \left(S_p(\boldsymbol{x}) \boldsymbol{d}_p \right) ,$$

where

$$S_p(\boldsymbol{x}) \;=\; sign(d_{0p} + \boldsymbol{d}_p{}^\top \boldsymbol{x}),$$

indicates on which side of the hyperplane p the point $\boldsymbol{x}$ is located. Note that $\boldsymbol{R}(\boldsymbol{x})$ is the derivative of (5.1), at all $\boldsymbol{x}$ at which $O(\cdot|\boldsymbol{X})$ is smooth. Hence, as $O(\cdot|\boldsymbol{X})$ is convex, the centered rank function is a subgradient of the outlyingness function, at all $\boldsymbol{x} \in \mathbb{R}^k$. Below, $-\boldsymbol{R}(\boldsymbol{x})$ will be used as a direction of descent at point $\boldsymbol{x}$. It is easily seen from (5.1) that the outlyingness function is also represented as

$$\begin{aligned} O(\boldsymbol{x}) \;&=\; \frac{1}{k!} \left(ave_{p \in P} (S_p(\boldsymbol{x}) d_{0p}) + ave_{p \in P} (S_p(\boldsymbol{x}) \boldsymbol{d}_p{}^\top \boldsymbol{x}) \right) \\ &=\; \frac{1}{k!} \frac{1}{\binom{n}{k}} \left(D_0(\boldsymbol{x}) + \boldsymbol{D}(\boldsymbol{x})^\top \boldsymbol{x}) \right) , \end{aligned} \tag{5.5}$$

where the sums,

$$D_0(\boldsymbol{x}) = \sum_{p \in P} S_p(\boldsymbol{x}) d_{0p} \,, \quad \boldsymbol{D}(\boldsymbol{x}) = \sum_{p \in P} S_p(\boldsymbol{x}) \boldsymbol{d}_p \,, \tag{5.6}$$

are piecewise constant. They change by $2d_{0p}$ and $2\boldsymbol{d}_p$, respectively, when a hyperplane p is crossed.

5.2.1 Calculating the median according to ROO

In what follows we assume that the data are in general position. Hettmansperger et al. (1999) have shown that a version of the Oja median is always found among the intersection points of observation hyperplanes. The exact algorithm of ROO iteratively optimizes the outlyingness function along the intersection lines of $k-1$ observation hyperplanes, called *observation lines*. At first a searching line is randomly selected among the observation lines and the outlyingness function is optimized along the line. When the point of the minimum is found, the next searching line through this point is chosen. The possible choices of lines depend on the type of the point: the smallest number of lines is obtained if the point is an intersection of hyperplanes that have no common observation points, the largest number is obtained if the point coincides with one of the observation points; see also the discussion before subsection 5.4.1.

Minimizing the outlyingness function along the searching line is the most time consuming task. The chosen line L is intersected with all hyperplanes and the outlyingness function (5.1) is calculated at each intersection point. At the first intersection point the constant terms d_{0p} and $\boldsymbol{d}_p$ are summed up along with the signs $S_p(\boldsymbol{x}_m)$, yielding the sums D_0, and $\boldsymbol{D}$ according to (5.6). Then the other intersections are considered step by step. The outlyingness function is calculated as in (5.5). In each new point one of the hyperplanes changes its sign and the sums D_p and $\mathbf{D}$ are updated. Note, that there are $\binom{n}{k}$ intersections, almost all of which have to be considered, which causes the great complexity of the algorithm.

This algorithm finds just one of the vertices of the median set. While searching for the median, the algorithm may pass through several vertices of the median set, although it is not guaranteed that it visits all of them. The reason is that, while minimizing the outlyingness function along the searching line, only the first of possibly two points having highest Oja depth is taken as $\mathbf{x}_c^*$. However, in case of a non-unique median, there exist two such points lying on an edge of the median set. To deliver all vertices of the median set, the algorithm can be modified as follows: it has to store both points as vertices and, in addition, check all lines passing through them.

5.3 A bounding approach

The centered rank function is a subgradient of the outlyingness function. Note that no unique gradient exists at intersections of the observation hyperplanes, hence the centered rank function will in general not vanish at the Oja median. The negative rank function (= negative subgradient) $-\boldsymbol{R}(\boldsymbol{x})$ is a vector that points in a direction of descent of the outlyingness function, hence ascent of the depth function. It defines a hyperplane through $\boldsymbol{x}$, which separates the space into two halfspaces. The positive side of the hyperplane is indicated by the negative subgradient, which equals the negative rank function. Therefore, the Oja median is always found on the positive side of these hyperplanes.

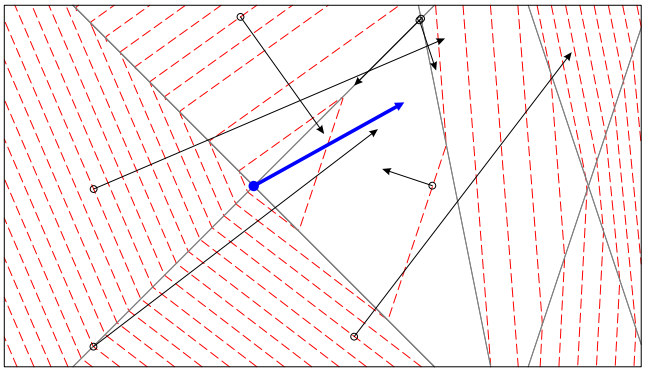


Figure 5.1: An example of Oja depth contours with values of the negative rank function. The median (unique) is shown at the intersection of the observation lines as a bold point, together with its subgradient.

Regarding the Oja depth function, observe that its subgradients have the same direction as the negative subgradients of the Oja outlyingness function,

$$\mathbf{grad}\ depth(\boldsymbol{x}) = -\boldsymbol{R}(\boldsymbol{x})\,(\det \boldsymbol{S}(\boldsymbol{X}))^{-1/2}\,(depth(\boldsymbol{x}))^2\,.$$

Their contour lines coincide since the depth function is a strictly decreasing transform of the outlyingness function.

An example of Oja depth contours and subgradients of the depth function is shown in Figure 5.1. As expected, all negative subgradients point to the halfspace containing the median, and the gradients are perpendicular to the depth contours.

The halfspaces defined by the negative rank function can be used to build a bounded region that contains the median. In our algorithm we select those halfspaces in an iterative way and restrict the further search to their intersection. The hyperplanes bordering such a search region will be called *bounding hyperplanes* or simply *bounds*. The bounded regions reduce the complexity of the searching procedure by reducing the number of hyperplanes that cross the searching lines as well as the number of their intersections actually considered in the minimization procedure.

The obtained hyperplanes form a bounded region, which is the intersection of the positive sides of the hyperplanes. Actually, such a bounded region is determined by part of these hyperplanes only, as bounds lying outside the region provide no additional information. In our algorithm, we adjust the bounded regions step by step. We begin with a rectangular region limited by hyperplanes that are perpendicular to the coordinate axes and go through the maximal and minimal coordinates of the data points on these axes. Then we add hyperplanes as new bounds. For each added new bound it is checked whether it is *efficient*, that is, actually crosses the bounded region, and thus reduces it. Then the intersection of the new hyperplane with the bounded region is determined, and all bounds that are made inefficient by the new one are removed. To check whether a hyperplane crosses the bounded region, it suffices to check if there exist any two bounds' intersections lying on different sides of the hyperplane. As

the calculation of the Oja rank function is itself a rather expensive operation, we will try to obtain the smallest possible central region by performing as few calculation as possible.

We have developed several approaches of the iterative bounds search. The divisive approach (A) is the simplest solution.

Approach A:

The bounded region is iteratively reduced by a divisive approach (A) *viz.* by iteratively adding hyperplanes that go through a properly chosen central point of the region and have their normal vectors equal to the corresponding negative rank function. The central point should be selected to cut a large amount of volume from the bounded region, and shall ideally be the center of the volume, so that any hyperplane through this point will approximately cut off half of the bounded region's volume. Here, we select the mean value of the bounds' intersection points as a central point. As the region is reduced by a hyperplane through the central point, it is expected that its volume shall become (on an average) twice smaller at each step. Ideally, after nine such steps, in any dimension k, a subspace volume of approximately 0.1% of the initial one should be obtained. The experiments in section 5.5 show that the volumes decrease slower in concrete calculations.

The divisive approach (A) considers only the directions of the subgradients, although their lengths also give the information about the location of the median. Another solution (approach B) consists in moving along the subgradients as it is shown in Figure 5.2.a. The length of $\boldsymbol{R}(\boldsymbol{x})$ decreases as $\boldsymbol{x}$ moves towards the median.

Approach B:

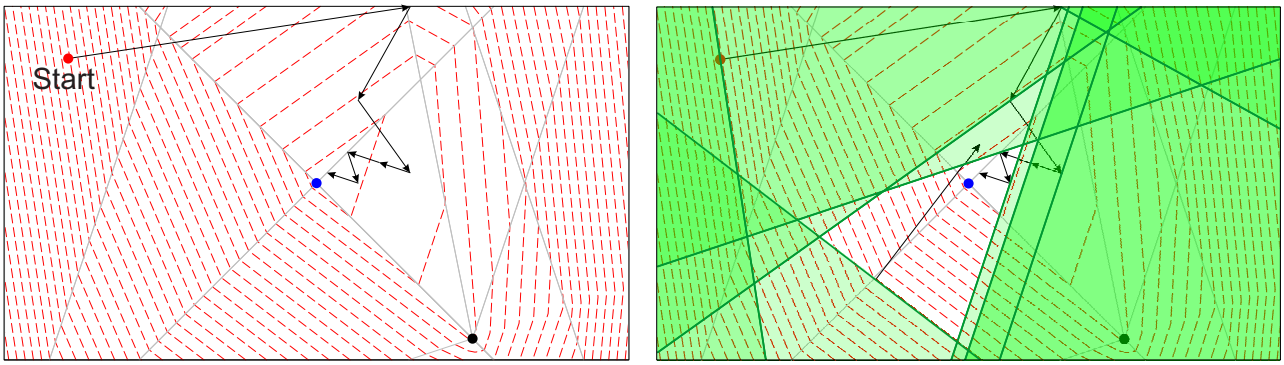
1) Start with $i = 0$. Select an initial point $\boldsymbol{x}_0$. Specifically, we choose the componentwise median of all observations.

2) Determine the subgradient $-\boldsymbol{R}(\boldsymbol{x}_i)$.

3) Add the subgradient vector, $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \boldsymbol{R}(\boldsymbol{x}_i)$, and continue.

We continue building such gradients, each time getting closer to the median, until they become either zero or increase in length. The zero case means that the point $\boldsymbol{x}_i$ lies in the median set, where the Oja depth assumes its minimal value. As it is seen in Figure 5.1, the subgradient's length depends not only on the distance from the median, but also on the subspace, formed by hyperplanes, that contains $\boldsymbol{x}_i$. Thus if the gradients become longer, their lengths may be restricted to the length of the shortest one, and this bound will consequently decrease.

Several of the gradients found may be used to build the bounded searching region, containing the median. The points having shortest gradients are closest to the median. An example of a bounded region built on such gradients is shown in Figure 5.2.b.



a b

Figure 5.2: A gradient path (a) and a bounded region (b), built using the gradients. The subspaces, cut off by each of the bounds are shaded.

The divisive approach needs an almost constant number of calculations to reach the intended volume. However, the efficiency of moving along the subgradients (approach A) strongly depends on the form of the data. In most cases, the subsequent gradients extend in rather different directions, and the volume of the bounded region decreases fast. But in certain cases, especially with asymmetric datasets, this is not true. The subgradients in the sequence may approach the median in a more common direction and thus leave too much space inside the bounded region. The gradients may also end outside the bounded region or jump between two subsets formed by the observation hyperplanes, providing not much information on each step.

It is therefore reasonable to start with moving along the subgradients, and then, as soon as this procedure slows down, shift to the divisive procedure, until the needed volume is reached:

Approach C:

This yields the following hybrid approach, where the next cutting point may be defined as the end of the subgradient, $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \boldsymbol{R}(\boldsymbol{x}_i)$, as long as it lies inside of the bounded region, or as the center of the bounded region otherwise.

Approach D:

Also the direction of the subgradients can be used to define the next cutting point as a central point of the segment between the subgradient's origin and its intersection with the bound.

Further, the calculation can be accelerated by using rougher bounds, *viz.* enlarging the given bounded region by a circumscribed k-variate box. Then *a fortiori* a point lies outside the bounded region if it lies outside the circumscribed box.

Once a bounded region is defined, the observation hyperplanes lying outside of it are excluded from the searching process, which decreases the number of intersections when minimizing on a line. A problem may occur if the bounded region contains no path through the intersections of the observation lines from the initial searching line to the line containing the median. Such a path connecting any two observation lines may be provided by including the

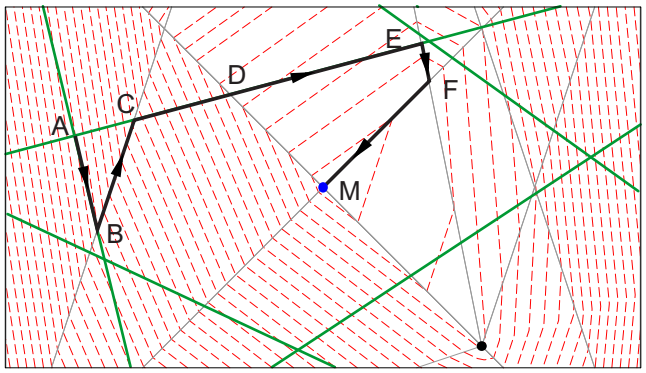


Figure 5.3: A path through the observation lines (thin) and the bounds (bold). We start from taking one of the bounds AB as the initial line, and find a minimum point B. Then the outlyingness function is minimized along the next line through this point. As it is seen from the line CE, the point of minimum E is not necessarily the closest one (D) to the median, and the selected path may be not the shortest one. The paths BC and EFM are isolated, as there are no observation lines inside the bounded region to connect them, but they are connected with the bound CE.

bounds themselves into the searching process as ordinary observation hyperplanes. Figure 5.3 shows an example of a path from the initial line through the observation lines and bounds to the median.

The bounding method may also be used to find the median in an approximative way with some given precision. The space may be cut until the bounded region has the proper size and its center may be taken as an approximation of the median. It is clear, that the median cannot lie outside the bounded region, so its center can be assumed to be the median with precision equal to half of the region's size. As the method considers all observation hyperplanes, it cannot be more efficient than existing approximative methods that consider subsamples of the data.

5.4 The algorithm

To start with, the first bounded region is created as described in the previous section. The desired size of the bounded region is selected as a part of the original volume. Here the volume is calculated as the volume of a minimal multivariate circumscribed rectangle with edges parallel to the coordinate axes. In subsequent iterations the first bounded region is reduced until the desired volume is reached. Here, the divisive approach (A) is considered, as it shows the best results in experiments (see section 5.5). Note that the bounds may cut off some of the vertices of the median set. Moreover, if the central point of the bounded region lies in the median set, its negative rank function is zero, and this point is directly returned as a median.

Next the initial line is determined. In a two-dimensional space any of the observation lines crossing the bounded region may be selected. In higher dimensions the search of the initial line is more complicated. All intersections of $(k-1)$ hyperplanes are inspected until a first intersection line that crosses the bounded region is found. For this, we start with the lines that border the initial bounded region, which makes the search for a fitting line much easier.

It is clear that all points inside the bounded region lie on the same side of any hyperplane which does not cross this region. Therefore, the respective parts of the sums in (5.6) can be calculated beforehand, which significantly decreases the number of calculations on each step. Thus, on every searching line we may restrict ourselves to iterating the remaining hyperplanes.

The bounded region reduces the procedure of minimization along a line to its part lying inside the region. The searching line is usually intersected by most of the bounds. Therefore the two bounds that cut the bounding region at the intersection line are of primary interest. In order to find these bounds, all bounds are sorted according to their intersections with the searching line. Then the intersection point of the first bound with the searching line is taken as a reference point. The first bound which has the reference point on its positive side is selected as well as the previous one. If the searching line goes through the bounded region, all other bounds must have the reference point on the positive side. This property is used to determine whether a searching line hits the bounded region in dimensions higher than two, as there exist hyperplanes that are crossing the bounded region, but whose intersection line lies outside of it, as it is shown in Figure 5.4 for a two-dimensional example and in Figure 5.5 for a higher dimensional one.

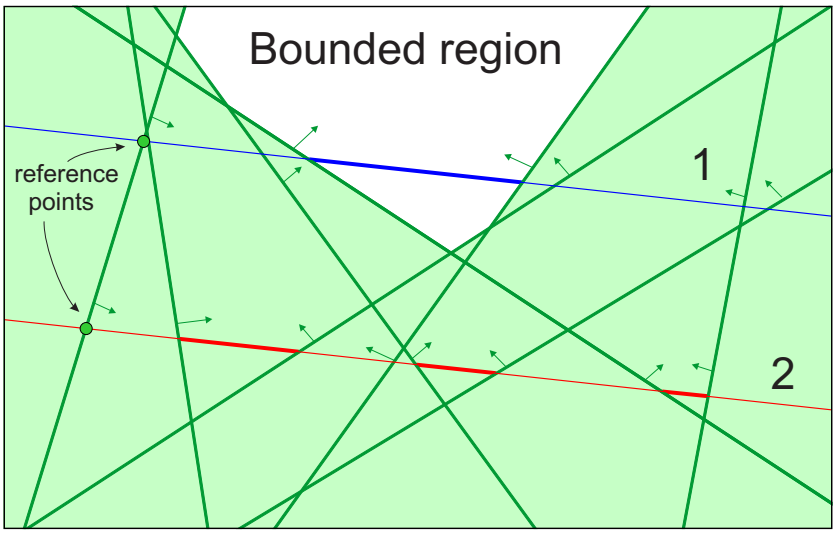


Figure 5.4: Two lines: crossing the bounded region (1) and lying outside of it (2). The arrows show the positive sides of the hyperplanes. The segments between the bounds, one having the reference point on the negative and another one on the positive side, are shown in bold. A line that hits the bounded region has only one such segment.

The searching line is intersected with the included hyperplanes, and the outlyingness function (5.1) is calculated at every intersection point that lies between the two bounds, found on the previous step. At first, the hyperplanes that intersect the line outside the rougher, i.e. more liberal, bound are filtered out and added to (5.6). Then the first bound's intersection is taken as a median candidate, and as a starting point for the minimization procedure. The left hyperplanes are added to (5.6) with the sign they have in the first bound's intersection. The intersection points are iterated, the corresponding hyperplanes change the sign in the sum (5.6) and the outlyingness function is calculated as in (5.5). The outlyingness function is also calcu-

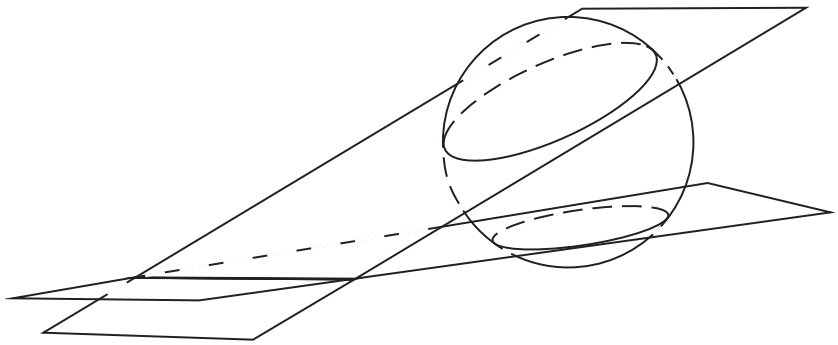


Figure 5.5: An example of an observation line lying outside of the bounded region (shown as a sphere) formed by two hyperplanes crossing the bounded region in a higher dimensional space.

lated at the intersections with the bounds. When the second bound's intersection is reached, the procedure is terminated. The outlyingness function has convex contours and therefore is unimodal on any line. However, in practice the outlyingness function may slightly fluctuate when it is optimized along a line. In this case, as soon as the outlyingness value begins to increase by a certain threshold amount, the minimization along the line is terminated.

When the minimum is found on the searching line, the next observation line is chosen among the lines that contain the minimum. In the simplest case, k hyperplanes, each defined by k unique observation points, define a point at their intersection and produce k observation lines through this point. More complex cases occur when some of the hyperplanes have observation points in common, and their intersection point lies in an affine subspace of dimension $d < k$, generated by these common points. Such a point may then be described by all possible observation hyperplanes that have the same common points, and thus the number of observation lines increases. If the number of observation lines exceeds the predefined maximum number max_{n_L}, ROO propose either to stop, or to take a random subset of these lines. Fischer et al. (2016) in their R-package **OjaNP** used to choose a new initial line in such cases.

If the minimum is defined with one or more bounds, we treat them like ordinary hyperplanes. In order to explicitly determine the bounded region's boarding lines and corners, the bounds are identified by k unique points that are found as intersections with the coordinate axes. If a bound is parallel to some of the axes, the diagonal axes in the space are taken. Thus the bounds do not have identifying points in common, and each intersection of the bounds and observation hyperplanes produces a minimum possible number of observation lines.

5.4.1 Formal description of the algorithm

The formal description has modifies the one of Ronkainen et al. (2003, *A.1, A.2*) and includes parts of it to make the comparison easier. In particular, Procedure 1 extends A.1 with the bounded region search (steps 2–14), and Procedure 3 modifies the minimization algorithm A.2 to be used in a bounded region (added steps 1-6, modified steps 18-31). Procedure 2 describes the bounded region construction as in the divisive approach A.

Procedure 1. Compute the exact Oja median.

Input: Data set $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ in $\mathbb{R}^k$.
The desired size s of the bounded box, $s = \frac{bounded\ box\ volume}{original\ volume}$.
Max number of observation lines to scan max_{n_L}.
Output: Exact Oja median $\boldsymbol{T} = \mathbf{Med}(\boldsymbol{X})$.

1: Precalculate all observation hyperplanes $p = (i_1, ..., i_k)$, $1 \leq i_1 < ... < i_k \leq n$.
2: Build the bounded region $\boldsymbol{B}$, that is, the set of bounds defining it, using procedure 2.
Chose the initial line L:
3: **for all** subsets $\boldsymbol{B}_s \subset \boldsymbol{B}$ with $|\boldsymbol{B}_s| = k - 1$ **do** ▷ find lines
4: Set $L \leftarrow \bigcap \boldsymbol{B}_s$.
5: Sort the bounds $\boldsymbol{b} \in \boldsymbol{B}$ according to their intersection points with L as ROO do in A.2, i.e. if $L = \{\boldsymbol{L}_0 + \beta \boldsymbol{u}_L : \beta \in \mathbb{R}\}$ and we have $\boldsymbol{b}_i \cap L = \{\boldsymbol{L}_0 + \beta_i \boldsymbol{u}_L\}$ and $\boldsymbol{b}_j \cap L = \{\boldsymbol{L}_0 + \beta_j \boldsymbol{u}_L\}$ for some $\boldsymbol{b}_i, \boldsymbol{b}_j \in \boldsymbol{B}$, then $i < j \iff \beta_i < \beta_j$. Denote the order $\boldsymbol{b}_{(1)}, \boldsymbol{b}_{(2)}, ..., \boldsymbol{b}_{(nb)}$, where $nb = |\boldsymbol{B}|$.
6: Set $\boldsymbol{y}_1 \leftarrow L \cap \boldsymbol{b}_{(1)}$.
7: $i \leftarrow$ smallest i at which $S_{\boldsymbol{b}_{(i)}}(\boldsymbol{y}_1) = 1$.
8: **if** $\exists j : j > i, S_{\boldsymbol{b}_{(j)}}(\boldsymbol{y}_1) \neq 1$ **then**
9: Continue ▷ the line is out of bounds
10: **else**
11: Break ▷ the line is found
12: **end if**
13: **end for**

14: Precalculate $\frac{1}{k!}\times$ the common part of (5.6), for given $\boldsymbol{t}$ in the bounded region:
$\boldsymbol{H} \leftarrow \sum_{p \notin \boldsymbol{B}} \frac{1}{k!} S_p(\boldsymbol{t}) \boldsymbol{d}_p$,
$H_0 \leftarrow \sum_{p \notin \boldsymbol{B}} \frac{1}{k!} S_p(\boldsymbol{t}) d_{0p}$.
15: Compute $\hat{\boldsymbol{T}} \leftarrow \arg\min_{\boldsymbol{t} \in L} O(\boldsymbol{t})$ using procedure 3.
16: Set the median candidate $\boldsymbol{T} \leftarrow \hat{\boldsymbol{T}}$.
17: Initialize the collection of investigated lines $\mathcal{L} \leftarrow \{L\}$.
18: Let n_L be the number of the observation lines containing $\hat{\boldsymbol{T}}$.
19: **if** $n_L > max_{n_L}$ **then**
20: There are too many possibilities. Goto 3.
21: **end if**
22: Construct the observation lines $\mathcal{L}' \leftarrow L_1, \dots L_{n_L}$.
23: Set $\mathcal{L}' \leftarrow \mathcal{L}' \setminus \mathcal{L}$.
24: **while** $\mathcal{L}' \neq \emptyset$ **do**
25: Find the line $L \in \mathcal{L}'$ of deepest descent.
26: Compute $\hat{\boldsymbol{T}} \leftarrow \arg\min_{\boldsymbol{t} \in L} O(\boldsymbol{t})$ using procedure 3.
27: Update $\mathcal{L} \leftarrow \mathcal{L} \cup \{L\}$ and $\mathcal{L}' \leftarrow \mathcal{L}' \setminus \{L\}$

28: **if** $O(\hat{\boldsymbol{T}}) < O(\boldsymbol{T})$ **then**
29: $\boldsymbol{T} \leftarrow \hat{\boldsymbol{T}}$
30: Goto 16.
31: **end if**
32: **end while**
33: **return** $\boldsymbol{T}$

Procedure 2. Build the bounded region as in the divisive approach A, sec 5.3.

Input: Data set $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ in $\mathbb{R}^k$.
Precalculated observation hyperplanes P.
The desired size s of the bounded box, $s = \frac{bounded\ box\ volume}{original\ volume}$.
Output: The bounded region $\boldsymbol{B}$.
Enclosing box E.

1: Define $\boldsymbol{B} \leftarrow \emptyset$. ▷ the set of bounds
2: Define $\boldsymbol{C} \leftarrow \emptyset$. ▷ the set of bounds' intersections
Build the Initial Box:
3: **for** $d = 1, ..., k$ **do** ▷ index $(-d)$ means all coordinates from 1 to k except of d
4: Set the seed of a bound $\boldsymbol{o}_d \leftarrow \max\{x_{1d}, ..., x_{nd}\}$, $\boldsymbol{o}_{-d} \leftarrow 0$.
5: Set the normal vector $\boldsymbol{n}_d \leftarrow -1$, $\boldsymbol{n}_{-d} \leftarrow 0$.
6: Define bound $\boldsymbol{b}$ with $\boldsymbol{o}$ and $\boldsymbol{n}$.
7: AddBound($\boldsymbol{b}$)
8: Set the seed of a bound $\boldsymbol{o}_d \leftarrow \min\{x_{1d}, ..., x_{nd}\}$, $\boldsymbol{o}_{-d} \leftarrow 0$.
9: Set the normal vector $\boldsymbol{n}_d \leftarrow +1$, $\boldsymbol{n}_{-d} \leftarrow 0$.
10: Define bound $\boldsymbol{b}$ with $\boldsymbol{o}$ and $\boldsymbol{n}$.
11: AddBound($\boldsymbol{b}$)
12: **end for** ▷ the Initial Box is now built
Proceed with the following divisions:
13: Calculate the original volume of the space as
$OriginalVolume = \prod_{d=1}^{k} (\max\{x_{1d}, ..., x_{nd}\} - \min\{x_{1d}, ..., x_{nd}\})$
14: **while** $NewVolume/OriginalVolume > s$ **do**
15: Define the center of $\boldsymbol{B}$ as $\bar{\boldsymbol{C}}$.
16: Calculate the negative rank function $\boldsymbol{g} = -\boldsymbol{R}(\bar{\boldsymbol{C}})$.
17: Define bound $\boldsymbol{b}$ with $\bar{\boldsymbol{C}}$ and $\boldsymbol{g}$.
18: AddBound($\boldsymbol{b}$)
19: Calculate $NewVolume = \prod_{d=1}^{k} \left(\max\{\boldsymbol{C}_{1d}, ..., \boldsymbol{C}_{|\boldsymbol{C}|d}\} - \min\{\boldsymbol{C}_{1d}, ..., \boldsymbol{C}_{|\boldsymbol{C}|d}\}\right)$ using the updated intersection points.
20: **end while**

21: **function** AddBound(new bound $\boldsymbol{b}$)
▷ here $(\boldsymbol{b} \cdot \boldsymbol{x})$ is the dot product of a point $\boldsymbol{x}$ and the normal vector of $\boldsymbol{b}$
22: **if** (Initial Box is built) and
$(sign(\boldsymbol{b} \cdot \boldsymbol{c}_1) = sign(\boldsymbol{b} \cdot \boldsymbol{c}_2) \forall \boldsymbol{c}_1, \boldsymbol{c}_2 \in \boldsymbol{C})$ **then**
23: **exit** without changes ▷ $\boldsymbol{b}$ lies outside of $\boldsymbol{B}$
24: **end if**
25: **for all** subsets $\boldsymbol{B}_s \subset \boldsymbol{B}$ with $|\boldsymbol{B}_s| = k - 1$ **do** ▷ find new intersections
26: Set $\boldsymbol{c} \leftarrow \bigcap(\boldsymbol{B}_s \cup \boldsymbol{b})$
27: **if** $\forall \boldsymbol{b} \in \boldsymbol{B}$ $sign(\boldsymbol{b} \cdot \boldsymbol{c})! = -1$ **then**
28: Add the new crossing point $\boldsymbol{C} \leftarrow \boldsymbol{C} \cup \boldsymbol{c}$.
29: **end if**
30: **end for**
31: Add the new bound $\boldsymbol{B} \leftarrow \boldsymbol{B} \cup \boldsymbol{b}$.
32: $\boldsymbol{C} \leftarrow \boldsymbol{C} \setminus \{\boldsymbol{c} : \boldsymbol{c} \in \boldsymbol{C}, sign(\boldsymbol{b} \cdot \boldsymbol{c}) = -1\}$. ▷ Remove the cut off intersections
33: $\boldsymbol{B} \leftarrow \boldsymbol{B} \setminus \{\boldsymbol{b} \in \boldsymbol{B} : sign(\boldsymbol{b} \cdot \boldsymbol{c}_1) = sign(\boldsymbol{b} \cdot \boldsymbol{c}_2) \forall \boldsymbol{c}_1, \boldsymbol{c}_2 \in \boldsymbol{C}\}$.
34: **end function**

Procedure 3. Minimize the outlyingness function O on the chosen line.

Input: Precalculated observation hyperplanes P.
Searching line L.
The bounded region $\boldsymbol{B}$.
Enclosing box E.

Output: The minimum $\hat{\boldsymbol{T}} \leftarrow \arg\min_{\boldsymbol{t} \in L} O(\boldsymbol{t})$ or an empty point if $L \cap \boldsymbol{B} = \emptyset$.

1: Sort bounds $\boldsymbol{b} \in \boldsymbol{B}$ according to their intersection points with L as in procedure 1.5, $\boldsymbol{b}_{(1)}, \boldsymbol{b}_{(2)}, ..., \boldsymbol{b}_{(nb)}$, where $nb = |\boldsymbol{B}|$.
2: Set $\boldsymbol{y}_1 \leftarrow L \cap \boldsymbol{b}_{(1)}$.
3: Set $\boldsymbol{y}_{b1} \leftarrow L \cap \boldsymbol{b}_{(i-1)}$ and $\boldsymbol{y}_{b2} \leftarrow L \cap \boldsymbol{b}_{(i)}$ where $i = \arg\min_i(S_{\boldsymbol{b}_{(i)}}(\boldsymbol{y}_1) = 1)$
4: **if** $\exists j : j > i, S_{\boldsymbol{b}_{(j)}}(\boldsymbol{y}_1) \neq 1$ **then**
5: **return** empty point. ▷ the line is out of bounds
6: **end if**
7: Chose any point $\boldsymbol{t}_0 \in \boldsymbol{B} \cap L$ (e.g. $\boldsymbol{t}_0 = \boldsymbol{y}_{b1}$).
8: Initialize $\boldsymbol{D} \leftarrow \boldsymbol{H}$, $D_0 \leftarrow H_0$, $\mathcal{H} \leftarrow \emptyset$.
9: **for all** $p \in \boldsymbol{B}$ **do** ▷ Compute the sum for hyperplanes, crossing L outside of E.
10: **if** $p \cap L \subset E$ **then**
11: $\mathcal{H} \leftarrow \mathcal{H} \cup p$
12: **else**
13: $\boldsymbol{D} \leftarrow \boldsymbol{D} + \frac{1}{k!} S_p(\boldsymbol{t}_0) \boldsymbol{d}_p$
14: $D_0 \leftarrow D_0 + \frac{1}{k!} S_p(\boldsymbol{t}_0) d_{0p}$.
15: **end if**
16: **end for**

17: Sort hyperplane indexes $p \in \mathcal{H}$ according to their intersection points with L as ROO do in A.2, $p_{(1)} \leq p_{(2)} \leq ... \leq p_{(np)}$, where $np = |\mathcal{H}|$ and $<$ resp. $\leq$ denote the order of intersection points.
18: Define $\mathcal{H}_1 \leftarrow \{p : p \in \mathcal{H}, p \cap L < \boldsymbol{y}_{b1}\}$,
$\mathcal{H}_2 \leftarrow \mathcal{H} \setminus \mathcal{H}_1$,
$\mathcal{H}_3 \leftarrow \{p : p \in \mathcal{H}_2, p \cap L \leq \boldsymbol{y}_{b2}\}$.
19: Set $\boldsymbol{y}_1 \leftarrow L \cap p_{(1)}$ and $\boldsymbol{y}_{np} \leftarrow L \cap p_{(np)}$.
20: Compute $\boldsymbol{D} \leftarrow \boldsymbol{D} + \sum_{p \in \mathcal{H}_1} \frac{1}{k!} S_p(\boldsymbol{y}_{np}) \boldsymbol{d}_p + \sum_{p \in \mathcal{H}_2} \frac{1}{k!} S_p(\boldsymbol{y}_1) \boldsymbol{d}_p$ and
$D_0 \leftarrow D_0 + \sum_{p \in \mathcal{H}_1} \frac{1}{k!} S_p(\boldsymbol{y}_{np}) d_{0p} + \sum_{p \in \mathcal{H}_2} \frac{1}{k!} S_p(\boldsymbol{y}_1) d_{0p}$.
21: Set potential minimum $\hat{\boldsymbol{T}} \leftarrow \boldsymbol{y}_{b1}$.
22: Evaluate $O(\hat{\boldsymbol{T}}) = \boldsymbol{D}^\top \hat{\boldsymbol{T}} + D_0$.
23: **for all** $\{i : p_{(i)} \in \mathcal{H}_3\}$ **do**
24: Set $\boldsymbol{D} \leftarrow \boldsymbol{D} - \frac{1}{k!} S_{p_{(i-1)}}(\boldsymbol{y}_1) \boldsymbol{d}_{p_{(i-1)}} + \frac{1}{k!} S_{p_{(i-1)}}(\boldsymbol{y}_{np}) \boldsymbol{d}_{p_{(i-1)}}$,
25: Set $D_0 \leftarrow D_0 - \frac{1}{k!} S_{p_{(i-1)}}(\boldsymbol{y}_1) d_{0p_{(i-1)}} + \frac{1}{k!} S_{p_{(i-1)}}(\boldsymbol{y}_{np}) d_{0p_{(i-1)}}$.
26: Set $\boldsymbol{t} \leftarrow L \cap p_{(i)}$.
27: Evaluate $O(\boldsymbol{t}) = \boldsymbol{D}^\top \boldsymbol{t} + D_0$
28: **if** $O(\boldsymbol{t}) < O(\hat{\boldsymbol{T}})$ **then**
29: Set $\hat{\boldsymbol{T}} \leftarrow \boldsymbol{t}$ and $O(\hat{\boldsymbol{T}}) \leftarrow O(\boldsymbol{t})$.
30: **end if**
31: **end for**
32: **return** $\hat{\boldsymbol{T}}$.

5.5 Numerical experience and conclusions

The new algorithm was implemented as a part of the R-package **OjaNP** of Fischer et al. (2016). A function `ojaMedianExB` was implemented to be used in place of the previous `ojaMedianEx` by ROO. A parameter `alg="exact_bounded"` was added to the function `ojaMedian`, and the corresponding C++ routines were modified. The benchmark values were measured inside the C++ routines, using file logging. This allows to easily compare the efficiency of the original and modified algorithms, excluding the data transformation and hyperplanes generation time.

The desired volume of the bounded region was set, and the calculation time was determined for the new exact algorithm as well as for the ROO procedure. Best results were received at around 10^{-8} of the original volume in most of tried datasets. The new algorithm showed to be three to six times faster than the one by ROO.

The new algorithm is able to calculate data sets of the same size and dimension as the ROO algorithm. It is mainly restricted by the amount of RAM, as it needs to store all $\binom{n}{k}$ hyperplanes. E.g. the calculation of the median in a data set of size 5×100 needs 12 GB RAM. A PC with an Intel Core i7-4770 (3.4 GHz) processor and 32 GB RAM was employed in the experimental studies. Only one processor core was used. The algorithm was able to find the

median in data sets of sizes 3×750, 4×150, 5×75, 6×50 in less than half an hour, and of sizes 4×200, 5×100 in less than an hour.

In constructing the bounded regions we have tried the different variants proposed in section 5.3. As it was observed, all proposed approaches (B, C, D) that use the subgradient's ending point or direction to define the next cutting point converge extremely slow, compared to the simple divisive approach (A). Although the subgradients may sometimes produce really good cutting points, which strongly reduce the bounded region, they often stick at the angles of the bounded region, so that the next steps reduce the bounded region by a narrow slice only, which is close to an existing bound. Particularly in higher dimensions, the subgradients also appear to be too short, so that the amount of the volume cut in each step becomes unsatisfying. Therefore in out search we desist from the lengths of the subgradients and use only their directions. All the numerical results provided in this chapter were received using the divisive approach starting with the initial rectangular bounded region. The number of cuts needed to obtain the desired volume appears to depend only moderately on the size and dimensionality of the data.

For both algorithms, the ROO and the new one, the performance of the searching procedure strictly depends on the selected initial line. As ROO select this line at random, their calculation times differ significantly between different launches. Our bounding algorithm selects the firstly found border line of the bounded region as the initial line, which makes the searching path completely deterministic, although in general not the fastest possible.

Employing bounds considerably decreases the complexity of the algorithm. The minimization along a line produces most of the complexity of the ROO algorithm. The line is intersected with $H = \binom{n}{k}$ hyperplanes, the intersections are sorted and all of them iterated, which has a complexity of $O(H^2 \log H)$. The bounding algorithm leaves a smaller amount $h < H$ of hyperplanes. Only b hyperplanes, $b < h$, that have intersections between the bounds remain to be considered. The rougher bound also strongly decreases the number of hyperplanes which need to be sorted to $s : b < s < h$. This provides a complexity of $O(b \times h \log s)$ only.

Tables 5.1, 5.2 and 5.3 exhibit a few exemplary results. The experimental data is an even mixture of two multidimensional normal distributions $N([15, 0, ..., 0]_k, diag([1, 25, 1, ..., 1]_k))$ and $N(\mathbf{0}_k, diag(\mathbf{1}_k))$ although the conclusions are the same for the data having other form and for the real data sets. They show how the performance parameters listed below depend on the data dimension and size, given the intended volume equal to 10^{-8} (for Tables 5.1 and 5.2), where *#Cuts* is the number of cuts needed to reach the intended volume using the divisive approach, *HP(%)* is the percent of the hyperplanes intersecting the bounded region, *#Steps* is the number of minimization steps needed to find the median, and time periods needed to: determine the bounded region T_{bounds}, calculate the median (after the bounded region is determined) T_{count}, perform the whole procedure T_{total}, and to find the median using the algorithm by ROO $T_{original}$. The given times do not include the generation of all observation hyperplanes, which is the same for both algorithms.

Table 5.1: The performance parameters for $n \in \{50, 75\}$ and intended volume 10^{-8}.

k	n	#Cuts	HP(%)	#Steps	T_{bounds}	T_{count}	T_{total}	$T_{original}$
2	50	29	0.16	3	0.009	0.001	0.010	0.018
3	50	39	0.64	9	0.216	0.053	0.269	0.557
4	50	42	3.33	34	3.015	2.433	5.448	25.529
5	50	42	9.65	45	31.359	42.876	74.235	476.600
6	50	45	17.65	77	345.128	774.382	1119.510	3149.010
2	75	32	0.11	2	0.023	0.002	0.025	0.038
3	75	36	0.34	14	0.658	0.243	0.901	3.033
4	75	42	2.11	39	15.353	13.720	29.073	110.291
5	75	45	7.17	70	281.888	474.461	756.349	2667.890

Table 5.2: The performance parameters for $k \in \{4, 5\}$ and intended volume 10^{-8}.

k	n	#Cuts	HP(%)	#Steps	T_{bounds}	T_{count}	T_{total}	$T_{original}$
4	25	38	3.26	25	0.159	0.095	0.254	0.707
4	50	42	3.33	34	3.015	2.433	5.448	25.529
4	75	42	2.11	39	15.353	13.720	29.073	110.291
4	100	43	2.60	35	49.360	41.691	91.051	338.950
5	25	44	11.77	39	0.930	0.932	1.862	6.171
5	50	42	9.65	45	31.359	42.876	74.235	476.600
5	75	45	7.17	70	281.888	474.461	756.349	2667.890
5	100	43	8.53	71	1166.930	2220.330	3387.260	9803.730

The part of the hyperplanes crossing the bounded region of the given volume grows quickly with dimension, as it is seen in Tables 5.1 and 5.3. On the other hand, the part of these hyperplanes that take part in the minimization process decreases, since many of their intersections with a searching line lie outside the bounded region. Note that the bounded region, being located in the middle of the data cloud, is intersected by most of the hyperplanes, so that the part of the included hyperplanes is much larger than the part of the final volume, compared to the initial one. Our calculations demonstrate that the part of included hyperplanes strongly depends on the dimensionality and the number of observations, which is also shown in Figure 5.6. However, the number of observations has less influence than the dimension.

These three tables also show that the new exact bounding algorithm finds the median much faster than the one of ROO. We observe that for each given data set the number of necessary minimization steps is almost the same in both algorithms. As the intended volume is reduced, the time needed to build the bounded region increases, while the minimization time decreases along with the number of hyperplanes and their intersections involved, and the total time also decreases (Table 5.3, Figure 5.7). However, beyond some point, usually at around 10^{-08} of the volume, this procedure becomes less efficient, and the total time increases. A smaller

Table 5.3: The performance parameters for data sets 4×100 and 6×50, with different intended volumes. Volume equal to one corresponds to the ROO algorithm.

k	n	Volume	#Cuts	HP(%)	#Steps	T_{bounds}	T_{count}	T_{total}
4	100	1			36			338.950
4	100	10^{-02}	14	67.45	50	17.648	283.936	301.584
4	100	10^{-03}	18	46.62	49	23.216	201.245	224.461
4	100	10^{-04}	22	28.47	39	28.304	112.429	140.733
4	100	10^{-05}	27	14.23	48	33.403	97.364	130.767
4	100	10^{-06}	31	9.76	29	37.707	49.499	87.206
4	100	10^{-07}	37	5.47	38	44.391	55.032	99.423
4	100	10^{-08}	43	2.60	35	49.360	41.691	91.051
4	100	10^{-09}	47	1.64	33	57.209	37.402	94.611
4	100	10^{-10}	52	0.83	33	59.709	35.347	95.056
4	100	10^{-20}	97	<0.01	22	108.205	21.089	129.294
4	100	10^{-30}	100	<0.01	13	109.004	12.778	121.782
6	50	1			73			3149.010
6	50	10^{-05}	31	53.86	91	215.747	1842.733	2058.480
6	50	10^{-06}	36	35.61	71	270.619	1272.041	1542.660
6	50	10^{-07}	40	26.68	75	301.413	885.707	1187.120
6	50	10^{-08}	45	17.65	77	345.128	774.382	1119.510
6	50	10^{-09}	49	12.97	70	373.170	545.055	918.225
6	50	10^{-10}	53	9.14	94	378.424	705.386	1083.810

volume may also contain a higher amount of isolated routes through the observation lines, which involves travelling along the bounds and additionally slows the procedure down.

If the volume is small enough, any point of it (e.g. the average of the bounds' intersections) may be taken as an approximate value of a median. For example, for a four-dimensional dataset bounded by a cube of side length 10, 10^{-8} of the volume was reached in 43 cuts, and the center of the final bounded region equalled the median ± 0.05 by each coordinate, which is quite precise. The precision of this approximative method depends on the volume of the bounded region and is controlled by it. The method yields as precise results as the approximative methods provided in the **OjaNP** R-package. In general, the approximate value of the median is found much faster than the exact one. We searched an approximative median with precision equal to half of the bounded region's volume, the computation times of which are given in the column T_{bounds} of Table 5.3. However, this approximation method is not really useful, as it considers all observation hyperplanes and is therefore largely outperformed by the approximative methods of ROO.

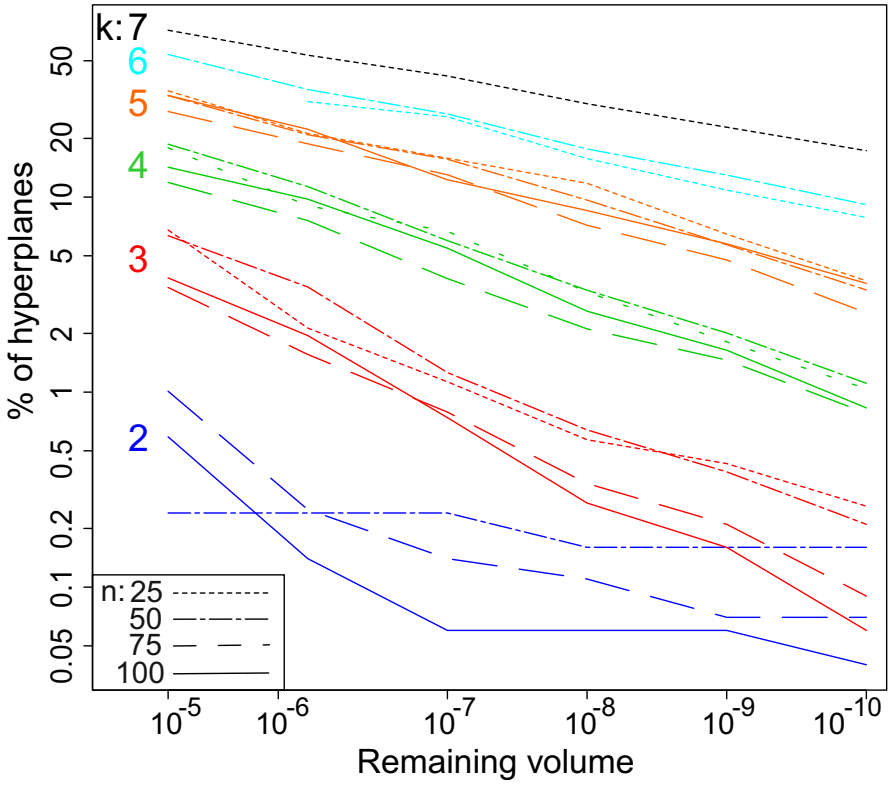


Figure 5.6: The dependence of the part of hyperplanes crossing the bounded region (log scale) on the size of the region for $k \in [2..7]$ and $n \in \{25, 50, 75, 100\}$.

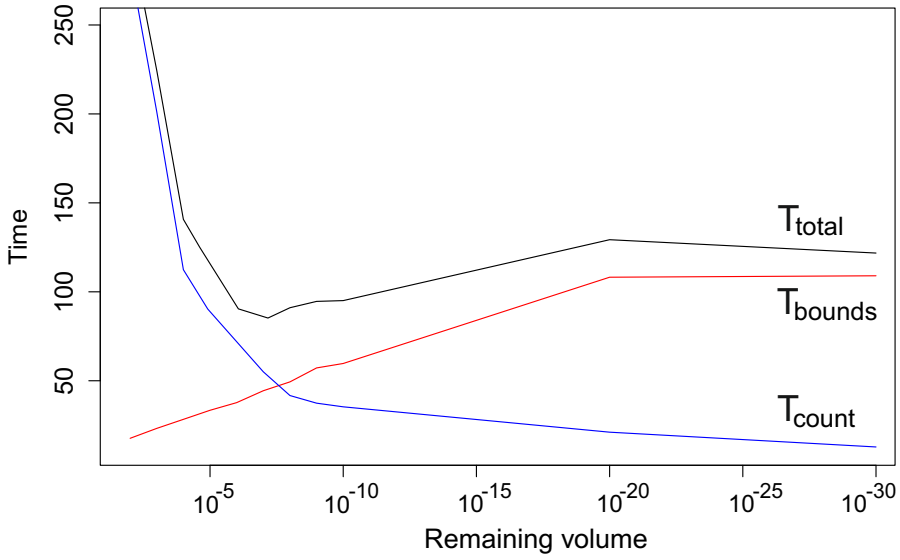


Figure 5.7: The dependence of calculation time on the size of the bounded region. Note that the ROO algorithm has total time of ca. 340 seconds.

Chapter 6

Outlook

The thesis has been devoted to depth and potential based classification and to the exact calculation of the Oja median. In Chapter 2, we introduced the pot-pot classification that may be seen as a combination of DD-classification and density-based classification. The DD-plot employs depth functions, that are focused on the center of a class. On the contrary, the density measures the mass of the data and is of a local nature. Further, local depths can be applied to build the DD-plots, by making allowance for local centers. Local depths allow to vary the rate of localization. Unlike density estimates they do not depend on bandwidth matrices that have many parameters, and thus the parameter tuning may be easier.

Chapter 3 is devoted to the R-Package **ddalpha** that implements various depth functions and classifiers for multivariate and functional data under one roof. The functionality of the package may be further extended with the implementations of other depth notions and separators, and data visualization instruments. The package is made expandable with user-defined custom depth methods and separators, which simplifies the usage of the package for scholars that implement their own methods and want to combine or compare them with those that are implemented in the package.

In Chapters 2 and 3 the α-procedure was applied in its DD-version. However, the α-procedure is itself interesting, as it efficiently reduces the space by selecting the most relevant properties. With slight modifications it can be successfully applied to the data with missing values. The α-procedure produces a linear solution in the extended space, and a polynomial solution in the original space. Together with Pavlo Mozharovskyi we are further exploring the α-procedure and space extension techniques. The idea of the kernel trick is to work in a rich extended space of features while performing the computation with kernel functions in the original lower-dimensional space. By introducing kernels to the generalized portrait method (now known as Support Vector Machine, SVM), Vapnik (1998) has demonstrated the power of kernels, which gave rise to numerous applications of the methodology. We introduce kernels to the α-procedure by employing it in a finite-dimensional explicit approximation of a reproducing kernel Hilbert space (RKHS). By its inductive character and robustness, the α-procedure selects a subspace of RKHS and evades the weights of misclassified observations.

The preliminary results were presented on the European Conference on Data Analysis 2015 in Colchester.

In Chapter 4 we propose a new objective function for classification, that weights the misclassified points with their depth. This approach focuses on the centers of the classes and enforces classification of more central points, that are more 'typical' for their classes. This objective function can also be used to measure the performance of other classifiers and in the cross-validation tuning procedures. We also introduced classifiers that minimize this objective function. We investigated the properties of these classifiers and compared them with those of the Bayes classifier for a two-class problem. The theoretical results may be extended to a more general classification problem with more classes and to multimodal and non-elliptical distributions. Also the class-specific weights shall be added and the behavior of the depth-weighted classifier shall be compared with that of the Bayes classifier. We hypothesize that due to underweighting the outliers of the minor class, the depth-weighted classifier shall not lead to misclassification of the major class in their neighbourhood, unlike the Bayes classifier.

In Chapter 5 we modified the algorithm of Ronkainen et al. (2003) for the exact calculation of the Oja median by employing bounded regions which contain the median. Employing bounds considerably decreases the complexity of the algorithm. Similar approaches can be used to accelerate the search of the medians based on other depth notions.

Appendix
Overview of local depth notions

Unlike the global depths, a local depth refers to local centers of the data, which allows to cope with multimodal distributions. Here I give an overview of the local depth notions found in the literature and present their population versions.

Local depths may be divided into two groups: restricted depths, and kernelized depths. Restricted depths in some way obtain a subsample around the point $\boldsymbol{x}$, where the depth is calculated. Kernelized depths apply kernels to weight the measure used by the depth function, or to weight the space around $\boldsymbol{x}$, or to find the depth in the reproduced kernel Hilbert space.

Restricted local depths

Agostinelli and Romanazzi (2011) introduced local variants of global depth functions, referring to the types coined by Zuo and Serfling (2000).

Type A depth functions are defined by the average closeness of $\boldsymbol{x}$ to r points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r$, measured by some bounded nonnegative function $h(\boldsymbol{x}; \boldsymbol{x}_1, \ldots, \boldsymbol{x}_r)$:

$$D(\boldsymbol{x}|X) = Eh(\boldsymbol{x}; X_1, \ldots, X_r),$$

where $X_1, \ldots, X_r$ is a random sample from $X \in \mathbb{R}^d$.

Type B depth functions are defined by the reciprocal average distance from $\boldsymbol{x}$ to r points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r$, the distance being measured by any unbounded nonnegative function $g(\boldsymbol{x}; \boldsymbol{x}_1, \ldots, \boldsymbol{x}_r)$:

$$D(\boldsymbol{x}|X) = \big(1 - Eg(\boldsymbol{x}; X_1, \ldots, X_r)\big)^{-1}.$$

Types A and B local depth functions are obtained by selecting these r points only in the neighbourhood of $\boldsymbol{x}$. For example, for simplicial and simplicial volume depths the volumes of the simplices built on $d+1$ points are restricted by some value τ. Simplicial depth (type A) is defined as $D_S(\boldsymbol{x}|X) = E(I\{\boldsymbol{x} \in S_{d+1}\}) = P(S_{d+1}|\boldsymbol{x} \in S_{d+1})$, with $S_{d+1} = S[X_1, \ldots, X_{d+1}]$. Defining the volume of the corresponding simplex as $v(S_{d+1})$, the local form is

$$LD_S(\boldsymbol{x}|X) = P(S_{d+1} \mid \boldsymbol{x} \in S_{d+1} \wedge v(S_{d+1}) \leq \tau),$$

The global Oja's simplicial volume depth (type B) is defined as
$D_O(x|X) = \big(1 - E(v(S[\boldsymbol{x}, X_1, \ldots, X_d]))\big)^{-1}$. Its local form is

$$LD_O(\boldsymbol{x}|X) = \big(1 - E(v(S[\boldsymbol{x}, X_1, \ldots, X_d])) \mid v(S[\boldsymbol{x}, X_1, \ldots, X_d]) \leq \tau\big)^{-1}.$$

Type D depth functions are defined as

$$D(\boldsymbol{x}|X) = \inf_{S \in \mathbb{S}} P(S|\boldsymbol{x} \in S),$$

where $\mathbb{S}$ is a suitable family of closed sets. A local version is obtained by employing proper subsets of S. The global halfspace depth is defined as $D_H(\boldsymbol{x}|X) = \inf_{||\boldsymbol{u}||=1} P(\boldsymbol{z} \in \mathbb{R}^d | \boldsymbol{u}^\top \boldsymbol{z} \geq \boldsymbol{u}^\top \boldsymbol{x})$. Replacing hyperspaces with closed slabs of width τ formed with two parallel hyperplanes, one of which goes through $\boldsymbol{x}$, we obtain a local version

$$LD_H(\boldsymbol{x}|X) = \inf_{||\boldsymbol{u}||=1} P(z \in \mathbb{R}^d \mid \boldsymbol{u}^\top \boldsymbol{x} \leq \boldsymbol{u}^\top \boldsymbol{z} \leq \boldsymbol{u}^\top \boldsymbol{x} + \tau).$$

In the limit, as τ tends to infinity, this local halfspace depth coincides with the global one.

Another version of a local halfspace depth using weighted probability in the halfspace was proposed by Hlubinka et al. (2010), Kotík (2014). They denote a measurable and bounded *weight function* by $w : \mathbb{R}^d \times \mathbb{S}^d \to [0, \infty)$. Then the *generalized halfspace depth* function is defined as

$$WD(\boldsymbol{x}|X) = \inf_{||\boldsymbol{u}||=1} E_P w(X - \boldsymbol{x}, \boldsymbol{u}).$$

It is also assumed to be symmetric and piecewise continuous. And the two variants of *weighted halfspace ratio depth* are defined as

$$WRD_1(\boldsymbol{x}|X) = \inf_{||\boldsymbol{u}||=1} \frac{E_P w(X - \boldsymbol{x}, \boldsymbol{u})}{E_P w(X - \boldsymbol{x}, -\boldsymbol{u})},$$

$$WRD_2(\boldsymbol{x}|X) = \inf_{||\boldsymbol{u}||=1} \frac{E_P w(X - \boldsymbol{x}, \boldsymbol{u})}{E_P w(X - \boldsymbol{x}, \boldsymbol{u}) + E_P w(X - \boldsymbol{x}, -\boldsymbol{u})},$$

where $0/0 = 1$ and $0/(0+0) = 1/2$. All these variants are equivalent in sense of the multivariate ordering: $WRD_1(\boldsymbol{x}_1|X) < WRD_1(\boldsymbol{x}_2|X) \Leftrightarrow WRD_2(\boldsymbol{x}_1|X) < WRD_2(\boldsymbol{x}_2|X)$, the same as with $WD(\boldsymbol{x}|X)$.

Usually a *spherically symmetric* about $\boldsymbol{u}$ function is chosen as a weight function, that is $w(\boldsymbol{x}, \boldsymbol{u}) = h(||\boldsymbol{x} - \langle \boldsymbol{u}, \boldsymbol{x} \rangle \boldsymbol{u}||, \langle \boldsymbol{u}, \boldsymbol{x} \rangle) = h(\sqrt{||\boldsymbol{x}||^2 - \langle \boldsymbol{u}, \boldsymbol{x} \rangle^2}, \langle \boldsymbol{u}, \boldsymbol{x} \rangle)$. All the weighted functions used below and illustrated in Figure A.1 are spherically symmetric.

The *halfspace depth* $D_H(\boldsymbol{x})$ is equal to $WD(\boldsymbol{x})$ and $WRD_2(\boldsymbol{x})$ with $w(\boldsymbol{x}, \boldsymbol{u}) \equiv \mathbb{I}\{\boldsymbol{u}^T \boldsymbol{x} \geq 0\}$ for the absolutely continuous distributions.

The *cylindrical weight function* is defined as

$$w(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{I}\{||\boldsymbol{x} - \langle \boldsymbol{x}, \boldsymbol{u} \rangle \boldsymbol{u}|| \leq h, \langle \boldsymbol{x}, \boldsymbol{u} \rangle \geq 0\}.$$

Here cylinders with radius $h > 0$ are built along unit vectors extending from $\boldsymbol{x}$.

The *cone weight function* is similar to the cylindric one. It is defined as

$$w(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{I}\{\angle(\boldsymbol{x}, \boldsymbol{u}) \leq \alpha\},$$

for an angle $\alpha \in [0, \pi/2]$. For a continuous distribution D_H equals WD for $\alpha = \pi/2$.

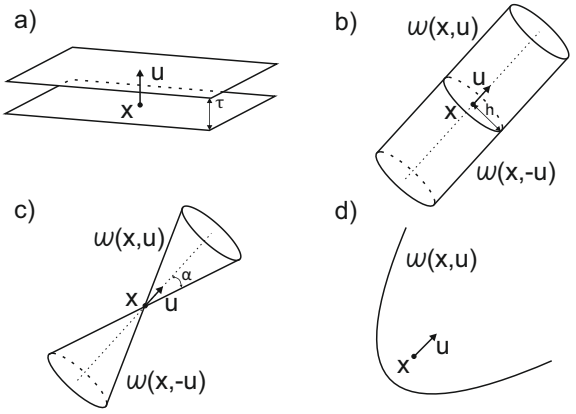


Figure A.1: The restricted subspaces for the local halfspace depths: a) of Agostinelli and Romanazzi; b-d)of Hlubinka, Kotík and Venćalek with the following weight functions b) cylindrical, c) cone, d) conic section.

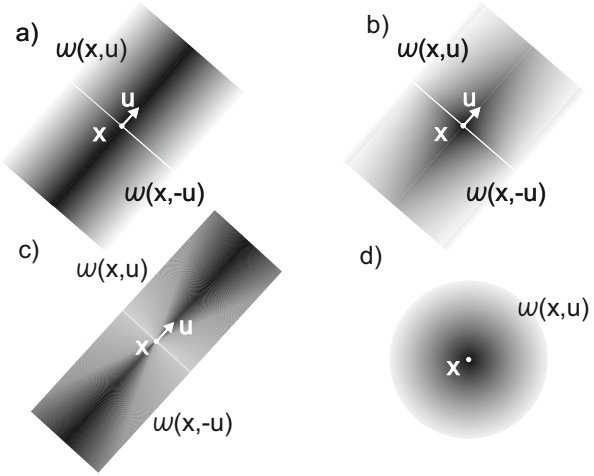


Figure A.2: The kernelized halfspace depths of Hlubinka, Kotík and Venćalek using the next weight functions: a) kernel cylinder with constant kernel bandwidth, b) kernel cylinder with bandwidth dependent on $\boldsymbol{x}$, c) kernel cone, d) local kernel.

The *conic section weight function* is defined using a conic section $C(\boldsymbol{u}, t)$ (sphere, ellipsoid, paraboloid or hyperboloid) instead of a halfspace, with its major axis in the direction of $\boldsymbol{u}$ and with the focus in the point $t\boldsymbol{u}$:

$$w(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{I}\{\boldsymbol{x} \in C(\boldsymbol{u}, t)\}.$$

The *conic section radius function* $r_e(\boldsymbol{x}, \boldsymbol{u}) = ||\boldsymbol{x}|| - e\langle \boldsymbol{x}, \boldsymbol{u}\rangle$ is used to describe conic sections. Here $e \geq 0$ is the eccentricity: $e = 0$ for a sphere, $e < 1$ for an ellipsoid, $e = 1$ for a paraboloid, $e > 1$ for a hyperboloid and $e = +\infty$ for a halfspace. The conic section weight function for the given radius l may be now rewritten as

$$w(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{I}\{r_e(\boldsymbol{x}, \boldsymbol{u}) \leq l\}.$$

A completely different notion of a restricted local depth is given by Paindaveine and Van Bever (2013), whose approach allows to turn any global depth into a local one. The produced depth provides a measure of local centrality at any level of localization. First the data is symmetrized around $\boldsymbol{x}$, so that $P_{\boldsymbol{x}} = \frac{1}{2}P^{\boldsymbol{X}} + \frac{1}{2}P^{2\boldsymbol{x}-\boldsymbol{X}}$. Then a neighbourhood of $\boldsymbol{x}$ is defined in this symmetrized data by the smallest depth region $R^{\beta}_{\boldsymbol{x}}(P) = R^{\beta}(P_{\boldsymbol{x}})$ with $P_{\boldsymbol{x}}$-probability greater or equal to the localization level $\beta \in [0, 1]$. Finally the local depth is calculated as a global depth of $\boldsymbol{x}$ restricted to this $R^{\beta}_{\boldsymbol{x}}(P)$ region,

$$LD^{\beta}(\boldsymbol{x}; P) = D(\boldsymbol{x}; P^{\beta}_{\boldsymbol{x}}),$$

where $P^{\beta}_{\boldsymbol{x}}$ is the conditional distribution of P on $R^{\beta}_{\boldsymbol{x}}(P)$. Figure A.3 shows this process.

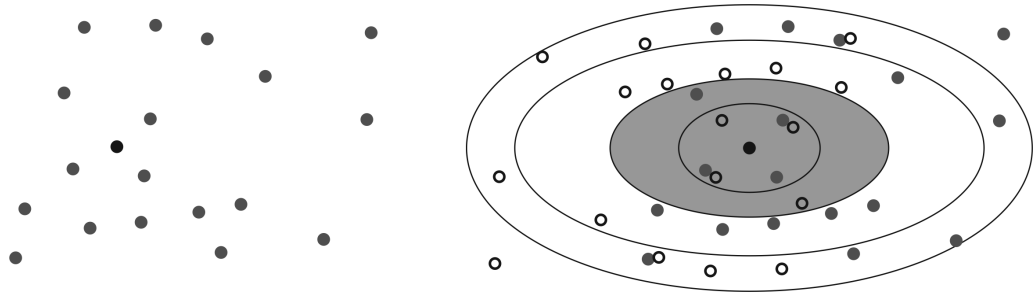


Figure A.3: The Local depth of Paindaveine and Van Bever: left – the original data; right – the symmetrized data, the depth contours and the restricted region $R^{\beta}_{\boldsymbol{x}}(P)$ (grey) in which the global depth of $\boldsymbol{x}$ is calculated.

Kernelized local depths

The restricted local depths consider a subsample or subspace depending on $\boldsymbol{x}$, where the depth is calculated. Instead, a kernelized depth at some point $\boldsymbol{x}$ includes, as a weight, a kernel estimate e.g. of a difference with or distance to $\boldsymbol{x}$.

Dutta and Ghosh (2015) proposed localized spatial depth. The global spatial depth calculates the expectation of the unit vectors, pointing from the point $\boldsymbol{x}$ in the directions of the data points $D_{SP}(\boldsymbol{x}|X) = 1 - ||E(u(\boldsymbol{t}))||$, where $\boldsymbol{t} = \boldsymbol{\Sigma}^{-1/2}(\boldsymbol{x} - X)$, $u(\boldsymbol{t}) = \frac{\boldsymbol{t}_i}{||\boldsymbol{t}||}$ is the unit vector in the direction of X, and $\boldsymbol{\Sigma}$ is the covariance matrix of X. The local version is obtained by kernelizing the distances

$$LD_{SP}(\boldsymbol{x}|X) = E[K_h(\boldsymbol{t})] - ||E[K_h(\boldsymbol{t})u(\boldsymbol{t})]||$$

with some spherical kernel $K_h(\boldsymbol{x})$ having a single parameter h. If $h > 1$, this depth has to be multiplied by h^d. For the classification task the classes are scaled (using sphering transformation) separately, but the same bandwidth parameter h is used for both of them. In the left part of Figure A.4 the average of the unit vectors is black, and the unit vectors pointing to the data points are gray. In the right part the kernelized versions are shown.

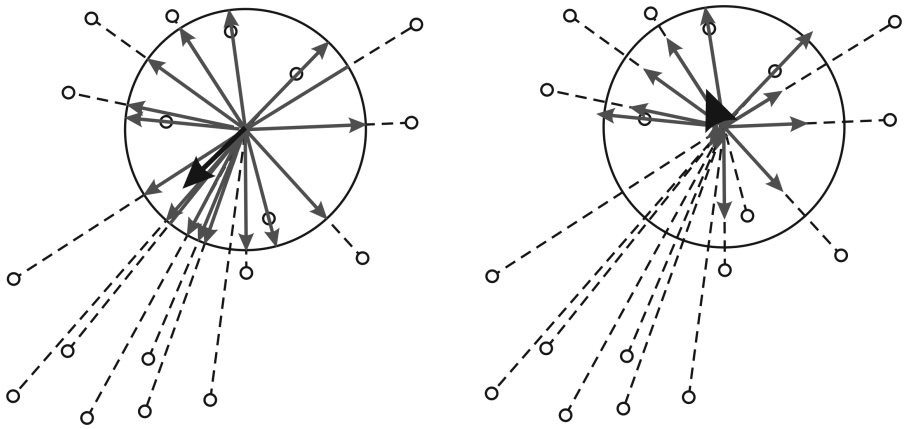


Figure A.4: Global and local spatial depth.

The weighted halfspace depth (Hlubinka et al., 2010) also employs kernel weight functions. For example, the *kernel cylinder weight function* weights the points in the halfspaces, so that the points lying in the neighbourhood of the axis $\boldsymbol{u}$ get more weight

$$w(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{I}\{\langle \boldsymbol{x}, \boldsymbol{u}\rangle \geq 0\} K_h(||\boldsymbol{x} - \langle \boldsymbol{x}, \boldsymbol{u}\rangle \boldsymbol{u}||).$$

The other given weight functions may also be kernelized. The *kernel cone weight function* is

$$w(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{I}\{\langle \boldsymbol{x}, \boldsymbol{u}\rangle \geq 0\} K_h(\angle(\boldsymbol{x}, \boldsymbol{u})),$$

and the *kernel conic section weight function* is defined as

$$w(\boldsymbol{x}, \boldsymbol{u}) = K_h(r_e(\boldsymbol{x}, \boldsymbol{u})).$$

The *local kernel weight function* $w(\boldsymbol{x}, \boldsymbol{u}) = K_h(\boldsymbol{x})$ does not depend on $\boldsymbol{u}$ and $WD(\boldsymbol{x}|X)$ equals the density estimated using this kernel. In these examples the kernel bandwidth may be either constant, or dependent on $\boldsymbol{x}$. The weight functions are shown in Figure A.2.

Another approach is using global depths in the Reproducing Kernel Hilbert Space (RKHS) as for example, the Mahalanobis depth in RHKS of Hu et al. (2011). Let X have an empirical distribution. The global Mahalanobis depth is defined as

$$MD(\boldsymbol{x}|X) = \left(1 + dm^2_\Sigma(\boldsymbol{x}, \overline{X})\right)^{-1},$$

where $dm^2_\Sigma(\boldsymbol{x}, \overline{X}) = (\boldsymbol{x} - \overline{X})^T \Sigma^{-1} (\boldsymbol{x} - \overline{X})$, is the Mahalanobis distance to the mean $\overline{X}$ and Σ is the covariance matrix. This requires Σ to be invertible. Hu et al. (2011) cope with this problem using the singular value decomposition of $A^T A$, where $A = (X_1 - \overline{X}, \ldots, X_n - \overline{X})^T$, $\overline{X}$ is the sample mean. Let the rank of $A^T A$ be r, then σ_i^2 and $\boldsymbol{v}_i$ are respectively its i_{th} eigenvalue and eigenvector, $i \leq r$. Then the generalized Mahalanobis depth is

$$GMD(\boldsymbol{x}|X) = \left(1 + \sum_{i=1}^{r} \frac{\left((\boldsymbol{x} - \overline{X})^T \boldsymbol{v}_i\right)^2}{\sigma_i^2}\right)^{-1}.$$

The generalized Mahalanobis depth is then used in the Hilbert space $\mathcal{H}_K$, generated by a mapping function $\phi : \mathbb{R}^p \to \mathcal{H}_K$, that is associated with some kernel function $K(\boldsymbol{x}, \boldsymbol{y}) = < \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) >$:

$$GMD(\phi(\boldsymbol{x})|X) = \left(1 + \sum_{i=1}^{r} \frac{\left((\phi(\boldsymbol{x}) - \overline{\phi(X)})^T \boldsymbol{v}_i\right)^2}{\sigma_i^2}\right)^{-1} = \left(1 + \sum_{i=1}^{r} \frac{\left(S(\boldsymbol{x}, \boldsymbol{x}_j)_{1\times n}^{j=1,\ldots,n} \boldsymbol{u}_i\right)^2}{\sigma_i^4}\right)^{-1},$$

where $S(\boldsymbol{x}, \boldsymbol{x}_j) = K(\boldsymbol{x}, \boldsymbol{x}_j) - \frac{1}{n}\sum_{l=1}^n K(\boldsymbol{x}, \boldsymbol{x}_l) - \frac{1}{n}\sum_{l=1}^n K(\boldsymbol{x}_j, \boldsymbol{x}_l) + \frac{1}{n^2}\sum_{k=1}^n \sum_{l=1}^n K(\boldsymbol{x}_k, \boldsymbol{x}_l)$, σ_i^2 is a nonzero eigenvalue of $S(\boldsymbol{x}_k, \boldsymbol{x}_l)_{n\times n}^{k,l=1,\ldots,n}$ with eigenvector $\boldsymbol{u}_i$, and r is the rank of $S(\boldsymbol{x}_k, \boldsymbol{x}_l)_{n\times n}^{k,l=1,\ldots,n}$.

Bibliography

Agostinelli, C. and Romanazzi, M. (2011). Local depth, *Journal of Statistical Planning and Inference* **141**: 817–830.

Agostinelli, C., Romanazzi, M. and SLATEC Common Mathematical Library (2013). **localdepth***: Local depth.* R package version 0.5-7.
URL: *http://CRAN.R-project.org/package=localdepth*

Aizerman, M. A., Braverman, E. M. and Rozonoer, L. I. (1970). *The Method of Potential Functions in the Theory of Machine Learning*, Nauka, Moscow.

Azzalini, A. (2013). *The Skew-Normal and Related Families*, Cambridge University Press.

Bazovkin, P. (2013). **WMTregions***: Exact calculation of WMTR.* R package version 3.2.6.
URL: *http://CRAN.R-project.org/package=WMTregions*

Bazovkin, P. and Mosler, K. (2012). An exact algorithm for weighted-mean trimmed regions in any dimension, *Journal of Statistical Software* **47**: 1–29.
URL: *http://www.jstatsoft.org/v47/i13/*

Boček, P. and Šiman, M. (2016). Directional quantile regression in octave (and matlab), *Kybernetika* **52**: 28–51.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines, *ACM Transactions on Intelligent Systems and Technology* **2**: 27:1–27:27.
URL: *Software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm*

Chaudhuri, P. (1996). On a geometric notion of quantiles for multivariate data, *Journal of the American Statistical Association* **91**: 862–872.

Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A. and Batista, G. (2015). The UCR time series classification archive.
URL: *http://www.cs.ucr.edu/˜eamonn/time_series_data/*

Cuesta-Albertos, J. A., Febrero-Bande, M. and de la Fuente, M. O. (2016). The DD^G-classifier in the functional setting, *arXiv:1501.00372* .

Cuesta-Albertos, J. A. and Nieto-Reyes, A. (2008). The random Tukey depth, *Computational Statistics and Data Analysis* **52**: 4979–4988.

Cuevas, A., Febrero, M. and Fraiman, R. (2007). Robust estimation and classification for functional data via projection-based depth notions, *Computational Statistics* **22**(3): 481–496.

Cui, X., Lin, L. and Yang, G. (2008). An extended projection data depth and its applications to discrimination, *Communications in Statistics – Theory and Methods* **37**: 2276–2290.

Delaigle, A., Hall, P. and Bathia, N. (2012). Componentwise classification and clustering of functional data, *Biometrika* **99**: 299–313.

Devroye, L., Györfi, L. and Lugosi, G. (1996). *A Probabilistic Theory of Pattern Recognition*, Springer. New York.

Donoho, D. (1982). *Breakdown Properties of Multivariate Location Estimators*, PhD thesis, Harvard University.

Donoho, D. L. and Gasko, M. (1992). Breakdown properties of location estimates based on halfspace depth and projected outlyingness, *The Annals of Statistics* **20**: 1803–1827.

Duong, T. (2007). ks: Kernel density estimation and kernel discriminant analysis for multivariate data in R, *Journal of Statistical Software* **21**: 1–16.
URL: *http://www.jstatsoft.org/v021/i07/*

Dutta, S., Chaudhuri, P. and Ghosh, A. K. (2012). Classification using localized spatial depth with multiple localization, *Mimeo* .

Dutta, S. and Ghosh, A. K. (2011). On classification based on L_p depth with an adaptive choice of p, *Technical Report R5/2011*, Statistics and Mathematics Unit, Indian Statistical Institute.

Dutta, S. and Ghosh, A. K. (2012). On robust classification using projection depth, *Annals of the Institute of Statistical Mathematics* **64**: 657–676.

Dutta, S. and Ghosh, A. K. (2015). Multi-scale classification using localized spatial depth, *arXiv:1504.03804* .

Dyckerhoff, R. (2004). Data depths satisfying the projection property, *AStA – Advances in Statistical Analysis* **88**: 163–190.

Dyckerhoff, R. and Mosler, K. (2011). Weighted-mean trimming of multivariate data, *Journal of Multivariate Analysis* **102**: 405–421.

Dyckerhoff, R., Mosler, K. and Koshevoy, G. (1996). Zonoid data depth: Theory and computation, *in* A. Prat (ed.), *COMPSTAT '96 – Proceedings in Computational Statistics*, Springer, pp. 235–240.

Dyckerhoff, R. and Mozharovskyi, P. (2016). Exact computation of the halfspace depth, *Computational Statistics and Data Analysis* **98**: 19–30.

Eddelbuettel, D., Emerson, J. W. and Kane, M. J. (2016). **BH***: Boost C++ header files*. R package version 1.60.0-2.
URL: *http://CRAN.R-project.org/package=BH*

Eddelbuettel, D., Francois, R., Allaire, J., Ushey, K., Kou, Q., Bates, D. and Chambers, J. (2016). **Rcpp***: Seamless R and C++ Integration*. R package version 0.12.5.
URL: *http://CRAN.R-project.org/package=Rcpp*

Febrero-Bande, M. and Oviedo de la Fuente, M. (2012). Statistical computing in functional data analysis: The R package **fda.usc**, *Journal of Statistical Software* **51**(4): 1–28.
URL: *http://www.jstatsoft.org/v51/i04/*

Fischer, D., Mosler, K., Möttönen, J., Nordhausen, K., Pokotylo, O. and Vogel, D. (2016). Computing the Oja median in R: The package OjaNP, *arXiv:1606.07620* .

Fraiman, R. and Meloche, J. (1999). Multivariate L-estimation, *Test* **8**: 255–317.

Friedman, J. H. (1997). On bias, variance, 0/1-loss, and the curse-of-dimensionality, *Data Mining and Knowledge Discovery* **1**: 55–77.

Genest, M., Masse, J.-C. and Plante, J.-F. (2012). **depth***: Depth functions tools for multivariate analysis*. R package version 2.0-0.
URL: *http://CRAN.R-project.org/package=depth*

Ghosh, A. K. and Chaudhuri, P. (2005a). On data depth and distribution-free discriminant analysis using separating surfaces, *Bernoulli* **11**: 1–27.

Ghosh, A. K. and Chaudhuri, P. (2005b). On maximum depth and related classifiers, *Scandinavian Journal of Statistics* **32**: 327–350.

Härdle, W., Müller, M., Sperlich, S. and Werwatz, A. (2004). *Nonparametric and Semiparametric Models*, Springer. New York.

Hayfield, T. and Racine, J. S. (2008). Nonparametric econometrics: The **np** package, *Journal of Statistical Software* **27**.
URL: *http://www.jstatsoft.org/v27/i05/*

Hettmansperger, T. P., Möttönen, J. and Oja, H. (1999). The geometry of the affine invariant multivariate sign and rank methods, *Journal of Nonparametric Statistics* **11**: 271–285.

Hlubinka, D., Kotík, L. and Vencalek, O. (2010). Weighted halfspace depth, *Kybernetika* **46**: 125–148.

Hu, Y., Wang, Y., Wu, Y., Li, Q. and Hou, C. (2011). Generalized Mahalanobis depth in the reproducing kernel Hilbert space, *Statistical Papers* **52**: 511–522.

Hubert, M. and Vakili, K. (2013). **MFHD***: Multivariate Functional Halfspace Depth*. R package version 0.0.1.
URL: *http://CRAN.R-project.org/package=MFHD*

Jörnsten, R. (2004). Clustering and classification based on the L_1 data depth, *Journal of Multivariate Analysis* **90**: 67–89.

Koltchinskii, V. (1997). M-estimation, convexity and quantiles, *The Annals of Statistics* **25**: 435–477.

Koshevoy, G. (2002). The Tukey depth characterizes the atomic measure, *Journal of Multivariate Analysis* **83**: 360–364.

Koshevoy, G. (2003). Lift-zonoid and multivariate depths, *in* R. Dutter, P. Filzmoser, U. Gather and P. Rousseeuw (eds), *Developments in Robust Statistics*, Physica-Verlag, Heidelberg, pp. 194–202.

Koshevoy, G. and Mosler, K. (1997). Zonoid trimming for multivariate distributions, *The Annals of Statistics* **25**: 1998–2017.

Kosiorowski, D., Bocian, M., Wegrzynkiewicz, A. and Zawadzki, Z. (2016). **DepthProc***: Statistical depth functions for multivariate analysis*. R package version 1.0.7.
URL: *http://CRAN.R-project.org/package=DepthProc*

Kotík, L. (2014). *Weighted halfspace depths and their properties*, PhD thesis, Charles University, Prague.

Lange, T., Mosler, K. and Mozharovskyi, P. (2014a). DDα-classification of asymmetric and fat-tailed data, *in* M. Spiliopoulou, L. Schmidt-Thieme and R. Janning (eds), *Data Analysis, Machine Learning and Knowledge Discovery*, Springer. Berlin, pp. 71–78.

Lange, T., Mosler, K. and Mozharovskyi, P. (2014b). Fast nonparametric classification based on data depth, *Statistical Papers* **55**: 49–69.

Lange, T. and Mozharovskyi, P. (2014). The alpha-procedure – a nonparametric invariant method for automatic classification of d-dimensional objects., *in* M. Spiliopoulou, L. Schmidt-Thieme and R. Janning (eds), *Data Analysis, Machine Learning and Knowledge Discovery*, Springer-Verlag, Berlin Heidelberg, pp. 79–86.

Li, J., Cuesta-Albertos, J. A. and Liu, R. Y. (2012). DD-classifier: Nonparametric classification procedure based on DD-plot, *Journal of the American Statistical Association* **107**: 737–753.

Li, Q. and Racine, J. S. (2007). *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Liu, R. Y. (1988). On a notion of simplicial depth, *Proceedings of the National Academy of Sciences* **85**: 1732–1734.

Liu, R. Y. (1990). On a notion of data depth based on random simplices, *The Annals of Statistics* **18**: 405–414.

Liu, R. Y. (1992). Data depth and multivariate rank tests, *in* Y. Dodge (ed.), L_1*-Statistical Analysis and Related Methods*, Elsevier, Amsterdam, pp. 279–294.

Liu, R. Y., Parelius, J. M. and Singh, K. (1999). Multivariate analysis by data depth: Descriptive statistics, graphics and inference, (with discussion and a rejoinder by Liu and Singh), *The Annals of Statistics* **27**: 783–858.

Liu, X. and Zuo, Y. (2014a). Computing halfspace depth and regression depth, *Communications in Statistics - Simulation and Computation* **43**: 969–985.

Liu, X. and Zuo, Y. (2014b). Computing projection depth and its associated estimators, *Statistics and Computing* **24**: 51–63.

Liu, X. and Zuo, Y. (2015). CompPD: A MATLAB package for computing projection depth, *Journal of Statistical Software* **65**: 1–21.
URL: *http://www.jstatsoft.org/v65/i02/*

Lopez-Pintado, S. and Torrente, A. (2013). **depthTools***: Depth tools package.* R package version 0.4.
URL: *http://CRAN.R-project.org/package=depthTools*

Mahalanobis, P. C. (1936). On the generalized distance in statistics, *Proceedings of the National Institute of Sciences of India* **12**: 49–55.

Mahalanobish, O. and Karmakar, S. (2015). **depth.plot***: Multivariate analogy of quantiles.* R package version 0.1.
URL: *http://CRAN.R-project.org/package=depth.plot*

Mizera, I. (2002). On depth and deep points: a calculus, *The Annals of Statistics* **30**: 1681–1736.

Mosler, K. (2002). *Multivariate Dispersion, Central Regions, and Depth: The Lift Zonoid Approach*, Springer-Verlag, New York.

Mosler, K. (2013). Depth statistics, *in* C. Becker, R. Fried and S. Kuhnt (eds), *Robustness and Complex Data Structures: Festschrift in Honour of Ursula Gather*, Springer. Berlin, pp. 17–34.

Mosler, K. and Hoberg, R. (2006). Data analysis and classification with the zonoid depth, *DIMACS. Data Depth: Robust Multivariate Analysis, Computational Geometry and Applications* pp. 49–59.

Mosler, K. and Mozharovskyi, P. (2015). Fast DD-classification of functional data, *Statistical Papers, to appear* .

Mosler, K. and Pokotylo, O. (2015). Computation of the Oja Median by Bounded Search, *in* K. Nordhausen and S. Taskinen (eds), *Modern Nonparametric, Robust and Multivariate Methods*, Springer-Verlag, pp. 185–203.

Mosler, K. and Polyakova, Y. (2012). General notions of depth for functional data, *arXiv:1208.1981* .

Mozharovskyi, P. (2015). *Contributions to Depth-based Classification and Computation of the Tukey Depth*, Verlag Dr. Kovač, Hamburg.

Mozharovskyi, P., Mosler, K. and Lange, T. (2015). Classifying real-world data with the $DD\alpha$-procedure, *Advances in Data Analysis and Classification* **9**: 287–314.

Mustafa, N., Ray, S. and Shabbir, M. (2014). **rsdepth***: Ray Shooting Depth (i.e. RS Depth) functions for bivariate analysis.* R package version 0.1-5.
URL: *http://CRAN.R-project.org/package=rsdepth*

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization, *The Computer Journal* **7**: 308–313.

Nieto-Reyes, A. and Battey, H. (2016). A topologically valid definition of depth for functional data, *Statistical Science* **31**: 61–79.

Niinimaa, A., Oja, H. and Nyblom, J. (1992). Algorithm AS 277: the Oja bivariate median, *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **41**: 611–617.

Niinimaa, A., Oja, H. and Tableman, M. (1990). The finite-sample breakdown point of the oja bivariate median and of the corresponding half-samples version, *Statistics & Probability Letters* **10**: 325–328.

Oja, H. (1983). Descriptive statistics for multivariate distributions, *Statistics and Probability Letters* **1**: 327–332.

Oja, H. (2013). Multivariate median, *in* C. Becker, R. Fried and S. Kuhnt (eds), *Robustness and Complex Data Structures: Festschrift in Honour of Ursula Gather*, Springer, pp. 3–15.

Paindaveine, D. and Van Bever, G. (2013). From depth to local depth: a focus on centrality, *Journal of the American Statistical Association* **108**: 1105–1119.

Paindaveine, D. and Van Bever, G. (2015). Nonparametrically consistent depth-based classifiers, *Bernoulli* **21**: 62–82.

Pokotylo, O. and Mosler, K. (2016). Classification with the pot-pot plot, *Statistical Papers* .

Pokotylo, O., Mozharovskyi, P. and Dyckerhoff, R. (2016). Depth and depth-based classification with R-package **ddalpha**, *arXiv:1608.04109* .

Ronkainen, T., Oja, H. and Orponen, P. (2003). Computation of the multivariate oja median, *in* R. Dutter (ed.), *Developments in robust statistics*, Springer, pp. 344–359.

Rousseeuw, P. J. and Leroy, A. M. (1987). *Robust Regression and Outlier Detection*, Wiley, New York.

Rousseeuw, P. J. and Ruts, I. (1996). Algorithm as 307: Bivariate location depth, *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **45**: 516–526.

Rousseeuw, P. J. and Van Driessen, K. (1999). A fast algorithm for the minimum covariance determinant estimator, *Technometrics* **41**: 212–223.

Scott, D. W. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization*, John Wiley & Sons.

Serfling, R. (2002). A depth function and a scale curve based on spatial quantiles, *in* Y. Dodge (ed.), *Statistical Data Analysis Based on the L_1-Norm and Related Methods*, Birkhäuser-Verlag, Basel, pp. 25–38.

Serfling, R. (2006). Depth functions in nonparametric multivariate inference, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **72**.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*, Chapman and Hall. London.

Šiman, M. and Boček, P. (2016). **modQR***: Multiple-output directional quantile regression.* R package version 0.1.1.
URL: *http://CRAN.R-project.org/package=modQR*

Small, C. G. (1997). Multidimensional medians arising from geodesics on graphs, *The Annals of Statistics* pp. 478–494.

Stahel, W. (1981). *Robust Estimation: Infinitesimal Optimality and Covariance Matrix Estimators (In German)*, PhD thesis, Swiss Federal Institute of Technology in Zurich.

Struyf, A. J. and Rousseeuw, P. J. (1999). Halfspace depth and regression depth characterize the empirical distribution, *Journal of Multivariate Analysis* **69**: 135–153.

Tukey, J. W. (1975). Mathematics and the picturing of data, *in* R. James (ed.), *Proceedings of the International Congress of Mathematicians*, Vol. 2, Canadian Mathematical Congress, pp. 523–531.

Vapnik, V. N. (1998). *Statistical Learning Theory*, Vol. 1, Wiley, New York.

Vapnik, V. N. and Chervonenkis, A. J. (1974). *Theory of Pattern Recognition (in Russian)*, Nauka, Moscow.

Vardi, Y. and Zhang, C. (2000). The multivariate L_1-median and associated data depth, *Proceedings of the National Academy of Sciences of the United States of America* **97**: 1423–1426.

Vasil'ev, V. (2003). The reduction principle in problems of revealing regularities i., *Cybernetics and Systems Analysis* **39**: 686–694.

Vasil'ev, V. and Lange, T. (1998). The duality principle in learning for pattern recognition (in russian), *Kibernetika i Vytschislitelnaya Technika* **121**: 7–16.

Vencalek, O. (2011). *Weighted Data Depth and Depth Based Discrimination*, PhD thesis, Charles University. Prague.

Vencalek, O. (2013). k-depth-nearest neighbour method and its performance on skew-normal distributons, *Acta Universitatis Palackianae Olomucensis. Facultas Rerum Naturalium. Mathematica* **52**: 121–129.

Vencalek, O. (2014). New depth-based modification of the k-nearest neighbour method, *SOP Transactions on Statistics and Analysis* **1**: 131–138.

Vencalek, O. and Pokotylo, O. (2016). Depth-weighted bayes classification, *Mimeo* .

Wand, M. P. and Jones, M. C. (1993). Comparison of smoothing parameterizations in bivariate kernel density estimation, *Journal of the American Statistical Association* **88**: 520–528.

Wolf, H. P. (2014). **aplpack***: Another Plot PACKage: stem.leaf, bagplot, faces, spin3R, plot-summary, plothulls, and some slider functions.* R package version 1.3.0.
URL: *http://CRAN.R-project.org/package=aplpack1.3.0*

Zuo, Y. and Serfling, R. (2000). General notions of statistical depth function, *The Annals of Statistics* **28**: 461–482.