Journal of Statistical Software

October 2019, Volume 91, Issue 5. doi: 10.18637/jss.v091.i05

Depth and Depth-Based Classification with R Package ddalpha

Oleksii Pokotylo
University of Cologne

Pavlo Mozharovskyi
CREST, Ensai

Rainer Dyckerhoff
University of Cologne

Abstract

Following the seminal idea of Tukey (1975), data depth is a function that measures how close an arbitrary point of the space is located to an implicitly defined center of a data cloud. Having undergone theoretical and computational developments, it is now employed in numerous applications with classification being the most popular one. The R package **ddalpha** is a software directed to fuse experience of the applicant with recent achievements in the area of data depth and depth-based classification.

ddalpha provides an implementation for exact and approximate computation of most reasonable and widely applied notions of data depth. These can be further used in the depth-based multivariate and functional classifiers implemented in the package, where the $DD\alpha$-procedure is in the main focus. The package is expandable with user-defined custom depth methods and separators. The implemented functions for depth visualization and the built-in benchmark procedures may also serve to provide insights into the geometry of the data and the quality of pattern recognition.

Keywords: data depth, supervised classification, DD-plot, outsiders, visualization, functional classification, **ddalpha**.

1. Introduction

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 lta, and thus allows for multivariate ordering of data regarding their centrality. More formally, given a data cloud $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^\top$ in $\mathbb{R}^d$, for a point $\boldsymbol{z}$ of the same space, a depth function $D(\boldsymbol{z} \mid \boldsymbol{X})$ 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. Many depth notions have

arisen during the last several 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; see Mosler (2013) for a recent survey with details on categorization and properties.

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.

1.1. Classification in the depth space

Consider the following setting for supervised classification: Given a training sample consisting of q classes $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_q$, each containing n_i, $i = 1, \ldots, q$, 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, Lin, and Yang (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, Cuesta-Albertos, and Liu (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, Mosler, and Mozharovskyi (2014b); Mozharovskyi, Mosler, and Lange (2015) it proved to be useful in practice, also in the functional setting (Mosler and Mozharovskyi 2017; Cuesta-Albertos, Febrero-Bande, and de la Fuente 2017).

The general depth-based supervised classification framework implemented in the R package **ddalpha** (Pokotylo, Mozharovskyi, and Dyckerhoff 2016) 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 $\boldsymbol{x}_i \mapsto (D(\boldsymbol{x}_i \mid \boldsymbol{X}_1), \ldots, D(\boldsymbol{x}_i \mid \boldsymbol{X}_q))$. 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, $(\boldsymbol{x}_0 \mapsto (D(\boldsymbol{x}_0 \mid \boldsymbol{X}_1), \ldots, D(\boldsymbol{x}_0 \mid \boldsymbol{X}_q)))$, 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** (Pokotylo *et al.* 2016).

1.2. 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. The R package **ddalpha** (Pokotylo *et al.* 2016) is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=ddalpha`. **ddalpha** 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 kNN-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 empirical 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, Hall, and Bathia (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 provides an interface to define custom (user-specific) 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 2019), a well known fast and widely applied library, and resorts to **Rcpp** (Eddelbuettel and François 2011) allowing for calls of R functions from C++.

1.3. 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 2019), **depth** (Genest, Masse, and Plante 2017), **localdepth** (Agostinelli, Romanazzi, and SLATEC Common Mathematical Library 2013), **fda.usc** (Febrero-Bande and Oviedo de la Fuente 2012), **rsdepth** (Mustafa, Ray, and Shabbir 2014), **depthTools** (Lopez-Pintado and Torrente 2013), **MFHD** (Hubert and Vakili 2013), **depth.plot** (Mahalanobish and Karmakar 2015), **DepthProc** (Kosiorowski and Zawadzki 2019), **WMTregions** (Bazovkin 2013), **modQR** (Šiman and Boček 2019), **OjaNP** (Fischer, Mosler, Möttönen, Nordhausen, Pokotylo, and Vogel 2018), and MATLAB (The MathWorks Inc. 2019) 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** (Liu and Zuo 2015). 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, Mosler, and Koshevoy 1996).

ddalpha not only contains a comprehensive implementation of seven most popular depth notions, and 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 article.

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** (Febrero-Bande and Oviedo de la Fuente 2012).

1.4. Outline of the article

To facilitate understanding and keep the presentation solid, the functionality of the R package **ddalpha** is illustrated through the article 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). Depth computation and depth-based classification are exemplified – through Sections 2 and 3 – on the finite-dimensional (bivariate) transform of these data, whose explanation we postpone to Section 5. In Section 2.2 considered depth notions are illustrated in this two-dimensional space (cf. Figures 1 and 2, and Figure 5, left). In Section 3, the DD-plot is constructed (Figure 5, right) where the classification is performed by the $DD\alpha$-separator. The steps of the α-procedure are illustrated in Figure 6.

The article is outlined as follows: Section 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 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 4. Section 5 addresses the classification of functional data. In Section 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.

The illustrative material and the examples have been prepared using R packages **bfp** (Sabanés Bové and Held 2011), **ggplot2** (Wickham 2016), **MASS** (Venables and Ripley 2002), **microbenchmark** (Mersmann 2018), **reshape2** (Wickham 2007), **rgl** (Adler, Murdoch *et al.* 2019), **Rmpi** (Yu 2017), **snowFT** (Sevcikova and Rossini 2017), **stringr** (Wickham 2019).

2. Data depth

This section regards depth functions. First (Section 2.1), we briefly review the concept of data depth and its fundamental properties. Then (Section 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 2.3). The section is concluded (Section 2.5) 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.

2.1. The concept

Consider a point $\boldsymbol{z} \in \mathbb{R}^d$ and a data sample $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^\top$ 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 $D(\boldsymbol{z} \mid \boldsymbol{X}) : \mathbb{R}^d \mapsto [0, 1]$ 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 $D(\boldsymbol{z} \mid \boldsymbol{X}) : \mathbb{R}^d \mapsto [0, 1]$ that is:

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

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

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

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

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

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

The first two properties state that $D(\cdot \mid \boldsymbol{X})$ 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 $D_{\max_{\boldsymbol{z} \in \mathbb{R}^d} D(\boldsymbol{z} \mid \boldsymbol{X})}(\boldsymbol{X})$ is convex. (D4) can be strengthened by requiring $D(\cdot \mid \boldsymbol{X})$ 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 $D_\alpha(\boldsymbol{X}) = \{\boldsymbol{x} \in \mathbb{R}^d : D(\boldsymbol{x} \mid \boldsymbol{X}) \geq \alpha\}$ 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 $D(\cdot \mid \boldsymbol{X})$ is additionally required).

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 1 and 2 for one class of the data set introduced in Section 1.4. The corresponding code can be found in the part *Depth visualization* of Section 6.3.

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:

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

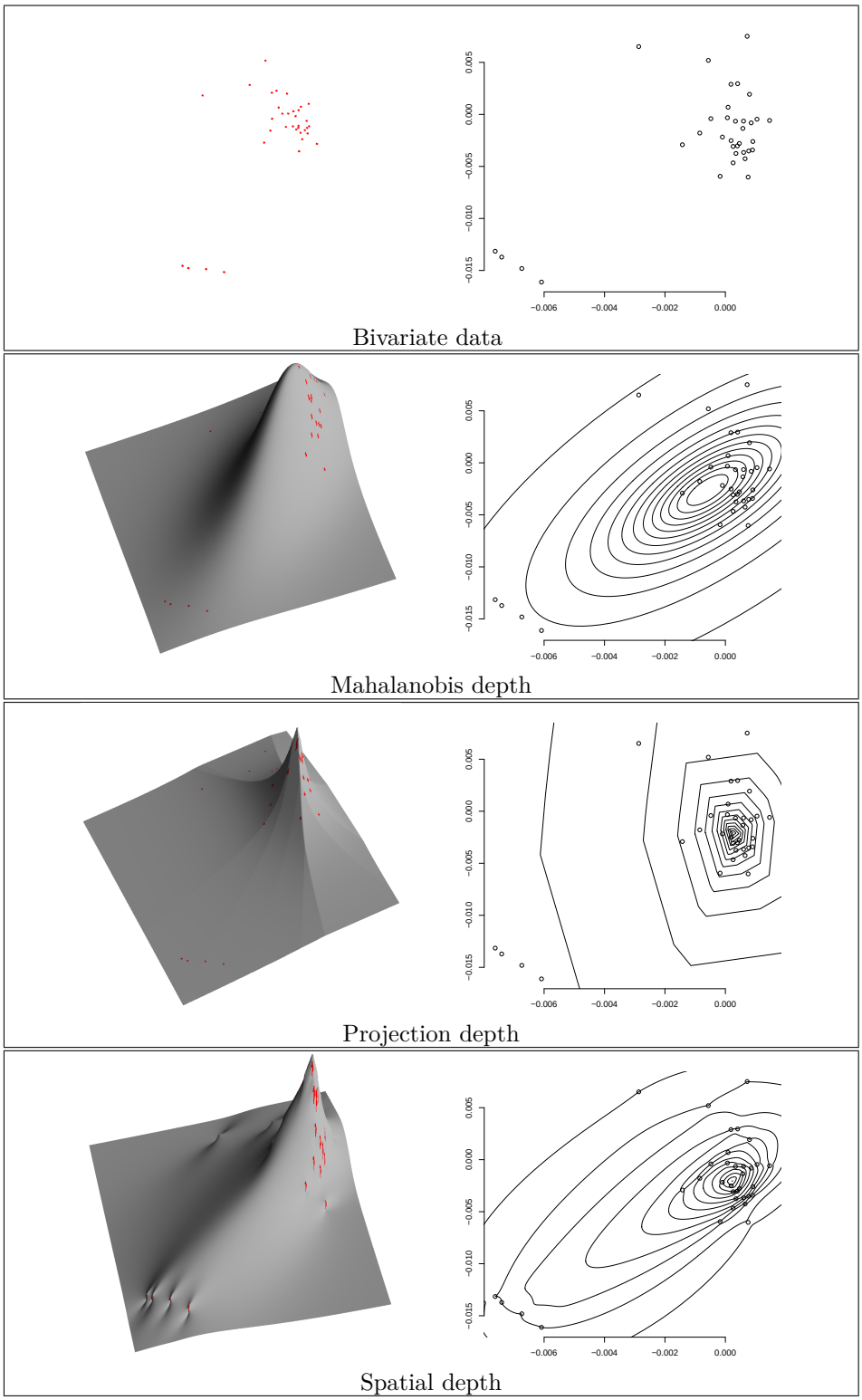

Figure 1: Depth plots and contours of bivariate data.

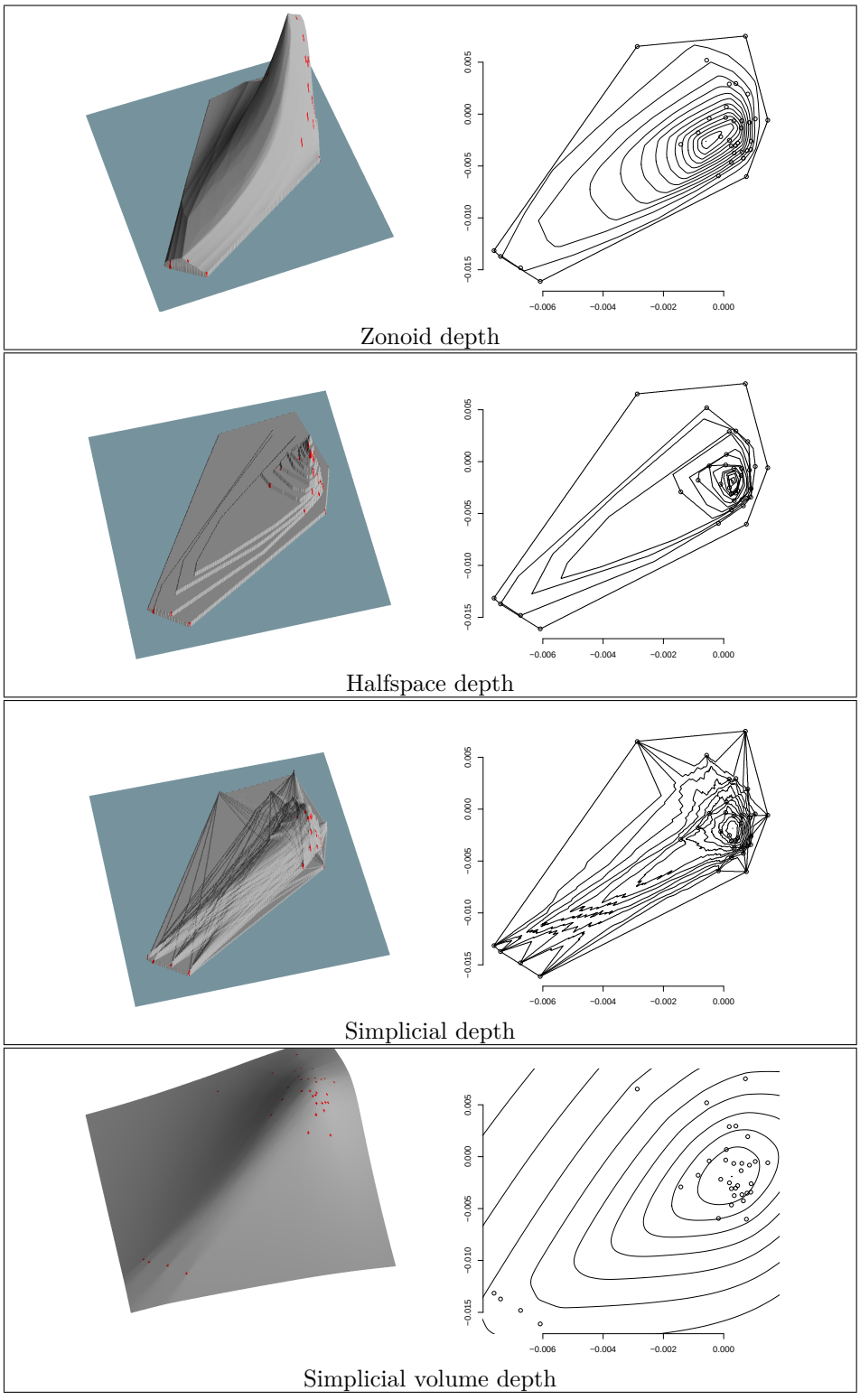

Figure 2: Depth plots and contours of bivariate data.

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

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

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 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:

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

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

$$D_{prj}(\boldsymbol{z} \mid \boldsymbol{X}) = \frac{1}{1 + o_{prj}(\boldsymbol{z} \mid \boldsymbol{X})}. \tag{2}$$

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 $\boldsymbol{z}$:

$$D_{spt}(\boldsymbol{z} \mid \boldsymbol{X}) = 1 - \left\| \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{v}(\boldsymbol{\Sigma}^{-\frac{1}{2}}(\boldsymbol{X})(\boldsymbol{z} - \boldsymbol{x}_i)) \right\|, \tag{3}$$

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

Affine invariant spatial depth satisfies postulates (D1), (D2), (D3), and (D5), but fails to satisfy (D4) (Nagy 2017) and is thus also 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:

$$D_{hs}(\boldsymbol{z} \mid \boldsymbol{X}) = \min_{\boldsymbol{u} \in S^{d-1}} \frac{1}{n} \#\{i : \boldsymbol{x}_i^\top \boldsymbol{u} \leq \boldsymbol{z}^\top \boldsymbol{u}; i = 1, \ldots, n\}. \tag{4}$$

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, $D_{hs}(\boldsymbol{z} \mid \boldsymbol{X})$ is approximated from above with time complexity $O(knd)$, and $D_{hs}(\boldsymbol{X} \mid \boldsymbol{X})$ with time complexity $O(kn(d + \log n))$, 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 $\boldsymbol{z}$:

$$D_{sim}(\boldsymbol{z} \mid \boldsymbol{X}) = \frac{1}{\binom{n}{d+1}} \sum_{1 \leq i_1 < i_2 < \ldots < i_{d+1} \leq n} I(\boldsymbol{z} \in \mathrm{conv}(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \ldots, \boldsymbol{x}_{i_{d+1}})) \tag{5}$$

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 $\boldsymbol{z}$:

$$D_{simv}(\boldsymbol{z} \mid \boldsymbol{X}) = \frac{1}{1 + \frac{1}{\binom{n}{d}\sqrt{\det\big(\boldsymbol{\Sigma}(\boldsymbol{X})\big)}} \sum_{1 \le i_1 < i_2 < \ldots < i_d \le n} \mathrm{vol}\big(\mathrm{conv}(\boldsymbol{z}, \boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \ldots, \boldsymbol{x}_{i_d})\big)} \tag{6}$$

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 $\alpha \in \left[\frac{k}{n}, \frac{k+1}{n}\right]$, $k = 1, \ldots, n-1$ the zonoid region is defined as

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

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:

$$D_{zon}(\boldsymbol{z} \mid \boldsymbol{X}) = \sup\{\alpha \in [0, 1] : \boldsymbol{z} \in Z_\alpha(\boldsymbol{X})\}, \tag{7}$$

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 above considered 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

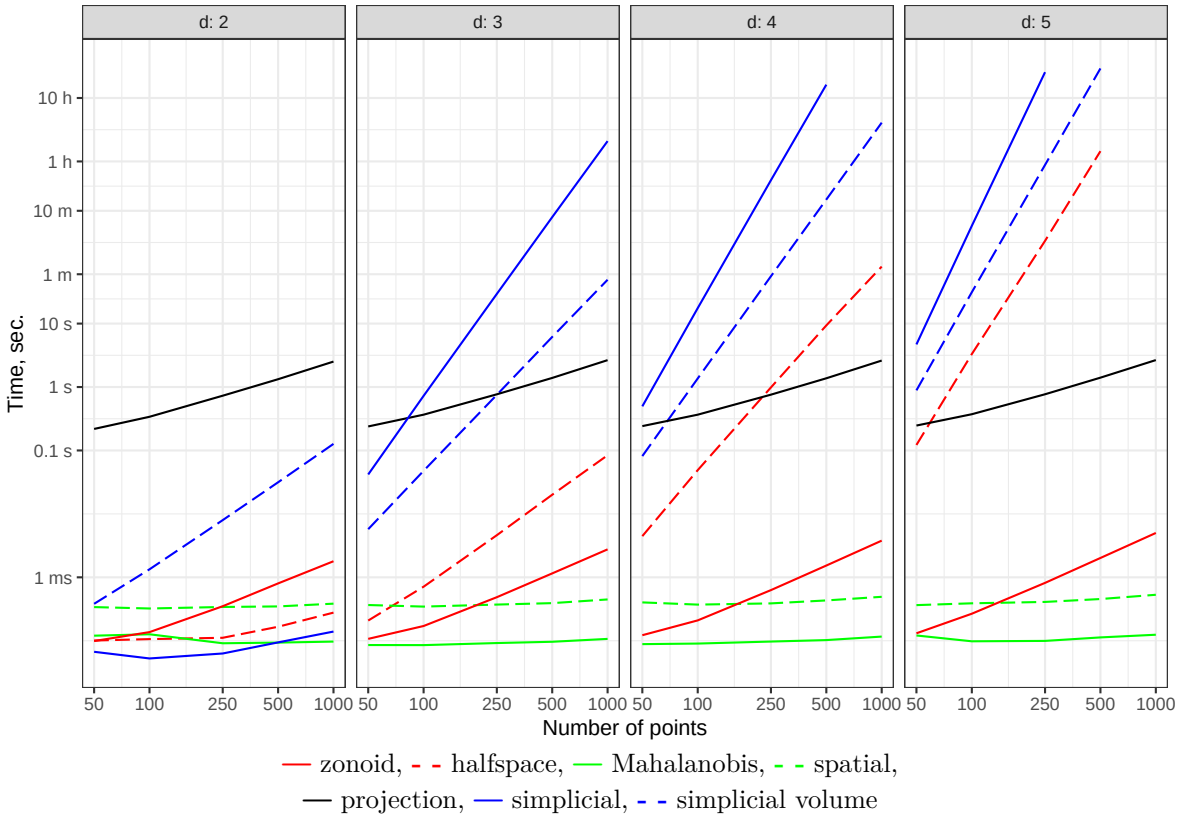

Figure 3: Calculation time of various depth functions, on the logarithmic time scale.

context, localized extension of the spatial depth (Dutta, Sarkar, and Ghosh 2016) and the data potential (Pokotylo and Mosler 2019). 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 article and relegate to the corresponding literature for theoretical and experimental results.

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. On the logarithmic time scale, the lines represent the time (in seconds) needed to compute the depth of a single point, averaged over 25 points w.r.t. 30 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 100'000 random projections, all other depths have been computed exactly. Here we used one kernel of the Macbook Pro laptop possessing processor Intel(R) Core(TM) i7-4980HQ (2.8 GHz) having enough physical memory.

One can see that, for all exactly computed depths and $n \leq 1000$, 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.

2.4. Choosing among depth notions

When employing the data depth function in multivariate statistical analysis, a notion – out of the existing variety – should be chosen by accounting for the nature of the question and application area, statistical properties of the data, available computational resources. Addressing this choice in a comprehensive way appears impossible, and doing it even in part is beyond the scope of this article. Nevertheless we hope that information provided throughout the article and the short discussion following right below will guide the practitioner.

Depending on the application of interest, when choosing a depth notion one usually deals with the trade-off between satisfaction of the depth's general statistical properties (postulates but also their extension/weakening possibly required by a particular application area), its robustness, and computational intensity. The importance of particular depth properties is mainly determined by the specific application (area). For example, affine invariance of a statistical depth function is generally attractive in multivariate analysis. On the other hand, it may make less sense if measurement units in all variables are connected in scaling, e.g., regard geometric coordinates of an object where stretching/shrinking one axis only may cause results contradicting with the reality; in this case orthogonal and translation invariance appear to be sufficient. As any general guidelines from our side would definitely be incomplete, we just refer the reader to Mosler (2013) for a discussion of the properties of the depth function from a statistical point of view.

If, on the other hand, affine invariance is required, this can be achieved by introducing the covariance matrix into the depth definition, see, e.g., Mahalanobis, spatial, and simplicial volume depths. While this is cheap computationally (see, e.g., the paragraph on Mahalanobis depth in Section 2.2), the moment estimator contradicts with robustness. Robust covariance estimators such as minimum volume ellipsoid (see, e.g., Rousseeuw and Leroy 1987) or minimum covariance determinant (already mentioned in Section 2.2) can be used instead; for the last one the authors recommend the fast algorithm developed by Rousseeuw and Van Driessen (1999) accessible, e.g., as function `covMcd` in the R package **robustbase** (Maechler *et al.* 2016). If the data set has affine dimension $< d$ ("is flat"), the algorithm cannot converge, which is usually reported by the implementation; the dimension reduction, say, by projecting data on the principal components with positive (one usually establishes a precision-defined threshold) eigenvalues should be done first. (R package **FactoMineR**, Lê, Josse, and Husson 2008, e.g., provides the necessary functionality.)

Another issue, that is often very important in practice, is the robustness of the depth function under consideration. Due to the chosen two-dimensional data possessing four outliers (in the left bottom corner), robustness of the implemented depth notions can be compared in Figures 1 and 2. For example, regard the depth contours (right part of the figures). One can see that the Mahalanobis depth with elliptical contours based on the moment estimates of mean and covariance poorly reflects the geometry of the data. Slightly better behave zonoid and simplicial volume depths, although their contours are still (substantially) distracted by the outliers, the first one being defined with weighted mean and the second one being based on volumes but also by exploiting the covariance moment estimator to achieve affine invariance. While being robust by its nature, the spatial depth (3) is sensible to outliers by involving the moment estimator of covariance for affine invariance reasons.

The three remaining notions, namely projection, halfspace, and simplicial depths, prove resistance against data pollution, while being in addition affine invariant (without being based on the covariance matrix). One can see that their (more central) depth contours are not distracted by the four outlying points. This comes at a high computational cost: exponential in (n, d) algorithmic time complexities are reflected visually by rapidly mounting straight lines in Figure 3 (projection depth is approximated, see Liu and Zuo 2014b, for the times of exact computation). For this reason it seems reasonable to approximate them, which is implemented in the R package **ddalpha**, too. Nevertheless properties of such approximations are not well studied yet, but due to their usefulness deserve future exploration.

As an example of an application revealing the properties of different depth notions we suggest a short comparative simulation study of the naïve maximum depth classifier in Section 2.5 right below, which will also serve as an introductory bridge to the subsequent material of this article. Additionally, we refer the reader to the recent work by Vencalek (2017) for a similar discussion on multivariate classification.

2.5. 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):

$$class(\boldsymbol{x}_0) = \underset{i \in \{1,\dots,q\}}{\operatorname{argmax}} \; D(\boldsymbol{x}_0 \mid \boldsymbol{X}_i). \tag{8}$$

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 4. The testing observations were sampled inside the convex hull of the training set. The problem of outsiders is addressed in Section 4. For $n = 1000$, experiments have not been conducted with the simplicial depth due to high computation time.

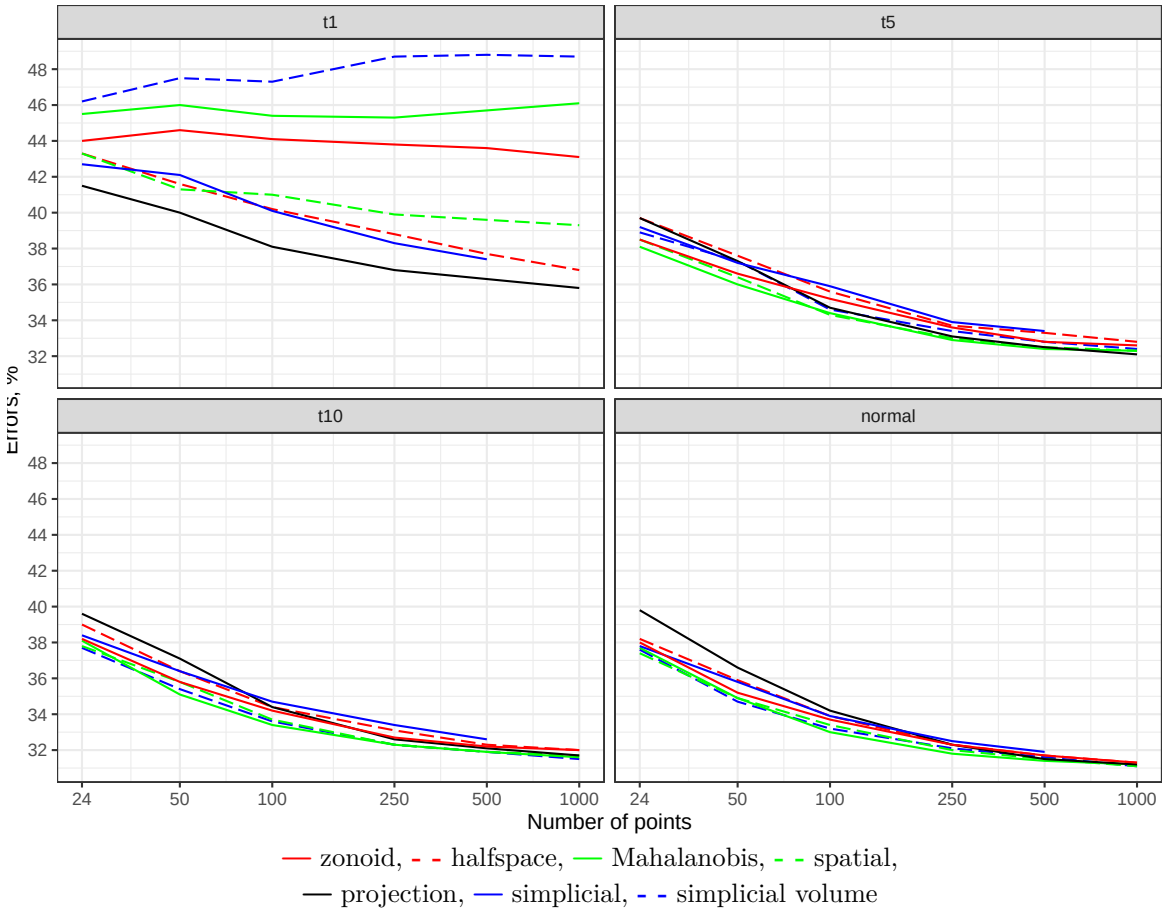

Figure 4: Average error rates of the maximum depth classifier with different data depths. The samples are simulated from the Student-t distribution possessing 1, 5, 10, and ∞ degrees of freedom.

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.

3. Classification in the DD-plot

In Section 2.5, 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 $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_q$, the depth space

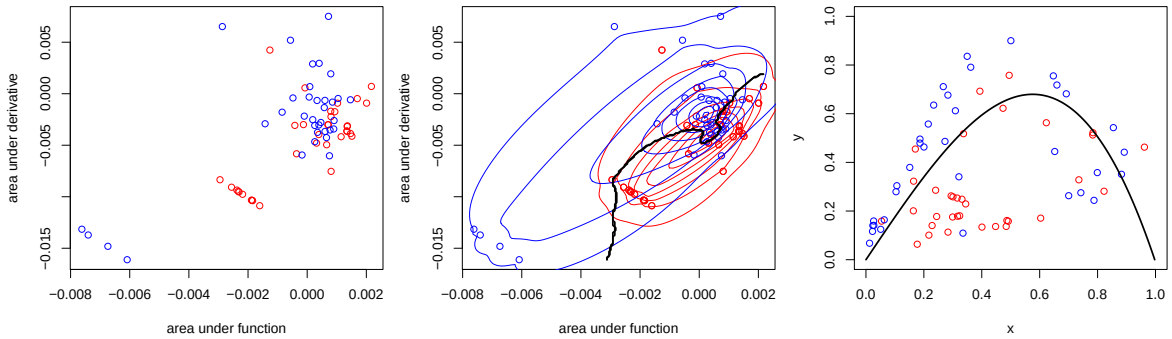

Figure 5: The discretized space (left), the depth contours with the separating rule (middle) and the DD-plot with the separating line in it (right), using spatial depth. Here we denote the depth of a point w.r.t. red and blue classes by x and y, respectively.

is constructed by applying the mapping $\mathbb{R}^d \to [0,1]^q : \boldsymbol{x} \mapsto (D(\boldsymbol{x} \mid \boldsymbol{X}_1), \ldots, D(\boldsymbol{x} \mid \boldsymbol{X}_q))$ 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 3.1, 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 3.2 we discuss application of alternative techniques in the depth space.

3.1. The $DD\alpha$-separator

The $DD\alpha$-separator is an extension of the α-procedure to the depth space, see Vasil'ev (2003); Vasil'ev and Lange (1998), also Lange and Mozharovskyi (2014). 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. The detailed algorithm is stated right below.

Regard the two-class sample illustrated on Figure 5, left, representing discretizations of the electrocardiogram curves. Explanation of the data is given in Section 1.4, we postpone the explanation of the discretization scheme till Section 5 and consider a binary classification in the DD-plot for the moment. Figure 5, middle, represents the depth contours of each class computed using the spatial depth. The DD-plot is obtained as a depth mapping $(\boldsymbol{X}_1, \boldsymbol{X}_2) \mapsto \boldsymbol{Z} = \{\boldsymbol{z}_i = (D_{i,1}, D_{i,2}),\ i = 1, \ldots, n_1 + n_2\}$, when the first class is indexed by $i = 1, \ldots, n_1$ and the second by $i = n_1 + 1, \ldots, n_2$, and writing $D_{spt}(\boldsymbol{x}_i \mid \boldsymbol{X}_1)$ (respectively $D_{spt}(\boldsymbol{x}_i \mid \boldsymbol{X}_2)$) 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. In the R package **ddalpha** we employ the polynomial kernel. Truncated series or another finitized basis of general reproducing kernel Hilbert spaces can be used alternatively. For the polynomial kernel we choose the degree of space extension (an integer > 0) 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.

Given the degree p of the polynomial extension, the dimension of the resulting extended depth space equals $r = \binom{p+q}{q} - 1$ (by default, we choose p among $\{1, 2, 3\}$ using 10-fold cross-validation). This extended depth space serves as the input to the $DD\alpha$-separator. Continuing the example, i.e., with $q = 2$, and taking $p = 3$, one gets the extended depth space $\boldsymbol{Z}^{(p)}$ of dimension $r = 9$ consisting of the observations $\boldsymbol{z}_i^{(p)} = (D_{i,1},\ D_{i,2},\ D_{i,1}^2,\ D_{i,1} \times D_{i,2},\ D_{i,2}^2,\ D_{i,1}^3,\ D_{i,1}^2 \times D_{i,2},\ D_{i,1} \times D_{i,2}^2,\ D_{i,2}^3) \in \mathbb{R}^9$ for $i = 1, \ldots, n_1 + n_2$.

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 $\boldsymbol{Z}^{(k,l)} = \{\boldsymbol{z}_i^{(k,l)} \mid \boldsymbol{z}_i^{(k,l)} = (\boldsymbol{z}_{ik}^{(p)}, \boldsymbol{z}_{il}^{(p)}),\ i = 1, \ldots, n_1+n_2\}$ 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

$$\alpha_1^{(k,l)} \in \operatorname*{argmin}_{\alpha \in [0;2\pi)} \Delta^{(k,l)}(\alpha) \tag{9}$$

with

$$\Delta^{(k,l)}(\alpha) = \sum_{i=1}^{n_1} I(\boldsymbol{z}_{ik}^{(p)} \cos\alpha - \boldsymbol{z}_{il}^{(p)} \sin\alpha < 0) + \sum_{i=n_1+1}^{n_1+n_2} I(\boldsymbol{z}_{ik}^{(p)} \cos\alpha - \boldsymbol{z}_{il}^{(p)} \sin\alpha > 0). \tag{10}$$

For the regarded example, this is demonstrated in Figure 6 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 whole data set is collapsed to a line, as shown in Figure 6. Finally, a triplet is chosen:

$$(\alpha_1^{(k^*,l^*)}, k^*, l^*) \in \operatorname*{argmin}_{1 \le k < l \le r,\ \alpha \in [0;2\pi)} \Delta^{(k,l)}(\alpha), \tag{11}$$

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

$$\boldsymbol{z}^{(1^*)} = \{z_i \mid z_i = \boldsymbol{z}_{ik^*}^{(p)} \cos\alpha_1^{(k^*,l^*)} - \boldsymbol{z}_{il^*}^{(p)} \sin\alpha_1^{(k^*,l^*)},\ i = 1, \ldots, n_1 + n_2\}, \tag{12}$$

— 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 $\boldsymbol{Z}^{((s-1)^*,k)} = \{\boldsymbol{z}_i^{((s-1)^*,k)} \mid \boldsymbol{z}_i^{((s-1)^*,k)} = (\boldsymbol{z}_i^{((s-1)^*)}, \boldsymbol{z}_{ik}^{(p)}),\ i = 1, \ldots, n_1 + n_2\}$ 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 (9) and (10). Out of all considered k, the one minimizing $\Delta^{((s-1)^*,k)}(\alpha_s^{(k)})$ is chosen, as in (11), and the corresponding $\boldsymbol{Z}^{((s-1)^*,k)}$ is convoluted to $\boldsymbol{z}^{(s^*)}$, as in (12). The second part of Figure 6 illustrates a possible second step of the algorithm.

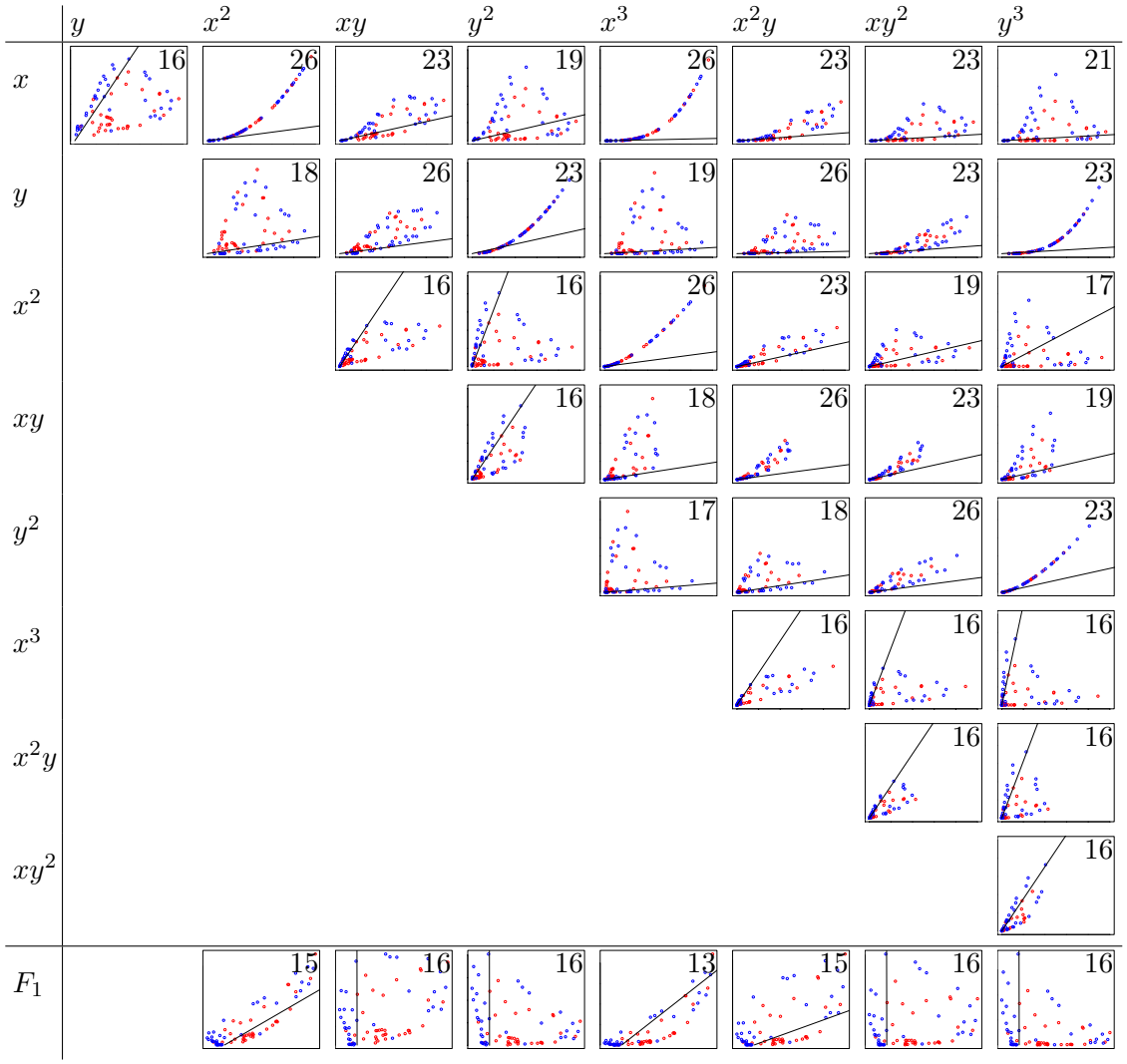

Figure 6: The steps of the α-procedure. The number of errors is shown in the right top corner of each plot. Here we denote the depth of a point w.r.t. red and blue classes by x and y, respectively. The two-dimensional spaces are shown for each pair of properties. On the first step all pairs of properties 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: $\tilde{\boldsymbol{X}} = \{\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n\}$, $\tilde{\boldsymbol{x}}_i \in \mathbb{R}^d$,
$\{y_1, \ldots, y_n\}$, $y_i \in \{-1, 1\}$ for all $i = 1, \ldots, m = m_{-1} + m_{+1}$.

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

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 $i = 1, \ldots, d$. 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
   $m_{l-} = \sharp\{i : \boldsymbol{p}_i.y = -1, \boldsymbol{p}_i.x \leq 0\}$,
   $m_{l+} = \sharp\{i : \boldsymbol{p}_i.y = +1, \boldsymbol{p}_i.x \leq 0\}$
   (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 $\boldsymbol{r}$.

Procedure GetMinError

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

1. Obtain angles:

(a) Calculate $\alpha_i = \arctan \frac{x_i}{f_j}$, $i = 1, \ldots, n$, 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: $O(\sum_{i=1}^q n_i \log(\sum_{i=1}^q n_i))$.

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., Mozharovskyi *et al.* (2015, Section 5.2).

For theoretical results on the $DD\alpha$-procedure the reader is referred to Lange *et al.* (2014b, Section 4). 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, Mosler, and Mozharovskyi (2014a) conduct a simulation study on asymmetric and heavy-tailed distributions.

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 (kNN) 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 Li *et al.* (2012, Sections 4 and 5). For a simulation comparison of the polynomial rule in the DD-plot and the $DD\alpha$-separator see Lange *et al.* (2014b, Section 5).

In his PhD-thesis, Vencalek (2011) suggests to perform the kNN 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 Vencalek (2011, Sections 3.4.3 and 3.7), respectively. It is worth to notice that the kNN-separator has another advantage – it is directly extendable to more than two classes.

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 7 showing scatter and DD-plot for the training sample of random vectors X_1 ("red") and X_2 ("blue") distributed normally with means $\mu_{X_1} = [0, 0]$ and $\mu_{X_2} = [1, 1]$, and covariances $\Sigma_{X_1} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 4 \end{smallmatrix}\right]$ and $\Sigma_{X_2} = \left[\begin{smallmatrix} 4 & 4 \\ 4 & 16 \end{smallmatrix}\right]$, 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 poorly tuned kNN (e.g., 1NN). The suggestion thus is to apply an additional fast classifier to the outsiders.

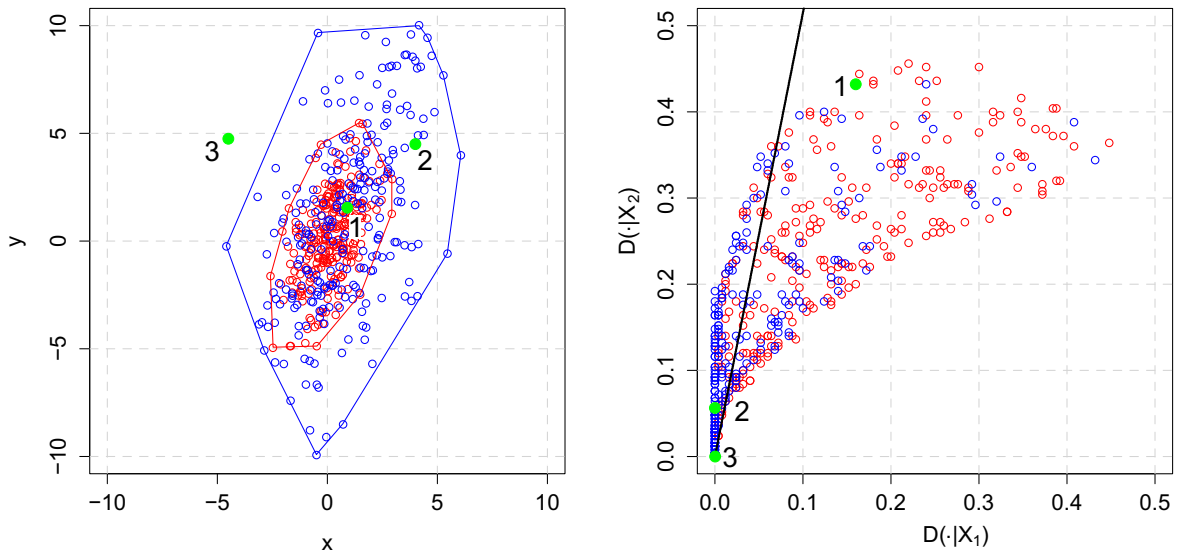

Figure 7: Points to be classified (green) in the original (left) and depth (right) space.

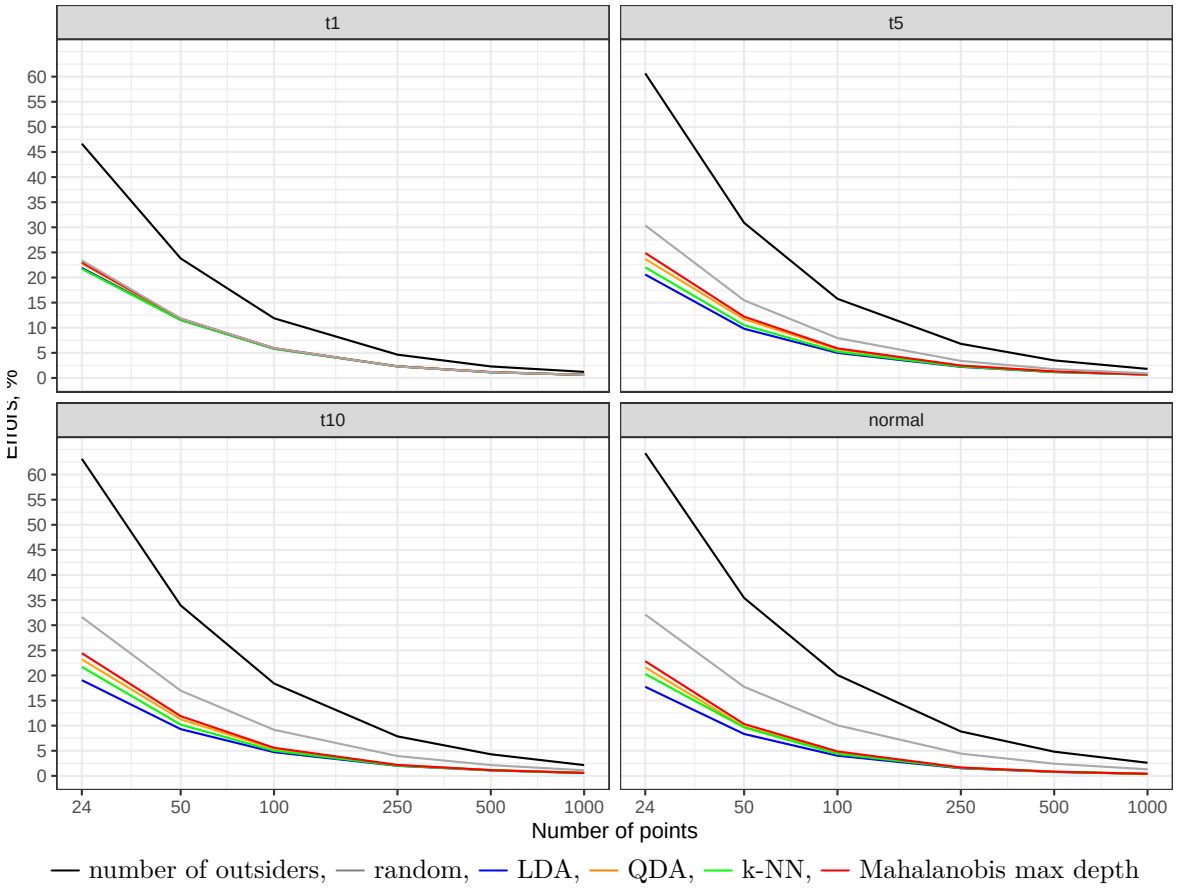

Figure 8: Error rates of various outsiders treatment. Only outsiders are classified.

The R package **ddalpha** implements a number of outsider treatments: linear (LDA) and quadratic (QDA) discriminant analysis, kNN, 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 2.5, we contrast these treatments in Figure 8, 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 treatments becomes negligible. For an extensive comparative study of different outsider treatments the reader is referred to Mozharovskyi *et al.* (2015), see also Lange *et al.* (2014b).

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.

5. An extension to functional data

Similar to Section 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: $\boldsymbol{\mathcal{F}}_1 = \{f_1, \ldots, f_{n_1}\}$ and $\boldsymbol{\mathcal{F}}_2 = \{f_1, \ldots, f_{n_2}\}$, again indexing observations by $i = 1, \ldots, n_1, n_1+1, \ldots, n_1+n_2$ 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 $(\boldsymbol{\mathcal{F}}_1, \boldsymbol{\mathcal{F}}_2) \mapsto \boldsymbol{Z} = \{\boldsymbol{z}_i = (D(f_i \mid \boldsymbol{\mathcal{F}}_1), D(f_i \mid \boldsymbol{\mathcal{F}}_2)),\ i = 1, \ldots, n_1 + n_2\}$ similar to that in Section 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.* (2017) 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 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 (2017). 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 $\tilde{\boldsymbol{f}}_i = [f_i(t_{i1}), f_i(t_{i2}), \ldots, f_i(t_{iN_i})]$, which are the measurements at ordered (time) points $t_{i1} < t_{i2} < \ldots < t_{iN_i}$, $i = 1, \ldots, n_1 + n_2$, 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:

$$\hat{f}_i \mapsto \boldsymbol{x}_i = \Big[\int_0^{T/L} \hat{f}_i(t)dt, \ldots, \int_{T(L-1)/L}^{T} \hat{f}_i(t)dt, \int_0^{T/S} \hat{f}_i'(t)dt, \ldots, \int_{T(S-1)/S}^{T} \hat{f}_i'(t)dt\Big], \quad (13)$$

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.

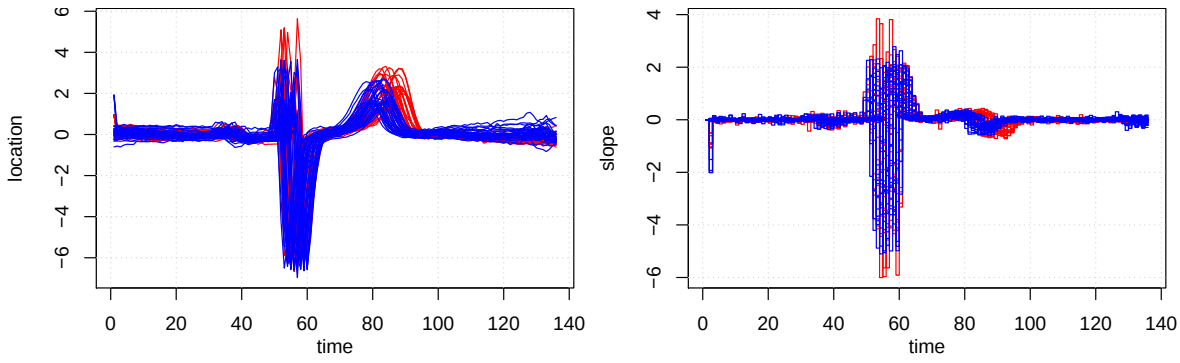

Figure 9: ECG Five Days data (left) and their derivatives (right).

The set of considered (L, S)-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 $d_{min} \leq L + S \leq d_{max}$. Cross-validation is then used to choose the best (L, S)-pair. **ddalpha** suggests to reduce the set of cross-validated (L, S)-pairs by employing the Vapnik-Chervonenkis bound. The idea behind is that, while being conservative, the bound can still provide insightful ordering of the (L, S)-pairs, especially in the case when the empirical risk and the bound have the same order of magnitude.

Given a set of pairs $\mathcal{S} = \{(l_i, s_i) \mid i = 1, \ldots, N_{ls}\}$, for each its element calculate the Vapnik-Chervonenkis bound (see Mosler and Mozharovskyi 2017, for this particular derivation)

$$b_i^{VC} = \epsilon\left(\boldsymbol{c}, \hat{\boldsymbol{\mathcal{F}}}_1^{(l_i,s_i)}, \hat{\boldsymbol{\mathcal{F}}}_2^{(l_i,s_i)}\right) + \sqrt{\frac{\ln 2 \sum_{k=0}^{l_i+s_i-1} \binom{n_1+n_2-1}{k} - \ln \eta}{2(n_1+n_2)}}, \tag{14}$$

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 (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 (Section 1.4), the functions' levels and slopes are shown in Figure 9; the LS-representation is selected by reduced cross-validation due to (13) having $(L, S) = (1, 1)$, and is depicted in Figure 5, left.

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 10. The current section describes the functionality of the package using code fragments. Section 6.5 contains comprehensive classification examples.

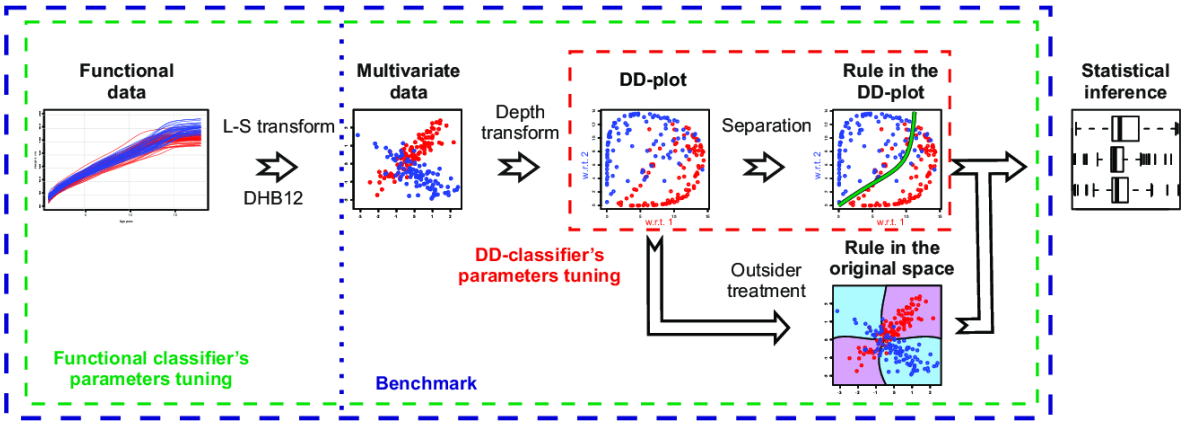

Figure 10: The structure of the package.

6.1. Basic functionality

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

Data depth is calculated by calling

```
depths <- depth.(x, data, notion = <depthName>, ...)
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`. These values, constituting a coordinate of the depth space, can be further used as a new feature of `x`, which is calculated taking into account information from the entire data set `data`. Most of the depth functions possess both exact and approximate versions that are toggled with parameters `exact` and `method`, see Table 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 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 (`https://sites.google.com/site/tijahbus/home`). 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, the entire multivariate depth representation of a data set consisting of several (labeled) classes can be calculated at once by

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

Depth	Exact	Approximate	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	+		

Table 1: Implemented depth algorithms.

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. This matrix can be seen (and further used) as additional information about `x` gained considering at once the entire data set `data`.

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 the `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 10.

```
ddalpha <- ddalpha.train(formula, data, subset, depth = "halfspace",
  separator = "alpha", outsider.methods = "LDA", outsider.settings = NULL,
  aggregation.method = "majority", use.convex = FALSE, seed = 0, ...)
classes <- ddalpha.classify(ddalpha, objects, subset, outsider.method,
  use.convex)
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 parameter `subset`. The notion of the depth function and the DD-separator are specified with 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 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 kNN (`"kNNAff"`), i.e., kNN 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 2017) and `compclassf.train` implementing componentwise classification (Delaigle *et al.* 2012).

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

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 se-

lected 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 used part of the observations may be additionally specified with parameter `subset` as in `ddalpha.train`.

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 (2017) in detail. The optimal values of L and S are stored in the *ddalphaf* object, that is returned from `ddalphaf.train`.

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 2 and 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

`.<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>, ...)`	

Table 2: Definition of a custom depth function.

case if the custom method is binary, the package takes care of the voting procedures, and 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

`.<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 = FALSE` 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>, ...)`

Table 3: Definition of a custom separator.

previously 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 = FALSE` from the validator, otherwise the method will be treated as a binary one, as by default `aggregation.method = "majority"`.

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:

```
ddalpha.test(learn, test, ...)
ddalpha.getErrorRateCV(data, numchunks = 10,  ...)
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:

```
ddalphaf.test(learn, learnlabels, test, testlabels, disc.type, ...)
ddalphaf.getErrorRateCV(dataf, labels, numchunks, disc.type, ...)
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.

Localized spatial depth and a classifier based on it, proposed by Dutta *et al.* (2016), 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) denoting $\boldsymbol{t}_i = \boldsymbol{\Sigma}^{-\frac{1}{2}}(\boldsymbol{X})(\boldsymbol{z}-\boldsymbol{x}_i)$

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

The local version is obtained by kernelizing the distances

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

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 potential-potential (pot-pot) plot (Pokotylo and Mosler 2019) 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)
+  }
```

The depth-based kNN (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

```
depth.graph(data, depth_f, main, xlim, ylim, zlim, xnum, ynum,
  theta, phi, bold = FALSE, ...)
```

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:

```
depth.contours(data, depth, main, xlab, ylab, drawplot = TRUE,
  frequency = 100, levels = 10, col, ...)
depth.contours.ddalpha(ddalpha, main, xlab, ylab, drawplot = TRUE,
  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 1 and 2 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:

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

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 are included in the package **ddalpha** to make the empirical comparison of different classifiers and data depths easier. 50 multivariate binary classification problems were collected and described by Mozharovskyi *et al.* (2015) and are also available at the web page https://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, Febrero, and Fraiman (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 S3 functions `plot.functional`, `lines.functional` and `points.functional`.

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 `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 ($0 <$ `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 2019), the classifiers' error rates grow in the following order: $DD\alpha$, polynomial classifier, kNN; although $DD\alpha$ and the polynomial classifier provide similar polynomial solutions and kNN 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)
+  }
```

6.5. Classification examples

In this section we present two examples of real data classification with the R package **ddalpha**.

In *the first example*, we train the `ddalpha` classifier using the Blood Transfusion Service Center Data Set (Yeh, Yang, and Ting 2009), see Figure 11 for the DD-plot obtained with the code fragment below. The data has been taken from the Blood Transfusion Service Center in Hsin-Chu City in Taiwan. It contains 748 donors that have three properties: Recency – months since last donation, Frequency – total number of donation, Monetary – total blood donated in c.c.. The class variable indicates whether the donor gave blood in March 2007. Following Li *et al.* (2012), we have removed the correlated property Time – months since first donation.

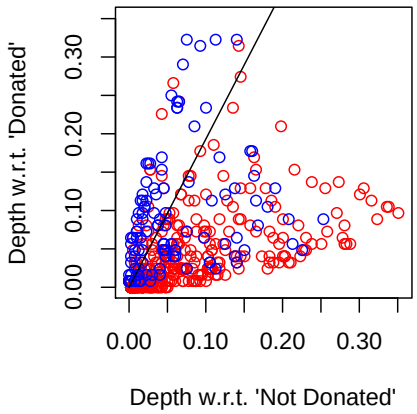

Figure 11: DD-plot of the Blood Transfusion data.

```
R> library("ddalpha")
R> data <- getdata("bloodtransfusion")
R> N <- nrow(data)
R> D <- ncol(data)
R> Ntrain <- 0.7 * N
R> Ntest <- N - Ntrain
R> learnSample <- sample(1:N, Ntrain)
```

We train the $DD\alpha$-classifier using:

```
R> ddalpha <- ddalpha.train(C ~ ., data, subset = learnSample,
+    depth = "halfspace", separator = "alpha",
+    outsider.methods = "kNNAff")

Selected columns:  C,   Recency,   Frequency,   Monetary
```

We draw the separation on the DD-plot and classify by means of the $DD\alpha$-classifier:

```
R> draw.ddplot(ddalpha, main = "", xlab = "Depth w.r.t. 'Not Donated'",
+    ylab = "Depth w.r.t. 'Donated'")
R> classes <- ddalpha.classify(ddalpha, data, subset = -learnSample)
R> cat("Classification error rate:",
+    sum(unlist(classes) != data[-learnSample, D]) / Ntest, "\n")

Classification error rate: 0.2228164
```

The second example illustrates the usage of the functional DD-classifier applying it to the Berkeley Growth Study Data (Tuddenham and Snyder 1954). The data set contains the heights of 39 boys and 54 girls of age from 1 to 18 and the ages at which they were collected. The measurements were done at 31 not equally spaced ages, see Figure 12 for the data itself and the DD-plot obtained with the code fragment below.

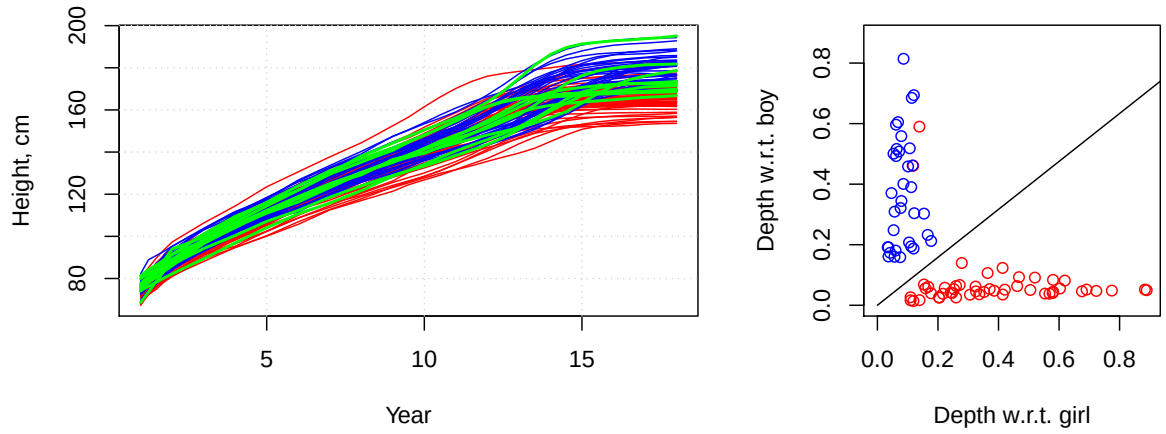

Figure 12: Berkeley Growth Study Data (left) and its DD-plot (right). Girls (red), boys (blue), classified observations (green).

Here we cross-validate all possible pairs of L and S, $L + S \leq$ `maxNumIntervals`. Having additional information about the functions, the user can explicitly specify reasonable (L, S)-pairs for cross-validation. For example, by examining the Growth data, one may notice that after 13 years boys start growing faster than girls, and the derivatives differ between the classes in the second part of the function, so the classes shall be well separated by setting $S = 2$. We load the package and the Growth dataset:

```
R> library("ddalpha")
R> dataf <- dataf.growth()
```

We take a sample as test data consisting of elements 50 to 59, which are 5 girls and 5 boys, and plot the data with colors assigned in alphabetical order of class labels:

```
R> testIndexes <- 50:59
R> plot(dataf, main = "", xlab = "Year", ylab = "Height, cm",
+    colors = c("blue", "red"))
R> testData <- structure(list(dataf = dataf$dataf[testIndexes]),
+    class = "functional")
R> lines(testData, colors = "green", lwd = 2)
```

We train the functional classifier using cross-validation over all variants up to dimension 3:

```
R> ddalphaf <- ddalphaf.train(dataf$dataf, dataf$labels,
+    subset = -testIndexes, classifier.type = "ddalpha",
+    depth = "spatial", maxNumIntervals = 3)
```

We draw the separation on the DD-plot and classify by means of the $DD\alpha$-classifier:

```
R> draw.ddplot(ddalphaf$classifier, main = "",
+    xlab = paste("Depth w.r.t.", ddalphaf$labels[[1]]),
+    ylab = paste("Depth w.r.t.", ddalphaf$labels[[2]]))
```

```
R> classified1 <- ddalphaf.classify(ddalphaf, dataf$dataf,
+    subset = testIndexes)
R> print(unlist(classified1))

[1] "girl" "boy"  "girl" "girl" "girl" "boy"  "boy"  "boy"  "boy"  "boy"

R> print(ddalphaf$adc.args)

$instance
[1] "avr"

$numFcn
[1] 1

$numDer
[1] 2
```

We cross-validate the sample to get error rate and running time for particular parameters:

```
R> ddalphaf.getErrorRateCV(dataf$dataf, dataf$labels,
+    disc.type = "LS", classifier.type = "ddalpha", depth = "spatial",
+    adc.args = list(instance = "avr", numFcn = 1, numDer = 2))

$errors
[1] 0.05376344

$time
[1] 0.212

$time_sd
[1] 0.02250926
```

Acknowledgments

The authors highly appreciate the help of Karl Mosler consisting in numerous remarks on earlier versions of this article and his notable contributions to the field of data depth. The authors want to thank Julie Josse, François Husson, Myriam Vimond, and Pierre Lafaye de Micheaux for their valuable suggestions that have substantially improved the present work. Stanislav Nagy is acknowledged for having pointed out absence of the monotonicity property for spatial depth. Thanks also go to Associate Editor Max Kuhn and the two anonymous referees. The authors would like to express their gratitude to the Cologne Graduate School of Management, Economics and Social Sciences who supported the work of Oleksii Pokotylo and to the Lebesgue Centre of Mathematics who supported the work of Pavlo Mozharovskyi (program PIA-ANR-11-LABX-0020-01). HPC cluster CHEOPS of the University of Cologne is acknowledged for providing necessary computational resources.

References

Adler D, Murdoch D, *et al.* (2019). ***rgl**: 3D Visualization Using OpenGL*. R package version 0.100.24, URL https://CRAN.R-project.org/package=rgl.

Agostinelli C, Romanazzi M, SLATEC Common Mathematical Library (2013). ***localdepth**: Local Depth*. R package version 0.5-7, URL https://CRAN.R-project.org/src/contrib/Archive/localdepth/.

Bazovkin P (2013). ***WMTregions**: Exact Calculation of WMTR*. R package version 3.2.6, URL https://CRAN.R-project.org/src/contrib/Archive/WMTregions.

Bazovkin P, Mosler K (2012). "An Exact Algorithm for Weighted-Mean Trimmed Regions in Any Dimension." *Journal of Statistical Software*, **47**(13), 1–29. doi:10.18637/jss.v047.i13.

Boček P, Šiman M (2016). "Directional Quantile Regression in Octave (and MATLAB)." *Kybernetika*, **52**(1), 28–51. doi:10.14736/kyb-2016-1-0028.

Chaudhuri P (1996). "On a Geometric Notion of Quantiles for Multivariate Data." *Journal of the American Statistical Association*, **91**(434), 862–872. doi:10.2307/2291681.

Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen A, Batista G (2015). "The UCR Time Series Classification Archive." www.cs.ucr.edu/~eamonn/time_series_data/.

Cuesta-Albertos JA, Febrero-Bande M, de la Fuente MO (2017). "The DD^G-Classifier in the Functional Setting." *TEST*, **26**(1), 119–142. doi:10.1007/s11749-016-0502-6.

Cuesta-Albertos JA, Nieto-Reyes A (2008). "The Random Tukey Depth." *Computational Statistics & Data Analysis*, **52**(11), 4979–4988. doi:10.1016/j.csda.2008.04.021.

Cuevas A, Febrero M, Fraiman R (2007). "Robust Estimation and Classification for Functional Data via Projection-Based Depth Notions." *Computational Statistics*, **22**(3), 481–496. doi:10.1007/s00180-007-0053-0.

Cui X, Lin L, Yang G (2008). "An Extended Projection Data Depth and Its Applications to Discrimination." *Communications in Statistics – Theory and Methods*, **37**, 2276–2290. doi:10.1080/03610920701858396.

Delaigle A, Hall P, Bathia N (2012). "Componentwise Classification and Clustering of Functional Data." *Biometrika*, **99**(2), 299–313. doi:10.1093/biomet/ass003.

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

Donoho DL, Gasko M (1992). "Breakdown Properties of Location Estimates Based on Halfspace Depth and Projected Outlyingness." *The Annals of Statistics*, **20**(4), 1803–1827. doi:10.1214/aos/1176348890.

Dutta S, Ghosh AK (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, Ghosh AK (2012). "On Robust Classification Using Projection Depth." *The Annals of the Institute of Statistical Mathematics*, **64**(3), 657–676. doi:10.1007/s10463-011-0324-y.

Dutta S, Sarkar S, Ghosh AK (2016). "Multi-Scale Classification Using Localized Spatial Depth." *Journal of Machine Learning Research*, **17**(218), 1–30.

Dyckerhoff R (2004). "Data Depths Satisfying the Projection Property." *Allgemeines Statistisches Archiv*, **88**(2), 163–190. doi:10.1007/s101820400167.

Dyckerhoff R, Mosler K (2011). "Weighted-Mean Trimming of Multivariate Data." *Journal of Multivariate Analysis*, **102**(3), 405–421. doi:10.1016/j.jmva.2010.10.002.

Dyckerhoff R, Mosler K, Koshevoy G (1996). "Zonoid Data Depth: Theory and Computation." In A Prat (ed.), *COMPSTAT '96 – Proceedings in Computational Statistics*, pp. 235–240. Springer-Verlag. doi:10.1007/978-3-642-46992-3_26.

Dyckerhoff R, Mozharovskyi P (2016). "Exact Computation of the Halfspace Depth." *Computational Statistics & Data Analysis*, **98**, 19–30. doi:10.1016/j.csda.2015.12.011.

Eddelbuettel D, Emerson JW, Kane MJ (2019). ***BH**: Boost C++ Header Files*. R package version 1.69.0-1, URL https://CRAN.R-project.org/package=BH.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Febrero-Bande M, 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. doi:10.18637/jss.v051.i04.

Fischer D, Mosler K, Möttönen J, Nordhausen K, Pokotylo O, Vogel D (2018). ***OjaNP**: Multivariate Methods Based on the Oja Median and Related Concepts*. R package version 0.9-12, URL https://CRAN.R-project.org/package=OjaNP.

Genest M, Masse JC, Plante JF (2017). ***depth**: Nonparametric Depth Functions for Multivariate Analysis*. R package version 2.1-1, URL https://CRAN.R-project.org/package=depth.

Ghosh AK, Chaudhuri P (2005a). "On Data Depth and Distribution-Free Discriminant Analysis Using Separating Surfaces." *Bernoulli*, **11**(1), 1–27. doi:10.3150/bj/1110228239.

Ghosh AK, Chaudhuri P (2005b). "On Maximum Depth and Related Classifiers." *Scandinavian Journal of Statistics*, **32**(2), 327–350. doi:10.1111/j.1467-9469.2005.00423.x.

Hubert M, Vakili K (2013). ***MFHD**: Multivariate Functional Halfspace Depth*. R package version 0.0.1, URL https://CRAN.R-project.org/package=MFHD.

Jörnsten R (2004). "Clustering and Classification Based on the L_1 Data Depth." *Journal of Multivariate Analysis*, **90**(1), 67–89. doi:10.1016/j.jmva.2004.02.013.

Koltchinskii VI (1997). "M-Estimation, Convexity and Quantiles." *The Annals of Statistics*, **25**(2), 435–477. doi:10.1214/aos/1031833659.

Koshevoy G, Mosler K (1997). "Zonoid Trimming for Multivariate Distributions." *The Annals of Statistics*, **25**(5), 1998–2017. doi:10.1214/aos/1069362382.

Koshevoy GA (2002). "The Tukey Depth Characterizes the Atomic Measure." *Journal of Multivariate Analysis*, **83**(2), 360–364. doi:10.1006/jmva.2001.2052.

Koshevoy GA (2003). "Lift-Zonoid and Multivariate Depths." In R Dutter, P Filzmoser, U Gather, PJ Rousseeuw (eds.), *Developments in Robust Statistics*, pp. 194–202. Physica-Verlag, Heidelberg. doi:10.1007/978-3-642-57338-5_16.

Kosiorowski D, Zawadzki Z (2019). ***DepthProc**: An R Package for Robust Exploration of Multidimensional Economic Phenomena*. R package version 2.1.1, URL https://CRAN.R-project.org/package=DepthProc.

Lange T, Mosler K, Mozharovskyi P (2014a). "DDα-Classification of Asymmetric and Fat-Tailed Data." In M Spiliopoulou, L Schmidt-Thieme, R Janning (eds.), *Data Analysis, Machine Learning and Knowledge Discovery*, pp. 71–78. Springer-Verlag. doi:10.1007/978-3-319-01595-8_8.

Lange T, Mosler K, Mozharovskyi P (2014b). "Fast Nonparametric Classification Based on Data Depth." *Statistical Papers*, **55**(1), 49–69. doi:10.1007/s00362-012-0488-4.

Lange T, Mozharovskyi P (2014). "The Alpha-Procedure – A Nonparametric Invariant Method for Automatic Classification of d-Dimensional Objects." In M Spiliopoulou, L Schmidt-Thieme, R Janning (eds.), *Data Analysis, Machine Learning and Knowledge Discovery*, pp. 79–86. Springer-Verlag. doi:10.1007/978-3-319-01595-8_9.

Lê S, Josse J, Husson F (2008). "**FactoMineR**: An R Package for Multivariate Analysis." *Journal of Statistical Software*, **25**(1), 1–18. doi:10.18637/jss.v025.i01.

Li J, Cuesta-Albertos JA, Liu RY (2012). "DD-Classifier: Nonparametric Classification Procedure Based on DD-Plot." *Journal of the American Statistical Association*, **107**(498), 737–753. doi:10.1080/01621459.2012.688462.

Liu RY (1990). "On a Notion of Data Depth Based on Random Simplices." *The Annals of Statistics*, **18**(1), 405–414. doi:10.1214/aos/1176347507.

Liu RY (1992). "Data Depth and Multivariate Rank Tests." In Y Dodge (ed.), *L_1-Statistical Analysis and Related Methods*, pp. 279–294. Elsevier, Amsterdam.

Liu X, Zuo Y (2014a). "Computing Halfspace Depth and Regression Depth." *Communications in Statistics – Simulation and Computation*, **43**(5), 969–985. doi:10.1080/03610918.2012.720744.

Liu X, Zuo Y (2014b). "Computing Projection Depth and Its Associated Estimators." *Statistics and Computing*, **24**(1), 51–63. doi:10.1007/s11222-012-9352-6.

Liu X, Zuo Y (2015). "**CompPD**: A MATLAB Package for Computing Projection Depth." *Journal of Statistical Software*, **65**(2), 1–21. doi:10.18637/jss.v065.i02.

Lopez-Pintado S, Torrente A (2013). ***depthTools**: Depth Tools Package*. R package version 0.4, URL https://CRAN.R-project.org/package=depthTools.

Maechler M, Rousseeuw P, Croux C, Todorov V, Ruckstuhl A, Salibian-Barrera M, Verbeke T, Koller M, Conceicao ELT, di Palma MA (2016). ***robustbase**: Basic Robust Statistics*. R package version 0.92-7, URL https://CRAN.R-project.org/package=robustbase.

Mahalanobis PC (1936). "On the Generalized Distance in Statistics." *Proceedings of the National Institute of Sciences of India*, **12**, 49–55.

Mahalanobish O, Karmakar S (2015). ***depth.plot**: Multivariate Analogy of Quantiles*. R package version 0.1, URL https://CRAN.R-project.org/package=depth.plot.

Mersmann O (2018). ***microbenchmark**: Accurate Timing Functions*. R package version 1.4-6, URL https://CRAN.R-project.org/package=microbenchmark.

Mizera I (2002). "On Depth and Deep Points: A Calculus." *The Annals of Statistics*, **30**(6), 1681–1736. doi:10.1214/aos/1043351254.

Mosler K (2002). *Multivariate Dispersion, Central Regions, and Depth: The Lift Zonoid Approach*. Springer-Verlag.

Mosler K (2013). "Depth Statistics." In C Becker, R Fried, S Kuhnt (eds.), *Robustness and Complex Data Structures: Festschrift in Honour of Ursula Gather*, pp. 17–34. Springer-Verlag. doi:10.1007/978-3-642-35494-6_2.

Mosler K, 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, Mozharovskyi P (2017). "Fast DD-Classification of Functional Data." *Statistical Papers*, **58**(4), 1055–1089. doi:10.1007/s00362-015-0738-3.

Mosler K, Polyakova Y (2012). "General Notions of Depth for Functional Data." arXiv:1208.1981 [stat.ME], URL https://arxiv.org/abs/1208.1981.

Mozharovskyi P (2015). *Contributions to Depth-Based Classification and Computation of the Tukey Depth*. Verlag Dr. Kovač, Hamburg.

Mozharovskyi P, Mosler K, Lange T (2015). "Classifying Real-World Data with the $DD\alpha$-Procedure." *Advances in Data Analysis and Classification*, **9**(3), 287–314. doi:10.1007/s11634-014-0180-8.

Mustafa N, Ray S, Shabbir M (2014). ***rsdepth**: Ray Shooting Depth (i.e. RS Depth) Functions for Bivariate Analysis*. R package version 0.1-5, URL https://CRAN.R-project.org/package=rsdepth.

Nagy S (2017). "Monotonicity Properties of Spatial Depth." *Statistics and Probability Letters*, **129**, 373–378. doi:10.1016/j.spl.2017.06.025.

Nelder JA, Mead R (1965). "A Simplex Method for Function Minimization." *The Computer Journal*, **7**, 308–313.

Nieto-Reyes A, Battey H (2016). "A Topologically Valid Definition of Depth for Functional Data." *Statistical Science*, **31**(1), 61–79. doi:10.1214/15-sts532.

Oja H (1983). "Descriptive Statistics for Multivariate Distributions." *Statistics and Probability Letters*, **1**(6), 327–332. doi:10.1016/0167-7152(83)90054-8.

Paindaveine D, Van Bever G (2015). "Nonparametrically Consistent Depth-Based Classifiers." *Bernoulli*, **21**(1), 62–82. doi:10.3150/13-bej561.

Pokotylo O, Mosler K (2019). "Classification with the Pot-Pot Plot." *Statistical Papers*, **60**(3), 903–931. doi:10.1007/s00362-016-0854-8.

Pokotylo O, Mozharovskyi P, Dyckerhoff R (2016). "Depth and Depth-Based Classification with R Package **ddalpha**." arXiv:1608.04109 [stat.CO], URL https://arxiv.org/abs/1608.04109.

Rousseeuw PJ, Leroy AM (1987). *Robust Regression and Outlier Detection*. John Wiley & Sons. doi:10.1002/0471725382.

Rousseeuw PJ, Ruts I (1996). "Algorithm AS 307: Bivariate Location Depth." *Journal of the Royal Statistical Society C*, **45**(4), 516–526. doi:10.2307/2986073.

Rousseeuw PJ, Van Driessen K (1999). "A Fast Algorithm for the Minimum Covariance Determinant Estimator." *Technometrics*, **41**(3), 212–223. doi:10.1080/00401706.1999.10485670.

Sabanés Bové D, Held L (2011). "Bayesian Fractional Polynomials." *Statistics and Computing*, **21**(3), 309–324. doi:10.1007/s11222-010-9170-7.

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*, pp. 25–38. Birkhäuser-Verlag, Basel. doi:10.1007/978-3-0348-8201-9_3.

Sevcikova H, Rossini AJ (2017). ***snowFT**: Fault Tolerant Simple Network of Workstations*. R package version 1.6-0, URL https://CRAN.R-project.org/package=snowFT.

Stahel WA (1981). *Robust Estimation: Infinitesimal Optimality and Covariance Matrix Estimators*. Ph.D. thesis, Swiss Federal Institute of Technology in Zurich. In German.

Struyf AJ, Rousseeuw PJ (1999). "Halfspace Depth and Regression Depth Characterize the Empirical Distribution." *Journal of Multivariate Analysis*, **69**(1), 135–153. doi:10.1006/jmva.1998.1804.

The MathWorks Inc (2019). *MATLAB – The Language of Technical Computing, Version R2019a*. Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

Tuddenham RD, Snyder MM (1954). "Physical Growth of California Boys and Girls from Birth to Eighteen Years." *Publications in Child Development. University of California, Berkeley*, **1**(2), 183–364.

Tukey JW (1975). "Mathematics and the Picturing of Data." In RD James (ed.), *Proceedings of the International Congress of Mathematicians*, volume 2, pp. 523–531. Canadian Mathematical Congress.

Vardi Y, Zhang CH (2000). "The Multivariate L_1-Median and Associated Data Depth." *Proceedings of the National Academy of Sciences of the United States of America*, **97**(4), 1423–1426. doi:10.1073/pnas.97.4.1423.

Vasil'ev VI (2003). "The Reduction Principle in Problems of Revealing Regularities I." *Cybernetics and Systems Analysis*, **39**(5), 686–694. doi:10.1023/b:casa.0000012089.39260.b3.

Vasil'ev VI, Lange TI (1998). "The Duality Principle in Learning for Pattern Recognition." *Kibernetika i Vytschislitelnaya Technika*, **121**, 7–16. In Russian.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York. URL http://www.stats.ox.ac.uk/pub/MASS4.

Vencalek O (2011). *Weighted Data Depth and Depth Based Discrimination*. Ph.D. thesis, Charles University, Prague.

Vencalek O (2017). "Depth-Based Classification for Multivariate Data." *Austrian Journal of Statistics*, **46**(3–4), 117–128. doi:10.17713/ajs.v46i3-4.677.

Šiman M, Boček P (2019). ***modQR**: Multiple-Output Directional Quantile Regression*. R package version 0.1.2, URL https://CRAN.R-project.org/package=modQR.

Wickham H (2007). "Reshaping Data with the **reshape** Package." *Journal of Statistical Software*, **21**(12), 1–20. doi:10.18637/jss.v021.i12.

Wickham H (2016). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag. URL https://ggplot2.tidyverse.org/.

Wickham H (2019). ***stringr**: Simple, Consistent Wrappers for Common String Operations*. R package version 1.4.0, URL https://CRAN.R-project.org/package=stringr.

Wolf HP (2019). ***aplpack**: Another Plot Package*. R package version 1.3.3, URL https://CRAN.R-project.org/package=aplpack.

Yeh IC, Yang KJ, Ting TM (2009). "Knowledge Discovery on RFM Model Using Bernoulli Sequence." *Expert Systems with Applications*, **36**(3), 5866–5871. doi:10.1016/j.eswa.2008.07.018.

Yu H (2017). ***Rmpi**: Interface (Wrapper) to MPI (Message-Passing Interface)*. R package version 0.6-6, URL https://CRAN.R-project.org/package=Rmpi.

Zuo Y, Serfling R (2000). "General Notions of Statistical Depth Function." *The Annals of Statistics*, **28**(2), 461–482. doi:10.1214/aos/1016218226.

Affiliation:

Oleksii Pokotylo
Faculty of Management, Economics and Social Sciences
University of Cologne
Albertus-Magnus-Platz, 50923 Cologne, Germany
E-mail: oleksii.pokotylo@gmail.com

Pavlo Mozharovskyi
Center for Research in Economics and Statistics
National School for Statistics and Information Analysis
Université Bretagne Loire
Rue Blaise Pascal, 35172 Bruz, France
E-mail: pavlo.mozharovskyi@ensai.fr
URL: http://www.ensai.fr/enseignant/alias/pavlo-mozharovskyi.html

Rainer Dyckerhoff
Institute of Econometrics and Statistics
Faculty of Management, Economics and Social Sciences
University of Cologne
Albertus-Magnus-Platz, 50923 Cologne, Germany
E-mail: rainer.dyckerhoff@statistik.uni-koeln.de
URL: http://www.wisostat.uni-koeln.de/dyckerhoff.html

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
October 2019, Volume 91, Issue 5 *Submitted:* 2016-08-11
doi:10.18637/jss.v091.i05 *Accepted:* 2018-07-07