An Object Oriented Framework for Robust Multivariate Analysis

Valentin Todorov
UNIDO

Peter Filzmoser
Vienna University of Technology

Abstract

This introduction to the R package **rrcov** is a (slightly) modified version of Todorov and Filzmoser (2009), published in the *Journal of Statistical Software*.
Taking advantage of the S4 class system of the programming environment R, which facilitates the creation and maintenance of reusable and modular components, an object oriented framework for robust multivariate analysis was developed. The framework resides in the packages **robustbase** and **rrcov** and includes an almost complete set of algorithms for computing robust multivariate location and scatter, various robust methods for principal component analysis as well as robust linear and quadratic discriminant analysis. The design of these methods follows common patterns which we call statistical design patterns in analogy to the design patterns widely used in software engineering. The application of the framework to data analysis as well as possible extensions by the development of new methods is demonstrated on examples which themselves are part of the package **rrcov**.

Keywords: robustness, multivariate analysis, MCD, R, statistical design patterns.

1. Introduction

Outliers are present in virtually every data set in any application domain, and the identification of outliers has a hundred years long history. Many researchers in science, industry and economics work with huge amounts of data and this even increases the possibility of anomalous data and makes their (visual) detection more difficult. Taking into account the multivariate aspect of the data, the outlyingness of the observations can be measured by the Mahalanobis distance which is based on location and scatter estimates of the data set. In order to avoid the masking effect, robust estimates of these parameters are called for, even more, they must possess a positive breakdown point. The estimates of the multivariate location vector $\boldsymbol{\mu}$ and the scatter matrix $\boldsymbol{\Sigma}$ are also a cornerstone in the analysis of multidimensional data, since they form the input to many classical multivariate methods. The most common estimators of multivariate location and scatter are the sample mean $\bar{\boldsymbol{x}}$ and the sample covariance matrix $\mathbf{S}$, i.e., the corresponding MLE estimates. These estimates are optimal if the data come from a multivariate normal distribution but are extremely sensitive to the presence of even a few outliers (atypical values, anomalous observations, gross errors) in the data. If outliers are present in the input data they will influence the estimates $\bar{\boldsymbol{x}}$ and $\mathbf{S}$ and subsequently worsen the performance of the classical multivariate procedure based on these estimates. Therefore it is important to consider robust alternatives to these estimators and actually in the last two decades much effort was devoted to the development of affine equivariant estimators

possessing a high breakdown point. The most widely used estimators of this type are the minimum covariance determinant (MCD) estimator of Rousseeuw (1985) for which also a fast computing algorithm was constructed—Rousseeuw and Van Driessen (1999), the S estimators (Davies 1987) and the Stahel-Donoho estimator introduced by Stahel (1981a,b) and Donoho (1982) and studied by Maronna and Yohai (1995). If we give up the requirement for affine equivariance, estimators like the one of Maronna and Zamar (2002) are available and the reward is an extreme gain in speed. Substituting the classical location and scatter estimates by their robust analogues is the most straightforward method for robustifying many multivariate procedures like principal components, discriminant and cluster analysis, canonical correlation, etc. The reliable identification of multivariate outliers which is an important task in itself, is another approach to robustifying many classical multivariate methods.

Some of these estimates and procedures became available in the popular statistical packages like S-PLUS, SAS, MATLAB as well as in R but nevertheless it is recognized that the robust methods have not yet replaced the ordinary least square techniques as it could be expected (Morgenthaler 2007; Stromberg 2004). One reason is the lack of easily accessible and easy to use software, that is software which presents the robust procedures as extensions to the classical ones—similar input and output, reasonable defaults for most of the estimation options and visualization tools. As far as the easiness of access is concerned, the robust statistical methods should be implemented in the freely available statistical software package R, (R Development Core Team 2009), which provides a powerful platform for the development of statistical software. These requirements have been defined in the project "Robust Statistics and R", see `http://www.statistik.tuwien.ac.at/rsr/`, and a first step in this direction was the initial development of the collaborative package **robustbase**, (Rousseeuw *et al.* 2009), with the intention that it becomes the *essential robust statistics* R package covering the methods described in the recent book Maronna *et al.* (2006).

During the last decades the object oriented programming paradigm has revolutionized the style of software system design and development. A further step in the software reuse are the object oriented frameworks (see Gamma *et al.* 1995) which provide technology for reusing both the architecture and the functionality of software components. Taking advantage of the new S4 class system (Chambers 1998) of R which facilitate the creation of reusable and modular components an object oriented framework for robust multivariate analysis was implemented. The goal of the framework is manyfold:

1. to provide the end-user with a flexible and easy access to newly developed robust methods for multivariate data analysis;

2. to allow the programming statisticians an extension by developing, implementing and testing new methods with minimum effort, and

3. to guarantee the original developers and maintainer of the packages a high level of maintainability.

The framework includes an almost complete set of algorithms for computing robust multivariate location and scatter, such as minimum covariance determinant, different S estimators (SURREAL, FAST-S, Bisquare, Rocke-type), orthogonalized Gnanadesikan–Kettenring (OGK) estimator of Maronna and Zamar (2002). The next large group of classes are the methods for robust principal component analysis (PCA) including ROBPCA of Hubert *et al.*

(2005), spherical principal components (SPC) of Locantore *et al.* (1999), the projection pursuit algorithms of Croux and Ruiz-Gazen (2005) and Croux *et al.* (2007). Further applications implemented in the framework are linear and quadratic discriminant analysis (see Todorov and Pires 2007, for a review), multivariate tests (Willems *et al.* 2002; Todorov and Filzmoser 2010) and outlier detection tools.

The application of the framework to data analysis as well as the development of new methods is illustrated on examples, which themselves are part of the package. Some issues of the object oriented paradigm as applied to the R object model (naming conventions, access methods, coexistence of S3 and S4 classes, usage of UML, etc.) are discussed. The framework is implemented in the R packages **robustbase** and **rrcov**, (Todorov 2009), which are available from Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org` under the GNU General Public License.

The rest of the paper is organized as follows. In the next Section 2 the design principles and the structure of the framework is presented as well as some related object oriented concepts are discussed. As a main tool for modeling of the robust estimation methods a statistical design pattern is proposed. Section 3 facilitates the quick start by an example session giving a brief overview of the framework. Section 4 describes the robust multivariate methods, their computation and implementation. The Sections 4.1, 4.2 and 4.3 are dedicated to the estimation of multivariate location and scatter, principal component analysis and discriminant analysis, respectively. For each domain the object model, the available visualization tools, an example, and other relevant information are presented. We conclude in Section 5 with discussion and outline of the future work.

2. Design approach and structure of the framework

In classical multivariate statistics we rely on parametric models based on assumptions about the structural and the stochastic parts of the model for which optimal procedures are derived, like the least squares estimators and the maximum likelihood estimators. The corresponding robust methods can be seen as extensions to the classical ones which can cope with deviations from the stochastic assumptions thus mitigating the dangers for classical estimators. The developed statistical procedures will remain reliable and reasonably efficient even when such deviations are present. For example in the case of location and covariance estimation the classical theory yields the sample mean $\bar{\boldsymbol{x}}$ and the sample covariance matrix $\mathbf{S}$, i.e., the corresponding MLE estimates as an optimal solution. One (out of many) robust alternatives is the minimum covariance determinant estimator. When we consider this situation from an object oriented design point of view we can think of an abstract base class representing the estimation problem, a concrete realization of this object—the classical estimates, and a second concrete derivative of the base class representing the MCD estimates. Since there exist many other robust estimators of multivariate location and covariance which share common characteristics we would prefer to add one more level of abstraction by defining an abstract "robust" object from which all other robust estimators are derived. We encounter a similar pattern in most of the other multivariate statistical methods like principal component analysis, linear and quadratic discriminant analysis, etc. and we will call it a *statistical design pattern*. A schematic representation as an *UML diagram* is shown in Figure 2. The following simple example demonstrates the functionality. We start with a generic object model of a robust and the corresponding classical multivariate method with all the necessary interfaces and

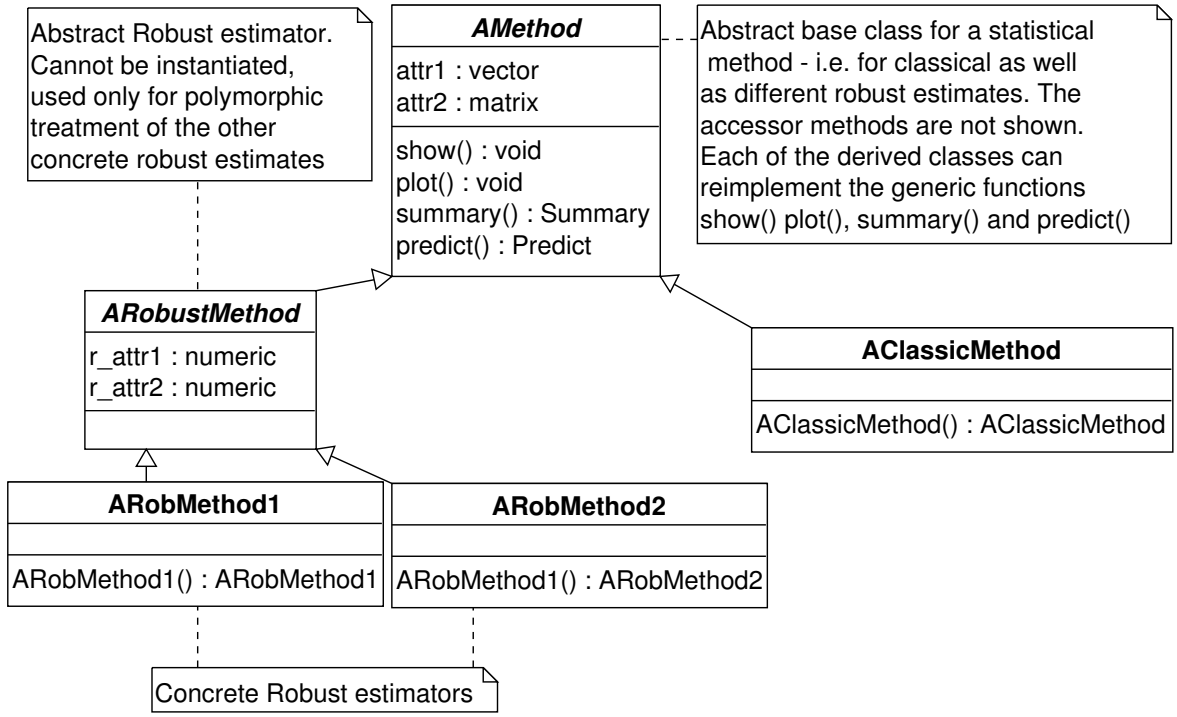

Figure 1: Class diagram of the statistical design pattern for robust estimation methods.

functionalities and then concretize it to represent the desired class hierarchy. The basic idea is to define an abstract S4 class which has as slots the common data elements of the result of the classical method and its robust counterparts (e.g., `Pca`). For this abstract class we can implement the standard in R generic functions like `print()`, `summary()`, `plot()` and maybe also `predict()`. Now we can derive and implement a concrete class which will represent the classical method, say `PcaClassic`. Further we derive another abstract class which represents a potential robust method we are going to implement, e.g., `PcaRobust`—it is abstract because we want to have a "placeholder" for the robust methods we are going to develop next. The generic functions that we implemented for the class `Pca` are still valid for `PcaRobust` but whenever necessary we can override them with new functionality. Now we have the necessary platform and of course we have had diligently documented everything we have implemented so far—this is our investment in the future development of robust methods from this family. The framework at its current expansion stage provides such platform for several important families of multivariate methods. It is time to dedicate our effort to the development and implementation of our new robust method/class, say `PcaHubert` and only to this—here comes the first obvious benefit from the framework—we do not need to care for the implementation of `print()`, `summary()`, `plot()` and `predict()` neither for their documentation or testing.

In contrast to the S3 class system the S4 system requires the creation of objects to be done by the `new()` function which will perform the necessary validity checks. We go one step further and require that the `new()` function is not used directly but only through special functions known in R as *generating functions* or as *constructors* in the conventional object oriented

programming languages. A constructor function has the same name as the corresponding class, takes the estimation options as parameters, organizes the necessary computations and returns an object of the class containing the results of the computation. It can take as a parameter also a *control object* which itself is an S4 object and contains the estimation options. More details on the generating functions and their application for structuring the user interface can be found in Ruckdeschel *et al.* (2009).

The main part of the framework is implemented in the package **rrcov** but it relies on code in the packages **robustbase** and **pcaPP** (Filzmoser *et al.* 2009). The structure of the framework and its relation to other R packages is shown in Figure 2. The framework can be used by other

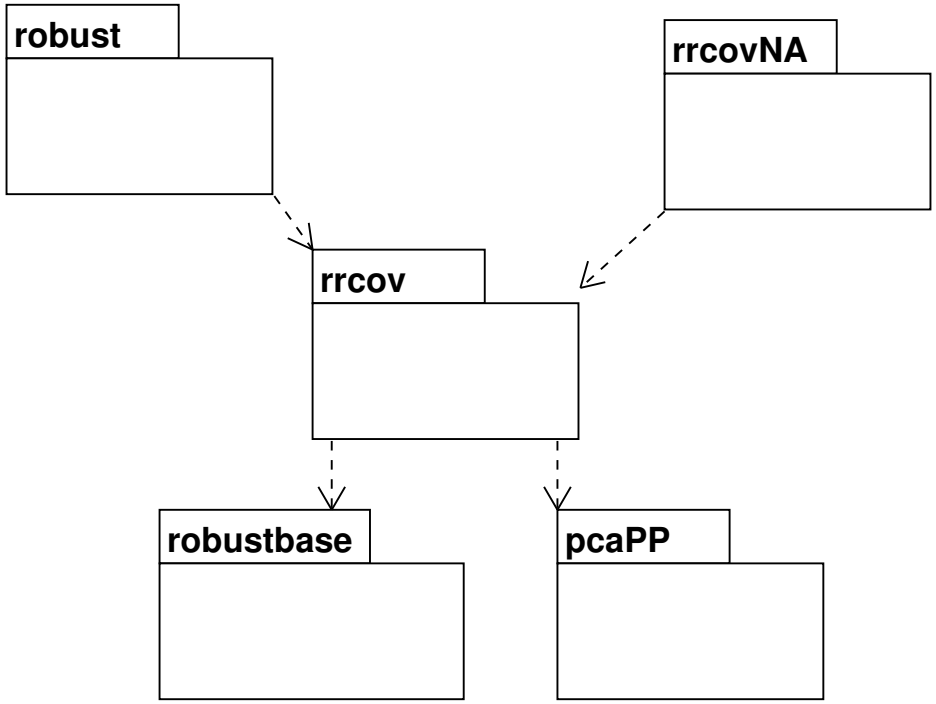

Figure 2: Class diagram: structure of the framework and relation to other R packages.

packages, like for example by **robust** (see Wang *et al.* 2008) or can be further extended. In Figure 2 a hypothetical package **rrcovNA** is shown, which could extend the available robust multivariate methods with options for dealing with missing values.

In the rest of this section some object-oriented programming (OOP) concepts will be discussed which are essential for understanding the framework.

2.1. UML diagrams

Throughout this paper we exploit UML class diagrams to give a clear picture of the framework and its components. UML stands for *Unified Modeling Language*—an object-oriented system of notation which has evolved from previous works of Grady Booch, James Rumbaugh and Ivar Jacobson to become a tool accepted by the Object Management Group (OMG) as the standard for modeling object oriented programs (see OMG 2009a,b). A class diagram models the structure and contents of a system by depicting classes, packages, objects and the relations among them with the aim to increase the ease of understanding the considered application. A class is denoted by a box with three compartments which contain the name, the attributes (slots) and operations (methods) of the class, respectively. The class name in italics indicates that the class is abstract. The bottom two compartments could be omitted or they can contain only the key attributes and operations which are useful for understanding the particular diagram. Each attribute is followed by its type and each operation—by the type of its return value. We use the R types like `numeric`, `logical`, `vector`, `matrix`, etc. but the type can be also a name of an S4 class.

Relationships between classes are denoted by lines or arrows with different form. The inheritance relationship is depicted by a large empty triangular arrowhead pointing to the base class. Composition means that one class contains another one as a slot (not to be mistaken with the keyword "contains" signalling inheritance in R). This relation is represented by an arrow with a solid diamond on the side of the composed class. If a class "uses" another one or depends on it, the classes are connected by a dashed arrow (dependence relation). Packages can also be present in a class diagram—in our case they correspond more or less to R packages—and are shown as tabbed boxes with the name of the package written in the tab (see Figure 2).

All UML diagrams of the framework were created with the open source UML tool **ArgoUML** (Robbins 1999; Robbins and Redmiles 2000) which is available for download from `http://argouml.tigris.org/`.

2.2. Design patterns

Design patterns are usually defined as general solutions to recurring design problems and refer to both the description of a solution and an instance of that solution solving a particular problem. The current use of the term design patterns originates in the writings of the architect Christopher Alexander devoted to urban planning and building architecture (Alexander *et al.* 1977) but it was brought to the software development community by the seminal book of Gamma *et al.* (1995).

A design pattern can be seen as a template for how to solve a problem which can be used in many different situations. Object-Oriented design patterns are about classes and the relationships between classes or objects at abstract level, without defining the final classes or objects of the particular application. In order to be usable, design patterns must be defined formally and the documentation, including a preferably evocative name, describes the context in which the pattern is used, the pattern structure, the participants and collaboration, thus presenting the suggested solution.

Design patterns are not limited to architecture or software development but can be applied in any domain where solutions are searched for. During the development of the here presented framework several design patterns were identified, which we prefer to call *statistical design*

patterns. The first one was already described earlier in this section and captures the relations among a classical and one or more alternative robust multivariate estimators. Another candidate is the control object encapsulating the estimation parameters and a third one is the factory-like construct which suggests selection of a robust estimation method and creation of the corresponding objects based on the data set characteristics (see Section 4.1). The formal description of these design patterns is beyond the scope of this work and we will limit the discussion to several examples.

2.3. Accessor methods

One of the major characteristics and advantages of object oriented programming is the encapsulation. Unfortunately real encapsulation (information hiding) is missing in R, but as far as the access to the member variables is concerned this could be mitigated by defining accessor methods (i.e., methods used to examine or modify the slots (member variables) of a class) and "advising" the users to use them instead of directly accessing the slots. The usual way of defining accessor functions in R is to use the same name as the name of the corresponding slot. For example for the slot `a` these are:

```
R> cc <- a(obj)
R> a(obj) <- cc
```

In many cases this is not possible, because of conflicts with other existing functions. For example it is not possible to define an accessor function `cov()` for the slot `cov` of class `Cov`, since the function `cov()` already exists in the `base` R. Also it is not immediately seen if a slot is "read only" or can be modified by the user (unfortunately, as already mentioned, every slot in R can be modified by simply using `obj@a <- cc`). In **rrcov** a notation was adopted, which is usual in Java: the accessors are defined as `getXxx()` and `setXxx()` (if a `setXxx()` method is missing, we are "not allowed" to change the slot). The use of accessor methods allows to perform computations on demand (`getMah(mcd)` computes the Mahalanobis distances, stores them into the object and returns them) or even have "virtual" slots which are not at all stored in the object (e.g., `getCorr(mcd)` computes each time and returns the correlation matrix without storing it).

2.4. Naming conventions

There is no agreed naming convention (coding rules) in R but to facilitate the framework usage several simple rules are in order, following the recommended Sun's Java coding style (see http://java.sun.com/docs/codeconv/):

- Class, function, method and variable names are alphanumeric, do not contain "-" or "." but rather use interchanging lower and upper case.
- Class names start with an uppercase letter.
- Methods, functions, and variables start with a lowercase letter.
- Exceptions are functions returning an object of a given class (i.e., generating functions or constructors)—they have the same name as the class.

- Variables and methods which are not intended to be seen by the user—i.e., private members—start with ".".
- Violate these rules whenever necessary to maintain compatibility.

3. Example session

In this section we will introduce the base functionalities of the framework by an example session. First of all we have to load the package **rrcov** which will cause all necessary packages to be loaded too. The framework includes many example data sets but here we will load only those which will be used throughout the following examples. For the rest of the paper it will be assumed that the package has been loaded already.

```
R> library("rrcov")

pcaPP 0.1-1 loaded
Scalable Robust Estimators with High Breakdown Point (version 1.0-00)

R> data("delivery")
R> delivery.x <- delivery[, 1:2]
R> data("hbk")
R> hbk.x <- hbk[, 1:3]
```

Most of the multivariate statistical methods are based on estimates of multivariate location and covariance, therefore these estimates play a central role in the framework. We will start with computing the robust *minimum covariance determinant* estimate for the data set `delivery` from the package **robustbase**. The data set (see Rousseeuw and Leroy 1987, Table 23, p. 155) contains delivery time data in 25 observations with 3 variables. The aim is to explain the time required to service a vending machine `(Y)` by means of the number of products stocked `(X1)` and the distance walked by the route driver `(X2)`. For this example we will consider only the $\mathbf{X}$ part of the data set. After computing its robust location and covariance matrix using the MCD method implemented in the function `CovMcd()` we can print the results by calling the default `show()` method on the returned object `mcd` as well as summary information by the `summary()` method. The standard output contains the robust estimates of location and covariance. The summary output contains additionally the eigenvalues of the covariance matrix and the robust distances of the data items (Mahalanobis type distances computed with the robust location and covariance instead of the sample ones).

```
R> mcd <- CovMcd(delivery.x)
R> mcd

Call:
CovMcd(x = delivery.x)
-> Method:  Minimum Covariance Determinant Estimator.

Robust Estimate of Location:
```

```
  n.prod  distance
   5.895   268.053

Robust Estimate of Covariance:
          n.prod    distance
n.prod       11.66    220.72
distance    220.72  53202.65

R> summary(mcd)

Call:
CovMcd(x = delivery.x)

Robust Estimate of Location:
  n.prod  distance
   5.895   268.053

Robust Estimate of Covariance:
          n.prod    distance
n.prod       11.66    220.72
distance    220.72  53202.65

Eigenvalues of covariance matrix:
[1]  53203.57     10.74

Robust Distances:
 [1]   1.6031   0.7199   1.0467   0.7804   0.2949   0.1391   1.4464   0.2321
 [9]  60.8875   2.6234   9.8271   1.7949   0.3186   0.7526   1.1267   5.2213
[17]   0.1010   0.6075   1.3597  12.3162   2.3099  35.2113   1.1366   2.5625
[25]   0.4458

R> plot(mcd, which = "dd")
```

Now we will show one of the available plots by calling the `plot()` method—in Figure 3 the Distance-Distance plot introduced by Rousseeuw and van Zomeren (1991) is presented, which plots the robust distances versus the classical Mahalanobis distances and allows to classify the observations and identify the potential outliers. The description of this plot as well as examples of more graphical displays based on the covariance structure will be shown in Section 4.1.

Apart from the demonstrated MCD method the framework provides many other robust estimators of multivariate location and covariance, actually almost all of the well established estimates in the contemporary robustness literature. The most fascinating feature of the framework is that one will get the output and the graphs in the same format, whatever estimation method was used. For example the following code lines will compute the S estimates for the same data set and provide the standard and extended output (not shown here).

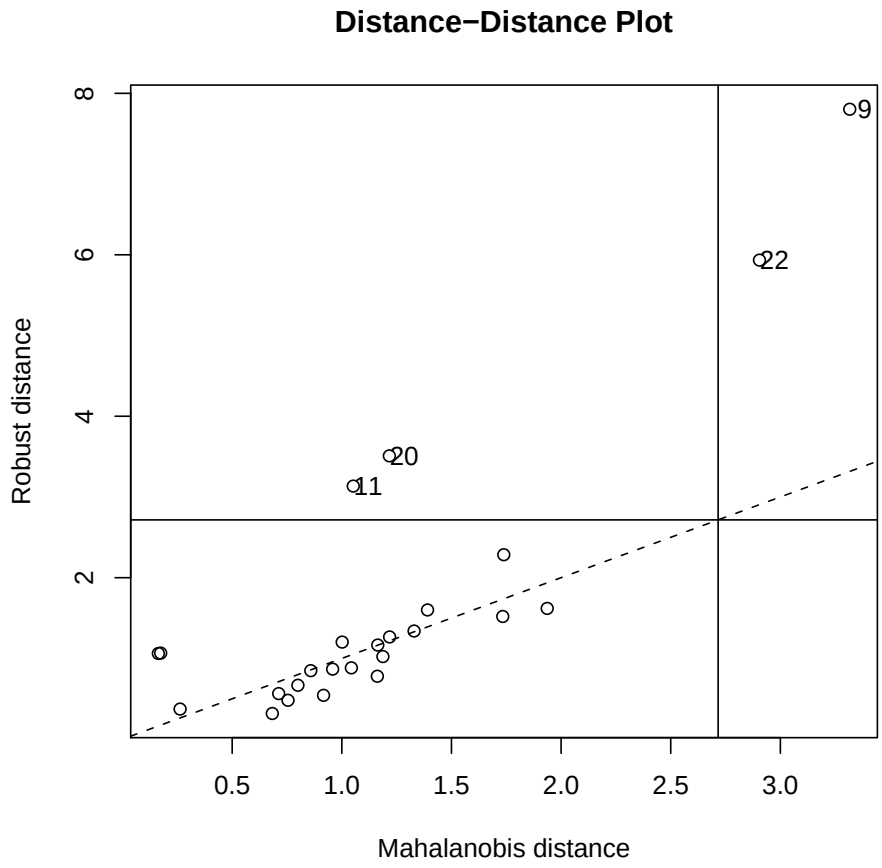

Figure 3: Example plot of the robust against classical distances for the `delivery` data set.

```
R> est <- CovSest(delivery.x, method = "bisquare")
R> est
R> summary(est)
```

Nevertheless, this variety of methods could pose a serious hurdle for the novice and could be quite tedious even for the experienced user. Therefore a shortcut is provided too—the function `CovRobust()` can be called with a parameter set specifying any of the available estimation methods, but if this parameter set is omitted the function will decide on the basis of the data size which method to use. As we see in the example below, in this case it selects the Stahel-Donoho estimates. For details and further examples see Section 4.1.

```
R> est <- CovRobust(delivery.x)
R> est

Call:
CovSde(x = x, control = obj)
-> Method:  Stahel-Donoho estimator
```

```
Robust Estimate of Location:
  n.prod  distance
   5.849   276.373

Robust Estimate of Covariance:
          n.prod   distance
n.prod        9.5    350.1
distance    350.1  47354.7
```

4. Robust multivariate methods

4.1. Multivariate location and scatter

The framework provides an almost complete set of estimators for multivariate location and scatter with high breakdown point. The first such estimator was proposed by Stahel (1981a,b) and Donoho (1982) and it is recommended for small data sets, but the most widely used high breakdown estimator is the minimum covariance determinant estimate (Rousseeuw 1985). Several algorithms for computing the S estimators (Davies 1987) are provided (Ruppert 1992; Woodruff and Rocke 1994; Rocke 1996; Salibian-Barrera and Yohai 2006). The minimum volume ellipsoid (MVE) estimator (Rousseeuw 1985) is also included since it has some desirable properties when used as initial estimator for computing the S estimates (see Maronna *et al.* 2006, p. 198). In the rest of this section the definitions of the different estimators of location and scatter will be briefly reviewed and the algorithms for their computation will be discussed. Further details can be found in the relevant references. The object model is presented and examples of its usage, as well as further examples of the graphical displays are given.

The Minimum covariance determinant estimator and its computation

The MCD estimator for a data set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ in $\Re^p$ is defined by that subset $\{\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_h}\}$ of h observations whose covariance matrix has the smallest determinant among all possible subsets of size h. The MCD location and scatter estimate $\mathbf{T}_{MCD}$ and $\mathbf{C}_{MCD}$ are then given as the arithmetic mean and a multiple of the sample covariance matrix of that subset

$$\begin{aligned} \mathbf{T}_{MCD} &= \frac{1}{h}\sum_{j=1}^{h} \boldsymbol{x}_{i_j} \\ \mathbf{C}_{MCD} &= c_{ccf}\, c_{sscf} \frac{1}{h-1}\sum_{j=1}^{h} (\boldsymbol{x}_{i_j} - \mathbf{T}_{MCD})(\boldsymbol{x}_{i_j} - \mathbf{T}_{MCD})^{\top}. \end{aligned} \tag{1}$$

The multiplication factors c_{ccf} (consistency correction factor) and c_{sscf} (small sample correction factor) are selected so that $\mathbf{C}$ is consistent at the multivariate normal model and unbiased at small samples (see Butler *et al.* 1993; Croux and Haesbroeck 1999; Pison *et al.* 2002; Todorov 2008). A recommendable choice for h is $\lfloor (n+p+1)/2 \rfloor$ because then the BP of the MCD is maximized, but any integer h within the interval $[(n+p+1)/2, n]$ can be

chosen, see Rousseeuw and Leroy (1987). Here $\lfloor z \rfloor$ denotes the integer part of z which is not less than z. If $h = n$ then the MCD location and scatter estimate $\mathbf{T}_{MCD}$ and $\mathbf{C}_{MCD}$ reduce to the sample mean and covariance matrix of the full data set.

The computation of the MCD estimator is far from being trivial. The naive algorithm would proceed by exhaustively investigating all subsets of size h out of n to find the subset with the smallest determinant of its covariance matrix, but this will be feasible only for very small data sets. Initially MCD was neglected in favor of MVE because the simple resampling algorithm was more efficient for MVE. Meanwhile several heuristic search algorithms (see Todorov 1992; Woodruff and Rocke 1994; Hawkins 1994) and exact algorithms (Agulló 1996) were proposed but now a very fast algorithm due to Rousseeuw and Van Driessen (1999) exists and this algorithm is usually used in practice. The algorithm is based on the C-step which moves from one approximation $(\mathbf{T}_1, \mathbf{C}_1)$ of the MCD estimate of a data set $\mathbf{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ to the next one $(\mathbf{T}_2, \mathbf{C}_2)$ with possibly lower determinant $\det(\mathbf{C}_2) \leq \det(\mathbf{C}_1)$ by computing the distances $d_1, \ldots, d_n$ relative to $(\mathbf{T}_1, \mathbf{C}_1)$, i.e.,

$$d_i = \sqrt{(\boldsymbol{x}_i - \mathbf{T}_1)^\top \mathbf{C}_1^{-1} (\boldsymbol{x}_i - \mathbf{T}_1)} \tag{2}$$

and then computing $(\mathbf{T}_2, \mathbf{C}_2)$ for those h observations which have smallest distances. "C" in C-step stands for "concentration" since we are looking for a more "concentrated" covariance matrix with lower determinant. Rousseeuw and Van Driessen (1999) have proven a theorem stating that the iteration process given by the C-step converges in a finite number of steps to a (local) minimum. Since there is no guarantee that the global minimum of the MCD objective function will be reached, the iteration must be started many times from different initial subsets, to obtain an approximate solution. The procedure is very fast for small data sets but to make it really "fast" also for large data sets several computational improvements are used.

- *initial subsets*: It is possible to restart the iterations from randomly generated subsets of size h, but in order to increase the probability of drawing subsets without outliers, $p + 1$ points are selected randomly. These $p + 1$ points are used to compute $(\mathbf{T}_0, \mathbf{C}_0)$. Then the distances $d_1, \ldots, d_n$ are computed and sorted in increasing order. Finally the first h are selected to form the initial $h-$subset H_0.

- *reduced number of C-steps*: The C-step involving the computation of the covariance matrix, its determinant and the relative distances, is the most computationally intensive part of the algorithm. Therefore instead of iterating to convergence for each initial subset only two C-steps are performed and the 10 subsets with lowest determinant are kept. Only these subsets are iterated to convergence.

- *partitioning*: For large n the computation time of the algorithm increases mainly because all n distances given by Equation (2) have to be computed at each iteration. An improvement is to partition the data set into a maximum of say five subsets of approximately equal size (but not larger than say 300) and iterate in each subset separately. The ten best solutions for each data set are kept and finally only those are iterated on the complete data set.

- *nesting*: Further decrease of the computational time can be achieved for data sets with n larger than say 1500 by drawing 1500 observations without replacement and performing

the computations (including the partitioning) on this subset. Only the final iterations are carried out on the complete data set. The number of these iterations depends on the actual size of the data set at hand.

The MCD estimator is not very efficient at normal models, especially if h is selected so that maximal BP is achieved. To overcome the low efficiency of the MCD estimator, a reweighed version can be used. For this purpose a weight w_i is assigned to each observation $\boldsymbol{x}_i$, defined as $w_i = 1$ if $(\boldsymbol{x}_i - \mathbf{T}_{MCD})^\top \mathbf{C}_{MCD}^{-1}(\boldsymbol{x}_i - \mathbf{T}_{MCD}) \leq \chi^2_{p,0.975}$ and $w_i = 0$ otherwise, relative to the raw MCD estimates $(\mathbf{T}_{MCD}, \mathbf{C}_{MCD})$. Then the reweighted estimates are computed as

$$\begin{aligned} \mathbf{T}_R &= \frac{1}{\nu}\sum_{i=1}^{n} w_i \boldsymbol{x}_i, \\ \mathbf{C}_R &= c_{r.ccf}\, c_{r.sscf} \frac{1}{\nu - 1}\sum_{i=1}^{n} w_i (\boldsymbol{x}_i - \mathbf{T}_R)(\boldsymbol{x}_i - \mathbf{T}_R)^\top, \end{aligned} \tag{3}$$

where ν is the sum of the weights, $\nu = \sum_{i=1}^{n} w_i$. Again, the multiplication factors $c_{r.ccf}$ and $c_{r.sscf}$ are selected so that $\mathbf{C}_R$ is consistent at the multivariate normal model and unbiased at small samples (see Pison *et al.* 2002; Todorov 2008, and the references therein). These reweighted estimates $(\mathbf{T}_R, \mathbf{C}_R)$ which have the same breakdown point as the initial (raw) estimates but better statistical efficiency are computed and used by default.

The Minimum volume ellipsoid estimates

The minimum volume ellipsoid estimator searches for the ellipsoid of minimal volume containing at least half of the points in the data set $\mathbf{X}$. Then the location estimate is defined as the center of this ellipsoid and the covariance estimate is provided by its shape. Formally the estimate is defined as these $\mathbf{T}_{MVE}, \mathbf{C}_{MVE}$ that minimize $\det(\mathbf{C})$ subject to

$$\#\{i : (\boldsymbol{x}_i - \mathbf{T})^\top \mathbf{C}^{-1}(\boldsymbol{x}_i - \mathbf{T}) \leq c^2\} \geq \left\lfloor \frac{n + p + 1}{2} \right\rfloor, \tag{4}$$

where $\#$ denotes the cardinality. The constant c is chosen as $\chi^2_{p,0.5}$.

The search for the approximate solution is made over ellipsoids determined by the covariance matrix of $p + 1$ of the data points and by applying a simple but effective improvement of the sub-sampling procedure as described in Maronna *et al.* (2006), p. 198. Although there exists no formal proof of this improvement (as for MCD and LTS), simulations show that it can be recommended as an approximation of the MVE. The MVE was the first popular high breakdown point estimator of location and scatter but later it was replaced by the MCD, mainly because of the availability of an efficient algorithm for its computation (Rousseeuw and Van Driessen 1999). Recently the MVE gained importance as initial estimator for S estimation because of its small maximum bias (see Maronna *et al.* 2006, Table 6.2, p. 196).

The Stahel-Donoho estimator

The first multivariate equivariant estimator of location and scatter with high breakdown point was proposed by Stahel (1981a,b) and Donoho (1982) but became better known after the analysis of Maronna and Yohai (1995). For a data set $\mathbf{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ in $\Re^p$ it is defined as a weighted mean and covariance matrix of the form given by Equation (3) where the

weight w_i of each observation is inverse proportional to the "outlyingness" of the observation. Let the univariate outlyingness of a point $\boldsymbol{x}_i$ with respect to the data set $\mathbf{X}$ along a vector $\mathbf{a} \in \Re^p, ||\mathbf{a}|| \neq \mathbf{0}$ be given by

$$r(\boldsymbol{x}_i, \mathbf{a}) = \frac{|\boldsymbol{x}^\top \mathbf{a} - m(\mathbf{a}^\top \mathbf{X})|}{s(\mathbf{a}^\top \mathbf{X})} \quad i = 1, \ldots, n \tag{5}$$

where $(\mathbf{a}^\top \mathbf{X})$ is the projection of the data set $\mathbf{X}$ on $\mathbf{a}$ and the functions $m()$ and $s()$ are robust univariate location and scale statistics, for example the median and MAD, respectively. Then the multivariate outlyingness of $\boldsymbol{x}_i$ is defined by

$$r_i = r(\boldsymbol{x}_i) = \max_{\mathbf{a}} r(\boldsymbol{x}_i, \mathbf{a}). \tag{6}$$

The weights are computed by $w_i = w(r_i)$ where $w(r)$ is a nonincreasing function of r and $w(r)$ and $w(r)r^2$ are bounded. Maronna and Yohai (1995) use the weights

$$w(r) = \min\left(1, \left(\frac{c}{t}\right)^2\right) \tag{7}$$

with $c = \sqrt{\chi^2_{p,\beta}}$ and $\beta = 0.95$, that are known in the literature as "Huber weights".

Exact computation of the estimator is not possible and an approximate solution is found by subsampling a large number of directions $\mathbf{a}$ and computing the outlyingness measures $r_i, i = 1, \ldots, n$ for them. For each subsample of p points the vector $\mathbf{a}$ is taken as the norm 1 vector orthogonal to the hyperplane spanned by these points. It has been shown by simulations (Maronna *et al.* 2006) that one step reweighting does not improve the estimator.

Orthogonalized Gnanadesikan/Kettenring

The MCD estimator and all other known affine equivariant high-breakdown point estimates are solutions to a highly non-convex optimization problem and as such pose a serious computational challenge. Much faster estimates with high breakdown point can be computed if one gives up the requirements of affine equivariance of the covariance matrix. Such an algorithm was proposed by Maronna and Zamar (2002) which is based on the very simple robust bivariate covariance estimator s_{jk} proposed by Gnanadesikan and Kettenring (1972) and studied by Devlin *et al.* (1981). For a pair of random variables Y_j and Y_k and a standard deviation function $\sigma()$, s_{jk} is defined as

$$s_{jk} = \frac{1}{4}\left(\sigma\left(\frac{Y_j}{\sigma(Y_j)} + \frac{Y_k}{\sigma(Y_k)}\right)^2 - \sigma\left(\frac{Y_j}{\sigma(Y_j)} - \frac{Y_k}{\sigma(Y_k)}\right)^2\right). \tag{8}$$

If a robust function is chosen for $\sigma()$ then s_{jk} is also robust and an estimate of the covariance matrix can be obtained by computing each of its elements s_{jk} for each $j = 1, \ldots, p$ and $k = 1, \ldots, p$ using Equation (8). This estimator does not necessarily produce a positive definite matrix (although symmetric) and it is not affine equivariant. Maronna and Zamar (2002) overcome the lack of positive definiteness by the following steps:

- Define $\boldsymbol{y}_i = \boldsymbol{D}^{-1}\boldsymbol{x}_i, i = 1, \ldots, n$ with $\mathbf{D} = \text{diag}(\sigma(X_1), \ldots, \sigma(X_p))$ where $X_l, l = 1, \ldots, p$ are the columns of the data matrix $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. Thus a normalized data matrix $\boldsymbol{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$ is computed.

- Compute the matrix $\boldsymbol{U} = (u_{jk})$ as $u_{jk} = s_{jk} = s(Y_j, Y_k)$ if $j \neq k$ or $u_{jk} = 1$ otherwise. Here $Y_l, l = 1, \ldots, p$ are the columns of the transformed data matrix $\boldsymbol{Y}$ and $s(.,.)$ is a robust estimate of the covariance of two random variables like the one in Equation (8).
- Obtain the "principal component decomposition" of $\boldsymbol{Y}$ by decomposing $\boldsymbol{U} = \boldsymbol{E}\boldsymbol{\Lambda}\boldsymbol{E}^\top$ where $\boldsymbol{\Lambda}$ is a diagonal matrix $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_p)$ with the eigenvalues λ_j of $\boldsymbol{U}$ and $\boldsymbol{E}$ is a matrix with columns the eigenvalues $\boldsymbol{e}_j$ of $\boldsymbol{U}$.
- Define $\boldsymbol{z}_i = \boldsymbol{E}^\top \boldsymbol{y}_i = \boldsymbol{E}^\top \boldsymbol{D}^{-1} \boldsymbol{x}_i$ and $\boldsymbol{A} = \boldsymbol{D}\boldsymbol{E}$. Then the estimator of $\boldsymbol{\Sigma}$ is $\boldsymbol{C}_{OGK} = \boldsymbol{A}\boldsymbol{\Gamma}\boldsymbol{A}^\top$ where $\boldsymbol{\Gamma} = \text{diag}(\sigma(Z_j)^2), j = 1, \ldots, p$ and the location estimator is $\boldsymbol{T}_{OGK} = \boldsymbol{A}\boldsymbol{m}$ where $\boldsymbol{m} = m(\boldsymbol{z}_i) = (m(Z_1), \ldots, m(Z_p))$ is a robust mean function.

This can be iterated by computing $\boldsymbol{C}_{OGK}$ and $\boldsymbol{T}_{OGK}$ for $\boldsymbol{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$ obtained in the last step of the procedure and then transforming back to the original coordinate system. Simulations (Maronna and Zamar 2002) show that iterations beyond the second did not lead to improvement.

Similarly as for the MCD estimator a one-step reweighting can be performed using Equations (3) but the weights w_i are based on the 0.9 quantile of the χ^2_p distribution (instead of 0.975) and the correction factors $c_r.ccf$ and $c_r.sscf$ are not used.

In order to complete the algorithm we need a robust and efficient location function $m()$ and scale function $\sigma()$, and one proposal is given in Maronna and Zamar (2002). Further, the robust estimate of covariance between two random vectors $s()$ given by Equation (8) can be replaced by another one. In the framework two such functions are predefined but the user can provide as a parameter an own function.

S estimates

S estimators of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ were introduced by Davies (1987) and further studied by Lopuhaä (1989) (see also Rousseeuw and Leroy 1987, p. 263). For a data set of p-variate observations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ an S estimate $(\mathbf{T}, \mathbf{C})$ is defined as the solution of $\sigma(d_1, \ldots, d_n) = \min$ where $d_i = (\boldsymbol{x} - \mathbf{T})^\top \mathbf{C}^{-1} (\boldsymbol{x} - \mathbf{T})$ and $\det(\mathbf{C}) = 1$. Here $\sigma = \sigma(\boldsymbol{z})$ is the M-scale estimate of a data set $\boldsymbol{z} = \{z_1, \ldots, z_n\}$ defined as the solution of $\frac{1}{n}\Sigma\rho(z/\sigma) = \delta$ where ρ is nondecreasing, $\rho(0) = 0$ and $\rho(\infty) = 1$ and $\delta \in (0, 1)$. An equivalent definition is to find the vector $\mathbf{T}$ and a positive definite symmetric matrix $\mathbf{C}$ that minimize $\det(\mathbf{C})$ subject to

$$\frac{1}{n} \sum_{i=1}^{n} \rho(d_i) = b_0 \tag{9}$$

with the above d_i and ρ.

As shown by Lopuhaä (1989) S estimators have a close connection to the M estimators and the solution $(\mathbf{T}, \mathbf{C})$ is also a solution to an equation defining an M estimator as well as a weighted sample mean and covariance matrix:

$$\begin{aligned} d_i^j &= [(\boldsymbol{x}_i - \mathbf{T}^{(j-1)})^\top (\mathbf{C}^{(j-1)})^{-1} (\boldsymbol{x} - \mathbf{T}^{(j-1)})]^{1/2} \\ \mathbf{T}^{(j)} &= \frac{\Sigma w(d_i^{(j)}) \boldsymbol{x}_i}{\Sigma w(d_i^{(j)})} \\ \mathbf{C}^{(j)} &= \frac{\Sigma w(d_i^{(j)})(\boldsymbol{x}_i - \mathbf{T}^{(j)})(\boldsymbol{x}_i - \mathbf{T}^{(j)})^\top}{\Sigma w(d_i^{(j)})} \end{aligned} \tag{10}$$

The framework implements the S estimates in the class `CovSest` and provides four different algorithms for their computation.

1. *SURREAL*: This algorithm was proposed by Ruppert (1992) as an analog to the algorithm proposed by the same author for computing S estimators of regression.

2. *Bisquare S estimation with HBDP start*: S estimates with the biweight ρ function can be obtained using the Equations (10) by a reweighted sample covariance and reweighted sample mean algorithm as described in Maronna *et al.* (2006). The preferred approach is to start the iteration from a bias-robust but possibly inefficient estimate which is computed by some form of sub-sampling. Since Maronna *et al.* (2006) have shown that the MVE has smallest maximum bias (Table 6.2, p. 196) it is recommended to use it as initial estimate.

3. *Rocke type S estimates*: In Rocke (1996) it is shown that S estimators in high dimensions can be sensitive to outliers even if the breakdown point is set to 50%. Therefore they propose a modified ρ function called translated biweight (or t-biweight) and replace the standardization step given in Equation (9) with a standardization step consisting of equating the median of $\rho(d_i)$ with the median under normality. The estimator is shown to be more outlier resistant in high dimensions than the typical S estimators. The specifics of the iteration are given in Rocke and Woodruff (1996), see also Maronna *et al.* (2006). As starting values for the iteration any of the available methods in the framework can be used. The recommended (and consequently the default) one is the MVE estimator computed by `CovMve()`.

4. *Fast S estimates*: Salibian-Barrera and Yohai (2006) proposed a fast algorithm for regression S estimates similar to the FAST-LTS algorithm of Rousseeuw and Van Driessen (2006) and borrowing ideas from the SURREAL algorithm of Ruppert (1992). Similarly, the FAST-S algorithm for multivariate location and scatter is based on modifying each candidate to improve the S-optimality criterion thus reducing the number of the necessary sub-samples required to achieve desired high breakdown point with high probability.

Object model for robust location and scatter estimation

The object model for the S4 classes and methods implementing the different multivariate location and scatter estimators follows the proposed class hierarchy given in Section 2 and is presented in Figure 4.

The abstract class `Cov` serves as a base class for deriving all classes representing classical and robust location and scatter estimation methods. It defines the common slots and the corresponding accessor methods, provides implementation for the general methods like `show()`, `plot()` and `summary()`. The slots of `Cov` hold some input or default parameters as well as the results of the computations: the location, the covariance matrix and the distances. The `show()` method presents brief results of the computations and the `summary()` method returns an object of class `SummaryCov` which has its own `show()` method. As in the other sections of the framework these slots and methods are defined and documented only once in this base class and can be used by all derived classes. Whenever new data (slots) or functionality (methods) are necessary, they can be defined or redefined in the particular class.

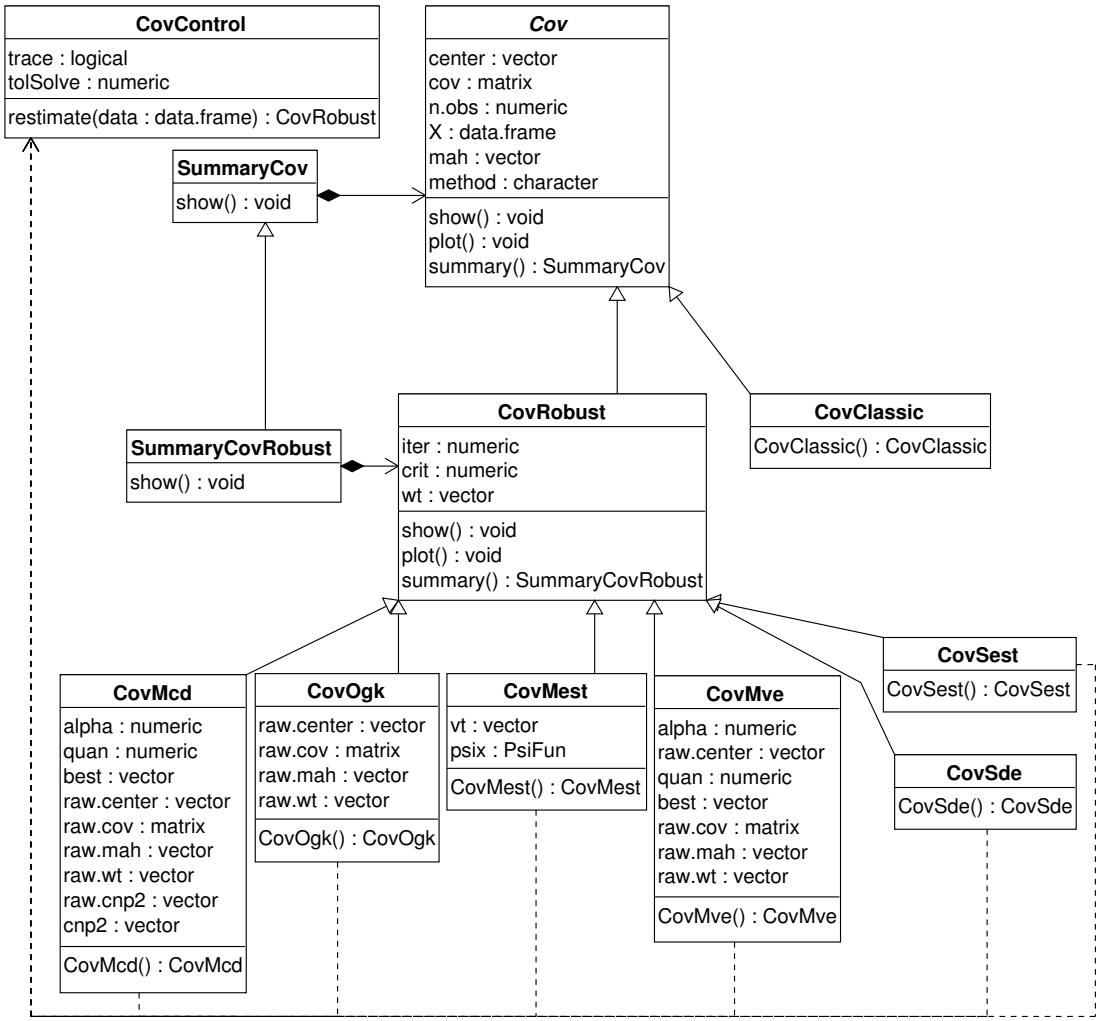

Figure 4: Object model for robust location and scatter estimation.

The classical location and scatter estimates are represented by the class `CovClassic` which inherits directly from `Cov` (and uses all slots and methods defined there). The function `CovClassic()` serves as a constructor (generating function) of the class. It can be called by providing a data frame or matrix. As already demonstrated in Section 3 the methods `show()` and `summary()` present the results of the computations. The `plot()` method draws different diagnostic plots which are shown in one of the next sections. The accessor functions like `getCenter()`, `getCov()`, etc. are used to access the corresponding slots.

Another abstract class, `CovRobust` is derived from `Cov`, which serves as a base class for all robust location and scatter estimators.

The classes representing robust estimators like `CovMcd`, `CovMve`, etc. are derived from `CovRobust` and provide implementation for the corresponding methods. Each of the constructor functions `CovMcd()`, `CovMve()`,`CovOgk()`, `CovMest()` and `CovSest()` performs the necessary computations and returns an object of the class containing the results. Similarly as the `CovClassic()` function, these functions can be called either with a data frame or a numeric matrix.

Controlling the estimation options

Although the different robust estimators of multivariate location and scatter have some controlling options in common, like the tracing flag `trace` or the numeric tolerance `tolSolve` to be used for inversion (`solve`) of the covariance matrix in `mahalanobis()`, each of them has more specific options. For example, the MCD and MVE estimators (`CovMcd()` and `CovMve()`) can specify `alpha` which controls the size of the subsets over which the determinant (the volume of the ellipsoid) is minimized. The allowed values are between 0.5 and 1 and the default is 0.5. Similarly, these estimators have parameters `nsamp` for the number of subsets used for initial estimates and `seed` for the initial seed for R's random number generator while the M and S estimators (`CovMest` and `CovSest`) have to specify the required breakdown point (allowed values between $(n-p)/(2*n)$ and 1 with default 0.5) as well as the asymptotic rejection point, i.e., the fraction of points receiving zero weight (Rocke and Woodruff 1996).

These parameters can be passed directly to the corresponding constructor function but additionally there exists the possibility to use a control object serving as a container for the parameters. The object model for the control objects shown in Figure 5 follows the proposed class hierarchy—there is a base class `CovControl` which holds the common parameters and from this class all control classes holding the specific parameters of their estimators are derived. These classes have only a constructor function for creating new objects and a method `restimate()` which takes a data frame or a matrix, calls the corresponding estimator to perform the calculations and returns the created class with the results.

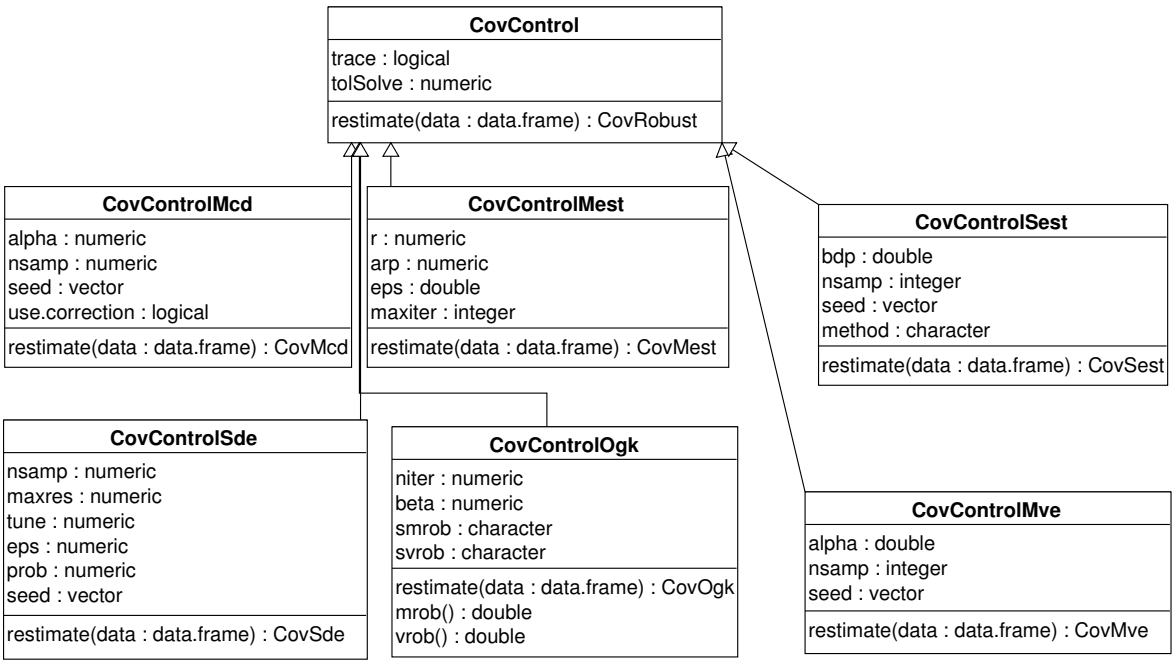

Figure 5: Object model of the control classes for robust location and scatter estimation.

Apart from providing a structured container for the estimation parameters this class hierarchy has the following additional benefits:

- the parameters can be passed easily to another multivariate method, for example the

principal components analysis based on a covariance matrix `PcaCov()` (see Section 4.2) can take a control object which will be used to estimate the desired covariance (or correlation) matrix. In the following example a control object holding the parameters for S estimation will be created and then `PcaCov()` will be called with this object.

```
R> control <- CovControlSest(method = "biweight")
R> PcaCov(hbk.x, cov.control = control)

Call:
PcaCov(x = hbk.x, cov.control = control)

Standard deviations:
[1] 1.409393 1.262539 1.142856

Loadings:
          PC1         PC2        PC3
X1 -0.5256781  0.74354209 -0.4132888
X2 -0.6219784 -0.66738398 -0.4095626
X3 -0.5803494  0.04175861  0.8132963
```

- the class hierarchy of the control objects allows to handle different estimator objects using a uniform interface thus leveraging one of the most important features of the object oriented programming, the polymorphism. In the following example we create a list containing different control objects and then via `sapply` we call the generic function `restimate()` on each of the objects in the list. The outcome will be a list containing the objects resulting from these calls (all are derived from `CovRobust`). This looping over the different estimation methods is very useful for implementing simulation studies.

```
R> cc <- list(CovControlMcd(), CovControlMest(), CovControlOgk(),
+     CovControlSest(), CovControlSest(method = "rocke"))
R> clist <- sapply(cc, restimate, x = delivery.x)
R> sapply(clist, data.class)

[1] "CovMcd"  "CovMest" "CovOgk"  "CovSest" "CovSest"

R> sapply(clist, getMeth)

[1] "Minimum Covariance Determinant Estimator."
[2] "M-Estimates"
[3] "Orthogonalized Gnanadesikan-Kettenring Estimator"
[4] "S-estimates: S-FAST"
[5] "S-estimates: Rocke type"
```

A generalized function for robust location and covariance estimation: `CovRobust()`

The provided variety of estimation methods, each of them with different parameters as well as the object models described in the previous sections can be overwhelming for the user, especially for the novice who does not care much about the technical implementation of

the framework. Therefore a function is provided which gives a quick access to the robust estimates of location and covariance matrix. This function is loosely modeled around the *abstract factory design pattern* (see Gamma *et al.* 1995, page 87) in the sense that it creates concrete objects of derived classes and returns the result over a base class interface. The class `CovRobust` is abstract (defined as *VIRTUAL*) and no objects of it can be created but any of the classes derived from `CovRobust`, such as `CovMcd` or `CovOgk`, can act as an object of class `CovRobust`. The function `CovRobust()` which is technically not a constructor function can return an object of any of the classes derived from `CovRobust` according to the user request. This request can be specified in one of three forms:

- If only a data frame or matrix is provided and the control parameter is omitted, the function decides which estimate to apply according to the size of the problem at hand. If there are less than 1000 observations and less than 10 variables or less than 5000 observations and less than 5 variables, Stahel-Donoho estimator will be used. Otherwise, if there are less than 50000 observations, either bisquare S estimates (in case of less than 10 variables) or Rocke type S estimates (for 10 to 20 variables) will be used. In both cases the S iteration starts at the initial MVE estimate. And finally, if there are more than 50000 observations and/or more than 20 variables the Orthogonalized Quadrant Correlation estimator (`CovOgk` with the corresponding parameters) is used. This is illustrated by the following example.

```
R> getMeth(CovRobust(matrix(rnorm(40), ncol = 2)))

[1] "Stahel-Donoho estimator"

R> getMeth(CovRobust(matrix(rnorm(16000), ncol = 8)))

[1] "S-estimates: bisquare"

R> getMeth(CovRobust(matrix(rnorm(20000), ncol = 10)))

[1] "S-estimates: Rocke type"

R> getMeth(CovRobust(matrix(rnorm(2e+05), ncol = 2)))

[1] "Orthogonalized Gnanadesikan-Kettenring Estimator"
```

- The simplest way to choose an estimator is to provide a character string with the name of the estimator—one of `"mcd"`, `"ogk"`, `"m"`, `"s-fast"`, `"s-rocke"`, etc.

```
R> getMeth(CovRobust(matrix(rnorm(40), ncol = 2), control = "rocke"))

[1] "S-estimates: Rocke type"
```

- If it is necessary to specify also some estimation parameters, the user can create a control object (derived from `CovControl`) and pass it to the function together with the data. For example to compute the OGK estimator using the median absolute deviation (MAD) as a scale estimate and the quadrant correlation (QC) as a pairwise correlation estimate we create a control object `ctrl` passing the parameters `s_mad` and `s_qc` to the

constructor function and then call `CovRobust` with this object. The last command line illustrates the accessor method for getting the correlation matrix of the estimate as well as a nice formatting method for covariance matrices.

```
R> data("toxicity")
R> ctrl <- CovControlOgk(smrob = "s_mad", svrob = "qc")
R> est <- CovRobust(toxicity, ctrl)
R> round(getCenter(est), 2)

toxicity   logKow      pKa    ELUMO    Ecarb     Emet       RM       IR
   -0.20     1.40     0.40     4.01    16.99     3.25    35.41     1.46
      Ts        P
   41.72     1.46

R> as.dist(round(getCorr(est), 2))

       toxicity logKow   pKa ELUMO Ecarb  Emet    RM    IR    Ts
logKow     0.72
pKa       -0.41  -0.13
ELUMO     -0.26   0.23  0.33
Ecarb      0.19   0.68  0.48  0.61
Emet       0.56   0.87  0.16  0.09  0.68
RM         0.37   0.81  0.35  0.30  0.90  0.88
IR        -0.20   0.17  0.41  0.18  0.58  0.22  0.59
Ts        -0.57  -0.49  0.45 -0.17 -0.04 -0.27  0.00  0.70
P         -0.20   0.17  0.41  0.18  0.58  0.22  0.59  1.00  0.70
```

Visualization of the results

The default plot accessed through the method `plot()` of class `CovRobust` is the Distance-Distance plot introduced by Rousseeuw and van Zomeren (1991). An example of this graph, which plots the robust distances versus the classical Mahalanobis distances is shown in Figure 3. The dashed line represents the points for which the robust and classical distances are equal. The horizontal and vertical lines are drawn at values $x = y = \sqrt{\chi^2_{p,0.975}}$. Points beyond these lines can be considered as outliers and are identified by their labels.

The other available plots are accessible either interactively or through the `which` parameter of the `plot()` method. The left panel of Figure 6 shows an example of the distance plot in which robust and classical Mahalanobis distances are shown in parallel panels. The outliers have large robust distances and are identified by their labels. The right panel of Figure 6 shows a Quantile-Quantile comparison plot of the robust and the classical Mahalanobis distances versus the square root of the quantiles of the chi-squared distribution.

The next plot shown in Figure 7 presents a scatter plot of the data on which the 97.5% robust and classical confidence ellipses are superimposed. Currently this plot is available only for bivariate data. The observations with distances larger than $\sqrt{\chi^2_{p,0.975}}$ are identified by their

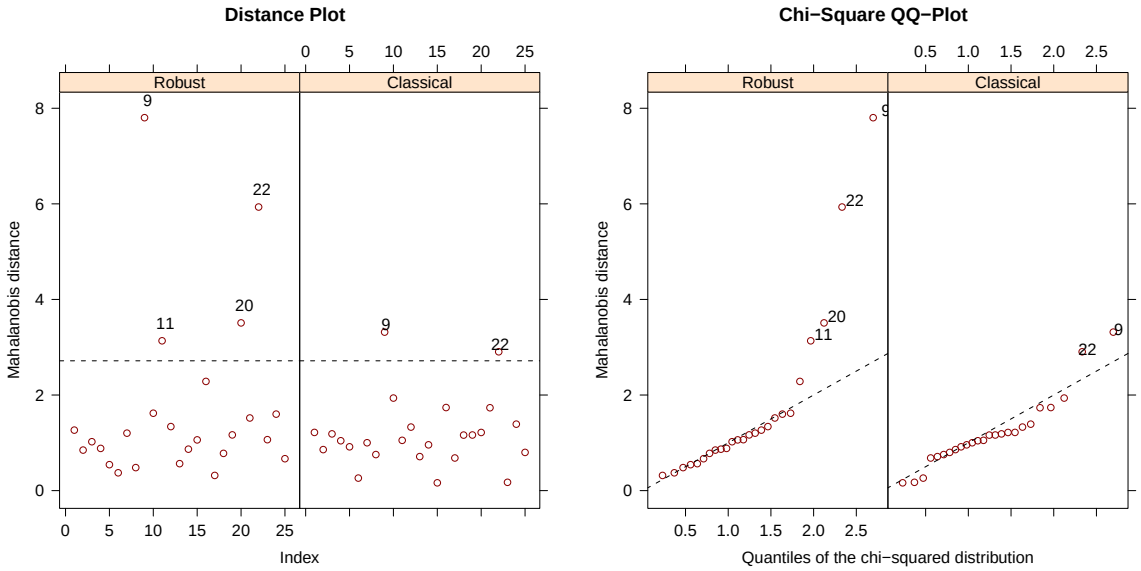

Figure 6: Distance plot and Chi-square Q-Q plot of the robust and classical distances.

subscript. In the right panel of Figure 7 a `screeplot` of the `milk` data set is shown, presenting the robust and classical eigenvalues.

```
R> data("milk")
R> usr <- par(mfrow = c(1, 2))
R> plot(CovMcd(delivery[, 1:2]), which = "tolEllipsePlot", classic = TRUE)
R> plot(CovMcd(milk), which = "screeplot", classic = TRUE)
R> par(usr)
```

4.2. Principal component analysis

Principal component analysis is a widely used technique for dimension reduction achieved by finding a smaller number q of linear combinations of the originally observed p variables and retaining most of the variability of the data. Thus PCA is usually aiming at a graphical representation of the data in a lower dimensional space. The classical approach to PCA measures the variability through the empirical variance and is essentially based on computation of eigenvalues and eigenvectors of the sample covariance or correlation matrix. Therefore the results may be extremely sensitive to the presence of even a few atypical observations in the data. These discrepancies will carry over to any subsequent analysis and to any graphical display related to the principal components such as the biplot.

The following example in Figure 8 illustrates the effect of outliers on the classical PCA. The data set `hbk` from the package **robustbase** consists of 75 observations in 4 dimensions (one response and three explanatory variables) and was constructed by Hawkins, Bradu and Kass in 1984 for illustrating some of the merits of a robust technique (see Rousseeuw and Leroy 1987). The first 10 observations are bad leverage points, and the next four points are good

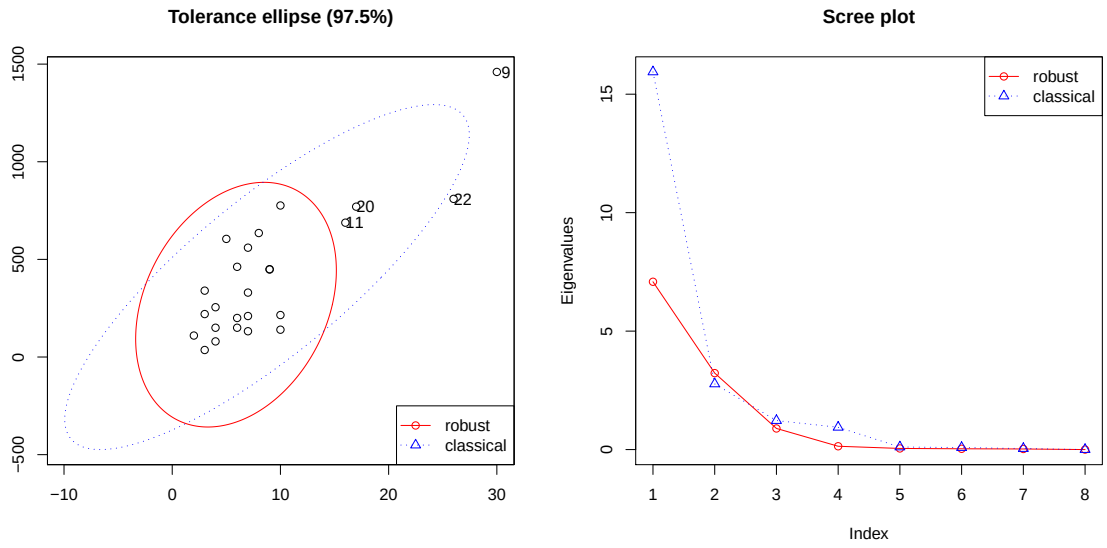

Figure 7: Robust and classical tolerance ellipse for the `delivery` data and robust and classical screeplot for the `milk` data.

leverage points (i.e., their $\boldsymbol{x}$ are outlying, but the corresponding y fit the model quite well). We will consider only the X-part of the data. The left panel shows the plot of the scores on the first two classical principal components (the first two components account for more than 98% of the total variation). The outliers are identified as separate groups, but the regular points are far from the origin (where the mean of the scores should be located). Furthermore, the ten bad leverage points 1–10 lie within the 97.5% tolerance ellipse and influence the classical estimates of location and scatter. The right panel shows the same plot based on robust estimates. We see that the estimate of the center is not shifted by the outliers and these outliers are clearly separated by the 97.5% tolerance ellipse.

PCA was probably the first multivariate technique subjected to robustification, either by simply computing the eigenvalues and eigenvectors of a robust estimate of the covariance matrix or directly by estimating each principal component in a robust manner. Different approaches to robust PCA are briefly presented in the next subsections with the emphasis on those methods which are available in the framework. Details about the methods and algorithms can be found in the corresponding references. The object model is described and examples are given.

PCA based on robust covariance matrix (MCD, OGK, MVE, etc.)

The most straightforward and intuitive method to obtain robust PCA is to replace the classical estimates of location and covariance by their robust analogues. In the earlier works M estimators of location and scatter were used for this purpose (see Devlin *et al.* 1981; Campbell 1980) but these estimators have the disadvantage of low breakdown point in high dimensions. To cope with this problem Naga and Antille (1990) used the MVE estimator and Todorov *et al.*

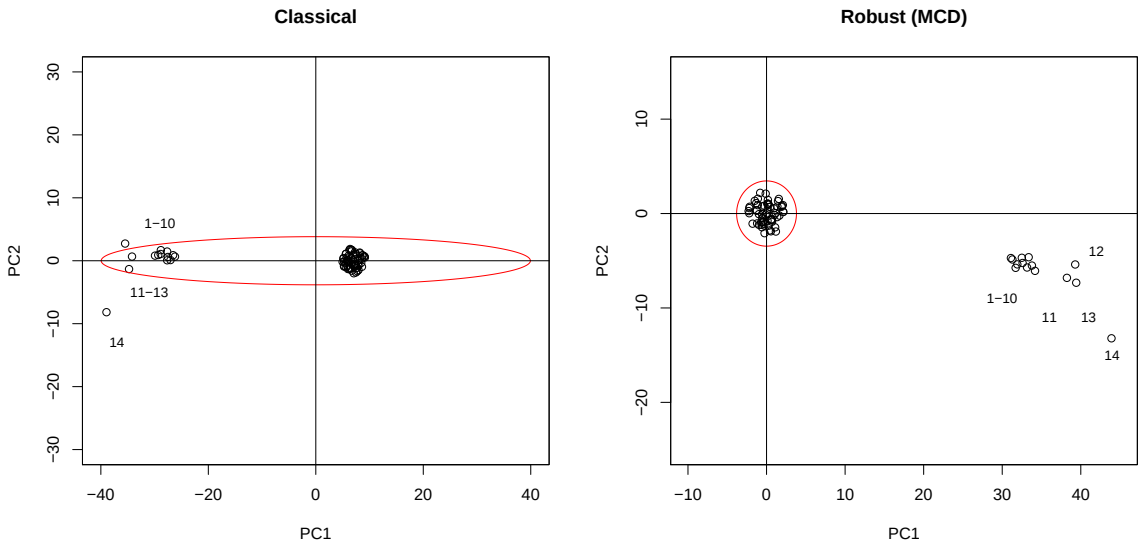

Figure 8: Plot of the first two principal components of the Hawkins, Bradu and Kass data set: classical and robust.

(1994b) used the MCD estimator. Croux and Haesbroeck (2000) investigated the properties of the MCD estimator and computed its influence function and efficiency.

The package **stats** in base R contains the function `princomp()` which performs a principal components analysis on a given numeric data matrix and returns the results as an object of S3 class `princomp`. This function has a parameter `covmat` which can take a covariance matrix, or a covariance list as returned by `cov.wt`, and if supplied, it is used rather than the covariance matrix of the input data. This allows to obtain robust principal components by supplying the covariance matrix computed by `cov.mve` or `cov.mcd` from the package **MASS**. One could ask why is it then necessary to include such type of function in the framework (since it already exists in the base package). The essential value added of the framework, apart from implementing many new robust multivariate methods is the unification of the interfaces by leveraging the object orientation provided by the S4 classes and methods. The function `PcaCov()` computes robust PCA by replacing the classical covariance matrix with one of the robust covariance estimators available in the framework—MCD, OGK, MVE, M, S or Stahel-Donoho, i.e., the parameter `cov.control` can be any object of a class derived from the base class `CovControl`. This control class will be used to compute a robust estimate of the covariance matrix. If this parameter is omitted, MCD will be used by default. Of course any newly developed estimator following the concepts of the framework can be used as input to the function `PcaCov()`.

Projection pursuit methods

The second approach to robust PCA uses *projection pursuit* (PP) and calculates directly the robust estimates of the eigenvalues and eigenvectors. Directions are seeked for, which maximize the variance (classical PCA) of the data projected onto them. Replacing the variance with a robust measure of spread yields robust PCA. Such a method was first introduced

by Li and Chen (1985) using an M estimator of scale S_n as a *projection index (PI)*. They showed that the PCA estimates inherit the robustness properties of the scale estimator S_n. Unfortunately, in spite of the good statistical properties of the method, the algorithm they proposed was too complicated to be used in practice. A more tractable algorithm in these lines was first proposed by Croux and Ruiz-Gazen (1996) and later improved by Croux and Ruiz-Gazen (2005). To improve the performance of the algorithm for high dimensional data a new improved version was proposed by Croux *et al.* (2007). The latter two algorithms are available in the package **pcaPP** (see Filzmoser *et al.* 2009) as functions `PCAproj()` and `PCAgrid()`.

In the framework these methods are represented by the classes `PcaProj` and `PcaGrid`. Their generating functions provide simple wrappers around the original functions from **pcaPP** and return objects of the corresponding class, derived from `PcaRobust`.

A major advantage of the PP-approach is that it searches for the eigenvectors consecutively and in case of high dimensional data when we are interested in only the first one or two principal components this results in reduced computational time. Even more, the PP-estimates cope with the main drawback of the covariance-based estimates—they can be computed for data matrices with more variables than observations.

Hubert method (ROBPCA)

The PCA method proposed by Hubert *et al.* (2005) tries to combine the advantages of both approaches—the PCA based on a robust covariance matrix and PCA based on projection pursuit. A brief description of the algorithm follows, for details see the relevant references (Hubert *et al.* 2008).

Let n denote the number of observations, and p the number of original variables in the input data matrix $\mathbf{X}$. The ROBPCA algorithm finds a robust center $\boldsymbol{m}$ of the data and a loading matrix $\mathbf{P}$ of dimension $p \times k$. Its columns are orthogonal and define a new coordinate system. The scores $\mathbf{T}$, an $n \times k$ matrix, are the coordinates of the centered observations with respect to the loadings:

$$\mathbf{T} = (\mathbf{X} - \mathbf{1}\boldsymbol{m}^\top)\mathbf{P} \tag{11}$$

where $\mathbf{1}$ is a column vector with all n components equal to 1. The ROBPCA algorithm yields also a robust covariance matrix (often singular) which can be computed as

$$\mathbf{S} = \mathbf{P}\mathbf{L}\mathbf{P}^\top \tag{12}$$

where $\mathbf{L}$ is the diagonal matrix with the eigenvalues $l_1, \ldots, l_k$. This is done in the following three main steps:

Step 1: The data are preprocessed by reducing their data space to the subspace spanned by the n observations. This is done by singular value decomposition of the input data matrix. As a result the data are represented in a space whose dimension is $rank(\mathbf{X})$, being at most $n - 1$ without loss of information.

Step 2: In this step a measure of outlyingness is computed for each data point. For this purpose the data points are projected on the $n(n-1)/2$ univariate directions through each two points. If n is too large, `maxdir` directions are chosen at random (`maxdir` defaults to

250 but can be changed by the user). On every direction the univariate MCD estimator of location and scale is computed and the standardized distance to the center is measured. The largest of these distances (over all considered directions) is the outlyingness measure of the data point. The h data points with smallest outlyingness measure are used to compute the covariance matrix $\mathbf{\Sigma}_h$ and to select the number k of principal components to retain. This is done by finding k such that $l_k/l_1 \geq 10^{-3}$ and $\Sigma_{j=1}^{k} l_j / \Sigma_{j=1}^{r} l_j \geq 0.8$. Alternatively the number of principal components k can be specified by the user after inspecting the scree plot.

Step 3: The data points are projected on the k-dimensional subspace spanned by the k eigenvectors corresponding to the largest k eigenvalues of the matrix $\mathbf{\Sigma}_h$. The location and scatter of the projected data are computed using the reweighted MCD estimator, and the eigenvectors of this scatter matrix yield the robust principal components.

Spherical principal components (SPC)

The spherical principal components procedure was first proposed by Locantore *et al.* (1999) as a method for functional data analysis. The idea is to perform classical PCA on the data, projected onto a unit sphere. The estimates of the eigenvectors are consistent if the data are elliptically distributed (see Boente and Fraiman 1999) and the procedure is extremely fast. Although not much is known about the efficiency of this method, the simulations of Maronna (2005) show that it has very good performance. If each coordinate of the data is normalized using some kind of robust scale, like for example the MAD, and then SPC is applied, we obtain "elliptical PCA", but unfortunately this procedure is not consistent.

Object model for robust PCA and examples

The object model for the S4 classes and methods implementing the principal component analysis methods follows the proposed class hierarchy given in Section 2 and is presented in Figure 9. The abstract class `Pca` serves as a base class for deriving all classes representing classical and robust principal components analysis methods. It defines the common slots and the corresponding accessor methods, provides implementation for the general methods like `show()`, `plot()`, `summary()` and `predict()`. The slots of `Pca` hold some input or default parameters like the requested number of components as well as the results of the computations: the eigenvalues, the loadings and the scores. The `show()` method presents brief results of the computations, and the `predict()` method projects the original or new data to the space spanned by the principal components. It can be used either with new observations or with the scores (if no new data are provided). The `summary()` method returns an object of class `SummaryPca` which has its own `show()` method. As in the other sections of the framework these slots and methods are defined and documented only once in this base class and can be used by all derived classes. Whenever new information (slots) or functionality (methods) are necessary, they can be defined or redefined in the particular class.

Classical principal component analysis is represented by the class `PcaClassic` which inherits directly from `Pca` (and uses all slots and methods defined there). The function `PcaClassic()` serves as a constructor (generating function) of the class. It can be called either by providing a data frame or matrix or a formula with no response variable, referring only to numeric variables. Let us consider the following simple example with the data set `hbk` from the package **robustbase**. The code line

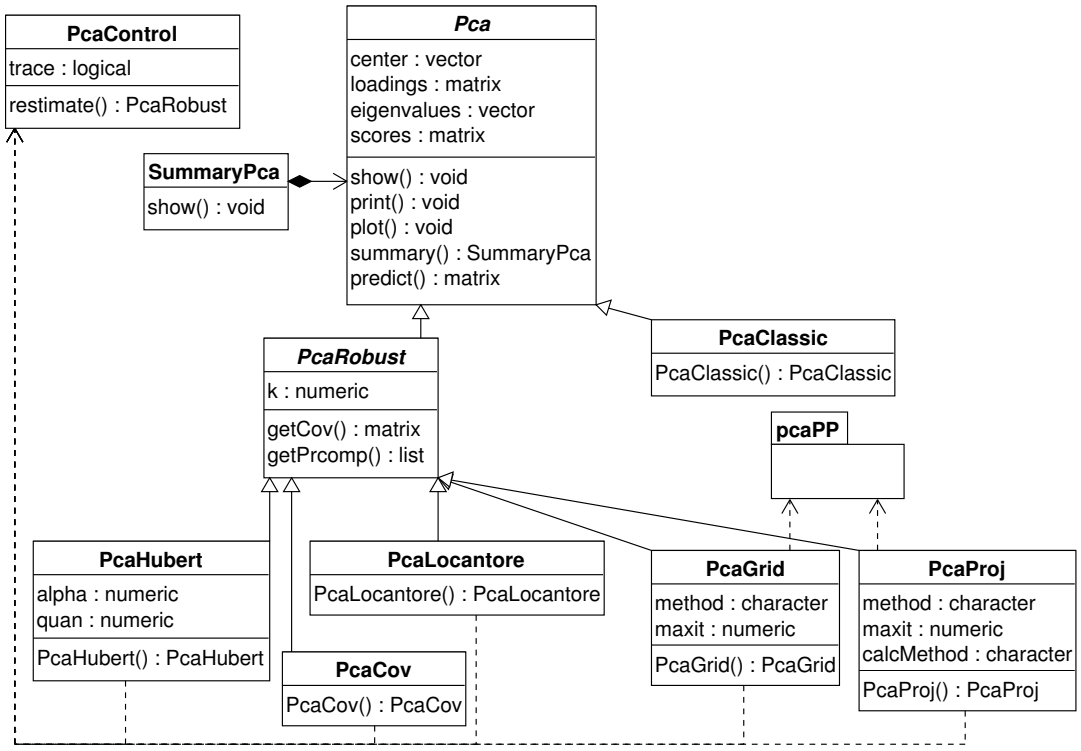

Figure 9: Object model for robust Principal Component Analysis.

```
R> PcaClassic(hbk.x)
```

can be rewritten as (and is equivalent to) the following code line using the formula interface

```
R> PcaClassic(~ ., data = hbk.x)
```

The function `PcaClassic()` performs the standard principal components analysis and returns an object of the class `PcaClassic`.

```
R> pca <- PcaClassic(~., data = hbk.x)
R> pca

Call:
PcaClassic(formula = ~., data = hbk.x)

Standard deviations:
[1] 14.7024532  1.4075073  0.9572508

Loadings:
          PC1        PC2         PC3
X1 -0.2398767  0.1937359 -0.95127577
X2 -0.5547042 -0.8315255 -0.02947174
X3 -0.7967198  0.5206071  0.30692969
```

```
R> summary(pca)

Call:
PcaClassic(formula = ~., data = hbk.x)
Importance of components:
                           PC1     PC2     PC3
Standard deviation     14.7025 1.40751 0.95725
Proportion of Variance  0.9868 0.00904 0.00418
Cumulative Proportion   0.9868 0.99582 1.00000

R> plot(pca)
R> getLoadings(pca)

          PC1        PC2         PC3
X1 -0.2398767  0.1937359 -0.95127577
X2 -0.5547042 -0.8315255 -0.02947174
X3 -0.7967198  0.5206071  0.30692969
```

The `show()` method displays the standard deviations of the resulting principal components, the loadings and the original call. The `summary()` method presents the importance of the calculated components. The `plot()` draws a PCA diagnostic plot which is shown and described later. The accessor functions like `getLoadings()`, `getEigenvalues()`, etc. are used to access the corresponding slots, and `predict()` is used to rotate the original or new data to the space of the principle components.

Another abstract class, `PcaRobust` is derived from `Pca`, which serves as a base class for all robust principal components methods.

The classes representing robust PCA methods like `PcaHubert`, `PcaLocantore`, etc. are derived from `PcaRobust` and provide implementation for the corresponding methods. Each of the constructor functions `PcaCov()`, `PcaHubert()`, `PcaLocantore()`, `PcaGrid()` and `PcaProj()` performs the necessary computations and returns an object of the class containing the results. In the following example the same data are analyzed using the projection pursuit method `PcaGrid()`.

```
R> rpca <- PcaGrid(~., data = hbk.x)
R> rpca

Call:
PcaGrid(formula = ~., data = hbk.x)

Standard deviations:
[1] 1.971022 1.698191 1.469397

Loadings:
          PC1         PC2         PC3
X1 0.99359515  0.11000558 -0.02583502
X2 0.03376011 -0.07080059  0.99691902
X3 0.10783752 -0.99140610 -0.07406093
```

```
R> summary(rpca)

Call:
PcaGrid(formula = ~., data = hbk.x)
Importance of components:
                          PC1    PC2    PC3
Standard deviation     1.9710 1.6982 1.4694
Proportion of Variance 0.4351 0.3230 0.2418
Cumulative Proportion  0.4351 0.7582 1.0000
```

Similar to the function `PcaClassic()`, these functions can be called either with a data frame or matrix or by a formula interface.

Visualization of PCA results

One of the most important applications of PCA, besides dimensionality reduction is data visualization. In the framework several plots for visualizing the results of the analysis are available. The `plot()` methods are implemented in the base class `Pca` and thus they are available for all objects derived from the class `Pca` no matter if classical and robust. The most straightforward plot is the *screeplot* which plots the variances against the number of principal components (similar to the screeplot for the standard `prcomp()` and `princomp()` functions). It is a useful tool for determining the number of relevant principal components. An example of the classical and robust screeplot for the `milk` data from **robustbase** is shown in Figure 10.

```
R> ##
R> ## Screeplot for classical and robust PCA of the milk data set.
R> ##
R> usr <- par(mfrow=c(1,2))
R> screeplot(PcaClassic(milk), type="lines",
+ main="Screeplot: classical PCA", sub="milk data")
R> screeplot(PcaHubert(milk), type="lines", main="Screeplot: robust PCA",
+ sub="milk data")
R> par(usr)
```

Another plot borrowed from standard R is the *biplot*. The biplot (Gabriel 1971) represents both the observations and variables in the plane of (the first) two principal components allowing the visualization of the magnitude and sign of each variable's contribution to these principal components. Each observation (row of scores) is represented as a point in the biplot and each variable is represented as an arrow. The arrows graphically indicate the proportion of the original variance explained by the (first) two principal components and their direction indicates the relative loadings on these components. Figure 11 shows an example of the robust biplot of the `un86` data set which contains seven socioeconomic variables observed for 73 countries. The data set is from World Statistics in Brief, Number 10, a 1986 UN publication. It was used in Daigle and Rivest (1992) to illustrate a robust biplot method.

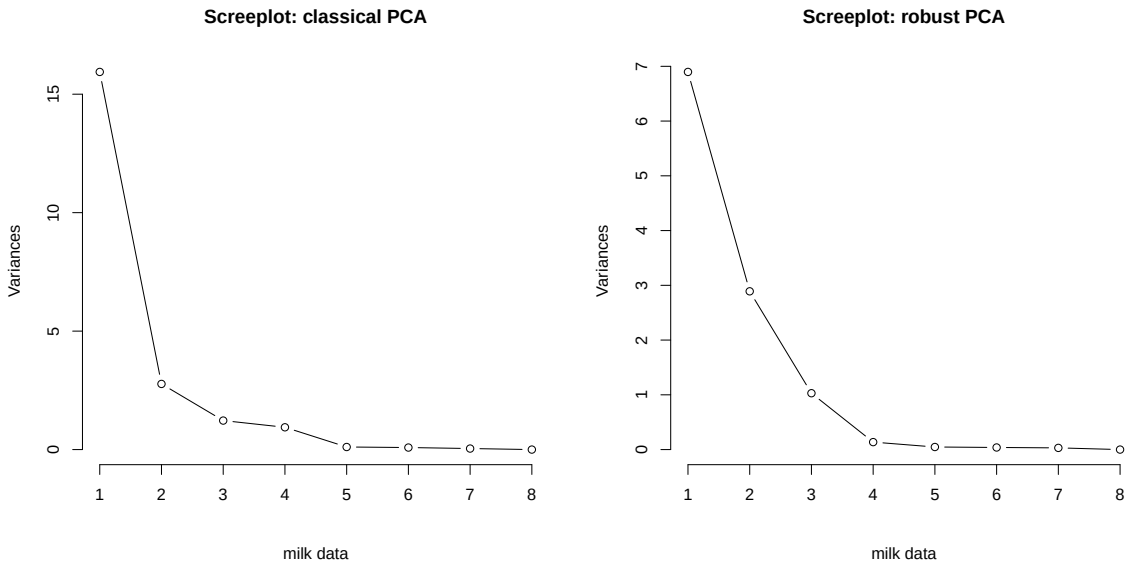

Figure 10: Screeplot for classical and robust PCA of the `milk` data set.

```
R> ##
R> ## Robust biplot for the UN86 data
R> ##
R> data("un86")
R> set.seed(9)
R> usr<-par(mfrow=c(1,2))
R> biplot(PcaCov(un86, corr=TRUE, cov.control=NULL),
+ main="Classical biplot", col=c("gray55", "red"))
R> biplot(PcaCov(un86, corr=TRUE), main="Robust biplot",
+ col=c("gray55", "red"))
R> par(usr)
```

In the context of PCA Hubert *et al.* (2005) defined a *diagnostic plot* or *outlier map* which helps to distinguish between the regular observations and the different types of outliers. This plot is based on the *score distances* and *orthogonal distances* computed for each observation. The score distances are given by

$$SD_i = \sqrt{\sum_{j=1}^{k} \frac{t_{ij}^2}{l_j}}, \qquad i = 1, \ldots, n \tag{13}$$

where t_{ij} are the elements of the scores from (11) and l_j are the eignevalues (the diagonal elements of the matrix $\mathbf{L}$ in (12)). The orthogonal distances OD_i of each observation to the subspace spanned by the first k ($1 \leq k \leq r$, r is the rank of the data) principal components are defined by

$$OD_i = ||\boldsymbol{x}_i - \hat{\boldsymbol{\mu}} - \mathbf{P}^{(k)}\boldsymbol{t}_i^{(k)}||, \qquad i = 1, \ldots, n \tag{14}$$

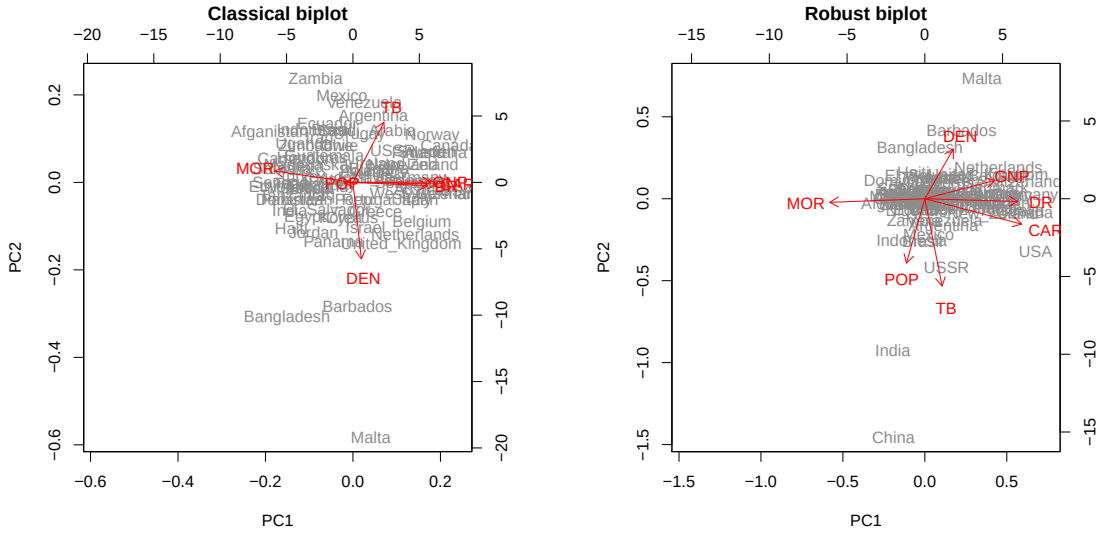

Figure 11: Classical (left panel) and robust (right panel) biplot for the `UN86` data.

where $\boldsymbol{x}_i$ denotes the ith observation, $\hat{\boldsymbol{\mu}}$ is the estimated center of the data, $\boldsymbol{t}_i^{(k)}$ is the ith score vector in the space of the first k principal components and the matrix $\mathbf{P}^{(k)}$ contains the first k estimated eigenvectors in its columns. The diagnostic plot is constructed by plotting the score distances on the horizontal axis, the orthogonal distances on the vertical axis and drawing two cutoff lines which will help to classify the observations. The cutoff value on the horizontal axis (for the score distances) is taken as the 97.5% quantile of χ^2_k distribution, i.e., $c_h = \sqrt{\chi^2_{k,0.975}}$. For the cutoff value on the vertical axis (for the orthogonal distances) the Wilson-Hilferty transformation for a χ^2 distribution is used (which assumes that the OD_i to the power of 2/3 are approximately normally distributed). The parameters μ and σ of the normal distribution can be estimated by the median and MAD of the values $OD_i^{2/3}$, and the critical value can be taken as $c_v = (\hat{\mu} + \hat{\sigma} z_{0.975})^{3/2}$ where $z_{0.975}$ is the the 97.5% quantile of the standard normal distribution.

An example of the classical and robust diagnostic plot for the `hbk` data set from **robustbase** is shown in Figure 12.

```
R> usr <- par(mfrow = c(1, 2))
R> plot(PcaClassic(hbk.x, k = 2), sub = "data set: hbk, k=2")
R> plot(PcaHubert(hbk.x, k = 2), sub = "data set: hbk, k=2")
R> par(usr)
```

If $k = p$ the orthogonal distances are not meaningful and the diagnostic plot shows a simple distance plot of the score distances (distances vs index). An example is shown in Figure 13.

```
R> usr <- par(mfrow = c(1, 2))
R> plot(PcaClassic(hbk.x, k = 3), sub = "data set: hbk.x, k=3")
```

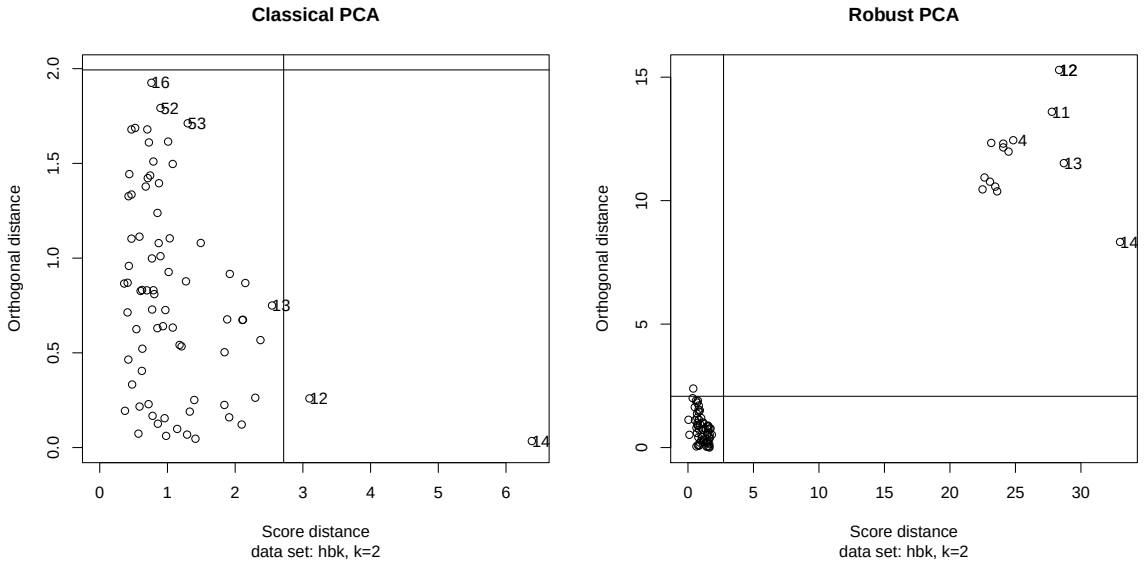

Figure 12: Classical and robust diagnostic plot for the `hbk` data with $k = 2$.

```
R> plot(PcaHubert(hbk.x, k = 3), sub = "data set: hbk.x, k=3")
R> par(usr)
```

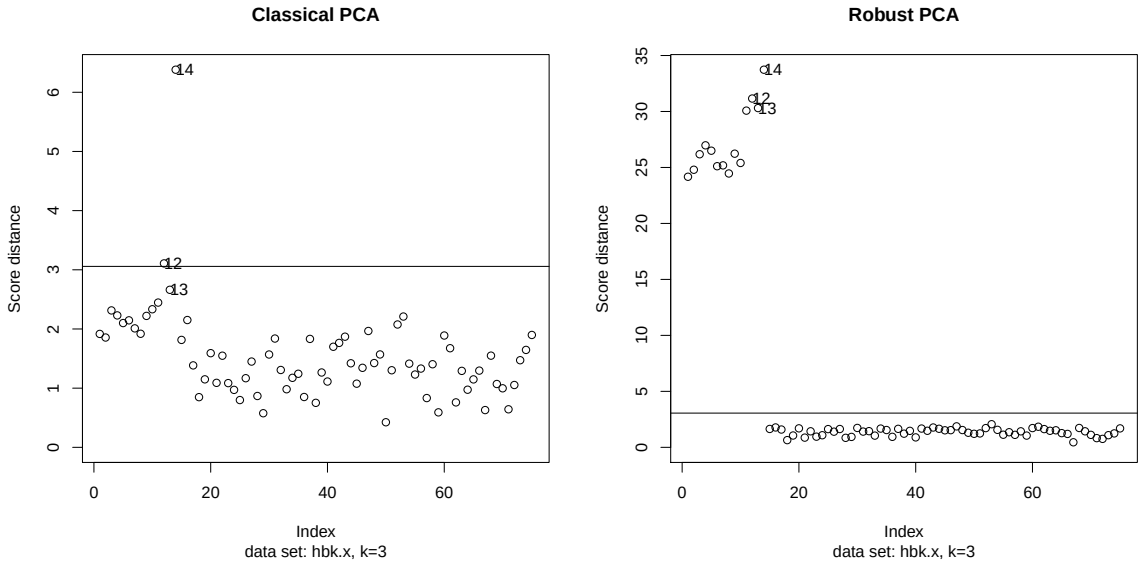

Figure 13: Classical and robust diagnostic plot for the x-part of the `hbk` data set with $k = 3 = p$.

4.3. Linear and quadratic discriminant analysis

The problem of discriminant analysis arises when one wants to assign an individual to one of g populations at the basis of a p-dimensional feature vector $\boldsymbol{x}$. Let the p-dimensional random variable $\boldsymbol{x}_k$ come from a population π_k with underlying density $\mathbf{f}_k$. Further let the prior probability of group k, i.e., the probability that an individual comes from population π_k be α_k, $\Sigma_{k=1}^{g}\alpha_k = 1$. The prior probabilities α_k are usually estimated by the empirical frequencies n_k in the k-th group of the training set, i.e., $\hat{\alpha}_k = n_k/\Sigma_{j=1}^{g} n_j$. Then the Bayesian discriminant rule assigns an observation $\boldsymbol{x}$ to that population π_k for which the expression $\ln(\alpha_k \mathbf{f}_k(\boldsymbol{x}))$ is maximal over all groups $k = 1, \ldots, g$. Usually it is assumed that the k populations π_k are p-dimensional normally distributed,

$$\pi_k \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \qquad k = 1, \ldots, g. \tag{15}$$

With this assumption the discriminant rule is equivalent to maximizing the discriminant scores $D_k(\boldsymbol{x})$ given by

$$D_k(\boldsymbol{x}) = -\frac{1}{2}\ln|\boldsymbol{\Sigma}_k| - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k) + \ln(\alpha_k) \qquad (k = 1, \ldots, g), \tag{16}$$

and individual $\boldsymbol{x}$ is assigned to π_k if

$$D_k(\boldsymbol{x}) = \arg\max_j D_j(\boldsymbol{x}). \tag{17}$$

The application of the discrimination rule given by Equations (16) and (17) is referred to as quadratic discriminant analysis (QDA), since the groups are separated by quadratic boundaries.

If it is further assumed that all group covariance matrices are equal ($\boldsymbol{\Sigma}_1 = \ldots = \boldsymbol{\Sigma}_g = \boldsymbol{\Sigma}$), then the overall probability of misclassification is minimized by assigning a new observation $\boldsymbol{x}$ to population π_k which maximizes

$$d_k(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln(\alpha_k) \qquad (k = 1, \ldots, g). \tag{18}$$

The application of the discriminant rule given by Equation (18) is referred to as linear discriminant analysis (LDA), since the scores $d_k(\boldsymbol{x})$ are linear in $\boldsymbol{x}$.

If the means $\boldsymbol{\mu}_k, k = 1, \ldots, g$, and the common covariance matrix $\boldsymbol{\Sigma}$ are unknown, which is usually the case, a training set consisting of samples drawn from each of the populations is required. In classical QDA and LDA the sample group means and sample covariance matrices are used to estimate $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and $\boldsymbol{\Sigma}$. The prior probabilities can be estimated by the relative frequencies of observations in each particular group. Both QDA and LDA using the classical estimates in (16) and (18) are vulnerable to the presence of outliers. The problem of the non-robustness of the classical estimates in the setting of the quadratic and linear discriminant analysis has been addressed by many authors: Todorov, Neykov, and Neytchev (1990, 1994a) replaced the classical estimates by MCD estimates; Chork and Rousseeuw (1992) used MVE instead; Hawkins and McLachlan (1997) defined the minimum within-group covariance determinant estimator (MWCD) especially for the case of linear discriminant analysis; He and Fung (2000) and Croux and Dehon (2001) used S estimates; Hubert and Van Driessen (2004) applied the MCD estimates computed by the FAST-MCD algorithm. For a recent review and comparison of these methods see Todorov and Pires (2007).

A robust version of quadratic discriminant analysis can be obtained by substituting the parameters $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ by their robust estimates. For this purpose the reweighted MCD estimates, S estimates or OGK can be used. In the case of linear discriminant analysis a robust version of the common covariance matrix $\boldsymbol{\Sigma}$ is necessary. There are several methods for estimating the common covariance matrix based on a high breakdown point estimator which are considered in one of the next subsections.

Object model for robust LDA and QDA

The object model for the S4 classes and methods implementing the linear and quadratic discriminant analysis methods follows the proposed class hierarchy given in Section 2 and is presented in Figure 14.

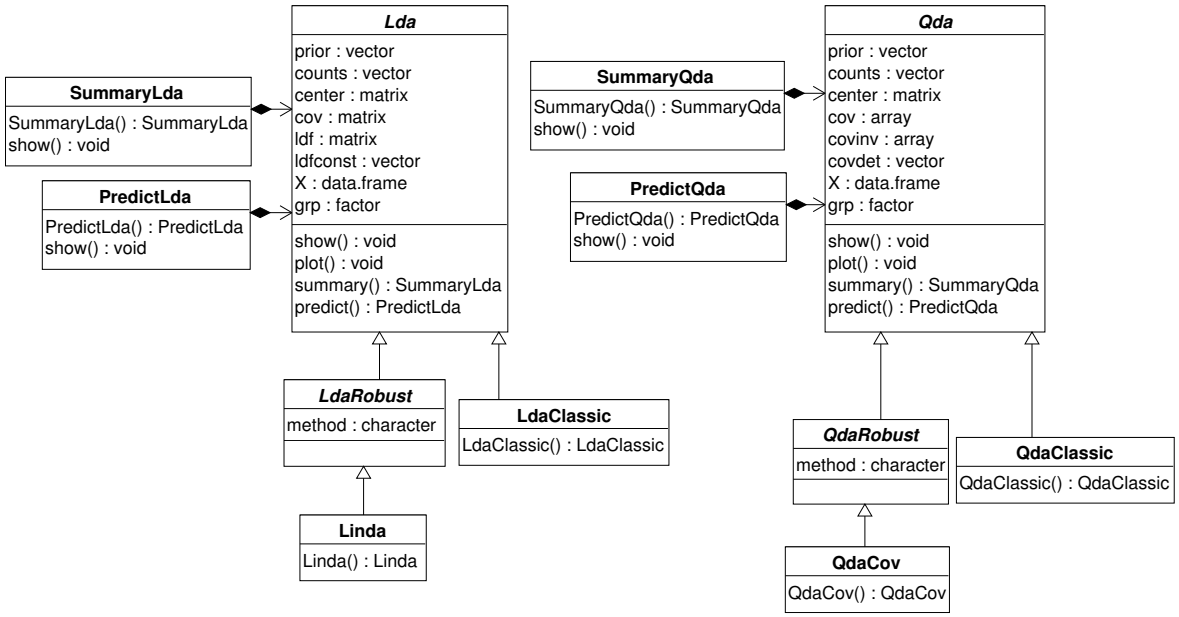

Figure 14: Object models for robust linear discriminant analysis and quadratic discriminant analysis.

The abstract classes `Lda` and `Qda` serve as base classes for deriving all classes representing classical and robust methods for linear and quadratic discriminant analysis methods. They define the common slots and the corresponding accessor methods, provide implementation for the general methods like `show()`, `plot()`, `summary()` and `predict()`. This base classes also host several utility functions which are not directly exported but are documented and can be used by quoting the namespace. The slots of `Lda` hold some input or default parameters like the prior probabilities, the original data matrix and the grouping variable as well as the results of the computations: the group means and the common covariance matrix, the linear discriminant functions and the corresponding constants. The `show()` method presents brief results of the computations, and `predict()` can be used either for classifying new observations or for the evaluation of the discriminant rules on the basis of the training sample. The method `predict()` returns an object of class `PredictLda` which has its own `show()` method to print the results of the classification or evaluation. The `summary()` method returns an object of class

`SummaryLda` which has its own `show()` method. As in the other sections of the framework these slots and methods are defined and documented only once in this base class and can be used by all derived classes. Whenever new data (slots) or functionality (methods) are necessary, they can be defined or redefined in the particular class.

Classical linear discriminant analysis is represented by the class `LdaClassic` which inherits directly from `Lda` (and uses all slots and methods defined there). The function `LdaClassic()` serves as a constructor (generating function) of the class. It can be called either by providing a data frame or matrix and a grouping variable (factor) or a formula specifying the model to be used. Let us consider the following simple example with the data set `diabetes` from package **mclust**: the grouping variable is `diabetes$class` and all remaining variables are the explanatory variables. The code

```
R> x <- diabetes[,-1]
R> grpvar <- diabetes$class
R> LdaClassic(x, grpvar)
```

can be rewritten as (and is equivalent to) the following code using the formula interface:

```
R> LdaClassic(class ~ ., data = diabetes)
```

The function `LdaClassic()` performs standard linear discriminant analysis and returns an object of class `LdaClassic`. Another abstract class, `LdaRobust` is derived from `Lda`, which serves as a base class for all robust linear discriminant analysis methods. The only slot added in this class is a character variable specifying the robust method to be used.

The class `Linda` is derived from `LdaRobust` and provides implementation for all methods for robust LDA currently available in the framework. If we wanted to be precisely object oriented, we should define a separate class for each robust method—for example `LdaRobustMcd`, `LdaRobustFsa`, etc. but this would lead to explosion of the necessary code and documentation. The constructor function `Linda()` takes a character parameter `method` specifying which robust location and scatter estimator to use and how to compute the common covariance matrix and returns an object of class `Linda`. Similarly as the function `LdaClassic()`, `Linda()` can be called either with a data matrix and grouping variable or by a formula interface.

Computing the common covariance matrix

The easiest way to estimate the common covariance matrix $\mathbf{\Sigma}$ is to obtain the estimates of the group means $\boldsymbol{\mu}_k$ and group covariance matrices $\mathbf{\Sigma}_k$ from the individual groups as $(\boldsymbol{m}_k, \mathbf{C}_k), k = 1, \ldots, g$, and then pool the estimates $\mathbf{C}_k, k = 1, \ldots, g$ to yield the common covariance matrix

$$\mathbf{C} = \frac{\sum_{k=1}^{g} n_k \mathbf{C}_k}{\sum_{k=1}^{g} n_k - g}. \tag{19}$$

This method, using MVE and MCD estimates, was proposed by Todorov *et al.* (1990, 1994a) and was also used, based on the MVE estimator by Chork and Rousseeuw (1992).
Croux and Dehon (2001) applied this procedure for robustifying linear discriminant analysis based on S estimates. A drawback of this method is that the same trimming proportions are applied to all groups which could lead to a loss of efficiency if some groups are outlier free. We will denote this method as "*A*" and the corresponding estimator as XXX-A. For example, in the case of the MCD estimator this will be MCD-A.

Another method was proposed by He and Fung (2000) for the S estimates and was later adapted by Hubert and Van Driessen (2004) for the MCD estimates. Instead of pooling the group covariance matrices, the observations are centered and pooled to obtain a single sample for which the covariance matrix is estimated. It starts by obtaining the individual group location estimates $\boldsymbol{t}_k, k = 1, \ldots, g$, as the reweighted MCD location estimates of each group. These group means are swept from the original observations $\boldsymbol{x}_{ik}$ $(i = 1, \ldots, n_k;\ k = 1, \ldots, g)$ to obtain the centered observations

$$\mathbf{Z} = \{\boldsymbol{z}_{ik}\}, \qquad \boldsymbol{z}_{ik} = \boldsymbol{x}_{ik} - \boldsymbol{t}_k. \tag{20}$$

The common covariance matrix $\mathbf{C}$ is estimated as the reweighted MCD covariance matrix of the centered observations $\mathbf{Z}$. The location estimate $\boldsymbol{\delta}$ of $\mathbf{Z}$ is used to adjust the group means $\boldsymbol{m}_k$ and thus the final group means are

$$\boldsymbol{m}_k = \boldsymbol{t}_k + \boldsymbol{\delta}. \tag{21}$$

This process could be iterated until convergence, but since the improvements from such iterations are negligible (see He and Fung 2000; Hubert and Van Driessen 2004) we are not going to use it. This method will be denoted by "B" and as already mentioned, the corresponding estimator as XXX-B, for example MCD-B.

The third approach is to modify the algorithm for high breakdown point estimation itself in order to accommodate the pooled sample. He and Fung (2000) modified Ruperts's SURREAL algorithm for S estimation in case of two groups. Hawkins and McLachlan (1997) defined the minimum within-group covariance determinant estimator which does not apply the same trimming proportion to each group but minimizes directly the determinant of the common within groups covariance matrix by pairwise swaps of observations. Unfortunately their estimator is based on the Feasible Solution Algorithm (see Hawkins and McLachlan 1997, and the references therein), which is extremely time consuming as compared to the FAST-MCD algorithm. Hubert and Van Driessen (2004) proposed a modification of this algorithm taking advantage of the FAST-MCD, but it is still necessary to compute the MCD for each individual group. This method will be denoted by MCD-C.

Using the estimates $\boldsymbol{m}_k^0$ and $\mathbf{C}_0$ obtained by one of the methods, we can calculate the initial robust distances (Rousseeuw and van Zomeren 1991)

$$RD_{ik}^0 = \sqrt{(\boldsymbol{x}_{ik} - \boldsymbol{m}_k^0)^\top \mathbf{C}_0^{-1} (\boldsymbol{x}_{ik} - \boldsymbol{m}_k^0)}. \tag{22}$$

With these initial robust distances we can define a weight for each observation $\boldsymbol{x}_{ik}$, $i = 1, \ldots, n_k$ and $k = 1, \ldots, g$, by setting the weight to 1 if the corresponding robust distance is less or equal to a suitable cut-off, usually $\sqrt{\chi^2_{p,0.975}}$, and to 0 otherwise, i.e.,

$$w_{ik} = \begin{cases} 1 & RD_{ik}^0 \leq \sqrt{\chi^2_{p,0.975}} \\ 0 & \text{otherwise.} \end{cases} \tag{23}$$

With these weights we can calculate the final reweighted estimates of the group means, $\boldsymbol{m}_k$, and the common within-groups covariance matrix, $\mathbf{C}$, which are necessary for constructing

the robust classification rules,

$$\begin{aligned} \boldsymbol{m}_k &= (\sum_{i=1}^{n_k} w_{ik}\boldsymbol{x}_{ik})/\nu_k, \\ \mathbf{C} &= \frac{1}{\nu - g}\sum_{k=1}^{g}\sum_{i=1}^{n_k} w_{ik}(\boldsymbol{x}_{ik} - \boldsymbol{m}_k)(\boldsymbol{x}_{ik} - \boldsymbol{m}_k)^\top, \end{aligned} \tag{24}$$

where ν_k are the sums of the weights within group k, for $k = 1, \ldots, g$, and ν is the total sum of weights,

$$\nu_k = \sum_{i=1}^{n_k} w_{ik}, \qquad \nu = \sum_{k=1}^{g} \nu_k.$$

Evaluation of the discriminant rules

In order to evaluate and compare different discriminant rules, their discriminating power has to be determined in the classification of future observations, i.e., we need an estimate of the overall probability of misclassification. A number of methods to estimate this probability exists in the literature—see for example Lachenbruch (1975). The *apparent error rate* (known also as resubstitution error rate or reclassification error rate) is the most straightforward estimator of the actual error rate in discriminant analysis and is calculated by applying the classification criterion to the same data set from which it was derived. The number of misclassified observations for each group is divided by the group sample size. An estimate of the apparent error rate is calculated by the method `predict()` of the class `Lda`. Examples are given in the next section.

It is well known that this method is too optimistic (the true error is likely to be higher). If there are plenty of observations in each class, the error rate can be estimated by splitting the data into training and validation set. The first one is used to estimate the discriminant rules and the second to estimate the misclassification error. This method is fast and easy to apply but it is wasteful of data. Another method is the *leave-one-out cross-validation* (Lachenbruch and Michey 1968) which proceeds by removing one observation from the data set, estimating the discriminant rule, using the remaining $n - 1$ observations and then classifying this observation with the estimated discriminant rule. For the classical discriminant analysis there exist updating formulas which avoid the re-computation of the discriminant rule at each step, but no such formulas are available for the robust methods. Thus the estimation of the error rate by this method can be very time consuming depending on the size of the data set. Nevertheless, **rrcov** provides an (internal, not exported) function `rrcov:::.CV()` which calculates the leave-one-out cross-validation error rate by "brute force", but the user should be aware that its usage is appropriate only for moderate data sets. An improvement will be the implementation of a cross-validation technique similar to the one proposed by Hubert and Engelen (2007).

Example: Diabetes data

As an example for demonstrating the usage of the robust discriminant analysis classes and methods we use the `diabetes` data set, which was analyzed by Reaven and Miller (1979) in an attempt to examine the relationship between chemical diabetes and overt diabetes in 145 nonobese adult subjects. Their analysis was focused on three primary variables and the 145

individuals were classified initially on the basis of their plasma glucose levels into three groups: normal subjects, chemical diabetes and overt diabetes. This data set was also analyzed by Hawkins and McLachlan (1997) in the context of robust linear discriminant analysis. The data set is available in several R packages: `diabetes` in package **mclust** (Fraley and Raftery 2009), `chemdiab` in package **locfit** (Loader 2007) and `diabetes.dat` in **Rfwdmv** (Atkinson *et al.* 2005). We are going to use the one from **mclust** in which the value of the second variable, *insulin*, on the 104th observation, is 45 while for the other data sets this value is 455 (note that 45 is more likely to be an outlier in this variable than 455).

We start with bringing the data set `diabetes` from package **mclust** into the workspace by typing

```
R> data("diabetes", package = "mclust")
```

Using the package **lattice** (Sarkar 2008) we produce a three dimensional cloud plot of the data (Figure 15).

```
R> library("lattice")
R> sup.sym <- trellis.par.get("superpose.symbol")
R> sup.sym$pch <- c(1, 2, 3, 4, 5)
R> trellis.par.set("superpose.symbol", sup.sym)
R> cloud.plt <- cloud(insulin ~ glucose + sspg, groups = class,
+     data = diabetes, auto.key = TRUE)
```

We will first apply the classical linear discriminant analysis as implemented in `LdaClassic()` by the formula interface of the function—the grouping variable is `class` and all the remaining variables in `diabetes` are used as predictor variables. The `show()` method will present the results of the computations: the group means, the (common) within group covariance matrix, the linear discriminant functions together with the corresponding constants. The prior probabilities (either provided by the user or estimated as a proportion of the groups) are also presented.

```
R> lda <- LdaClassic(class ~ ., data = diabetes)
R> lda

Call:
LdaClassic(class ~ ., data = diabetes)

Prior Probabilities of Groups:
 chemical    normal     overt
0.2482759 0.5241379 0.2275862

Group means:
           glucose   insulin     sspg
chemical  99.30556  482.5556 288.0000
normal    91.18421  349.9737 172.6447
overt    217.66667 1043.7576 106.0000
```

```
R> print(cloud.plt)
```

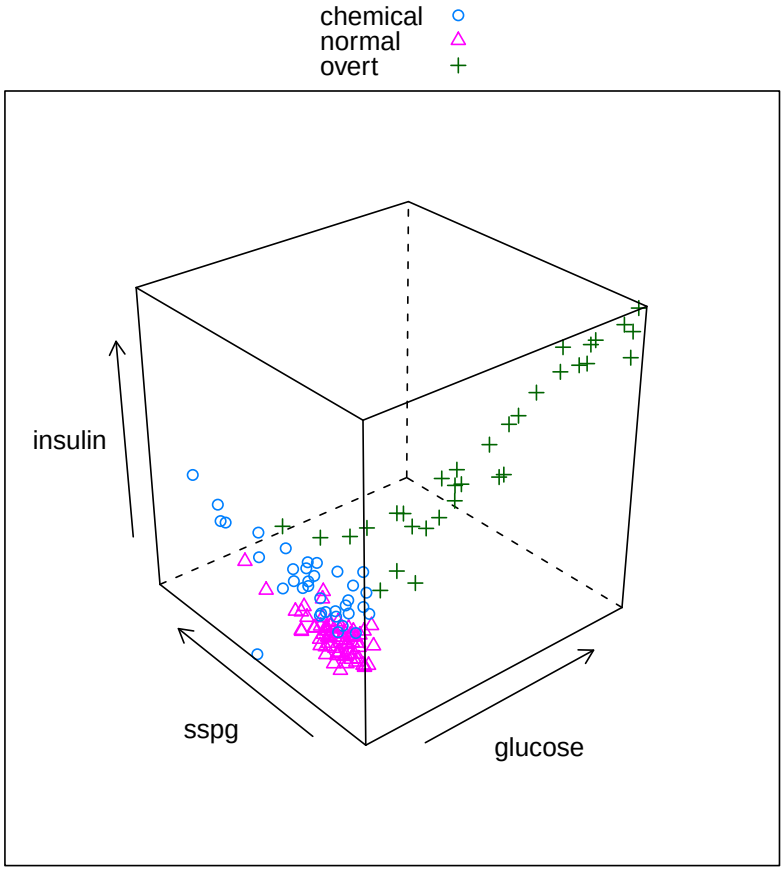

Figure 15: Three dimensional scatter plot of the `diabetes` data.

```
Within-groups Covariance Matrix:
          glucose   insulin       sspg
glucose 1378.9464  5292.673  -961.4298
insulin 5292.6732 24422.556 -4201.6529
sspg    -961.4298 -4201.653 10610.8972

Linear Coeficients:
             glucose    insulin       sspg
chemical -0.01450687 0.02934485 0.03744731
normal    0.07138094 0.00297536 0.02391635
overt    -0.03046774 0.05428174 0.02872334

Constants:
  chemical     normal      overt
```

```
-13.145581  -6.485573 -28.015148
```

Now the `predict()` method can be used on the `Lda` object (`Lda` is the base class for both `LdaClassic` and `LdaRobust`) in order to classify new observations. The method returns an object of class `PredictLda` which has its own `show()` method. If no new data are supplied, the training sample will be reclassified and a classification table will be produced to estimate the apparent error rate of the classification rules.

```
R> predict(lda)

Apparent error rate 0.131

Classification table
          Predicted
Actual     chemical normal overt
  chemical       26     10     0
  normal          2     74     0
  overt           5      2    26

Confusion matrix
          Predicted
Actual     chemical normal overt
  chemical    0.722  0.278 0.000
  normal      0.026  0.974 0.000
  overt       0.152  0.061 0.788
```

Robust linear discriminant analysis can be performed in a similar way but using the function `Linda` (which will create an object of class `Linda`). As before the `predict()` method called without new data returns a classification table of the training subsample. Using the internal convenience function `rrcov:::.AER()` we can calculate and print the apparent error rate (which now is equal to 0.1103 and is lower than the obtained with the classical LDA 0.1310).

```
R> rlda <- Linda(class ~ ., data = diabetes)
R> rlda

Call:
Linda(class ~ ., data = diabetes)

Prior Probabilities of Groups:
 chemical    normal     overt
0.2482759 0.5241379 0.2275862

Group means:
           glucose   insulin      sspg
chemical  98.73864  478.7077 241.82154
normal    91.09493  349.4047 163.20130
overt    225.82330 1093.3096  75.27524
```

```
Within-groups Covariance Matrix:
          glucose   insulin       sspg
glucose 119.60739  269.1706   83.71639
insulin 269.17062 2917.5431  575.31632
sspg     83.71639  575.3163 6734.33458

Linear Coeficients:
           glucose    insulin        sspg
chemical 0.5694809 0.10766800  0.01963126
normal   0.6173666 0.06055679  0.01138617
overt    1.3272936 0.25767141 -0.02733508

Constants:
  chemical     normal      overt
 -57.65248  -40.27402 -291.17562

R> rlda.predict <- predict(rlda)
R> cat("\nApparent error rate: ", round(rrcov:::.AER(rlda.predict@ct),
+     4))

Apparent error rate:  0.1103
```

In the above example we did not specify which method for computing the common covariance matrix should be used (thus using the default method "MCD-B" described above). We could choose the method by providing the `method` parameter to the function `Linda()`. For example the following call

```
R> rlda <- Linda(class ~ ., data = diabetes, method = "fsa")
```

will use the Hawkins and McLachlan (1997) *feasible solution algorithm* method.

The variance-covariance structures of the three classes in the `diabetes` data set differ substantially and we can expect improved results if quadratic discriminant analysis is used. Robust quadratic discriminant analysis is performed by the function `QdaCov()` which will return an object of class `QdaCov`.

```
R> rqda <- QdaCov(class ~ ., data = diabetes)
R> rqda

Call:
QdaCov(class ~ ., data = diabetes)

Prior Probabilities of Groups:
 chemical    normal     overt
0.2482759 0.5241379 0.2275862

Group means:
```

```
           glucose   insulin     sspg
chemical  99.62963  477.1481 243.2963
normal    91.98592  347.8451 164.6761
overt    226.71429 1091.7500  76.7500

Group:  chemical
         glucose  insulin       sspg
glucose 131.1589  517.766   217.1186
insulin 517.7660 3083.794   880.0170
sspg    217.1186  880.017 16767.5321

Group:  normal
          glucose   insulin       sspg
glucose  68.83554  132.3141   61.27949
insulin 132.31412 1621.7423  311.38160
sspg     61.27949  311.3816 3445.56538

Group:  overt
          glucose   insulin       sspg
glucose  7373.007  28865.49  -3361.691
insulin 28865.486 123104.82 -13689.866
sspg    -3361.691 -13689.87   3046.795

R> rqda.predict <- predict(rqda)
R> cat("\nApparent error rate: ", round(rrcov:::.AER(rqda.predict@ct),
+     4))

Apparent error rate:  0.0759
```

The accuracy of the prediction improves compared to the linear discriminant analysis.

5. Conclusions

In this paper we presented an object oriented framework for robust multivariate analysis developed in the S4 class system of the programming environment R. The main goal of the framework is to support the usage, experimentation, development and testing of robust multivariate methods as well as simplifying comparisons with related methods. It minimizes the effort for performing any of the following tasks:

- application of the already existing robust multivariate methods for practical data analysis;
- implementation of new robust multivariate methods or variants of the existing ones;
- evaluation of existing and new methods by carrying out comparison studies.

A major design goal was the openness to extensions by the development of new robust methods in the package **rrcov** or in other packages depending on **rrcov**. Further classes implementing

robust multivariate methods like principal component regression and partial least squares will follow but the user is encouraged to develop own methods using the proposed reusable statistical design patterns.

Acknowledgements

We wish to express our thanks to the organizers of and participants in the "Robust Statistics and R" workshops for the valuable comments and suggestions which were a major input for the development of this framework. We are also grateful to many people, notably Matias Salibian-Barrera, Victor Yohai, Kjell Konis, and Christophe Croux for the provided code. The careful review and constructive comments of the editor and the anonymous reviewers helped us to substantially improve the manuscript.

The views expressed herein are those of the authors and do not necessarily reflect the views of the United Nations Industrial Development Organization (UNIDO).

References

Agulló JJ (1996). "Exact Iterative Computation of the Multivariate Minimum Volume Ellipsoid Estimator with a Branch and Bound Algorithm." In A Prat (ed.), "Proceedings in Computational Statistics, COMPSTAT'96," pp. 175–180. Physica Verlag, Heidelberg.

Alexander C, Ishikawa S, Silverstein M (1977). *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press.

Atkinson A, Cerioli A, Riani M (2005). ***Rfwdmv**: Forward Search for Multivariate Data*. R package version 0.72-2, URL http://CRAN.R-project.org/package=Rfwdmv.

Boente G, Fraiman R (1999). "Discussion of 'Robust principal components for functional data' by Locantore et al." *Test*, **8**, 1–28.

Butler R, Davies P, Jhun M (1993). "Asympthotic for the Minimum Covariance Determinant Estimator." *The Annals of Statistics*, **21**, 1385–1401.

Campbell NA (1980). "Procedures in Multivariate Analysis I: Robust Covariance Estimation." *Applied Statistics*, **29**, 231–237.

Chambers JM (1998). *Programming with Data: A Guide to the S Language*. Springer-Verlag, New York.

Chork CY, Rousseeuw PJ (1992). "Integrating a High Beakdown Option into Discriminant Analysis in Exploration Geochemistry." *Journal of Geochemical Exploration*, **43**, 191–203.

Croux C, Dehon C (2001). "Robust Linear Discriminant Analysis Using S-Estimators." *The Canadian Journal of Statistics*, **29**, 473–492.

Croux C, Filzmoser P, Oliveira M (2007). "Algorithms for Projection-pursuit Robust Principal Component Analysis." *Chemometrics and Intelligent Laboratory Systems*, **87**(218), 218–225.

Croux C, Haesbroeck G (1999). "Influence Function and Efficiency of the Minimum Covariance Determinant Scatter Matrix Estimator." *Journal of Multivariate Analysis*, **71**, 161–190.

Croux C, Haesbroeck G (2000). "Principal Components Analysis Based on Robust Estimators of the Covariance or Correlation Matrix: Influence Functions and Efficiencies." *Biometrika*, **87**, 603–618.

Croux C, Ruiz-Gazen A (1996). "A Fast Algorithm for Robust Principal Components Based on Projection Pursuit." In A Prat (ed.), "Proceedings in Computational Statistics, COMPSTAT'96," pp. 211–216. Physica Verlag, Heidelberg.

Croux C, Ruiz-Gazen A (2005). "High Breakdown Estimators for Principal Components: The Projection-pursuit Approach Revisited." *Journal of Multivariate Analysis*, **95**, 206–226.

Daigle G, Rivest L (1992). "A Robust Biplot." *The Canadian Journal of Statistics*, **20**(3), 241–255.

Davies P (1987). "Asymptotic Behavior of S-Estimators of Multivariate Location Parameters and Dispersion Matrices." *The Annals of Statistics*, **15**, 1269–1292.

Devlin SJ, Gnanadesikan R, Kettenring JR (1981). "Robust Estimation of Dispersion Matrices and Principal Components." *Journal of the American Statistical Association*, **76**, 354–362.

Donoho DL (1982). "Breakdown Properties of Multivariate Location Estimators." *Technical report*, Harvard University, Boston. URL http://www-stat.stanford.edu/~donoho/Reports/Oldies/BPMLE.pdf.

Filzmoser P, Fritz H, Kalcher K (2009). ***pcaPP**: Robust PCA by Projection Pursuit*. R package version 1.7, URL http://CRAN.R-project.org/package=pcaPP.

Fraley C, Raftery A (2009). ***mclust**: Model-Based Clustering / Normal Mixture Modeling*. R package version 3.2, URL http://CRAN.R-project.org/package=mclust.

Gabriel KR (1971). "The Biplot Graphical Display of Matrices with Application to Principal Component Analysis." *Biometrika*, **58**, 453–467.

Gamma E, Helm R, Johnson R, Vlissides J (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, Reading.

Gnanadesikan R, Kettenring JR (1972). "Robust Estimates, Residuals, and Outlier Detection with Multiresponse Data." *Biometrics*, **28**, 81–124.

Hawkins DM (1994). "The Feasible Solution Algorithm for the Minimum Covariance Determinant Estimator in Multivariate Data." *Computational Statistics & Data Analysis*, **17**(2), 197–210. ISSN 0167-9473.

Hawkins DM, McLachlan GJ (1997). "High Breakdown Linear Discriminant Analysis." *Journal of the American Statistical Association*, **92**, 136–143.

He X, Fung W (2000). "High Breakdown Estimation for Multiple Populations with Applications to Discriminant Analysis." *Journal of Multivariate Analysis*, **72**, 151–162.

Hubert M, Engelen S (2007). "Fast Cross-Validation of High Breakdown Resampling Methods for PCA." *Computational Statistics & Data Analysis*, **51**, 5013–5024.

Hubert M, Rousseeuw PJ, van Aelst S (2008). "High-Breakdown Robust Multivariate Methods." *Statistical Science*, **23**, 92–119.

Hubert M, Rousseeuw PJ, Vanden Branden K (2005). "ROBPCA: A New Approach to Robust Principal Component Analysis." *Technometrics*, **47**, 64–79.

Hubert M, Van Driessen K (2004). "Fast and Robust Discriminant Analysis." *Computational Statistics & Data Analysis*, **45**, 301–320.

Lachenbruch PA (1975). *Discriminant Analysis*. Hafner, New York.

Lachenbruch PA, Michey MR (1968). "Estimation of Error Rates in Discriminant Analysis." *Technometrics*, **10**, 1–11.

Li G, Chen Z (1985). "Projection-Pursuit Approach to Robust Dispersion Matrices and Principal Components: Primary Theory and Monte Carlo." *Journal of the American Statistical Association*, **80**, 759–766.

Loader C (2007). ***locfit**: Local Regression, Likelihood and Density Estimation.* R package version 1.5-4, URL http://CRAN.R-project.org/package=locfit.

Locantore N, Marron J, Simpson D, Tripoli N, Zhang J, Cohen K (1999). "Robust Principal Components for Functional Data." *Test*, **8**, 1–28.

Lopuhaä HP (1989). "On the Relation Between S-Estimators and M-estimators of Multivariate Location and Covariance." *The Annals of Statistics*, **17**, 1662–1683.

Maronna RA (2005). "Principal Components and Orthogonal Regression Based on Robust Scales." *Technometrics*, **47**, 264–273.

Maronna RA, Martin D, Yohai V (2006). *Robust Statistics: Theory and Methods*. John Wiley & Sons, New York.

Maronna RA, Yohai VJ (1995). "The Behavoiur of the Stahel-Donoho Robust Multivariate Estimator." *Journal of the American Statistical Association*, **90**, 330–341.

Maronna RA, Zamar RH (2002). "Robust Estimation of Location and Dispersion for High-Dimensional Datasets." *Technometrics*, **44**, 307–317.

Morgenthaler S (2007). "A Survey of Robust Statistics." *Statistical Methods and Applications*, **15**(3), 271–293.

Naga R, Antille G (1990). "Stability of Robust and Non-robust Principal Component Analysis." *Computational Statistics & Data Analysis*, **10**, 169–174.

OMG (2009a). "OMG Unified Modeling Language (OMG UML), Infrastructure, V2.2." *Current formally adopted specification*, Object Management Group. URL http://www.omg.org/spec/UML/2.2/Infrastructure/PDF.

OMG (2009b). "OMG Unified Modeling Language (OMG UML), Superstructure, V2.2." *Current formally adopted specification*, Object Management Group. URL http://www.omg.org/spec/UML/2.2/Superstructure/PDF.

Pison G, Van Aelst S, Willems G (2002). "Small Sample Corrections for LTS and MCD." *Metrika*, **55**, 111–123.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Reaven GM, Miller RG (1979). "An Attempt to Define the Nature of Chemical Diabetes Using a Multidimensional Analysis." *Diabetologia*, **16**, 17–24.

Robbins JE (1999). "Cognitive Support Features for Software Development Tools." *Ph.d. thesis*, University of California, Irvine. URL http://argouml.tigris.org/docs/robbins_dissertation/.

Robbins JE, Redmiles DF (2000). "Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML." *Information and Software Technology*, **42**, 79–89.

Rocke DM (1996). "Robustness Properties of S-Estimators of Multivariate Location and Shape in High Dimension." *The Annals of Statistics*, **24**, 1327–1345.

Rocke DM, Woodruff DL (1996). "Identification of Outliers in Multivariate Data." *Journal of the American Statistical Association*, **91**, 1047–1061.

Rousseeuw PJ (1985). "Multivariate Estimation with High Breakdown Point." In W Grossmann, G Pflug, I Vincze, W Wertz (eds.), "Mathematical Statistics and Applications Vol. B," pp. 283–297. Reidel Publishing, Dordrecht.

Rousseeuw PJ, Croux C, Todorov V, Ruckstuhl A, Salibian-Barrera M, Verbeke T, Maechler M (2009). ***robustbase**: Basic Robust Statistics*. R package version 0.4-5, URL http://CRAN.R-project.org/package=robustbase.

Rousseeuw PJ, Leroy AM (1987). *Robust Regression and Outlier Detection*. John Wiley & Sons, New York.

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

Rousseeuw PJ, Van Driessen K (2006). "Computing LTS Regression for Large Data Sets." *Data Mining and Knowledge Discovery*, **12**(1), 29–45.

Rousseeuw PJ, van Zomeren BC (1991). "Robust Distances: Simulation and Cutoff Values." In W Stahel, S Weisberg (eds.), "Directions in Robust Statistics and Diagnostics, Part II," Springer-Verlag, New York.

Ruckdeschel P, Kohl M, Todorov V (2009). "Structured User Interfaces via Generating Functions." Unpublished manuscript, in preparation.

Ruppert D (1992). "Computing S-Estimators for Regression and Multivariate Location/Dispersion." *Journal of Computational and Graphical Statistics*, **1**, 253–270.

Salibian-Barrera M, Yohai VJ (2006). "A Fast Algorithm for S-regression Estimates." *Journal of Computational and Graphical Statistics*, **15**, 414–427.

Sarkar D (2008). ***lattice**: Multivariate Data Visualization with R*. Springer-Verlag, New York.

Stahel WA (1981a). "Breakdown of Covariance Estimators." *Research Report 31*, ETH Zurich. Fachgruppe für Statistik.

Stahel WA (1981b). "Robuste Schätzungen: Infinitesimale Optimalität und Schätzungen von Kovarianzmatrizen." *Ph.d. thesis no. 6881*, Swiss Federal Institute of Technology (ETH), Zürich. URL http://e-collection.ethbib.ethz.ch/view/eth:21890.

Stromberg A (2004). "Why Write Statistical Software? The Case of Robust Statistical Methods." *Journal of Statistical Software*, **10**(5), 1–8. URL http://www.jstatsoft.org/v10/i05.

Todorov V (1992). "Computing the Minimum Covariance Determinant Estimator (MCD) by Simulated Annealing." *Computational Statistics & Data Analysis*, **14**, 515–525.

Todorov V (2008). "A Note on the MCD Consistency and Small Sample Correction Factors." Unpublished manuscript, in preparation.

Todorov V (2009). ***rrcov**: Scalable Robust Estimators with High Breakdown Point*. R package version 0.5-03, URL http://CRAN.R-project.org/package=rrcov.

Todorov V, Filzmoser P (2009). "An Object Oriented Framework for Robust Multivariate Analysis." *Journal of Statistical Software*, **32**(3), 1–47. URL http://www.jstatsoft.org/v32/i03/.

Todorov V, Filzmoser P (2010). "Robust Statistic for the One-way MANOVA." *Computational Statistics & Data Analysis*, **54**, 37–48.

Todorov V, Neykov N, Neytchev P (1990). "Robust Selection of Variables in the Discriminant Analysis Based on MVE and MCD Estimators." In K Momirovic, V Mildner (eds.), "Proceedings in Computational Statistics, COMPSTAT 1990," pp. 193–198. Physica Verlag, Heidelberg.

Todorov V, Neykov N, Neytchev P (1994a). "Robust Two-group Discrimination by Bounded Influence Regression." *Computational Statistics & Data Analysis*, **17**, 289–302.

Todorov V, Neykov N, Neytchev P (1994b). "Stability of (High-breakdown Point) Robust Principal Components Analysis." In R Dutter, W Grossmann (eds.), "Short Communications in Computational Statistics, COMPSTAT 1994," pp. 90–92. Physica Verlag, Heidelberg.

Todorov V, Pires AM (2007). "Comparative Performance of Several Robust Linear Discriminant Analysis Methods." *REVSTAT Statistical Journal*, **5**, 63–83.

Wang J, Zamar R, Marazzi A, Yohai V, Salibian-Barrera M, Maronna R, Zivot E, Rocke D, Martin D, Konis K (2008). ***robust**: Insightful Robust Library*. R package version 0.3-4, URL http://CRAN.R-project.org/package=robust.

Willems G, Pison G, Rousseeuw PJ, Van Aelst S (2002). "A Robust Hotelling Test." *Metrika*, **55**, 125–138.

Woodruff DL, Rocke DM (1994). "Computable Robust Estimation of Multivariate Location and Shape in High Dimension Using Compound Estimators." *Journal of the American Statistical Association*, **89**, 888–896.

Affiliation:

Valentin Todorov
Research and Statistics Branch
United Nations Industrial Development Organization (UNIDO)
Vienna International Centre
P.O.Box 300, 1400, Vienna, Austria
E-mail: valentin.todorov@chello.at