Algorithms for Regression and Classification

Robust Regression and Genetic Association Studies

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund

an der Fakultät für Informatik

von

Robin Nunkesser

Dortmund

2009

Tag der mündlichen Prüfung: 24.02.2009

Dekan: Professor Dr. Peter Buchholz

Gutachter: Juniorprofessor Dr. Thomas Jansen
Professor Dr. Roland Fried

Abstract

Regression and classification are statistical techniques that may be used to extract rules and patterns out of data sets. Analyzing the involved algorithms comprises interdisciplinary research that offers interesting problems for statisticians and computer scientists alike. The focus of this thesis is on robust regression and classification in genetic association studies.

In the context of robust regression, new exact algorithms and results for robust online scale estimation with the estimators Q_n and S_n and for robust linear regression in the plane with the estimator least quartile difference (LQD) are presented. Additionally, an evolutionary computation algorithm for robust regression with different estimators in higher dimensions is devised. These estimators include the widely used least median of squares (LMS) and least trimmed squares (LTS).

For classification in genetic association studies, this thesis describes a Genetic Programming algorithm that outpeforms the standard approaches on the considered data sets. It is able to identify interesting genetic factors not found before in a data set on sporadic breast cancer and to handle larger data sets than the compared methods. In addition, it is extendible to further application fields.

Acknowledgments

First and foremost, I would like to thank my former advisor Ingo Wegener (4 December 1950 – 26 November 2008) for providing the topic of this dissertation, giving valuable advice and supporting and encouraging me. I am grateful that I had the fortune to work with him.

I would also like to thank my colleagues at the chair of *Efficient Algorithms and Complexity Theory* at the Faculty of Computer Science of the TU Dortmund University and my colleagues in the SFB 475 *Reduction of complexity in multivariate data structures*. Especially I would like to thank my advisors Thomas Jansen and Roland Fried and my not yet mentioned co-authors Thorsten Bernholt, Ursula Gather, Katja Ickstadt, Joachim Kunert, Oliver Morell, Karen Schettlinger, and Holger Schwender.

Many proofreaders gave valuable advise that improved this thesis. I also thank every proofreader and reviewer that was not mentioned here by name.

I gratefully acknowledge the financial support of the Deutsche Forschungsgemeinschaft (SFB 475, Reduction of complexity in multivariate data structures).

Molecular graphics images were produced using the UCSF Chimera package from the Resource for Biocomputing, Visualization, and Informatics at the University of California, San Francisco (supported by NIH P41 RR-01081).

Finally, I thank my family and friends for all the things in life besides computer science.

Contents

I Introductory Part 1

1 Introduction 3

2 Overview of Publications 7

II Robust Statistics 9

3 Preliminaries 11
- 3.1 Robust Linear Regression 13
- 3.2 Robust Scale Estimation 16

4 Online Computation of Two Estimators of Scale 19
- 4.1 Motivation 20
- 4.2 Online Computation with Moving Windows 21
- 4.3 Computation of Q_n 22
- 4.4 Computation of S_n 33

5 Computing the Least Quartile Difference Estimator in the Plane 39
- 5.1 The Concept of Geometric Duality 40
- 5.2 Computing the LQD Geometrically 42
- 5.3 Solving the Underlying Decision Problem 45
- 5.4 Searching for the Optimal Point 46
- 5.5 Running Time Simulations 51

6 An Evolutionary Algorithm for Robust Linear Regression 53
- 6.1 Evolutionary Computation 54
- 6.2 Outline of the Algorithm 55
- 6.3 Comparison 58

III Genetic Association Studies 61

7 Preliminaries 63
- 7.1 Single Nucleotide Polymorphisms . . . 63
- 7.2 Genetic Programming . . . 64
- 7.3 Hypothesis Testing . . . 65

8 Genetic Programming for Association Studies 69
- 8.1 Boolean Concept Learning . . . 69
- 8.2 Genetic Programming Algorithm . . . 71
- 8.3 Automated Rules to Select the Best Individual . . . 77
- 8.4 Results on Real and Simulated Data . . . 79
- 8.5 Logic Minimization . . . 86

IV Concluding Part 91

9 Conclusions 93

10 Classifying with Decision Diagrams 95
- 10.1 Ordered Decision Diagrams . . . 97
- 10.2 Ideas for Adaptions . . . 98

Bibliography 99

List of Algorithms 111

List of Figures 113

List of Tables 115

Index 117

Part I

Introductory Part

1 Introduction

In an early article dealing with the interface of computer science and statistics Shamos (1976) wrote:

> From the viewpoint of applied computational complexity, statistics is a gold mine, for it provides a rich and extensive source of unanalyzed algorithms and computational procedures.

Based on this "gold rush" of the late 1960s and early 1970s a two-way feedback between computer science and statistics has since been established (see e. g. Gentle *et al.*, 2004 for more historic background). In 1967, the first joint conference *Symposium on the Interface of Computer Science and Statistics* was held. The attendance at the Interface Symposia grew rapidly in the 1970s. Besides these Interface Symposia, the 1970s also saw the formation of further collaborative conferences and societies. In the 1980s, when rapidly growing computing capabilities and computer availability changed many sciences, the importance of interdisciplinary research grew even more. In statistics, the advances in computer technology enabled statistical methods, that were not practicable before. However, statistical methods with high computational complexity may probably never be computed exactly in a reasonable amount of time. The high computational complexity of many statistical methods is only one of many reasons why statistics still offers many interesting problems for computer scientists. In fact, the rich and extensive source of unanalyzed algorithms and computational procedures will not run dry for a long time to come.

This thesis deals with two topics at the interface of computer science and statistics from a computer scientist's perspective. One topic is computational biology, which emerged very recently as a multidisciplinary field involving both sciences. The other topic is robust statistics, in particular robust regression.

Regression analysis is a technique to model the relationship between a response variable and one or more explanatory variables. In linear regression, the mean or the median of the response variable is modeled as a linear combination of the explanatory variables. The best known linear regression method is least squares, which dates back to Gauss and Legendre (for historical discussions of least squares see Plackett, 1972; Stigler, 1981). One of the advantages of least squares is that it is has a closed form solution. A disadvantage is that a single outlying value can have arbitrarily large effects on the estimation. The aim of robust regression is to bound unduly effects of influence factors like outlying values. One of the earliest linear regression methods to be more robust than least squares is least absolute deviations, which is attributed to Laplace (for a brief history and comparison with least squares see Portnoy and Koenker, 1997). Although published earlier it is not as popular as least squares, which is mostly due to its computational complexity. In fact, Bernholt (2005) proves that many robust regression methods are NP-hard, which underlines the need for good heuristics and for algorithms for manageable cases like low dimensional data. After the introductory Part I of this thesis, we consider new algorithms for some robust methods in Part II. Apart from the regression methods, we also consider two other estimators that are helpful in the context of regression. When conducting regression analyses, one frequently needs an auxiliary estimation of the variability of a variable or a probability distribution. For this purpose, estimators of scale like the standard deviation are used. The standard deviation is a non-robust estimator and algorithms for robust alternatives are of high interest. Thus, we provide new algorithms for two robust estimators of scale in Part II in addition to the algorithms for robust regression methods.

Part III deals with genetic association studies which is a topic from computational biology. The aim of these studies is to identify genetic factors that may contribute to a medical condition, for example, a specific type of cancer. Among the most important genetic factors considered are single nucleotide polymorphisms, i. e. genetic variations that occur when different base alternatives exist at a single base pair position of the DNA. In this thesis, we mainly cover the classification problem of identifying genetic factors that distinguish between having the medical condition under consideration and not having it. To this end, we consider a genetic programming (Koza, 1992) algorithm for finding prediction models based on single nucleotide polymorphism data.

In detail, following this introduction, Chapter 2 gives an overview of the publications underlying this thesis and the contribution of the author. The part of the thesis concerned with robust regression starts with Chapter 3, which introduces the needed fundamentals of robust statistics. The first new results are presented in Chapter 4, which contains new online algorithms for the robust scale estimators Q_n and S_n outperforming existing algorithms. In Chapter 5, we consider an estimator called least quartile difference (LQD) which is based on the scale estimator Q_n. We show that computation of the LQD in the plane is possible in time $O(n^2 \log^2 n)$ or expected time $O(n^2 \log n)$ and state known algorithms achieving this bounds. The bounds are far lower than the hitherto known time bound of $O(n^4)$. Additionally, we present two new algorithms that almost perform within the theoretically possible runtime. Both algorithms are advantageous in terms of applicability and simplicity of implementation.

In Chapter 6 we use evolutionary computation for NP-hard robust regression methods in higher dimensions. The provided algorithm enables us to compute good solutions for least median of squares, least quantile of squares, least quartile difference, least trimmed squares, least trimmed sum of absolute values, and similar estimators. We also compare this new heuristic to existing ones and show its advantages.

Part III starts with Chapter 7 giving an introduction to genetic association studies. After these preliminaries, Chapter 8 introduces a new genetic programming algorithm for genetic association studies called GPAS. The comparison of GPAS with state-of-the-art algorithms indicates a superior performance in the investigated situations. In addition, the algorithm is not restricted to the intended application field, but may also be applied to other tasks, for example, logic minimization.

In the final part, Chapter 9 provides some conclusions and research outlooks. The concluding Chapter 10 gives a detailed outlook on a genetic programming algorithm for another classification problem in genetic association studies, namely classification with more than two classes.

2 Overview of Publications

Some of the material covered in this thesis has previously been published in journals, conference proceedings, or as technical reports. Table 2.1 gives an overview of these publications, the contribution of the author to the respective publications, and the part of this thesis each publication contributes to. The first of the listed publications is also the basis for a section in the Ph. D. thesis of Bernholt (2006).

Publication	Contribution of the author	Corresponding chapter
Bernholt, Nunkesser, and Schettlinger (2007)	45%	Chapter 5
Nunkesser, Bernholt, Schwender, Ickstadt, and Wegener (2007)	45%	Chapter 8
Nunkesser, Schettlinger, and Fried (2008)	40%	Chapter 4
Nunkesser (2008)	100%	Chapter 8
Morell, Bernholt, Fried, Kunert, and Nunkesser (2008)	25%	Chapter 6
Nunkesser and Morell (2008)	90%	Chapter 6
Nunkesser, Fried, Schettlinger, and Gather (2009)	40%	Chapter 4

TABLE 2.1: *Overview of underlying publications and the contribution of the author*

List of Publications

1. Bernholt, T., Nunkesser, R., and Schettlinger, K. (2007). Computing the least quartile difference estimator in the plane. *Computational Statistics & Data Analysis*, **52**(2), 763–772. `http://dx.doi.org/10.1016/j.csda.2006.12.039`

2. Nunkesser, R., Bernholt, T., Schwender, H., Ickstadt, K., and Wegener, I. (2007). Detecting high-order interactions of single nucleotide polymorphisms using genetic programming. *Bioinformatics*, **23**(24), 3280–3288. `http://dx.doi.org/10.1093/bioinformatics/btm522`

3. Nunkesser, R., Schettlinger, K., and Fried, R. (2008). Applying the Q_n estimator online. In C. Preisach, H. Burkhardt, L. Schmidt-Thieme, and R. Decker, editors, *Data Analysis, Machine Learning and Applications*, Studies in Classifica-

tion, Data Analysis, and Knowledge Organization, pages 277–284, Berlin, Heidelberg. Springer-Verlag. `http://dx.doi.org/10.1007/978-3-540-78246-9_33`

4. Nunkesser, R. (2008). Analysis of a genetic programming algorithm for association studies. In *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 1259–1266, New York. ACM. `http://doi.acm.org/10.1145/1389095.1389339`

5. Morell, O., Bernholt, T., Fried, R., Kunert, J., and Nunkesser, R. (2008). An evolutionary algorithm for lts-regression: A comparative study. In P. Brito, editor, *COMPSTAT 2008: Proceedings in Computational Statistics*, volume II (Contributed Papers), pages 585–593, Heidelberg. Physica-Verlag.

6. Nunkesser, R. and Morell, O. (2008). Evolutionary algorithms for robust methods. Technical Report 29/2008, SFB 475, Technische Universität Dortmund.

7. Nunkesser, R., Fried, R., Schettlinger, K., and Gather, U. (2009). Online analysis of time series by the Q_n estimator. *Computational Statistics & Data Analysis*, **53**(6), 2354–2362. `http://dx.doi.org/10.1016/j.csda.2008.02.027`

Part II

Robust Statistics

3 Preliminaries

In this chapter, we introduce some fundamentals that are necessary for the following chapters. Most of the notation and basic definitions are based on Mood *et al.* (1974). In the following, we use capital Latin letters to denote random variables and the corresponding small letters to denote the value of a random variable.

Classical statistical methods are often unduly affected by small influence factors like aberrant values. To cope with this instability, robust statistics aims at bounding the influence of such factors. A simple example for a non-robust estimator is the *sample mean* $\bar{X}$ of a finite sample $X_1, \ldots, X_n$ which is defined as

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i \ .$$

A single bad observation adjoined to the sample may cause a so called *breakdown* of the estimator because its effect on the estimate is unbounded. The *breakdown point* measures an estimator's robustness against breaking down. First defined by Hodges (1967), the definition nowadays commonly used in robust statistics is the one for finite samples (Donoho and Huber, 1983). For the sample mean we use the *contamination breakdown point* ε_n^*.

Definition 3.1. Let $\mathcal{X} = (X_1, \ldots, X_n)$ be a sample of size n. The *contamination breakdown point* $\varepsilon_n^* (T, \mathcal{X})$ of an estimator T at a sample $\mathcal{X}$ is defined by

$$\varepsilon_n^*(T, \mathcal{X}) = \min \left\{ m/(n+m); \sup_{\mathcal{X}'} \|T(\mathcal{X}) - T(\mathcal{X}')\| = \infty \right\} \ ,$$

where $\mathcal{X}'$ is an arbitrary sample that is built by adding m observations to $\mathcal{X}$ and $\|\cdot\|$ is the Euclidean norm.

Thus, we consider the minimum fraction of adjoined observations leading to an unbounded change of the estimate. The contamination breakdown point of the sample mean is $1/(n+1)$ because one additional observation suffices for breakdown of the estimator. It is common to state the contamination breakdown point of the sample mean as its asymptotic value of 0%. One of the best known robust alternatives to the sample mean is the *sample median.* This and other estimators can be defined by means of the *order statistic*.

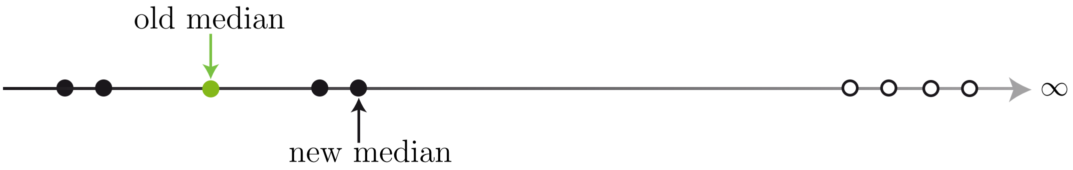


FIGURE 3.1: *Bounded change of the sample median after the insertion of $n-1$ observations.*

Definition 3.2. Let $X_1, \dots, X_n$ be a sample of size n. Then $X_{(1)} \leq \dots \leq X_{(n)}$, where $X_{(i)}$ are the X_i arranged in order of increasing magnitudes are defined to be the *order statistics* corresponding to $X_1, \dots, X_n$.

Definition 3.3. Let $X_1, \dots, X_n$ be a sample of size n. The *sample median* of $X_1, \dots, X_n$ is defined by

$$\mathrm{med}\{X_1, \dots, X_n\} = \begin{cases} X_{((n+1)/2)}, & \text{if } n \text{ is odd} \\ (X_{(n/2)} + X_{(n/2+1)})/2, & \text{if } n \text{ is even} \end{cases} .$$

The contamination breakdown point of the sample median is $n/(n+n) = 50\%$ because the estimator can only break down after the addition of at least n observations. Figure 3.1 demonstrates the robustness of the sample median. Only when n bad observations are added, the estimation will get unbounded. This first example already demonstrates advantages of robust estimation.

In this thesis, we consider linear model estimators and scale estimators for which Donoho and Huber (1983) give a different notion of breakdown point.

Definition 3.4. Let $\mathcal{X} = (X_1, \dots, X_n)$ be a sample of size n. The *replacement breakdown point* $\varepsilon_n^*(T, \mathcal{X})$ of an estimator T at a finite sample $\mathcal{X}$ is the smallest fraction of replaced values in $\mathcal{X}$ that can cause estimations arbitrarily far away from $T(\mathcal{X})$. More precisely, for linear model estimators it is defined as

$$\varepsilon_n^*(T, \mathcal{X}) = \min \left\{ m/n; \sup_{\mathcal{X}'} \|T(\mathcal{X}) - T(\mathcal{X}')\| = \infty \right\}$$

and for scale estimators as

$$\varepsilon_n^*(T, \mathcal{X}) = \min \left\{ m/n; \sup_{\mathcal{X}'} \|\log\left(T\left(\mathcal{X}\right)\right) - \log(T(\mathcal{X}'))\| = \infty \right\}$$

where $\mathcal{X}'$ is obtained by replacing any m observations in $\mathcal{X}$ by arbitrary points and $\|\cdot\|$ is the Euclidean norm.

A scale estimator is also said to break down if contamination drives the estimation to 0. The introduction of the logarithm in Definition 3.4 factors this in.

The robustness of estimators is not for free and we have at least two typical disadvantages. First, the computational complexity is often higher (although the sample median is computable in time $O(n)$, see e. g. Cormen *et al.*, 2001). To make statements about the computational complexity of methods, we use *asymptotic notation.*

Definition 3.5. Let $f : \mathbb{N} \to \mathbb{R}^+$ and $g : \mathbb{N} \to \mathbb{R}^+$ denote functions. We say that

1. $f(x) = O(g(x))$ as $x \to \infty$ if and only if there exist $c > 0$ and $x_0 \in \mathbb{N}$ such that
$$f(x)/g(x) \leq c$$
for all $x > x_0$,
2. $f(x) = \Omega(g(x))$ as $x \to \infty$ if and only if $g(x) = O(f(x))$,
3. $f(x) = \Theta(g(x))$ as $x \to \infty$ if and only if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.

A second typical disadvantage of robust estimators is that they mostly have lower *efficiency* at the normal distribution than non-robust estimators. Efficiency measures the variance of an estimator in relation to the minimum possible variance for an unbiased estimator. The efficiency of course depends on the distribution underlying the data. Here, we only consider efficiency for a Gaussian model which is sometimes called *Gaussian efficiency.* We denote the Gaussian distribution with mean μ and variance σ^2 by $\mathcal{N}(\mu, \sigma^2)$. Consider as an example (without going into detail) a sample $x_1, \ldots, x_n$ drawn from $\mathcal{N}(\mu, 1)$. The minimum possible variance for an unbiased estimator is $1/n$ which is equal to the variance of the sample mean, leading to an efficiency of 100%. The asymptotic variance of the sample median on the other hand is $\pi/2n$ corresponding to an efficiency of $2/\pi \approx 64\%$.

3.1 Robust Linear Regression

In regression analysis, we consider the relationship between a *response variable* Y and one or more *explanatory variables* X_i. The estimators we consider are linear regression methods, i. e. they assume that the response variable can be modeled by a *linear model* of the explanatory variables.

Definition 3.6. Let $Y_1, \ldots, Y_n$ be a sample of a continuous variable and let $x_{i1}, \ldots, x_{ip}$ for $i = 1, \ldots, n$ be observed values of explanatory variables $X_1, \ldots, X_p$. The *linear model* is given by

$$Y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i \quad i = 1, \ldots, n$$

where $\beta_0 \in \mathbb{R}$ is an intercept term, $\beta_1, \ldots, \beta_p$ are slope parameters, and $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ models statistical errors.

Using linear models, we automatically make certain model assumptions. These assumptions include

- linearity of the relationship between response and explanatory variables,
- continuousness of the response variable,
- independence of the errors,
- normality of the error distribution.

Applying a regression estimator to an observed data set

$$\begin{bmatrix} x_{11} & \cdots & x_{1p} & y_1 \\ \vdots & & \vdots & \vdots \\ x_{i1} & \cdots & x_{ip} & y_i \\ \vdots & & \vdots & \vdots \\ x_{n1} & \cdots & x_{np} & y_n \end{bmatrix} \tag{3.1}$$

yields estimates $\hat{\beta}_1, \ldots, \hat{\beta}_p$ and typically also $\hat{\beta}_0$ for the parameters $\beta_0, \ldots, \beta_p$. The *predicted* value of Y_i by that estimation then is

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \ldots + \hat{\beta}_p x_{ip} \quad .$$

The difference between the observation and the estimation is called *residual*.

Definition 3.7. Let y_i be the ith observed value of a data set structured like (3.1) and let $\hat{Y}_i$ be the predicted value of the corresponding random variable Y_i. The *residual* of the ith observed value is defined as

$$r_i = y_i - \hat{Y}_i$$

or in a parameterized form as

$$r_i\left(\hat{\beta}_0, \ldots, \hat{\beta}_p\right) = y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \ldots + \hat{\beta}_p x_{ip}\right) \quad .$$

Figure 3.2 shows an example for linear regression with the standard non-robust linear regression method least squares.

Definition 3.8. The least squares (LS) estimates $\hat{\beta}_0, ..., \hat{\beta}_p = \hat{\beta}_{\mathrm{LS}}$ of the regression parameters $\beta_0, ..., \beta_p$ are given by

$$\hat{\beta}_{\mathrm{LS}} = \min_{\beta_0,\ldots,\beta_p} \sum_{i=1}^{n} \left(r_i\left(\beta_0, \ldots, \beta_p\right)\right)^2 \quad .$$

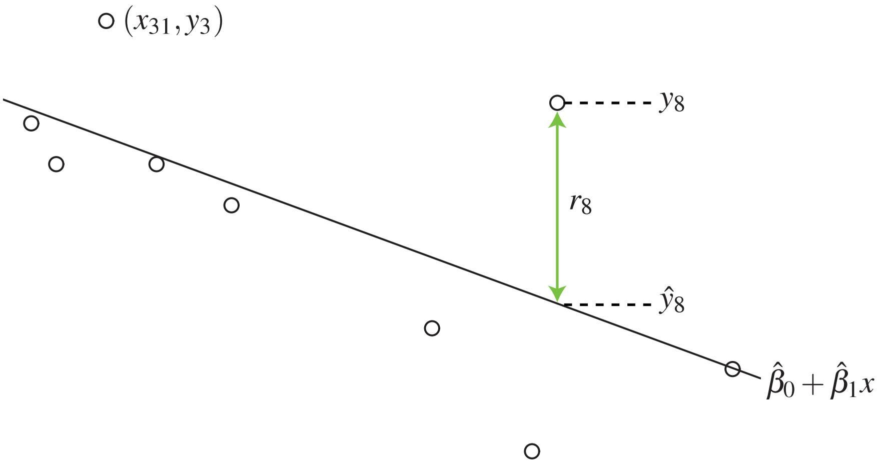


FIGURE 3.2: *Linear regression example.*

Similar to the sample mean, the breakdown point of LS is $1/n$ because it suffices to replace one observation to cause breakdown. A long known more robust alternative is least absolute deviations (LAD) where the estimates $\hat{\beta}_0, ..., \hat{\beta}_p = \hat{\beta}_{\text{LAD}}$ are defined by

$$\hat{\beta}_{\text{LAD}} = \min_{\beta_0,\dots,\beta_p} \sum_{i=1}^{n} |r_i(\beta_0, \dots, \beta_p)| \quad .$$

The LAD shows robustness when the response variable is contaminated. A classic example for the difference between robust and non-robust estimation for such contamination is an estimation on data of international phone calls made in Belgium between 1950 and 1973 (Rousseeuw and Leroy, 1987). The underlying data contains contamination between 1964 and 1969. The reason for this is that in these years the total duration of the calls was recorded instead of the total number. The years 1963 and 1970 are also partially affected by this change in recording. Figure 3.3 shows an LS and an LAD estimation for this data.

The breakdown point of LAD for a contaminated response variable depends on the design of the explanatory variables. Giloni and Padberg (2004) state that the LAD estimation in Figure 3.3 is not bounded when more than six outliers are present. However, LAD also breaks down even if only one value of the explanatory variables is replaced by an aberrant value. Rousseeuw (1984) therefore introduces the *least median of squares* (LMS) estimator which is also robust when the explanatory data is contaminated.

Definition 3.9. The least median of squares (LMS) estimates $\hat{\beta}_0, ..., \hat{\beta}_p = \hat{\beta}_{\text{LMS}}$ of the regression parameters $\beta_0, ..., \beta_p$ are given by

$$\hat{\beta}_{\text{LMS}} = \min_{\beta_0,\dots,\beta_p} \operatorname{med}\{r_1(\beta_0, \dots, \beta_p)^2, \dots, r_n(\beta_0, \dots, \beta_p)^2\} \quad .$$

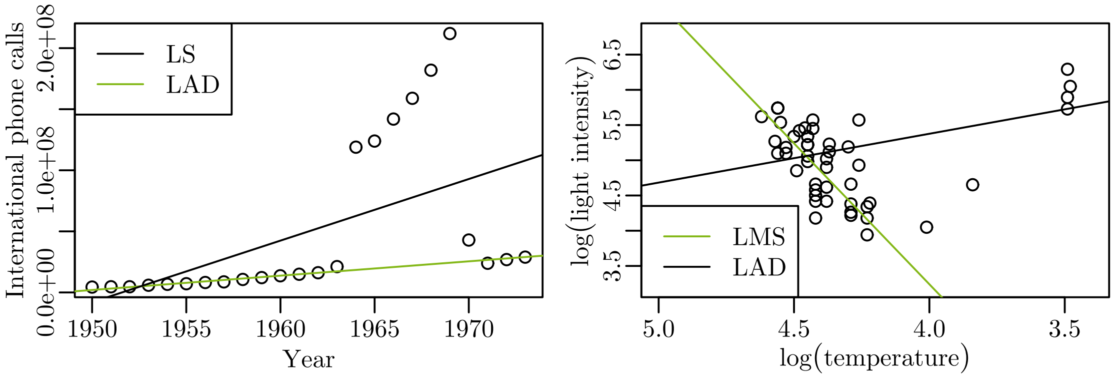


Figure 3.3: *Data on international phone calls from Belgium between 1950 and 1973 with LS and LAD fit (left) and data on light intensity and temperature of the star cluster CYG OB1 with LAD and LMS fit (right).*

Similar to the sample median, the breakdown point of LMS is 50%. This breakdown point holds for observations in *general position*.

Definition 3.10. A set of points in the d-dimensional Euclidean space is said to be in *general position* if no $d+1$ of them lie on a common hyperplane.

When the observations come from continuous distributions, the probability that they are in general position is 1 (Rousseeuw and Leroy, 1987).

Rousseeuw and Leroy (1987) give an example from astronomy that demonstrates the robustness of the LMS estimator against contamination in the explanatory data. Figure 3.3 shows the temperature and light intensity of 47 stars. Four of them are giant stars which pull away an LS or LAD estimation. The LMS estimation is unaffected by the giant stars.

In Chapter 5 and Chapter 6, we consider new algorithms for further robust estimators. Chapter 5 deals with a particular estimator in the plane and Chapter 6 presents an evolutionary algorithm for several estimators (including the LMS estimator) in higher dimensions.

3.2 Robust Scale Estimation

In robust estimation one frequently needs an initial or auxiliary estimate of scale or variability. The usual non-robust estimate of scale is the *sample standard deviation* defined by

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \bar{X}\right)^2}$$

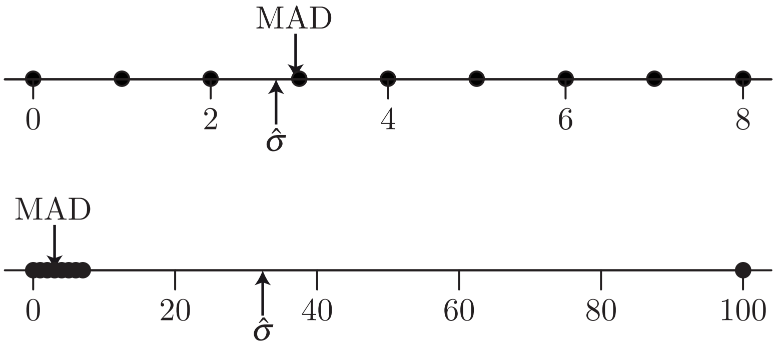


FIGURE 3.4: *Estimation by standard deviation and by MAD without and with an outlier.*

for a sample $X_1, \ldots, X_n$. Like the non-robust estimators in the previous sections, the sample standard deviation cannot withstand a single outlier and therefore has a breakdown point of $1/n$. Like in the previous section, the introduction of the median robustifies the estimation. Hampel (1974) suggests the *median absolute deviation (about the median)* (MAD), given by

$$\operatorname{med}\left\{|X_i - \operatorname{med}\{X_1, \ldots, X_n\}|\,; i = 1, \ldots, n\right\}$$

as a robust estimator which attains a breakdown point of 50% if there are no identical values in the sample (Donoho and Huber, 1983). Note that the MAD needs to be multiplied by a correction factor of 1.4826 in large samples to ensure consistency for the estimation of the standard deviation σ at normal distributed data.

As an example, consider the sample $X = \{0, \ldots, 8\}$ and another sample where the 8 is replaced by 100. The MAD is unaffected by the outlier and estimates the scale in both cases as 2.97 while the standard deviation changes from 2.74 to 32.25 in the presence of the outlier. Figure 3.4 depicts this example.

Rousseeuw and Croux (1993) point out two drawbacks of the MAD. First, its asymptotic Gaussian efficiency is only 37%. Second, the MAD is aimed at symmetric distributions, because it attaches equal importance to positive and negative deviations from a central value. Thus, the next section deals with two alternatives to the MAD.

4 Online Computation of Two Estimators of Scale

In the previous chapter, we encountered the MAD scale estimator. The MAD has a simple explicit formula, only needs $O(n)$ computation time, and is very robust. On the other hand, its low Gaussian efficiency and the fact that it is aimed at symmetric distributions are drawbacks. Rousseeuw and Croux (1993) propose two alternatives called S_n and Q_n that use order statistics (see Definition 3.2).

Definition 4.1. Let $X_1, \ldots, X_n$ be a sample of size n.

1. The scale estimator S_n is defined as

$$S_n = c \cdot \operatorname{med}\left\{\operatorname{med}\left\{|X_i - X_j|\,; j = 1, \ldots, n\right\}; i = 1, \ldots, n\right\} \quad .$$

2. The scale estimator Q_n is defined as

$$Q_n = c \cdot \left\{|X_i - X_j|\,; i < j\right\}_{(k)} \text{ with } k = \binom{\lfloor n/2 \rfloor + 1}{2} \quad .$$

The factor c is a consistency factor which is typically different for S_n and Q_n.

Like the MAD, S_n and Q_n typically need to be multiplied by a correction factor. When we want to ensure consistency for the estimation of the standard deviation σ at normal distributed data, this factor is 1.1926 for the S_n and 2.2219 for the Q_n in large samples (Rousseeuw and Croux, 1993). S_n and Q_n can attain the same optimal breakdown point as the MAD, but the asymptotic Gaussian efficiency of S_n and Q_n is 58% and 82%, respectively (Rousseeuw and Croux, 1993). This is superior to the 37% of the MAD. In addition, S_n and Q_n do not presuppose a symmetric distribution. The computation time needed is $O\left(n \log n\right)$ for both (Croux and Rousseeuw, 1992). This is asymptotically optimal for Q_n, because—as we will see later—computing Q_n is equivalent to a problem described in Johnson and Kashdan (1978) that needs $\Omega\left(n \log n\right)$ time. For S_n no better lower bound than the trivial bound $\Omega\left(n\right)$ for median computation (Blum *et al.*, 1973) is known. Our focus in this thesis is on efficient online computation of these estimators.

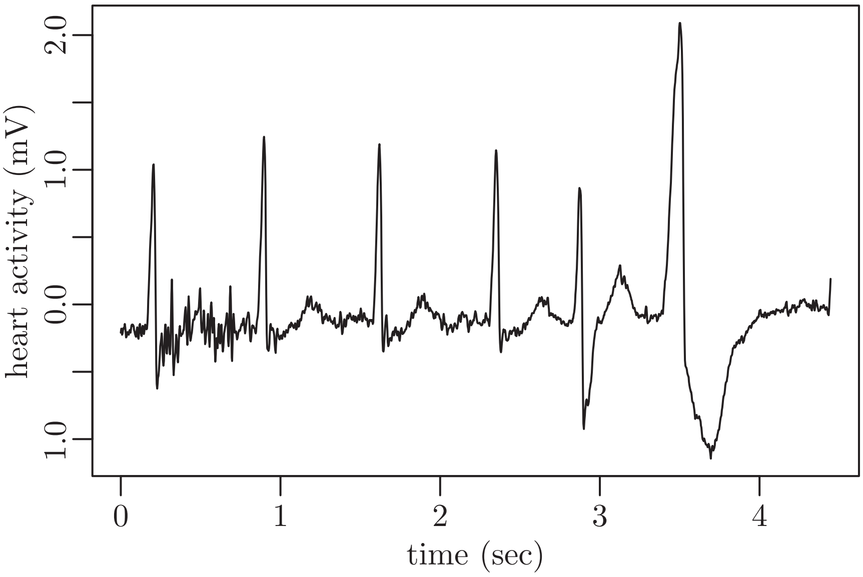


FIGURE 4.1: *Heart activity measured by ECG.*

4.1 Motivation

A *time series* is a sequence of data points typically measured at consecutive points of time with uniform time intervals in between. In many application fields, time series data has to be analyzed online to support real-time decisions. In computer science, this corresponds to the online analysis of *data streams* (see e. g. Muthukrishnan, 2005). High frequency data streams often arise in monitoring applications in which their analysis is time critical. The analysis therefore needs to be done in near real time to keep pace with the data updates and accurately reflect rapidly changing trends in the data. For this task, fast online algorithms are needed.

Our main motivation for considering the online computation of robust estimators of scale are automatic alarm systems in intensive care. The alarm systems currently used produce a high rate of false alarms due to measurement artifacts, patient movements, or transient fluctuations around the chosen alarm limit. Preprocessing the data by extracting the underlying *signal* (the time-varying level observed) and variability of the monitored physiological time series such as heart rate or blood pressure can improve the false alarm rate. Additionally, it is necessary to detect relevant changes in the extracted signal since they might point at serious changes in the patient's condition. Figure 4.1 shows an example of a patient's heart rate taken from PhysioNet (Goldberger *et al.*, 2000). The high number of artifacts observed in many time series requires the application of *robust* methods which are able to withstand some largely deviating values. Gather and Fried (2003) recommend to use the Q_n estimator in robust signal extraction to measure the variability of the statistical error.

Another application where near-real time monitoring is of interest is the analysis of financial data. High frequency data are especially susceptible to errors as for example stated by Brownlees and Gallo (2006):

> The higher the velocity in trading, the higher the probability that some error will be committed in reporting trading information.

In the financial context, we need preprocessing procedures for tasks like automatic data cleaning and outlier detection. Non-robust estimators can be strongly misled by outliers. Preprocessing the data with robust methods has the advantage that the robust analysis resists isolated outliers and patches of outlying values. Robust scale estimators allow us to extract possibly time-varying *volatilities* (the standard deviation of returns for a given financial parameter, for example a stock market index) in the presence of outliers, see Gather and Fried (2003) and Gelper *et al.* (2009).

Further online applications of robust scale estimators include the estimation of autocorrelations within the process (Ma and Genton, 2000) and the standardization of test statistics (Fried, 2007).

4.2 Online Computation with Moving Windows

To analyze the scale of an observed time series $x_1, \ldots, x_N$ online, we apply the scale estimator at each time point t to a time window of length $n \leq N$, which contains the observations $x_{t-n+1}, \ldots, x_t$. Instead of calculating the estimate for each window from scratch, we use an online algorithm. This means that for each move of the window from t to $t+1$ all stored information concerning the oldest observation x_{t-n+1} is deleted and new information concerning the incoming observation x_{t+1} is inserted. Insertions and deletions are called *updates*. Note that the online algorithms we propose in the next sections are not restricted to moving time windows; they can also handle arbitrary sequences of deletions and insertions of data points.

When dealing with online algorithms, dynamic data structures are helpful. The standard operations that a data structure storing a multiset S of elements has to offer according to Cormen *et al.* (2001) include

1. `Search`(S, x), which searches for x in S,
2. `Insert`(S, x), which inserts x into S,
3. `Delete`(S, x), which deletes x from S.

Depending on the data structure and the specific requirements, x could be an element for the data structure, the key of an element, or a pointer to an element. An additional operation we will need is `Rank`(S, x), which gives the rank of x in S, i. e. the position of x in sorted S. For the first algorithm, we will need fast insertion, deletion, searching and ranking of an element. Every balanced binary search tree guarantees insertion, deletion, and searching in time $O(\log n)$. The rank of an element may also be queried in $O(\log n)$ if we store ranking information in each node of the search tree (see e. g. Knuth, 1973 or Cormen *et al.*, 2001). We use AVL trees (Adel'son-Vel'skiĭ and Landis, 1962) as balanced binary search trees. In the second algorithm, ranking will be more important than insertion and deletion. For that case, we may use simple array or list structures that allow rank queries in time $O(1)$ but come with no better bound than $O(n)$ for insertion and deletion.

4.3 Computation of Q_n

Q_n is a high breakdown estimator with very good Gaussian efficiency and therefore a highly relevant robust scale estimator. Section 4.1 demonstrates that it is desirable to have fast online algorithms for computing this estimator. We present a fast online algorithm based on the optimal offline algorithm in the following. As a first step, we show that Q_n may be computed by solving selection in the multiset $X + Y$.

Problem 4.1 (Selection in $X + Y$)**.**

Given: Two multisets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ and a parameter $k \in \mathbb{N}$.

Goal: Compute

$$\{x_i + y_j; x_i \in X \text{ and } y_j \in Y\}_{(k)} \quad .$$

For this problem, Johnson and Kashdan (1978) state an upper bound of $O(n \log n)$ and a lower bound of $\Omega(n + \sqrt{k} \log k)$ for computation in a decision tree. Problem 4.1 may be used to solve different problems from statistics. In the considered problems, $k = \Theta(n^2)$ and thus the upper and the lower bound match.

Corollary 4.1. *The complexity of computing the Hodges-Lehmann estimator (Hodges and Lehmann, 1963) defined by*

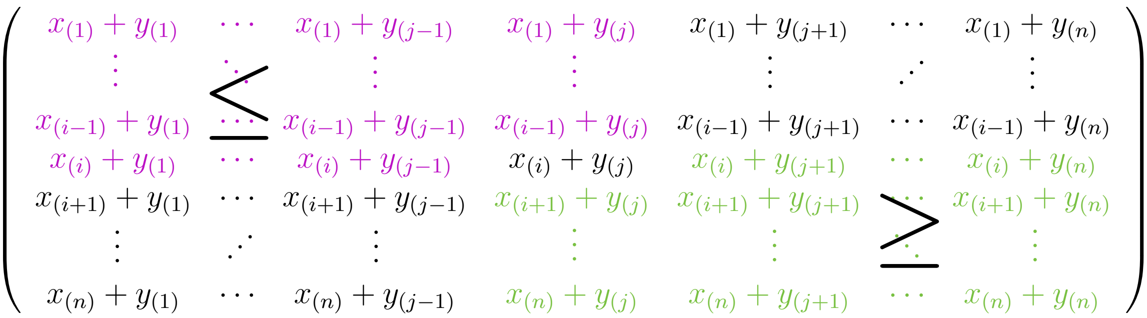
$$\frac{\operatorname{med}\{X_i + X_j; 1 \leq i, j \leq n\}}{2}$$

for a sample $X_1, \ldots, X_n$ *is* $\Theta\left(n \log n\right)$.

Theorem 4.1 (Croux and Rousseeuw, 1992)**.** The complexity of computing Q_n for a sample of size n is $\Theta\left(n \log n\right)$.

Proof. Starting with Definition 4.1 we obtain

$$\begin{aligned} Q_n &= c \cdot \{|X_i - X_j|\,; i < j\}_{(k)} \\ &= c \cdot \{\{-|X_i - X_j|\,; i > j\} \cup \{|X_i - X_j|\,; i = j\} \cup \{|X_i - X_j|\,; i < j\}\}_{\left(\binom{n}{2}+n+k\right)} \\ &= c \cdot \{X_i - X_j; 1 \leq i, j \leq n\}_{\left(\binom{n}{2}+n+k\right)} \quad . \end{aligned}$$



$$\begin{pmatrix} x_{(1)}+y_{(1)} & \cdots & x_{(1)}+y_{(j-1)} & x_{(1)}+y_{(j)} & x_{(1)}+y_{(j+1)} & \cdots & x_{(1)}+y_{(n)} \\ \vdots & \ddots \;\leq & \vdots & \vdots & \vdots & \iddots & \vdots \\ x_{(i-1)}+y_{(1)} & \cdots & x_{(i-1)}+y_{(j-1)} & x_{(i-1)}+y_{(j)} & x_{(i-1)}+y_{(j+1)} & \cdots & x_{(i-1)}+y_{(n)} \\ x_{(i)}+y_{(1)} & \cdots & x_{(i)}+y_{(j-1)} & x_{(i)}+y_{(j)} & x_{(i)}+y_{(j+1)} & \cdots & x_{(i)}+y_{(n)} \\ x_{(i+1)}+y_{(1)} & \cdots & x_{(i+1)}+y_{(j-1)} & x_{(i+1)}+y_{(j)} & x_{(i+1)}+y_{(j+1)} & \geq \cdots & x_{(i+1)}+y_{(n)} \\ \vdots & \iddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{(n)}+y_{(1)} & \cdots & x_{(n)}+y_{(j-1)} & x_{(n)}+y_{(j)} & x_{(n)}+y_{(j+1)} & \cdots & x_{(n)}+y_{(n)} \end{pmatrix}$$

FIGURE 4.2: *Regions in the matrix M with definitely smaller or certainly greater values than $x_{(i)} + y_{(i)}$.*

This corresponds to Problem 4.1 for a multiset of type $X + (-X)$ and the parameter $\binom{n}{2} + n + k$. □

Note that there are further estimators, e. g. the *medcouple* (Brys *et al.*, 2004), that may be computed with an algorithm for Problem 4.1.

4.3.1 Offline Computation

In Theorem 4.1, we saw that it is possible to compute the Q_n by solving Problem 4.1. This is why we only consider Problem 4.1 in the following. Shamos (1976) provides an algorithm he devised with Jefferson and Tarjan for computing

$$\operatorname{med}\{x_i + y_j; x_i \in X \text{ and } y_j \in Y\}$$

in $O(n \log n)$ and also states a lower bound of $\Omega(n \log n)$ communicated by Tarjan.

Johnson and Mizoguchi (1978) adapt this algorithm to compute an arbitrary order statistic, i. e. to solve Problem 4.1. It is convenient to visualize the algorithm of Johnson and Mizoguchi working on a partially sorted matrix $M = (m_{ij})$ with $m_{ij} = x_{(i)} + y_{(j)}$, although M is of course never constructed. Recall that $x_{(1)} \leq \ldots \leq x_{(n)}$ and $y_{(1)} \leq \ldots \leq y_{(n)}$ denote the elements of X and Y ordered according to size. Thus, the matrix M has monotonicity in the rows and columns, i. e. $m_{ij} = x_{(i)} + y_{(j)} \leq x_{(i)} + y_{(\ell)} = m_{i\ell}$ and $m_{ji} = x_{(j)} + y_{(i)} \leq x_{(\ell)} + y_{(i)} = m_{\ell i}$ for $j \leq \ell$. The algorithm uses this monotonicity to compute the desired order statistic (see Figure 4.2 for an example of this monotonicity).

Algorithm 4.1 sketches the algorithm of Johnson and Mizoguchi (1978). Steps 1–3 take time $O(n \log n)$ because of the sorting of X and Y. No computation is done in line 2. If the element in line 5 is carefully selected which takes time $O(n)$ (we omit the details), the while loop will run $O(\log n)$ times. Line 10 comprises a selection problem which can be carried out in time $O(n)$ (see e. g. Cormen *et al.*, 2001).

Algorithm 4.1: Sketch of the algorithm of Johnson and Mizoguchi (1978)

```
Input: Multisets $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_n\}$, parameter $k$
Output: The $k$th order statistic of $X + Y$
Sort $X$ and $Y$
Let $M = (m_{ij})$ with $m_{ij} = x_{(i)} + y_{(j)}$ denote the partially sorted matrix of $X + Y$
Set $c := n^2$
while $c > n$ do
    Select an element $m$ of $M$ that is still a candidate for the $k$th order statistic
    Determine regions in the matrix definitely smaller or certainly greater than $m$
    Exclude all parts of these regions that cannot contain the sought order
      statistic
    Update $c$, which denotes the number of matrix elements still under
      consideration
end
Determine directly which of the remaining elements of $M$ is the $k$th order
  statistic of $X + Y$
return the determined order statistic
```

4.3.2 Online Computation

The offline algorithm has optimal running time. In this section, we try to be faster for online computation and present a new algorithm. Online algorithms for a problem similar to computing Q_n exist. Bespamyatnikh (1998) states an online algorithm for computing

$$\{|x_i - x_j|\,; i < j\}_{(k)} \quad \text{with } k = 1 \tag{4.1}$$

that achieves $O(\log n)$ per update step. This is optimal, because the lower bound for offline compuataion is $\Omega(n \log n)$ (Preparata and Shamos, 1985). Note, that (4.1) is the well known problem of computing the minimal distance between two numbers in a set. In an earlier work on minimal distance computation, Smid (1991) suggests to use a buffer of possible solutions to obtain an online algorithm for this problem. The buffer contains the elements $\{|x_i - x_j|\,; i < j\}_{(1)}, \{|x_i - x_j|\,; i < j\}_{(2)}, \dots$ and enables fast updates because the new minimal distance after an update is likely to be found in the buffer. However, the computation of the Q_n estimator requires larger k. Hence, we generalize the idea of using a buffer to arbitrary values of k in the following, because it is easy to implement and achieves a good running time in practice. The resulting algorithm will have a worst case amortized time per update that is the same as for the offline algorithm. However, we show that our algorithm runs substantially faster under certain data assumptions and also runs faster for many data sets not fulfilling these assumptions.

Algorithm 4.1 may easily be extended to compute a *buffer* B of s matrix elements

$m_{(k-\lfloor (s-1)/2 \rfloor)}, \ldots, m_{(k+\lfloor s/2 \rfloor)}$ from M next to the solution $m_{(k)}$. We discuss the size s of the buffer B later. We first describe the framework of the online algorithm in Algorithm 4.2 and later give the details on how to insert and delete elements.

Algorithm 4.2: Moving window selection in $X + Y$

```
   Input: Samples $X$ and $Y$, parameter $k$, window width $n$, buffer size $s$
   Output: The $k$th order statistic of each window in $X + Y$
 1 Set $X_w := \{x_1, \ldots, x_n\}$ and $Y_w := \{y_1, \ldots, y_n\}$
 2 Compute the $k$th order statistic of $X_w + Y_w$ and a buffer $B$ of size $s$ offline
 3 for $t \leftarrow n$ to $|X| - 1$ do
 4     Call Insert($x_{t+1}$, $X_w$, $Y_w$, $B$) and Delete($x_{t-n+1}$, $X_w$, $Y_w$, $B$)
 5     Call Insert($y_{t+1}$, $Y_w$, $X_w$, $B$) and Delete($y_{t-n+1}$, $Y_w$, $X_w$, $B$)
 6     if Delete($y_{t-n+1}$, $Y_w$, $B$) returned that the $k$th order statistic of
       $\{x_{t-n+1}, \ldots, x_{t+1}\} + \{y_{t-n+1}, \ldots, y_{t+1}\}$ is in $B$ then
 7         Determine the $k$th order statistic directly
 8     else
 9         Recalculate the $k$th order statistic and the buffer $B$ offline
10     end
11 end
12 return the determined order statistics
```

To speed up online computation, we ensure fast insertion and deletion and few of the recalculations done in line 9. To achieve this, we use indexed AVL trees as the main data structure. As mentioned in Section 4.2, inserting, deleting, finding and determining the rank of an element takes $O(\log n)$ time in this data structure. Moreover—in the data structure we use—every element in the balanced tree has two pointers allowing access to the element pointed at in time $O(1)$. In detail, we store X, Y and B in separate balanced trees and manage the following pointers:

1. Each element $m_{ij} = x_{(i)} + y_{(j)}$ in the buffer B gets two pointers: one pointer to $x_{(i)} \in X$ and one pointer to $y_{(j)} \in Y$.

2. Each element $x_{(i)}$ in X (corresponding to elements in the ith row of M) gets one pointer to the smallest and one pointer to the largest element such that $m_{ij} \in B$ for $1 \leq j \leq n$.

3. Each element $y_{(j)}$ in Y (corresponding to elements in the jth column of M) gets one pointer to the smallest and one pointer to the largest element such that $m_{ij} \in B$ for $1 \leq i \leq n$.

Figure 4.3 shows an example for these pointers where the arrows with solid tips mark the pointers of a matrix element and the "≫"-tips mark the pointers to the smallest and largest element. Note that the buffer may of course be much larger and the

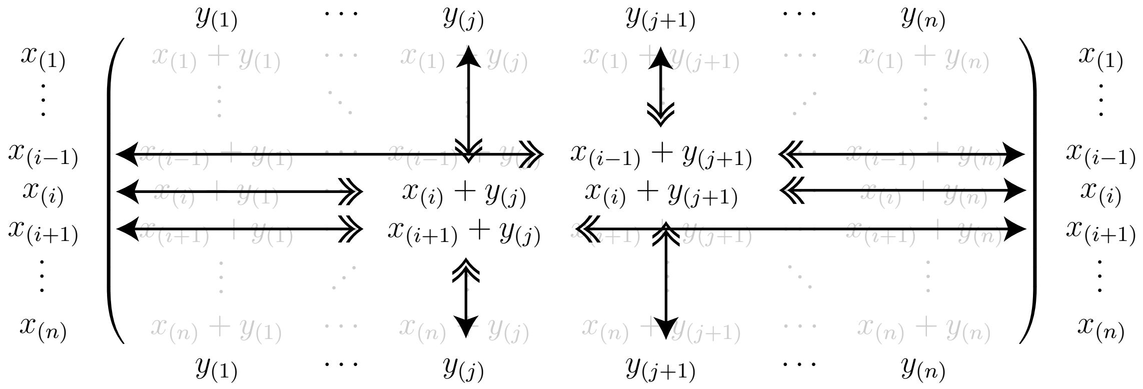


FIGURE 4.3: *Example for the pointer structure.*

pointers of elements in X and Y only point to the boundaries of the buffer. The offline Algorithm 4.1 may easily be extended to return this data structure and the buffer without using more time than $O(n \log n)$.

The following procedures `Insert` and `Delete` handle insertions into and deletions from X. Insertions into and deletions from Y work analogously with slightly different procedures.

```
Procedure Insert($x_{\text{ins}}$, sample $X$, sample $Y$, buffer $B$)
  Input: Element to insert $x_{\text{ins}}$, sample $X$, sample $Y$, buffer $B$
  Output: changed sample $X$ and buffer $B$, information whether the $k$th element
          of $X+Y$ is still in $B$
  // Compute all elements in the new matrix row defined by $x_{\text{ins}}$ that
     belong into the buffer
  Determine the smallest element $b_s$ and the greatest element $b_g$ in $B$
  Determine with a binary search the smallest $j$ such that $x_{\text{ins}} + y_{(j)} \geq b_s$ and the
  greatest $\ell$ such that $x_{\text{ins}} + y_{(\ell)} \leq b_g$
  Compute all elements $B_m := \{x_{\text{ins}} + y_{(m)} \mid j \leq m \leq \ell\}$
  // Insert these elements into the buffer and $x_{\text{ins}}$ into $X$
  Insert the elements in $B_m$ into the buffer $B$
  Insert $x_{\text{ins}}$ into $X$
  // Update the data structure
  Update pointers to and from the inserted elements accordingly
  Compute the new position of the $k$th element of $X+Y$ in $B$ by counting how
  many smaller and how many greater elements were inserted
  return $X$, $B$ and whether the $k$th element of $X+Y$ is still in $B$
```

The basic operations in our data structure need time $O(\log n)$, following a pointer needs time $O(1)$. These time bounds apply for most of the algorithm steps. The binary

```
Procedure Delete($x_{\text{del}}$, sample $X$, sample $Y$, buffer $B$)
  Input: Element to delete $x_{\text{del}}$, sample $X$, sample $Y$, buffer $B$
  Output: changed sample $X$ and buffer $B$, information whether the $k$th element
          of $X+Y$ is still in $B$
  // Get the element to delete and its rank in the data structure
1 Search in $X$ for $x_{\text{del}}$
2 Determine the rank $i$ of $x_{\text{del}}$ and the elements $b_s$ and $b_g$ pointed at
  // Determine all elements in the matrix row defined by $x_{\text{del}}$ that are
     in the buffer
3 Determine $y_{(j)}$ and $y_{(\ell)}$ with the help of the pointers of $b_s$ and $b_g$ such that
  $b_s = x_{(i)} + y_{(j)}$ and $b_g = x_{(i)} + y_{(\ell)}$
4 Find $B_m := \{x_{(i)} + y_{(m)} \in B \mid j \leq m \leq \ell\}$
  // Delete these elements from the buffer and $x_{\text{del}}$ from $X$
5 Delete the elements in $B_m$ from the buffer $B$
6 Delete $x_{\text{del}}$ from $X$
  // Update the data structure
7 Update the affected pointers accordingly
8 Compute the new position of the $k$th element of $X+Y$ in $B$ by counting how
  many smaller and how many greater elements were deleted
9 return $X$, $B$ and whether the $k$th element of $X+Y$ is still in $B$
```

search in line 2 of `Insert` needs time $O(\log n)$. The only operations needing more time than $O(\log n)$ are the ones concerning the set B_m. Thus, we see that `Insert` and `Delete` need a maximum of $O(|B_m| \cdot \log n)$ time for insertion and deletion, where B_m is defined as in the procedures. It is possible to introduce bounds on the size of B and to recompute B if these bounds are violated to limit the space requirement of the online algorithm. We will use a bound of $O(n)$ in the following. For arbitrary sequences of insertions and deletions, the size of B can vary more and we may have to recompute the buffer more often than for moving time windows. To assess the running time we have to consider first the number of elements in the buffer that depend on the inserted or deleted observation, i. e. the typical size of B_m.

Theorem 4.2. For a constant signal with identically distributed noise variables the expected time needed for insertion or deletion of data points is $O(\log n)$.

Proof. For a constant signal with identically distributed error terms, data points are *exchangeable* in the sense that each rank of a data point in the set of all n data points occurs with equal probability $1/n$. Assume without loss of generality that we only insert into and delete from X. We define random variables $D_{x_{(i)}} = \left|\{x_{(i)} + y_{(j)} \in B; 1 \leq j \leq n\}\right|$ for $x_{(i)} \in X$ that describe the number of buffer elements in the buffer B depending on $x_{(i)}$. As a first step, we consider the deletion of x_{del} from

X with equiprobable ranks. The expected value $\mathrm{E}\left(D_{x_{\mathrm{del}}}\right)$ is given by

$$\mathrm{E}\left(D_{x_{\mathrm{del}}}\right) = \frac{1}{n} D_{x_{(1)}} + \ldots + \frac{1}{n} D_{x_{(n)}} = \frac{1}{n} \sum_{i=1}^{n} D_{x_{(i)}} = \frac{1}{n} |B| \ .$$

When we insert an element x_{ins} into X, it is inserted between two elements $x_{(i)}$ and $x_{(i+1)}$. Because of the monotonicity of the matrix M, the number of buffer elements depending on x_{ins} after its insertion cannot be greater than $D_{x(i)} + D_{x(i+1)}$. Let $S_{x_{(i)}} = D_{x(i)} + D_{x(i+1)}$ with $S_{x(0)} = D_{x(1)}$ and $S_{x(n)} = D_{x(n)}$. We consider the insertion of x_{ins} with equiprobable ranks. The expected value $\mathrm{E}\left(D_{x_{\mathrm{ins}}}\right)$ is given by

$$\mathrm{E}\left(D_{x_{\mathrm{ins}}}\right) \leq \frac{1}{n+1} S_{x_{(0)}} + \ldots + \frac{1}{n+1} S_{x_{(n)}} = \frac{2}{n+1} \sum_{i=1}^{n} D_{x_{(i)}} = \frac{2}{n+1} |B| \ .$$

The size of the buffer $|B|$ is $O(n)$. Thus, $\mathrm{E}\left(D_{x_{\mathrm{del}}}\right)$ and $\mathrm{E}\left(D_{x_{\mathrm{ins}}}\right)$ are $O(1)$ and we expect to spend $O(\mathrm{E}\left(D_{x_{\mathrm{ins}}}\right) \log n) = O(\log n)$ time for the insertion and $O(\mathrm{E}\left(D_{x_{\mathrm{del}}}\right) \log n) = O(\log n)$ for the deletion of a data point. □

At times, we have to recompute the buffer which needs $O(n \log n)$ time and increases the amortized time per update. The frequency of recomputations depends on how much the kth element of $X + Y$ may move in the buffer. With equiprobable ranks as in Theorem 4.2, the expected position of the kth element in the buffer after a deletion and a subsequent insertion is the same as before the deletion and the insertion. Thus, we expect to recompute the buffer very rarely.

The next section contains running time simulations for data that fulfills the assumptions of Theorem 4.2 and for more complex data situations.

4.3.3 Running Time Simulations

To demonstrate the good performance of the new online algorithm in practice, we conduct running time simulations for online computation of the Q_n estimator. We use three different simulated time series and a real time series and consider the extraction of possibly time-varying volatilities.

The basis of the first time series (depicted in Figure 4.4) is a benchmark model which is commonly used to estimate and predict volatility processes. This benchmark model is given by the GARCH(1,1) model proposed by Bollerslev (1986)

$$X_t = \sigma_t \varepsilon_t, \ \ t \in \mathbb{Z} \ ,$$

where $\varepsilon_t \sim \mathcal{N}(0,1)$ is an error term and σ_t is a time-varying volatility coefficient. More precisely, the conditional variance $\sigma_t^2 = \mathrm{Var}(X_t | X_{t-1}, X_{t-2}, \ldots)$ is given by

$$\sigma_t^2 = \alpha_0 + \alpha_1 X_{t-1}^2 + \beta_1 \sigma_{t-1}^2$$

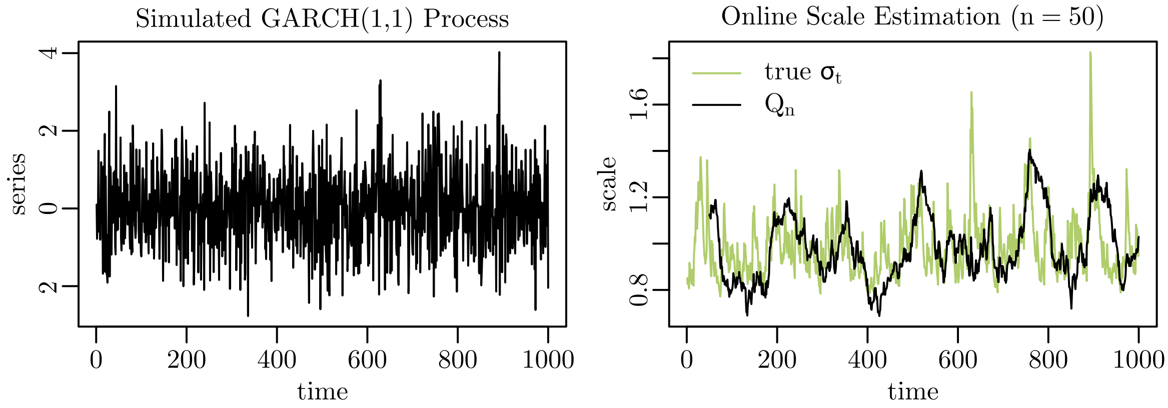


FIGURE 4.4: *Simulated GARCH(1,1) series (left) and estimated volatilities (right).*

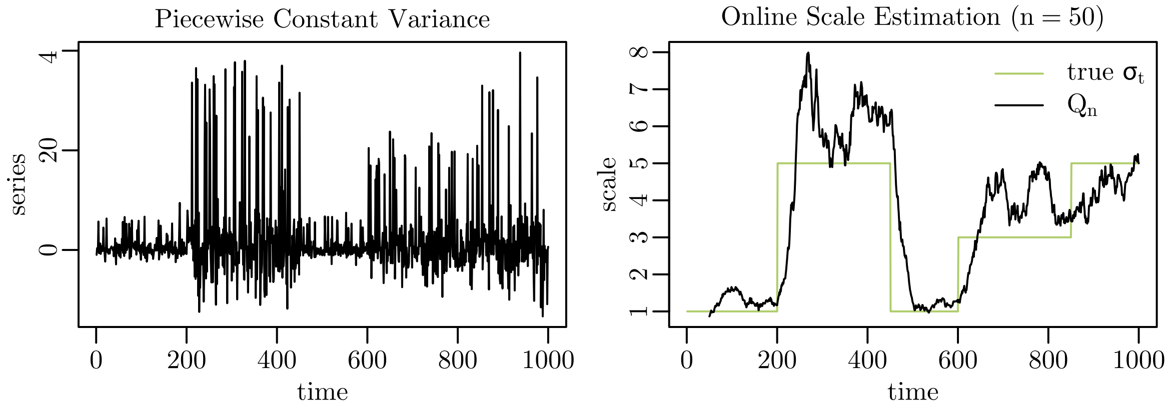


FIGURE 4.5: *Time series with piecewise constant variance (left) and estimated volatilities (right).*

with parameters $\alpha_0 > 0$ and $\alpha_1, \beta_1 \geq 0$. Note that GARCH(1,1) processes with the restriction $\alpha_1 + \beta_1 < 1$ are stationary and thus fulfill the assumption of Theorem 4.2. Figure 4.4 shows such a time series of length $N = 1000$ generated from a GARCH(1,1) model with coefficients $\alpha_0 = 0.1$, $\alpha_1 = 0.1$ and $\beta_1 = 0.8$. Additionally, we see the volatilities estimated by the Q_n when using a window width of $n = 50$. Q_n tracks the volatility rather well. However, since for online analyses only past observations are taken into account, a sudden increase or decrease in volatility is traced with some time delay. This time delay is relatively small for the Q_n leading to a smooth sequence of estimates.

The second time series of length $N = 1000$ (shown in Figure 4.5) possesses a piecewise constant volatility σ_t, which equals 1, 5, 1, 3, and 5, in time periods of length 200, 250, 150, 250, and 150, respectively. Such models with piecewise constant volatility

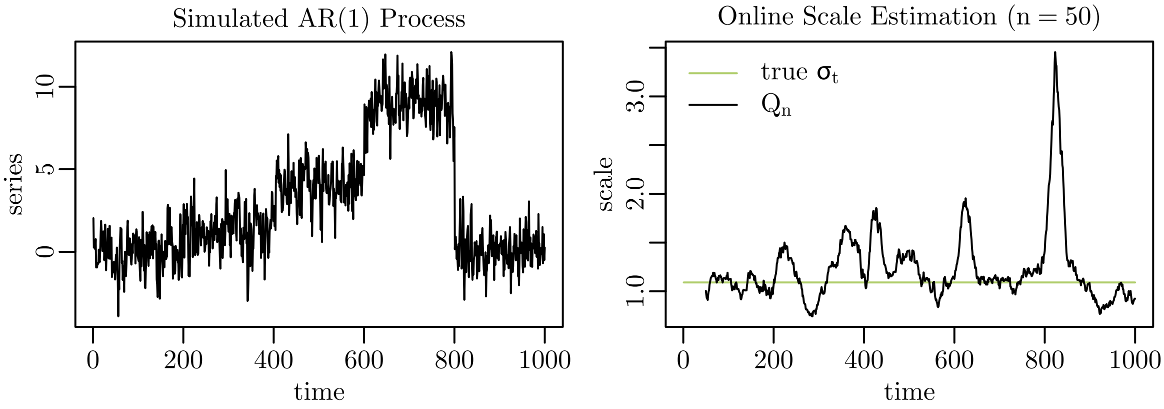


FIGURE 4.6: *Time series with piecewise constant level plus additive noise generated from an AR*(1) *model (left) and estimated volatilities (right).*

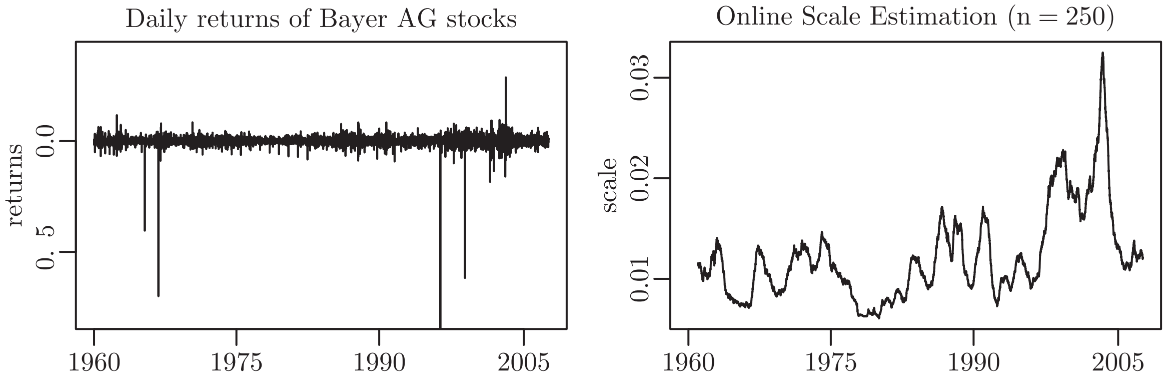


FIGURE 4.7: *Daily returns of Bayer AG stocks between January 4, 1960 and July 17, 2007 (left) and estimated volatilities (right).*

are suggested by Mercurio and Spokoiny (2004) to approximate financial time series. The simulated time series shown in Figure 4.5 consists of independent normal errors with standard deviations given above, plus 10% positive additive outliers of size $6\sigma_t$ at random time points. The Q_n can cope with the outliers and yields a stable scale estimation over time.

The third example (depicted in Figure 4.6) consists of a time series of length $N = 1000$ with positive level shifts of size 1, 3, and 5 at times 201, 401, 601, respectively, and a negative level shift of size -9 at time 801. Thus, the observations vary around the values 0, 1, 4, 9, and 0. The errors are generated from an AR(1) model which is defined by

$$X_t = \varphi X_{t-1} + \varepsilon_t \quad t \in \mathbb{Z} \ ,$$

where $\varepsilon_t \sim \mathcal{N}(0,1)$. The variance in this model is $\sigma_t^2 = 1/\left(1-\varphi^2\right)$, which is indepen-

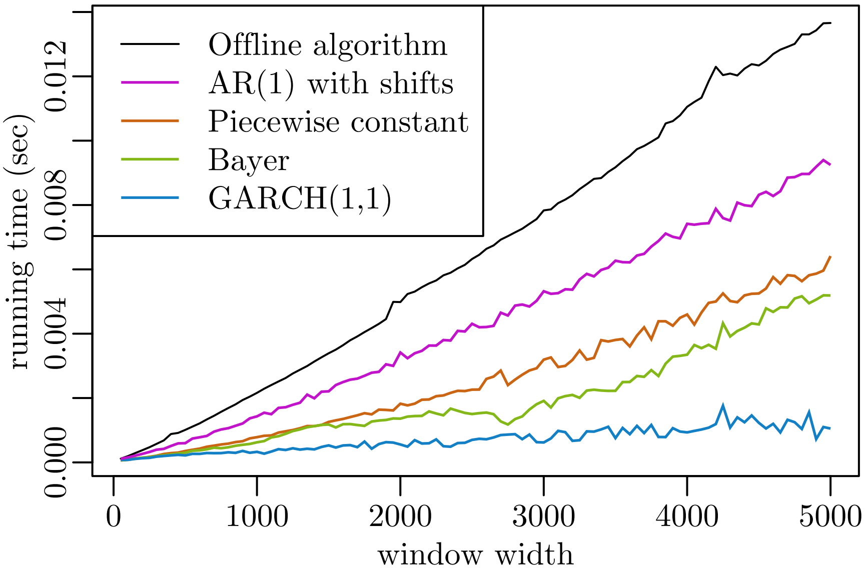


FIGURE 4.8: *Running time needed for the online analyses of the four considered data situations in comparison to the offline algorithm for different window widths.*

dent of t, see e. g. Brockwell and Davis (2002). Figure 4.6 shows a time series simulated according to the settings above with $\varphi = 0.4$. We observe that the estimations lie close to the true value of σ_t but that shifts cause strong biases. Larger shifts have larger impact on the online scale estimation.

The fourth time series is a real data set depicted in Figure 4.7. It consists of the *returns* (the first differences of the logarithms of daily closing prices) of Bayer AG stocks between January 4, 1960 and July 17, 2007. The variability of these returns is important for assessing derivate finance products like options. Because of unexpected events like regulatory changes, technological advances, natural catastrophes or accidents, financial time series can contain arbitrarily large outliers. We therefore should apply a robust scale estimator to evaluate the (local) variability. In this particular case, this is especially reasonable since the underlying prices are not adjusted for dividends and splits. Figure 4.7 shows that the time series obviously contains periods of increased or decreased variability as well as some outliers, a few of them being rather large. It also shows the estimates of the time-varying volatility obtained from applying the Q_n to a moving window of width $n = 250$ corresponding to the commonly assumed number of trading days within one year. Q_n resists the outliers well since there are only a few very large ones.

On these four data sets, we analyze the average time needed for an update of Q_n when using windows of width $50 \leq n \leq 5000$ (shown in Figure 4.8). For windows with width $n < 50$, the difference in running times is too small for accurate measurement, i. e. smaller than 0.1ms, and there is little difference in using the offline or the online

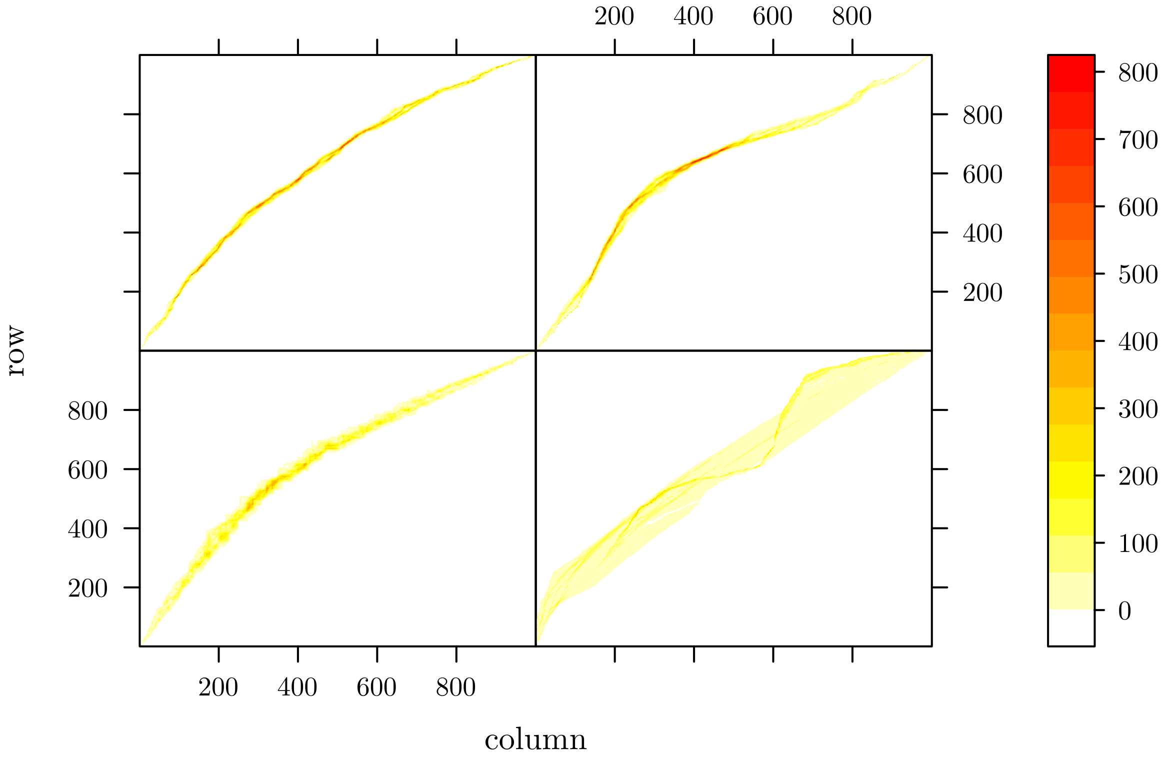


FIGURE 4.9: *Positions of the buffer B in the matrix M for the GARCH$(1,1)$ data (top left), the model with piecewise constant volatility (top right), the Bayer AG data (bottom left), and the data with level shifts and AR(1) errors (bottom right).*

algorithm.

In order to get similar situations for all window widths, we generate new data sets for each n. For the GARCH model we simulate data sets of size $N = n + 2500$. For the time series with a piecewise constant σ_t and the AR(1) process with level shifts we set the length of each part to $n + 500$.

Figure 4.8 illustrates a very good online running time for the GARCH(1,1) series, as could be expected from Theorem 4.2. In case of the real data example and the data set with piecewise constant variances, we also notice a considerable improvement in running time. The improvement for the time series with level shifts is not as good. The offline computation time is nearly the same for all data sets. Therefore, we include only the overall average of the offline computation times for each window width for comparison.

To gain some insight into the different runtimes needed, we analyze the position of the buffer B in the matrix M over time when performing 1000 updates with a window of size 1000. We see in Figure 4.9 that the more complex data situations result in a more spread buffer and therefore more computation time per update.

4.4 Computation of S_n

So far, we compared robust scale estimators in terms of breakdown point and Gaussian efficiency. Under these criteria, Q_n is superior to S_n. However, these properties are asymptotic and not advantageous in all cases. In fact, Rousseeuw and Croux (1993) recommend using S_n for most applications. Thus, computing S_n fast online is also of high interest. Similar to Section 4.3, we start with a description of the existing offline algorithm and then present a faster online algorithm.

4.4.1 Offline Computation

As a first step, we formulate the computation of S_n as an algorithmic problem. Croux and Rousseeuw (1992) use a slightly different definition of S_n for computation.

Problem 4.2 (Computation of S_n)**.**

Given: A sample $X_1, \ldots, X_n$ and a correction factor c.

Goal: Compute

$$\Big\{\{|X_i - X_j|\,; j \neq i\}_{(\lfloor (n+1)/2 \rfloor)}\,; i = 1, \ldots, n\Big\}_{(\lfloor (n+1)/2 \rfloor)}$$

and multiply it with the correction factor c.

Croux and Rousseeuw (1992) also describe an algorithm to compute S_n that needs time $O(n \log n)$.

Theorem 4.3 (Croux and Rousseeuw, 1992)**.** Computing S_n for a sample $X_1, \ldots, X_n$ is possible in time $O(n \log n)$.

Proof. In a first step, we sort the sample in time $O(n \log n)$. We may compute the inner order statistics

$$\{|X_i - X_j|\,; j \neq i\}_{(\lfloor (n+1)/2 \rfloor)}$$

of Problem 4.2 for each $i = 1, \ldots, n$ by computing

$$\left(\left\{X_{(i)} - X_{(i-1)}, \ldots, X_{(i)} - X_{(1)}\right\} \cup \left\{X_{(i+1)} - X_{(i)}, \ldots, X_{(n)} - X_{(i)}\right\}\right)_{(\lfloor (n+1)/2 \rfloor)} .$$

Note that we do not need to compute all values of these sorted sets. Shamos (1976) presents an algorithm by Jefferson to compute the common median (or $\lfloor (n+1)/2 \rfloor$th order statistic in this case) of two sorted sets in time $O(\log n)$. This has to be done n times and afterwards, the $\lfloor (n+1)/2 \rfloor$th order statistic of the n computed elements has to be computed. Finally, the result is multiplied with the correction factor c. None of these steps takes more time than $O(n \log n)$. □

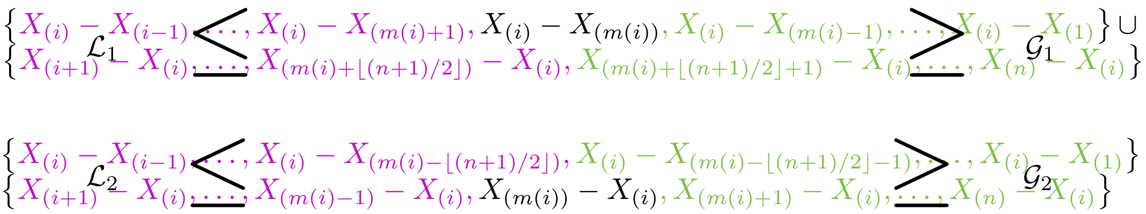


FIGURE 4.10: *Illustration of $\mathcal{L}_1$, $\mathcal{G}_1$, $\mathcal{L}_2$, and $\mathcal{G}_2$.*

4.4.2 Online Computation

To our knowledge, there is no online algorithm for S_n, yet. Fried *et al.* (2006) present an online algorithm for the *repeated median estimator*, which also contains an inner and an outer median and therefore bears similarity to the S_n. The online algorithm for the repeated median needs time $O(n)$ per update. However, we state a much simpler algorithm for the S_n which also needs time $O(n)$ per update.

To construct this online algorithm, we review the offline algorithm sketched in the proof of Theorem 4.3. The offline algorithm works on multisets

$$\mathcal{C}(i) = \left\{X_{(i)} - X_{(i-1)}, \ldots, X_{(i)} - X_{(1)}\right\} \cup \left\{X_{(i+1)} - X_{(i)}, \ldots, X_{(n)} - X_{(i)}\right\} \quad .$$

The two sets in $\mathcal{C}(i)$ are sorted in increasing order. The algorithm presented by Shamos (1976) that computes the common $\lfloor (n+1)/2 \rfloor$th order statistic of these sets $\mathcal{C}(i)$ returns the information which elements are smaller or equal and which elements are greater or equal than the computed value. Thus, it is convenient to imagine that $\left\{X_{(i)} - X_{(i-1)}, \ldots, X_{(i)} - X_{(1)}\right\}$ and $\left\{X_{(i+1)} - X_{(i)}, \ldots, X_{(n)} - X_{(i)}\right\}$ are split into halves by the algorithm.

We can easily adapt the offline algorithm to return values $m(i)$ that implicitly define multisets reflecting this split. The multiset $\mathcal{L}_1$ contains the elements smaller or equal than $\mathcal{C}(i)_{\lfloor (n+1)/2 \rfloor}$ and $\mathcal{G}_1$ contains the elements greater or equal than $\mathcal{C}(i)_{\lfloor (n+1)/2 \rfloor}$ in the case that $\mathcal{C}(i)_{\lfloor (n+1)/2 \rfloor}$ is in the first set $\left\{X_{(i)} - X_{(i-1)}, \ldots, X_{(i)} - X_{(1)}\right\}$. $\mathcal{L}_2$ and $\mathcal{G}_2$ comprise the case that $\mathcal{C}(i)_{\lfloor (n+1)/2 \rfloor}$ is in the second set $\left\{X_{(i+1)} - X_{(i)}, \ldots, X_{(n)} - X_{(i)}\right\}$. Figure 4.10 illustrates this idea. More precisely, the multisets $\mathcal{L}_1$, $\mathcal{G}_1$, $\mathcal{L}_2$, and $\mathcal{G}_2$ are defined as

$$\begin{aligned}
\mathcal{L}_1(m(i)) &= \left\{X_{(i)} - X_{(i-1)}, \ldots, X_{(i)} - X_{(m(i)+1)}\right\} \cup \\
&\quad \left\{X_{(i+1)} - X_{(i)}, \ldots, X_{(m(i)+\lfloor (n+1)/2 \rfloor)} - X_{(i)}\right\} \\
\mathcal{G}_1(m(i)) &= \left\{X_{(i)} - X_{(m(i)-1)}, \ldots, X_{(i)} - X_{(1)}\right\} \cup \\
&\quad \left\{X_{(m(i)+\lfloor (n+1)/2 \rfloor+1)} - X_{(i)}, \ldots, X_{(n)} - X_{(i)}\right\} \\
\mathcal{L}_2(m(i)) &= \left\{X_{(i)} - X_{(i-1)}, \ldots, X_{(i)} - X_{(m(i)-\lfloor (n+1)/2 \rfloor)}\right\} \cup \\
&\quad \left\{X_{(i+1)} - X_{(i)}, \ldots, X_{(m(i)-1)} - X_{(i)}\right\} \\
\mathcal{G}_2(m(i)) &= \left\{X_{(i)} - X_{(m(i)-\lfloor (n+1)/2 \rfloor-1)}, \ldots, X_{(i)} - X_{(1)}\right\} \cup \\
&\quad \left\{X_{(m(i)+1)} - X_{(i)}, \ldots, X_{(n)} - X_{(i)}\right\}
\end{aligned}$$

such that

$$\begin{aligned} \exists j \in \{1,2\} \quad : \quad & (|\mathcal{L}_j(m(i))| = \lfloor (n+1)/2 \rfloor - 1) \wedge \\ & \big(\forall s \in \mathcal{L}_j(m(i)), g \in \mathcal{G}_j(m(i)) : s \leq \left|X_{(i)} - X_{(m(i))}\right| \leq g\big) \quad . \end{aligned}$$

It is possible to let the algorithm return the values $m\,(i)$, because $\left|X_{(i)} - X_{(m(i))}\right|$ is what the original offline algorithm computes as $\mathcal{C}(i)_{\lfloor (n+1)/2 \rfloor}$. The algorithm only has to report the position of $\left|X_{(i)} - X_{(m(i))}\right|$ in $\mathcal{C}(i)$ instead of its value. After executing such an adapted offline algorithm on a sample $X_1, \ldots, X_n$ all $m(i)$ are known for each i with $1 \leq i \leq n$. This leads to the possibility of a fast update for the $\lfloor (n+1)/2 \rfloor$th order statistics. We will see, that it is possible to update each of the inner order statistics $\{|X_i - X_j|\,; j \neq i\}_{(\lfloor (n+1)/2 \rfloor)}$ in time $O(1)$ with this information. In principle, only three cases can occur: the order statistic does not change, the biggest element in $\mathcal{L}_j\,(m(i))$ takes its place, or the smallest element in $\mathcal{G}_j\,(m(i))$ takes its place. The update of the outer order statistic then takes time $O\,(n)$ leading to time $O\,(n)$ per update. Let us first look at the framework of the online algorithm working on the observed values $x_1, \ldots, x_N$ of a sample $X_1, \ldots, X_N$ in Algorithm 4.5.

Algorithm 4.5: Moving window computation of S_n

```
Input: observed sample $X = \{x_1, \ldots, x_N\}$, window width $n$
Output: Scale estimate $S_n$ for each window in $X$
Set $X_w := \{x_1, \ldots, x_n\}$
Compute $S_n$ of $X_w$ offline
Store the returned $S_n$ value in $s_n$ and the returned $m\,(i)$ values in $m$
for $t \leftarrow n$ to $|X| - 1$ do
    Call Insert($x_{t+1}$, $X_w$, $m$)
    Call Delete($x_{t-n+1}$, $X_w$, $m$)
    Calculate $\{\left|x_{(i)} - x_{m(i)}\right|; 1 \leq i \leq |X|\}_{(\lfloor (n+1)/2 \rfloor)}$
    Store the calculated value in $s_{t+1}$
end
return the determined scale estimates $s_n, \ldots, s_N$
```

To achieve the desired runtime, we need a data structure S for $X_w = x_1, \ldots, x_n$ that allows the operations `Rank`(S, x_i) and `Search`(S, x_i) for an element x_i in time $O\,(1)$ if the index i of either x_i or $x_{(i)}$ is known, but may take up to $O\,(n)$ time for `Insert`(S, x_i) and `Delete`(S, x_i), respectively. As the first window X_w in Algorithm 4.5 gets sorted in the offline algorithm, it suffices to use two arrays (one sorted by index, one sorted by rank) with a function mapping between index and rank of an element.

As a first observation, we see that the time needed to update the S_n value after an insertion and a deletion is $O(n)$ in Algorithm 4.5 because selecting the $\lfloor (n+1)/2 \rfloor$th order statistic of $\{\left|X_{(i)} - X_{m(i)}\right|; 1 \leq i \leq |X|\}$ determines the runtime. To obtain the overall update time, we have to look at the time needed for insertion and deletion.

We confine insertion and deletion to the case where $\left|X_{(i)} - X_{m(i)}\right| = X_{(i)} - X_{m(i)}$ (the searched element is in the left part of $\mathcal{C}(i)$) because the other case works analogously.

```
Procedure Insert($x_{\text{ins}}$, sample $X$, $m$)
Input: Element to insert $x_{\text{ins}}$, sample $X$, $m$
Output: changed sample $X$, changed $m$
// Desired order statistic before and after the insertion
$k = \lfloor (|X| + 1)/2 \rfloor$, $k' = \lfloor (|X| + 1 + 1)/2 \rfloor$
Insert $x_{\text{ins}}$ into $X$ and determine $p$ such that $x_{(p)} = x_{\text{ins}}$
// Adapt the indices in $m$ to the change caused by insertion
for $i \leftarrow 1$ to $|X| - 1$ do if $m(i) \geq p$ then $m(i) = m(i) + 1$
for $i \leftarrow |X|$ downto $p + 1$ do $m(i) = m(i - 1)$
Compute $m(p)$ by computing $\mathcal{C}(p)_{\lfloor (n+1)/2 \rfloor}$
foreach $i \in \{1, \ldots, |X|\} \setminus \{p\}$ do
    // Insertion into $\mathcal{L}_1(m(i))$ and no change in $k$
    if $\left|x_{(i)} - x_{(p)}\right| \leq x_{(i)} - x_{(m(i))}$ and $m(i) + k + 1 \geq p > m(i)$ and $k = k'$ then
        if $m(i) + k + 1 > |X|$ or
        $\big(m(i) + 1 < i$ and $x_{(i)} - x_{(m(i)+1)} \geq x_{(m(i)+k+1)} - x_{(i)}\big)$ then
            $m(i) = m(i) + 1$
        else
            $m(i) = m(i) + k + 1$
        end
    // Insertion into $\mathcal{G}_1(m(i))$ and a change in $k$
    else if $\left|x_{(i)} - x_{(p)}\right| \geq x_{(i)} - x_{(m(i))}$ and $(p < m(i)$ or $p > m(i) + k)$ and $k \neq k'$
    then
        if $m(i) + k + 1 > |X|$ or
        $\big(m(i) - 1 < i$ and $x_{(i)} - x_{(m(i)-1)} \leq x_{(m(i)+k+1)} - x_{(i)}\big)$ then
            $m(i) = m(i) - 1$
        else
            $m(i) = m(i) + k + 1$
        end
    end
end
return $X$, $m$
```

```
Procedure Delete($x_{\mathrm{del}}$, sample $X$, $m$)
Input: Element to delete $x_{\mathrm{del}}$, sample $X$, $m$
Output: changed sample $X$, changed $m$
// Desired order statistic before and after the deletion
$k = \lfloor (|X|+1)/2 \rfloor$, $k' = \lfloor (|X|-1+1)/2 \rfloor$
Determine $p$ such that $x_{(p)} = x_{\mathrm{del}}$ and delete $x_{\mathrm{del}}$ from $X$
// Adapt the indices in $m$ to the change caused by deletion and mark
   the cases where the element $m(i)$ points at was deleted
for $i \leftarrow 1$ to $|X|+1$ do
    if $m(i) = p$ then $\mathsf{del}(i) =$ true else $\mathsf{del}(i) =$ false
    if $m(i) > p$ then $m(i) = m(i) - 1$
end
for $i \leftarrow p$ to $|X|$ do $m(i) = m(i+1)$
foreach $i \in \{1, \ldots, |X|\}$ do
    // Deletion of $x_{(i)} - x_{(m(i))}$ or from $\mathcal{L}_1(m(i))$ and no change in $k$
    if $(m(i) + k \geq p > m(i)$ or $\mathsf{del}(i))$ and $k = k'$ then
        if $m(i) + k > |X|$ or $\big(m(i) - 1 < i$ and $x_{(i)} - x_{(m(i)-1)} \geq x_{(m(i)+k)} - x_{(i)}\big)$
        then
            $m(i) = m(i) - 1$
        else
            $m(i) = m(i) + k$
        end
    // Deletion of $x_{(i)} - x_{(m(i))}$ or from $\mathcal{G}_1(m(i))$ and a change in $k$
    else if $(p \leq m(i)$ or $p > m(i) + k$ or $\mathsf{del}(i))$ and $k \neq k'$ then
        if $\mathsf{del}(i)$ then $m(i) = m(i) - 1$
        if $m(i) + k > |X|$ or $\big(m(i) + 1 < i$ and $x_{(i)} - x_{(m(i)+1)} \leq x_{(m(i)+k)} - x_{(i)}\big)$
        then
            $m(i) = m(i) + 1$
        else
            $m(i) = m(i) + k$
        end
    end
end
return $X$, $m$
```

Theorem 4.4. The time needed to update an S_n scale estimation on a sample of size n after an insertion or deletion is $O(n)$.

Proof. In the first part of the proof, we show the correctness of `Insert` and `Delete`. In the second part, we consider the runtime.

Let X be the sample before insertion or deletion and let $k = \lfloor (|X| + 1)/2 \rfloor$, $k_{\text{ins}} = \lfloor (|X| + 1 + 1)/2 \rfloor$, and $k_{\text{del}} = \lfloor (|X| - 1 + 1)/2 \rfloor$.

Lines 2–4 of `Insert` ensure that m temporarily points to the same elements as before, although some order statistics (indices) changed. Line 5 computes m for the newly inserted value. Lines 6–21 contain the loop doing this for all other values. Lines 7–12 handle the case, that the element $x_{(p)} - x_{(i)}$ defined by the newly inserted value belongs into $\mathcal{L}_1(m(i))$ (the test is for $m\,(i) + k + 1 \geq p > m(i)$ and not $m\,(i) + k \geq p > m(i)$ because the additional element changes the ranks). If $k_{\text{ins}} = k + 1$, we have to do nothing because we already have one additional element in $\mathcal{L}_1(m(i))$. Otherwise, we have to search for $\max \mathcal{L}_1(m(i))$ which is clearly the same as $\max\{x_{(i)} - x_{(m(i)+1)}, x_{(m(i)+k+1)} - x_{(i)}\}$. Lines 14–20 handle insertion into $\mathcal{G}_1(m(i))$ analogously.

Lines 2–7 of `Delete` ensure that m temporarily points to the same elements as before, and marks when the element pointed at is deleted. Lines 8–24 contain the loop correcting the m values. Lines 9–14 handle the case, that an element was deleted from $\mathcal{L}_1(m(i))$. If $k_{\text{del}} = k - 1$, we have nothing to do. Otherwise, we have to search for $\min \mathcal{G}_1(m(i))$, which is clearly the same as $\max\{x_{(i)} - x_{(m(i)-1)}, x_{(m(i)+k)} - x_{(i)}\}$ ($x_{(m(i)+k)} - x_{(i)}$ is an element of $\mathcal{G}_1(m(i))$ and not of $\mathcal{L}_1(m(i))$ because one element of $\mathcal{L}_1(m(i))$ was deleted). Lines 9–14 handle deletions from $\mathcal{G}_1(m(i))$ analogously. A special case occurs, when the element originally pointed at by $m(i)$ is deleted. If $k_{\text{del}} = k$ we have to do the same as if deleting from $\mathcal{L}_1(m(i))$. Otherwise, we have to care about the ranks. Decreasing the old value of $m\,(i)$ and beyond that doing the same as if deleting from $\mathcal{G}_1(m(i))$ handles this case correctly. Therefore, all cases are handled correctly and `Insert` and `Delete` work correctly.

The runtime is determined by the operations that select elements of X by rank. In `Insert` and `Delete` this is done $O(n)$ times. When we use the described data structure, this needs time $O\,(n)$. After `Insert` and `Delete` Algorithm 4.5 spends $O(n)$ time to update the S_n value. Thus, the overall time is as claimed.

□

5 Computing the Least Quartile Difference Estimator in the Plane

The main reason why Rousseeuw and Croux (1993) suggest Q_n and S_n as alternatives to the MAD for robust scale estimation is their advantage in Gaussian efficiency. For the case of linear regression, we saw in Section 3.1 that the LMS estimator shows robustness advantages over LAD and LS. However, the LMS also has the disadvantage of a low Gaussian efficiency, which is asymptotically 0% (Croux *et al.*, 1994). Croux *et al.* (1994) propose the *least quartile difference* estimator as an alternative.

Definition 5.1. The *least quartile difference* (LQD) estimates $\hat{\beta}_0, ..., \hat{\beta}_p = \hat{\beta}_{\text{LQD}}$ of the regression parameters $\beta_0, ..., \beta_p$ are given by

$$\hat{\beta}_{\text{LQD}} = \min_{\beta_0,\ldots,\beta_p} \{|r_i(\beta_0,\ldots,\beta_p) - r_j(\beta_0,\ldots,\beta_p)|\,; i < j\}_{\binom{h_p}{2}} \text{ with } h_p = \left\lfloor \frac{n+p+2}{2} \right\rfloor \ .$$

Note, that computing $\{|r_i(\beta_0,\ldots,\beta_p) - r_j(\beta_0,\ldots,\beta_p)|\,; i < j\}_{\binom{h_p}{2}}$ corresponds to computing the Q_n estimator on residuals. The LQD is a high-breakdown method with a breakdown point of 50%, it has an asymptotic Gaussian efficiency of 67.1% and does not presuppose a symmetric distribution. An additional property is that the function LQD minimizes does not depend on the intercept term. Taking a look at the definition of a residual difference $r_i(\beta_0,\ldots,\beta_p) - r_j(\beta_0,\ldots,\beta_p)$ (residuals were defined in Definition 3.7) we see why this is the case. The residual difference of the estimates $\hat{\beta}_0, ..., \hat{\beta}_p$

$$\begin{aligned} r_i - r_j &= y_i - \hat{Y}_i - \left(y_j - \hat{Y}_j\right) \\ &= y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \ldots + \hat{\beta}_p x_{ip}\right) - y_j + \hat{\beta}_0 + \hat{\beta}_1 x_{j1} + \ldots + \hat{\beta}_p x_{jp} \\ &= (y_i - y_j) - \hat{\beta}_1 (x_{i1} - x_{j1}) - \ldots - \hat{\beta}_p (x_{ip} - x_{jp}) \qquad (5.1) \end{aligned}$$

does not depend on $\hat{\beta}_0$. Therefore, the intercept of the LQD regression has to be estimated afterwards, e. g. by

$$\text{med}\{y_i - (\hat{\beta}_1 x_{i1} + \ldots + \hat{\beta}_p x_{ip}) \mid 1 \leq i \leq n\} \ .$$

Several algorithms exist for the LQD. In the following, we state results for the case $p = 1$, i. e. estimation in the plane. In their article, introducing the LQD estimator, Croux *et al.* (1994) propose to use the subset algorithm developed by Rousseeuw and Leroy (1987). The subset algorithm is based on examining subsets of the data points that determine local solutions, i. e. estimates $\hat{\beta}_0, ..., \hat{\beta}_p$ that are not necessarily the global solution. The $\binom{h_1}{2}$th order statistic of the absolute residual differences of a local solution can be computed in time $O(n \log n)$ as seen in Section 4.3.1. Croux *et al.* (1994) propose to examine all $O(n^2)$ or alternatively just $O(n)$ randomly chosen 2-subsets of the data points, which needs overall time $O(n^3 \log n)$ or $O(n^2 \log n)$, respectively. However, the resulting algorithm is not exact because the global solution is not necessarily determined by a 2-subset. The exact algorithm they propose needs time $O(n^5 \log n)$. Another possibility to compute the LQD regression fit is to adapt LMS or *least quantile of squares* (LQS) algorithms. LQS is a generalization of LMS.

Definition 5.2. The *least quantile of squares* (LQS) estimates $\hat{\beta}_0, ..., \hat{\beta}_p = \hat{\beta}_{\text{LQS}}$ of the regression parameters $\beta_0, ..., \beta_p$ are given by

$$\hat{\beta}_{\text{LQS}} = \min_{\beta_0,\ldots,\beta_p} \{r_1(\beta_0, \ldots, \beta_p)^2, \ldots, r_n(\beta_0, \ldots, \beta_p)^2\}_{(h_p)} \text{ with } 1 \leq h_p \leq n \ .$$

The adaption proposed by Croux *et al.* (1994) leads to a running time of $O(n^4)$, if the algorithms of Souvaine and Steele (1987) or Edelsbrunner and Souvaine (1990) for computing LMS in time $O(n^2)$ are used. Agulló (2002) proposes an approximation algorithm for LQD, but only gives empirical running time results.

Due to the high computational effort needed when using common algorithms, the LQD is not widely used, yet. However, Dryden and Walker (1999) propose to use it for object matching in biology and Mebane, Jr. and Sekhon (2004) use the LQD fit to detect outliers in vote counts.

5.1 The Concept of Geometric Duality

To construct a faster algorithm for LQD estimation, we utilize *geometric duality.* Classically, *geometric duality* is defined in the plane. Points in the plane and lines in the plane are both described by two parameters. It is therefore possible to map a set of points to a set of lines, and vice versa, in a one-to-one manner. Such a mapping from a *primal space* to a *dual space* is called *duality transform* and typically preserves certain properties and relations existing in the primal space. It is widely believed, that it is easier to search for a point in an arrangement of lines, than to search for a line through a set of points. As a matter of fact, the data structures and algorithms used in computational geometry strongly support that belief.

We concentrate on the version of geometric duality that Chazelle *et al.* (1985) propose for solving geometrical problems. In particular, we state the formulation of de Berg *et al.* (2008). Their geometric transform maps a primal point $p = (\beta_1, \beta_0)$ to a dual line $T_p : y = \beta_1 x - \beta_0$ and a primal line $\ell : y = \beta_1 x + \beta_0$ to a dual point

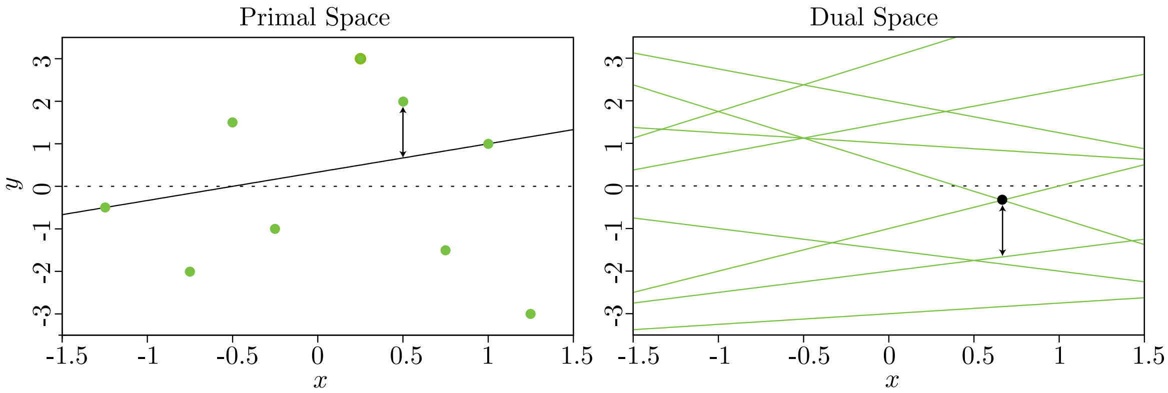


FIGURE 5.1: *Nine points and a line in primal space and the corresponding dual space.*

$T_\ell = (\beta_1, -\beta_0)$. The merit of this transformation is that relative orientations are preserved. We denote relative orientation with the help of half-spaces:

Definition 5.3. Let $h : y = \beta_1 x_1 + \ldots + \beta_p x_p + \beta_0$ be a hyperplane. The *open half-spaces* above and below H are defined by

$$\begin{aligned} h^+ : y &> \beta_1 x_1 + \ldots + \beta_p x_p + \beta_0 \\ h^- : y &< \beta_1 x_1 + \ldots + \beta_p x_p + \beta_0 \ . \end{aligned}$$

The corresponding *closed half-spaces* are defined by

$$\begin{aligned} h^\oplus : y &\geq \beta_1 x_1 + \ldots + \beta_p x_p + \beta_0 \\ h^\ominus : y &\leq \beta_1 x_1 + \ldots + \beta_p x_p + \beta_0 \ . \end{aligned}$$

We call h^+ and $h^\oplus$ upper half-spaces and h^- and $h^\ominus$ lower half-spaces.

Under the geometric transformation we use, a primal point p lies above the primal line ℓ, i. e. in ℓ^+, if and only if the dual point T_ℓ lies above the dual line T_p, i. e. in T_p^+. Incidence is also preserved: a primal point p is incident to a primal line ℓ if and only if the dual line T_p is incident to the dual point T_ℓ.

Figure 5.1 shows an example of primal and dual space. This duality transform can easily be extended to higher dimensions, where a primal point $(\beta_1, \ldots, \beta_p, \beta_0)$ is mapped to the dual hyperplane $y = \beta_1 x_1 + \ldots + \beta_p x_p - \beta_0$ and a primal hyperplane $y = \beta_1 x_1 + \ldots + \beta_p x_p + \beta_0$ is mapped to the dual point $(\beta_1, \ldots, \beta_p, -\beta_0)$.

Starting with the work of Chazelle *et al.* (1985) geometric duality is a frequently used concept in computational geometry and Johnstone and Velleman (1985) introduced the concept to robust regression.

5.2 Computing the LQD Geometrically

Methods from computational geometry frequently encounter problems with degenerate cases like parallel hyperplanes. To avoid the handling of degenerate cases, we adopt the typical geometric assumption that the inputs are in general position (see Definition 3.10). For most situations general position is given with overwhelming probability (see e. g. Rousseeuw and Leroy, 1987). If general position is not given, it is possible to use well-known techniques to handle that case (see e. g. Edelsbrunner and Mücke, 1990).

Using a duality transform, we are able to compute the LQD with the more general problem of finding the minimum of the k-level in an arrangement of hyperplanes.

Definition 5.4. Let H be a set of n non-vertical hyperplanes in $\mathbb{R}^d$. For a point $p \in \mathbb{R}^d$, let $a\,(p)$ and $b\,(p)$ be the number of hyperplanes h in H such that p is in h^- and h^+, respectively. For $1 \leq k \leq n$, define the *k-level* as the set of points p with

$$a(p) \leq n-k \text{ and } b(p) \leq k-1 \ .$$

Problem 5.1 (Minimum of k-level).

Given: A set H of n non-vertical hyperplanes in $\mathbb{R}^d$ and a parameter k.

Goal: Let the y-axis be the axis that distinguishes between upper and lower half-space of the given hyperplanes (as in Definition 5.3). Find a point on the k-level with minimal y-coordinate.

We call a point on the k-level a *local solution* and the minimum point on the k-level the *global solution*. To compute the LQD, we use a modified concept of geometric duality and the fact that the LQD estimator is independent of the intercept. This enables us to redefine the LQD as a dual problem:

Problem 5.2 (Dual LQD).

Given: A data set of n explanatory data points $x_1, \ldots, x_n \in \mathbb{R}^p$, n responses $y_1, \ldots, y_n \in \mathbb{R}$, and a positive integer h_p.

Transformation: We transform the points for $1 \leq i < j \leq n$ to $2\binom{n}{2}$ hyperplanes

$$\begin{aligned} h_{i,j} \quad &: \quad v = +(x_{i1} - x_{j1})u_1 + \ldots + (x_{ip} - x_{jp})u_p - (y_i - y_j) \\ \bar{h}_{i,j} \quad &: \quad v = -(x_{i1} - x_{j1})u_1 - \ldots - (x_{ip} - x_{jp})u_p + (y_i - y_j) \ . \end{aligned}$$

Goal: Find a point $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ with minimal v-coordinate $r > 0$ that lies in at least $\binom{n}{2} + \binom{h_p}{2}$ closed half-spaces $h^{\oplus}_{i,j}$ or $\bar{h}^{\oplus}_{i,j}$, i. e. the minimum of the $\binom{n}{2} + \binom{h_p}{2}$-level.

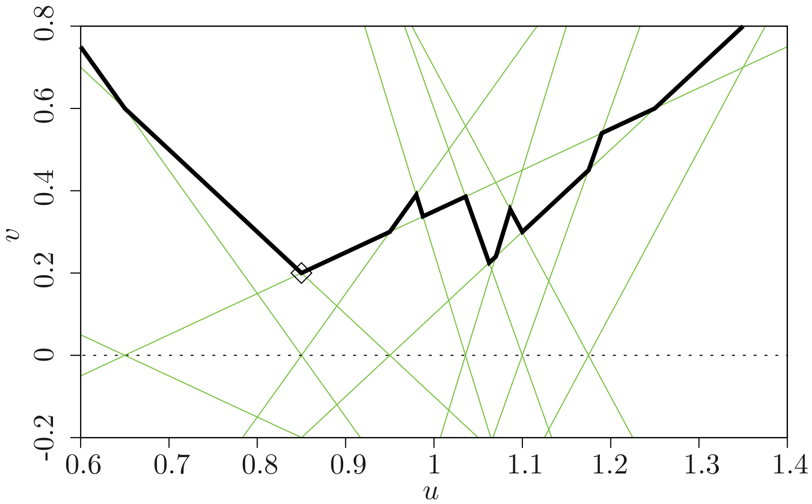


FIGURE 5.2: *An example for the mapping of the points* $(0, 0.15)$*,* $(1, 0.8)$*,* $(3, 2.7)$*, and* $(7, 7.4)$ *to 12 dual lines. The LQD solution for* $h_p = 3$ *is determined by the minimum 9-level point, here:* $(0.85, 0.2)$*. The bold line shows local solutions. The LQD regression line is therefore* $y = 0.85x + \beta_0$*, and the corresponding third order statistic of the absolute residual differences takes on its minimal value of* 0.2*.*

We will show in the next lemma, that an optimal LQD solution is obtained by solving Problem 5.2. An example in the plane is given in Figure 5.2.

Lemma 5.1. *Let* $h_p = \lfloor (n + p + 2)/2 \rfloor$*. If the point* $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ *is an optimal solution of Problem 5.2, then* $\hat{\beta}_1, \ldots, \hat{\beta}_p$ *are the LQD estimates of the regression parameters* $\beta_1, ..., \beta_p$ *in primal space and* r *is the minimal* $\binom{h_p}{2}$*th order statistic of* $\{|r_i(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p) - r_j(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)|; i < j\}$ *for arbitrary intercept* $\hat{\beta}_0$*.*

Proof. Let $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ be an optimal solution of Problem 5.2 and consider arbitrary i and j with $1 \leq i < j \leq n$ and the corresponding hyperplanes $h_{i,j}$ and $\bar{h}_{i,j}$. Now, consider the following three possible cases:

1. $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ is in $h_{i,j}^{\oplus} \cap \bar{h}_{i,j}^{\oplus}$.
2. $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ is in $h_{i,j}^{\oplus} \cap \bar{h}_{i,j}^{-}$ or in $h_{i,j}^{-} \cap \bar{h}_{i,j}^{\oplus}$.
3. $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ is in $h_{i,j}^{-} \cap \bar{h}_{i,j}^{-}$.

Case 3 does not occur because

$$\begin{aligned} & (\hat{\beta}_1, \ldots, \hat{\beta}_p, r) \text{ is in } h_{i,j}^{-} \cap \bar{h}_{i,j}^{-} \\ \Leftrightarrow\quad & +(x_{i1} - x_{j1})\hat{\beta}_1 + \ldots + (x_{ip} - x_{jp})\hat{\beta}_p - (y_i - y_j) > r \text{ and} \\ & -(x_{i1} - x_{j1})\hat{\beta}_1 - \ldots - (x_{ip} - x_{jp})\hat{\beta}_p + (y_i - y_j) > r \\ \Rightarrow\quad & r < 0 \end{aligned}$$

but $r \geq 0$. In case 1, the stated relations translate to the original problem as follows:

$$\begin{aligned}
& (\hat{\beta}_1, \ldots, \hat{\beta}_p, r) \text{ is in } h^{\oplus}_{i,j} \cap \bar{h}^{\oplus}_{i,j} \\
\Leftrightarrow \quad & +(x_{i1} - x_{j1})\hat{\beta}_1 + \ldots + (x_{ip} - x_{jp})\hat{\beta}_p - (y_i - y_j) \leq r \text{ and} \\
& -(x_{i1} - x_{j1})\hat{\beta}_1 - \ldots - (x_{ip} - x_{jp})\hat{\beta}_p + (y_i - y_j) \leq r \\
\Leftrightarrow \quad & \left|(y_i - y_j) - (x_{i1} - x_{j1})\hat{\beta}_1 - \ldots - (x_{ip} - x_{jp})\hat{\beta}_p\right| \leq r \\
\overset{(5.1)}{\Leftrightarrow} \quad & \text{For arbitrary } \hat{\beta}_0 : |r_i(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p, r) - r_j(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p, r)| \leq r \ . \qquad (5.2)
\end{aligned}$$

Now, recall that $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ is in at least $\binom{n}{2} + \binom{h_p}{2}$ closed upper half-spaces. Because of counting arguments, there are at least $\binom{h_p}{2}$ pairs (i, j) such that $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ is in $h^{\oplus}_{i,j} \cap \bar{h}^{\oplus}_{i,j}$. Due to Equation (5.2), we obtain at least $\binom{h_p}{2}$ absolute residual differences smaller than or equal to r with respect to a hyperplane with parameters $\hat{\beta}_1, \ldots, \hat{\beta}_p$ and an arbitrary parameter $\hat{\beta}_0$. In addition, $r \geq 0$ is the minimal value, such that $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ is in at least $\binom{n}{2} + \binom{h_p}{2}$ closed upper half-spaces. Therefore, $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ is in at most $\binom{n}{2} + \binom{h_p}{2} - 1$ open upper half-spaces ($(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ has to be incident to a hyperplane) and due to counting arguments at most $\binom{h_p}{2} - 1$ absolute residual differences are strictly smaller than r. Hence, r is the $\binom{h}{2}$th order statistic of $\{|r_i(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p) - r_j(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)|; i < j\}$.

We claim that no other hyperplane $y = \beta'_1 x_1 + \ldots + \beta'_p x_p + \beta'_0$ leads to a smaller $\binom{h_p}{2}$th order statistic r'. Assume for the sake of contradiction that there are parameters $\beta'_1, \ldots, \beta'_p$ leading to a smaller $\binom{h_p}{2}$th order statistic r'. Due to Equation (5.2), $(\beta'_1, \ldots, \beta'_p, r')$ is in $\binom{h_p}{2}$ half-space intersections of the type $h^{\oplus}_{i,j} \cap \bar{h}^{\oplus}_{i,j}$, that are defined by $2\binom{h_p}{2}$ hyperplanes. $(\beta'_1, \ldots, \beta'_p, r')$ lies in at least $(2\binom{n}{2} - 2\binom{h_p}{2})/2$ upper half-spaces defined by part of the remaining $2\binom{n}{2} - 2\binom{h_p}{2}$ hyperplanes (recall, that case three does not occur). Thus, $(\beta'_1, \ldots, \beta'_p, r')$ lies in at least $2\binom{h_p}{2} + (2\binom{n}{2} - 2\binom{h_p}{2})/2 = \binom{n}{2} + \binom{h_p}{2}$ closed upper half-spaces and therefore is a solution to Problem 5.2 with $r' < r$. That is a contradiction, because $(\hat{\beta}_1, \ldots, \hat{\beta}_p, r)$ is the global solution to Problem 5.2 (and therefore the local solution with the smallest v-coordinate). Hence, $y = \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_p x_p + \hat{\beta}_0$ minimizes the $\binom{h_p}{2}$th order statistic of $\{|r_i(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p) - r_j(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)|; i < j\}$. □

Lemma 5.1 enables us to compute the LQD by using a duality transform and solving Problem 5.1. Thus, we can concentrate on solving Problem 5.1. All of the following algorithms work for $p = 1$, i. e. estimation in the plane. As a first step, we use the known algorithms for Problem 5.1.

Theorem 5.1. It is possible to compute the LQD estimator for n data points in the plane in expected running time $O(n^2 \log n)$ or deterministic running time $O(n^2 \log^2 n)$.

Proof. Problem 5.1 and the equivalent *k-violation linear programming* are solved with *parametric search* (see Megiddo, 1979) by Cole *et al.* (1987) and by Roos and Widmayer (1994) in time $O(n \log^2 n)$. Chan (1999) states a randomized algorithm using *cuttings* that needs expected time $O(n \log n)$ for Problem 5.1. As we consider an input transformation to $O(n^2)$ hyperplanes in Problem 5.2 and Lemma 5.1 shows that solving Problem 5.1 on a transformed input computes the LQD, the theorem is proven. □

Regression through the origin names regression with fixed intercept 0. It is easy to see that a transformation of n points $p_i = (x_{i1}, \ldots, x_{ip}, y_i)$ to $2n$ hyperplanes

$$\begin{aligned} h_i &: v = +x_{i1}u_1 + \ldots + x_{ip}u_p - y_i \\ \bar{h}_i &: v = -x_{i1}u_1 - \ldots - x_{ip}u_p + y_i \end{aligned}$$

enables us to compute LQS and LMS regression through the origin with Problem 5.1.

Corollary 5.1. *It is possible to compute LQS and LMS regression through the origin for n data points in the plane in expected running time $O(n \log n)$ or deterministic running time $O(n \log^2 n)$.*

This improves a result of Barreto and Maharry (2006), who state an algorithm with running time $O(n^2 \log n)$ for LMS regression through the origin.

The shortcomings of the algorithms proposed in Chan (1999), Cole *et al.* (1987), and Roos and Widmayer (1994) are the rather complicated techniques involved (see e. g. Agarwal and Sharir, 1998 or van Oostrum and Veltkamp, 2002 for drawbacks of *parametric search* and e. g. Section 4 of Har-Peled, 1998 for problems in the implementation of *cutting* algorithms). To overcome these shortcomings, we provide two easy to implement algorithms with similar running times. The algorithms are based on solving a decision problem that underlies Problem 5.1.

5.3 Solving the Underlying Decision Problem

In the following, we specify the method `SearchLocalSolution` which is used in the next sections to solve the underlying decision problem: Given a set of non-vertical lines L and a fixed value $r \in \mathbb{R}$, we need to decide whether a point (x, r) on the k-level of L exists. Let x and y be the axes of the plane and let the y axis distinguish between the upper half-space and the lower half-space of a line. Using this convention, the terms *above* and *below* are clear. We will also use *left* and *right* as relative positions on the x axis. Further, let $v\,(r) : y = r$ denote the vertical line with intercept r.

The idea to solve this decision problem is to use a *plane sweep algorithm* (see e. g. de Berg *et al.*, 2008). We sweep an imaginary *sweep line* from left to right over the plane and change the count of lines above or below the sweep line when we encounter a line in the plane.

The procedure named `SearchLocalSolution` to do this is described in the following. Figure 5.3 additionally illustrates the idea of using a sweep line in the procedure `SearchLocalSolution`.

```
Procedure SearchLocalSolution(set of lines $L$, value $r$, level parameter $k$)
Input: Set of lines $L$, value $r$, level parameter $k$
Output: Information whether a local solution exists at height $r$
Compute all intersections of $v(r)$ with the lines in $L$
Let $s(i)$ be the slope of the line in $L$ that caused intersection $i$
Sort the intersections from left to right resulting in $(i_1, \dots, i_{|L|})$ (intersections $i$
    with the same $x$-value are ordered ascending according to $s(i)$)
Determine $b$, the number of different closed upper half-spaces of $L$ that $i_1$ is in
if $s(i_1) < 0$ then $b = b - 1$
foreach $i \in (i_2, \dots, i_{|L|})$ do
    if $s(i) < 0$ then $b = b + 1$ else $b = b - 1$
    if $b \geq k$ then
        return true, i. e. a local solution exists at height $r$
    end
end
return false, i. e. no local solution exists at height $r$ or a smaller height
```

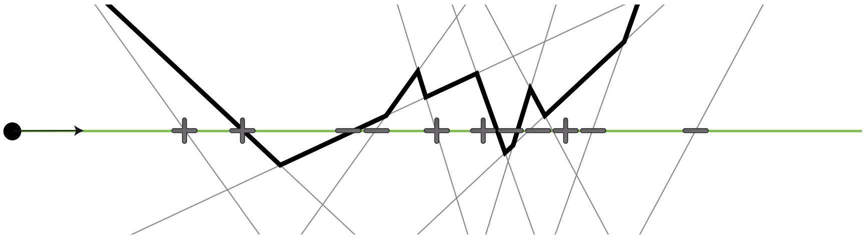


FIGURE 5.3: *The idea behind* ***SearchLocalSolution****.*

Lemma 5.2. *The procedure* `SearchLocalSolution` *needs time $O(n \log n)$ to decide whether a local solution to Problem 5.1 in the plane exists at a given height.*

Proof. It is easy to see, that `SearchLocalSolution` is correct. Of all computation steps, sorting costs most time, namely $O(n \log n)$. ☐

Note, that `SearchLocalSolution` can additionally report the encountered local solution if necessary. We will need this in the following algorithms.

5.4 Searching for the Optimal Point

To search for the minimum point on the k-level, we propose two methods:

1. A search based on the geometric mean to get an approximative solution.
2. A randomized search leading to a Las Vegas algorithm.

In both proposed methods we denote the upper bound for the height of the optimal solution by $r_{\max}$ and the lower bound by $r_{\min}$. To obtain an approximative solution with *approximation ratio* $1+\varepsilon$ the inequality $r_{\max}/r_{\min} \leq 1+\varepsilon$ has to hold (the approximation ratio refers to the height of the computed solution in comparison to the height of the optimal solution and therefore—concerning the LQD—to the ratio between the computed solution and the minimal $\binom{h_1}{2}$th order statistic of the absolute residual differences). We state the approximation algorithm in Algorithm 5.2.

Algorithm 5.2: Approximate minimum of k-level

```
   Input: Set of lines $L$, approximation ratio $\varepsilon$, level parameter $k$
   Output: A point on the $k$-level
1  $r_{\max} = \infty$ and $r_{\min} = 0$
2  foreach $r \in \{0, 1/(1+\varepsilon), 1, 1+\varepsilon\}$ do
3      found = SearchLocalSolution ($L, r, k$)
4      if found and $r < r_{\max}$ then $r_{\max} = r$
5      else if not found and $r > r_{\min}$ then $r_{\min} = r$
6  end
7  if $r_{\max} = 0$ or $r_{\min}, r_{\max} \in \{1/(1+\varepsilon), 1, 1+\varepsilon\}$ then go to 19
8  if $r_{\max} = \infty$ then
9      while not SearchLocalSolution ($L, r_{\min}^2, k$) do $r_{\min} = r_{\min}^2$
10     $r_{\max} = r_{\min}^2$
11 else if $r_{\min} = 0$ then
12     while SearchLocalSolution ($L, r_{\max}^2, k$) do $r_{\max} = r_{\max}^2$
13     $r_{\min} = r_{\max}^2$
14 end
15 while $r_{\max}/r_{\min} > 1+\varepsilon$ do
16     if SearchLocalSolution ($L, \sqrt{r_{\min} r_{\max}}, k$) then $r_{\max} = \sqrt{r_{\min} r_{\max}}$
17     else $r_{\min} = \sqrt{r_{\min} r_{\max}}$
18 end
19 return the local solution at height $r_{\max}$
```

The basic ideas of this algorithm and the randomized algorithm we present later are illustrated in Figure 5.4.

Theorem 5.2. Algorithm 5.2 computes the minimum k-level point with approximation ratio $1+\varepsilon$ $(0 < \varepsilon \leq 1)$ (concerning the y-coordinate) on n lines in the plane in worst case time

$$\begin{cases} O(n \log n \log \log_{(1+\varepsilon)} \max\{\frac{1}{r^*}, r^*\}), & \text{whenever } \max\{\frac{1}{r^*}, r^*\} > 1+\varepsilon \\ O(n \log n), & \text{otherwise} \end{cases},$$

where r^* is the height of the optimal solution.

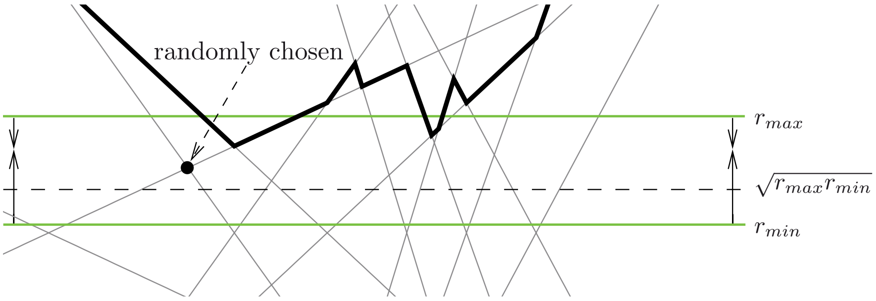


FIGURE 5.4: *The basic idea behind the approximation and the randomized algorithm.*

Proof. The correctness of the algorithm follows from the correctness of the procedure `SearchLocalSolution`. As the running time is bounded (we show this below), the algorithm terminates.

If $r^* = 0$ or $r^* \in [1/(1+\varepsilon), 1+\varepsilon] \Leftrightarrow \max\{\frac{1}{r^*}, r^*\} \leq 1+\varepsilon$, the four initial tests at heights 0, $1/(1+\varepsilon)$, 1, and $1+\varepsilon$ suffice to attain the desired approximation ratio.

If $\max\{\frac{1}{r^*}, r^*\} > 1+\varepsilon$ either $r_{\max}$ is still ∞ or $r_{\min}$ is still 0 after these tests. We only consider the case that $r_{\max}$ is still ∞, because the calculations for the other case are similar. If $r_{\max}$ is still ∞, $r_{\min}$ has to be the greatest of the initially tested values, namely $1+\varepsilon$. After the while loop in line 9 terminates, $r_{\max} = r_{\min}^2$ (recall, that $r_{\min}$ is updated if no local solution is found) and therefore $(r^*)^2 > r_{\max}$ (recall, that $r_{\min} \leq r^* \leq r_{\max}$). Hence, the maximum number of times `SearchLocalSolution` is called to obtain $r_{\max}$ is determined by the smallest integer k_1 that is a solution to

$$(1+\varepsilon)^{2^{k_1}} \geq (r^*)^2 \quad .$$

Therefore, the maximum number of times `SearchLocalSolution` is called is

$$\lceil 2 \log\log r^* - \log\log(1+\varepsilon) \rceil \quad .$$

Since $r_{\max} = r_{\min}^2$, we obtain $r_{\max}/r_{\min} = r_{\min}$. In lines 15–18, the geometric mean of $r_{\min}$ and $r_{\max}$ is tested until the approximation ratio $1+\varepsilon$ is reached. In each loop cycle, we obtain new bounds $r_{\min}$ and $r_{\max}$. One is identical to the former bound, the other is the geometric mean of the former bounds. For the case that $r_{\max}$ is updated, the new ratio between $r_{\max}$ and $r_{\min}$ is

$$\frac{r_{\max}}{r_{\min}} = \frac{\sqrt{r'_{\max} r_{\min}}}{r_{\min}} = \sqrt{\frac{r'_{\max}}{r'_{\min}}} \quad ,$$

where $r'_{\min}$ and $r'_{\max}$ denote the old values of $r_{\min}$ and $r_{\max}$, respectively. The other case leads to the same ratio. Hence, the new ratio is the square root of the old ratio. Since the ratio we begin with is less than r^*, the maximum number of times the procedure

`SearchLocalSolution` is called until we reach a ratio of $1+\varepsilon$ is determined by the smallest integer k_2 that is a solution to

$$(r^*)^{(1/2)^{k_2}} \leq 1+\varepsilon \ .$$

Therefore, the maximum number of times `SearchLocalSolution` is called is

$$\lceil \log\log r^* - \log\log\left(1+\varepsilon\right)\rceil \ .$$

Each call of `SearchLocalSolution` costs time $O\left(n\log n\right)$. □

Corollary 5.2. *Algorithm 5.2 finds the LQD fit with approximation ratio* $1+\varepsilon$ $(0<\varepsilon\leq 1)$ *on* n *points in the plane in worst case time*

$$\begin{cases} O(n^2\log n \log\log_{(1+\varepsilon)}\max\{\frac{1}{r^*},r^*\}), & \text{whenever } \max\{\frac{1}{r^*},r^*\} > 1+\varepsilon \\ O(n^2\log n), & \text{otherwise} \end{cases} ,$$

where r^* *is the* $\binom{h_1}{2}$*th order statistic of the absolute residual differences of the LQD fit.*

Note, that with input values not larger than 2^n, we may also write

$$O\left(n^2\log n\left(\log n - \log\log\left(1+\varepsilon\right)\right)\right)$$

as a simpler but less precise term for the running time.

For the randomized algorithm, we need the following definition.

Definition 5.5. An *inversion* in a permutation π is a pair of values where $i>j$ and $\pi\left(i\right)<\pi\left(j\right)$ or $i<j$ and $\pi\left(i\right)>\pi\left(j\right)$. An *inversion table* contains the number of inversions for each element i, denoted by $\mathrm{inv}(i)$.

We use the inversion table to calculate the number of intersections of an arrangement of lines between two vertical lines.

Procedure `NoOfIntersectionsBetween`(*set of lines* L*, intercept* r_1*, intercept* r_2)

```
Input: Set of lines $L$, intercept $r_1$, intercept $r_2$
Output: Number of intersections of lines in $L$ that lie between $v(r_1)$ and $v(r_2)$
Compute all intersections of $v(r_1)$ with the lines in $L$
Compute all intersections of $v(r_2)$ with the lines in $L$
Label the lines in $L$ according to their intersections on $v(r_1)$ from left to right
Interpret the intersections on $v(r_2)$ as a permutation $\pi$ of the labels
Compute the inversion table of $\pi$
return $\frac{1}{2}\sum_{i\in\pi}\mathrm{inv}(i)$
```

Taking Figure 5.4 as an example, we get $\pi=(3,1,4,2,8,5,7,6,10,9,11)$ if we label the intersections on $v\left(r_{\max}\right)$ (in the boundaries of the figure) from left to right. The corresponding inversion table $(2,1,1,2,3,1,2,2,1,1,0)$ clearly reflects the intersections each line has between the vertical lines. As each intersection is counted two times, the total number of intersections equals half the sum of the inversion table.

Lemma 5.3. *`NoOfIntersectionsBetween` computes the number of intersections of an arrangement of n lines between two vertical lines in time $O(n \log n)$*

Proof. The correctness of `NoOfIntersectionsBetween` is clear. The calculation of the number of intersections between two vertical lines is possible in time $O(n \log n)$, because the inversion table can be computed in this time (for example with an extended merge sort algorithm) and no other step takes more time than $O(n \log n)$. □

We are now able to state the randomized algorithm.

Algorithm 5.4: Randomized algorithm for minimum of k-level

```
Input: Set of lines $L$, level parameter $k$
Output: Minimum point on the $k$-level
$r_{\min} = 0$
Randomly choose a point $(x, 0)$ that is incident to a line in $L$
$r_{\max} = \{y \mid (x, y)$ is a local solution$\}$
$I =$ NoOfIntersectionsBetween $(L, r_{\min}, r_{\max})$
while $I > 0$ do
    // Randomly choose an intersection between $v(r_{\min})$ and $v(r_{\max})$
    Choose a line $\ell$ in $L$ (line $i$ is chosen with probability $\mathrm{inv}(i)/I$)
    Choose an intersection $(\cdot, r_{\mathrm{mid}})$ on $\ell$ uniformly at random ($r_{\min} < r_{\mathrm{mid}} < r_{\max}$)
    if SearchLocalSolution $(L, r_{\mathrm{mid}}, k)$ then $r_{\max} = r_{\mathrm{mid}}$ else $r_{\min} = r_{\mathrm{mid}}$
    $I =$ NoOfIntersectionsBetween $(L, r_{\min}, r_{\max})$
end
return the local solution at height $r_{\max}$
```

Theorem 5.3. Algorithm 5.4 computes the minimum k-level point of n lines in the plane in expected running time $O(n \log^2 n)$.

Proof. The correctness of the algorithm follows from the correctness of the procedures `NoOfIntersectionsBetween` and `SearchLocalSolution`. In the following, we show that the expected running time of the algorithm is $O(n \log^2 n)$.

It is possible to compute a starting value for $r_{\max}$ by computing the best local solution for a fixed x-coordinate in time $O(n \log n)$. To do this, we have to calculate all intersections of the lines with the chosen x-coordinate, sort them according to their y-coordinate, and sift through the intersection points increasing the count of closed upper half-spaces it lies in until we reach the value k.

Randomly choosing an intersection between $v(r_{\min})$ and $v(r_{\max})$ can be done in time $O(n \log n)$, because computing the intersections on the randomly chosen line is the costliest step.

Let $I(v(r_1), v(r_2))$ denote the number of intersections between two horizontal lines $v(r_1)$ and $v(r_2)$ and let $m := I(v(r_{\min}), v(r_{\max}))$. Furthermore, assume that no two intersections have the same y-coordinate. The algorithm choses each intersection between $v(r_{\min})$ and $v(r_{\max})$ with probability $1/m$, therefore the expected number of remaining intersections is

$$\mathrm{E}(I\left(v(r_{\min}), v(r_{\mathrm{mid}})\right)) = \mathrm{E}(I\left(v(r_{\mathrm{mid}}), v(r_{\max})\right)) = \sum_{i=0}^{m-1} \frac{1}{m} i < \frac{m}{2}.$$

Intersections with the same y-coordinate can only lead to a smaller value. Hence, we expect to execute $O\left(\log n\right)$ cycles of the while loop and expect an overall running time of $O(n \log^2 n)$. □

Corollary 5.3. *Algorithm 5.4 finds the LQD fit on n points in expected running time $O(n^2 \log^2 n)$.*

5.5 Running Time Simulations

In contrast to Chan (1999), Cole *et al.* (1987), and Roos and Widmayer (1994), where no implementation of the algorithms in Theorem 5.1 is mentioned, we have implemented Algorithm 5.2 and Algorithm 5.4. The implementation is available as part of the R (R Development Core Team, 2008) package *robfilter* (Fried and Schettlinger, 2008).

While it is theoretically possible to choose ε in such a way that the approximation algorithm is slower than the randomized algorithm, trial runs with our implementations show that the approximative version is generally faster in practice. For the conducted experiments, we used 64 bit floating point numbers according to IEEE 754-1985. If we choose ε sufficiently small and wait until $r_{\min}$ and $r_{\max}$ are indistinguishable from their geometric mean the approximative version computes the same results as the randomized version.

The experiments show that even with such a precision, the approximative version is faster than the randomized one. However, for greater ε it is of course much faster. We compare the approximative version with maximal precision for 64 bit floating point numbers to the approximative version with $\varepsilon = 0.01$ and to the randomized version on two types of data sets with n points. The first type of data set is

$$\left\{(x_i, y_i) \mid x_i = 2\left(i-1\right)/\left(n-1\right); y_i = -x_i + 1.2 + e_1; 1 \le i \le n\right\}$$

and the second is

$$\begin{cases} \left\{(x_i, y_i) \middle| x_i = \frac{2(i-1)}{n-1}; y_i = e_2; 1 \le i \le n\right\}, & \text{whenever } i \le \left\lfloor \frac{n}{2} \right\rfloor + 1 \\ \left\{(x_i, y_i) \middle| x_i = \frac{2(i-1)}{n-1}; y_i = -\frac{1}{10} x_i + \frac{3}{2} + e_2; 1 \le i \le n\right\}, & \text{otherwise} \end{cases}$$

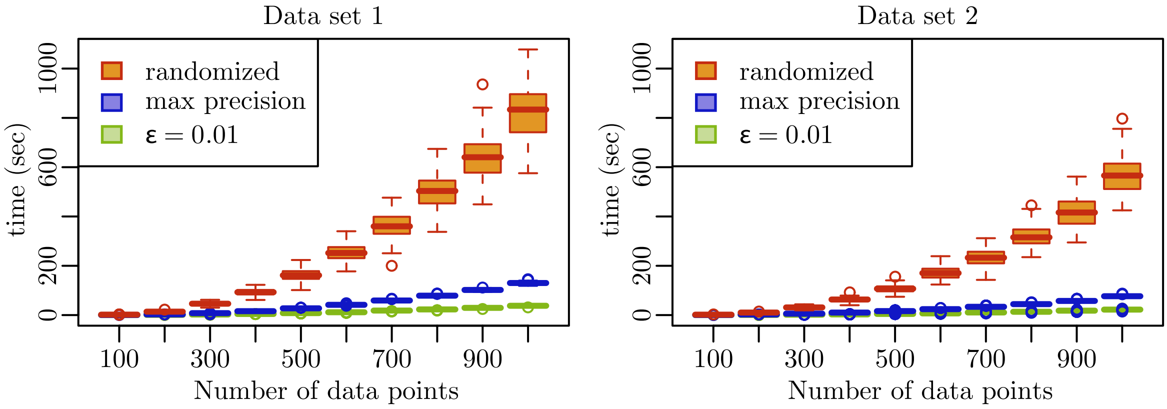


FIGURE 5.5: *Boxplots of the running time in seconds on a Pentium 4 CPU with* 2.56*GHz and* 1024*MB of RAM for the data sets.*

where $e_1 \sim \mathcal{N}(0, 10^{-4})$ and $e_2 \sim \mathcal{N}(0, 10^{-560})$. While the first type of data set represents uncontaminated normal data, the second type contains $\lceil n/2 \rceil - 1$ outliers. Thus, data set number two can result in local solutions that are far from the optimum.

Computing times of these three versions of the algorithm are measured for each n in $\{100, 200, \dots, 1000\}$ for 100 different data sets. The results are shown in Figure 5.5. The figure shows *boxplots* of the running times for each n and each algorithm. These boxplots illustrate the minimal and maximal running time for each n as well as the first and third *quartile* and the median of the running times. In detail, the upper and lower boundary of a box in a boxplot mark values such that 25% of the data is greater or equal and 25% of the data is smaller or equal. The middle line of a box marks the median of the data. The so called *whiskers* above and below a box denote the greatest and smallest value in the data that is not further away from the box than 1.5 times the distance between upper and lower boundary of the box. Points that are further away are marked by a circle.

The boxplots in Figure 5.5 clearly show that the randomized version has a considerably larger variance in its computation time and needs much more time than the approximative version. Another noticeable fact is that the two figures do not differ very much. The high number of outliers and local solutions in the second data set does not slow down the algorithms. On the contrary, the possibility to start at a local solution that is far below other local solutions leads to better performance.

In conclusion, the randomized version of the algorithms presented in Section 5.4 provides a large improvement in computation time on currently available LQD algorithms. However, the experiments show that the proposed approximation algorithm yields even better results. Therefore, these algorithms might increase the practical relevance of LQD regression in the future.

6 An Evolutionary Algorithm for Robust Linear Regression

The algorithms in Chapter 5 work only in the plane, but it is of course desirable to compute robust linear estimators in higher dimensions. Unfortunately, the computation of these estimators is quite hard. More precisely, Bernholt (2005) shows that estimators with the *exact fit property* (whenever a sufficiently large majority of the observations lies on a common hyperplane, an estimator with the exact fit property yields that hyperplane, see e. g. Rousseeuw, 1984) are NP-hard to compute. Under the assumption that the complexity classes NP and P are not equal, we therefore have little hope to compute exact solutions for large high dimensional data sets. All of the so far considered robust estimators—namely LMS, LQS, and LQD—possess the exact fit property (Rousseeuw, 1984; Croux *et al.*, 1994). Here, we additionally consider *least trimmed squares* proposed by Rousseeuw (1984) which also possesses the exact fit property.

Definition 6.1. The *least trimmed squares* (LTS) estimates $\hat{\beta}_0, ..., \hat{\beta}_p = \hat{\beta}_{\text{LTS}}$ of the regression parameters $\beta_0, ..., \beta_p$ are given by

$$\hat{\beta}_{\text{LTS}} = \min_{\beta_0,\ldots,\beta_p} \sum_{i=1}^{h_p} \{r_1\left(\beta_0, \ldots, \beta_p\right)^2, \ldots, r_n\left(\beta_0, \ldots, \beta_p\right)^2\}_{(i)} \text{ with } 1 \leq h_p \leq n \ .$$

The very similar *least trimmed sum of absolute values* (LTA) proposed by Hössjer (1994) is defined as

$$\hat{\beta}_{\text{LTA}} = \min_{\beta_0,\ldots,\beta_p} \sum_{i=1}^{h_p} \{\left|r_1\left(\beta_0, \ldots, \beta_p\right)\right|, \ldots, \left|r_n\left(\beta_0, \ldots, \beta_p\right)\right|\}_{(i)} \text{ with } 1 \leq h_p \leq n$$

and may also be computed by the algorithm we propose in the following.

All of the estimators LMS, LQS, LQD, LTS, and LTA are high-breakdown methods, possessing the optimal finite sample breakdown point of almost 50% for the appropriate choice of h_p.

LQS and LMS have the disadvantage of a low efficiency at Gaussian samples, which is asymptotically 0% (Croux *et al.*, 1994). Arguments for LQS/LMS used to be the easier computation of the objective function compared to LTS and LQD—though the computation of the estimators is still NP-hard—and the intuitive definition. LTS and

LQD are alternatives with higher Gaussian efficiencies of 7.1% and 67.1%, respectively. An additional advantage of LTS is its smooth objective function leading to a lower sensitivity to local effects (Rousseeuw and Van Driessen, 2006).

Because of the above mentioned advantages and disadvantages, the considered estimators are relevant for different data situations. Therefore, algorithms for them are of high interest. We consider situations where exact algorithms are not feasible and therefore concentrate on heuristics. The heuristics most commonly used are PROGRESS for LQS/LMS and LTS (Rousseeuw and Leroy, 1987; Rousseeuw and Hubert, 1997) and FAST-LTS for LTS (Rousseeuw and Van Driessen, 2006). LQD may be computed with LQS algorithms with a quadratic blow-up of computation time (Croux *et al.*, 1994). Aside from the best known heuristics, Hawkins and Olive (1999) propose a so called feasible solution algorithm (FSA) for LQS/LMS and LTS. The FSA is based on sampling data subsets of size h_p and iterative swapping of data points in the sample with data points outside the sample.

We propose to use *evolutionary computation* to compute the named robust estimators. As a matter of fact, all algorithms proposed in the remainder of this thesis are evolutionary computation algorithms.

6.1 Evolutionary Computation

The concept of evolutionary computation is well known in computer science and gaining importance in statistics as well. Examples for the application of evolutionary computation in computational statistics include evolutionary clustering (Hruschka *et al.*, 2006), computation of robust estimators (Meyer, 2003), time series modeling (Baragona *et al.*, 2004), and many more.

The idea of evolutionary computation is to mimic the Darwin–Wallace principle of natural selection in order to obtain an efficient search heuristic. Evolution's main principle is that *populations* of *individuals* evolve through variational inheritance where a concept of *fitness* reflects the ability to survive. Transfered to optimization, the population of individuals is a collection of candidate solutions, where the fitness reflects the goodness of the candidate solution, e. g. the objective value.

Individuals may be characterized by *genotypes* (the genetic makeup) or *phenotypes* (observed qualities). In algorithmic terms, these are the computer representation of the candidate solution and its (mathematical) interpretation if not apparent. Essential modules of evolutionary algorithms are therefore the fitness function that maps individuals to fitness values, the genotype search space and an optional mapping between genotype and phenotype.

Further, selection schemes determining which individuals are selected for variation, and concepts to modify individuals are needed (typically called *mutation* or *recombination/crossover* depending on whether one or more individuals are involved).

By nature, an evolutionary process is infinite. To obtain an algorithm that terminates, we additionally need a *stopping criterion.*

The basic evolutionary process used by evolutionary algorithms is described in Algorithm 6.1.

Algorithm 6.1: Basic Evolutionary Algorithm

```
1  Create an initial random population
2  Perform the following steps on the current generation of individuals
3  begin
4      Select individuals in the population based on a selection scheme
5      Adapt the selected individuals.
6      Evaluate the fitness value of the adapted individuals
7      Select individuals for the next generation according to a selection scheme
8  end
9  if stopping criterion is not fulfilled then set next generation as current and go to 3
10 return the final population
```

Evolutionary computation is a very general and adaptive framework and ideas used in evolutionary algorithms can also be found in existing algorithms for robust regression. For example, the point interchange Hawkins (1993) uses in his feasible set algorithm to compute LMS can be seen as a mutation operator.

6.2 Outline of the Algorithm

As a first step, we have to appoint the genotype of the individuals we work on. In order to have a limited number of candidate solutions, we restrict ourselves to candidate solutions uniquely determined by a data subsample of fixed size. In the most common algorithms PROGRESS and FAST-LTS the subsamples are for sound reasons of size $p+1$ (the value $p+1$ fits the notation used in Definition 3.6; in the notation used by Rousseeuw and Leroy, 1987, the value would be p). These reasons include the fact, that $p+1$ linear independent points uniquely define a hyperplane. Additionally, smaller subsamples decrease the possibility of having outliers in the subsample. Although, strictly speaking, we are computing a different estimator when using subsamples of size $p+1$, it remains being a high breakdown estimator (Rousseeuw and Basset Jr., 1991).

We will adopt this in defining for explanatory data $Z_e = \{x_1, \dots, x_n\}$ with $x_i \in \mathbb{R}^p$ for $1 \le i \le n$ the genotype of our individuals as

$$G = \left\{ (g_1, \dots, g_n) \in \{0,1\}^n : \sum_{i=1}^{n} g_i = p+1 \right\} .$$

A genotype in G is mapped to its phenotype by $m : G \to \{U \subseteq \mathbb{R}^p : |U| = p+1\}$ with

$$m\left((g_1, \ldots, g_n)\right) = \{x_i \in Z_e; g_i = 1\} \ .$$

Thus, we obtain $\binom{n}{p+1}$ different possible individuals. The determination of the fitness or goodness of these individuals comprises two steps leading to Algorithm 6.2:

1. Compute a unique candidate solution hyperplane H from the given individual (defining an estimate $\hat{\beta}_0, ..., \hat{\beta}_p$ of the regression parameters $\beta_0, ..., \beta_p$).
2. Compute—depending on the estimator chosen—one of the objective functions

$$\text{LQS: } \{r_1(\hat{\beta}_0, ..., \hat{\beta}_p)^2, \ldots, r_n(\hat{\beta}_0, ..., \hat{\beta}_p)^2\}_{(h_p)}$$

$$\text{LTS: } \sum_{i=1}^{h_p} \{r_1(\hat{\beta}_0, ..., \hat{\beta}_p)^2, \ldots, r_n(\hat{\beta}_0, ..., \hat{\beta}_p)^2\}_{(i)}$$

$$\text{LQD: } \{|r_i(\hat{\beta}_0, ..., \hat{\beta}_p) - r_j(\hat{\beta}_0, ..., \hat{\beta}_p)|; i < j\}_{\binom{h_p}{2}}$$

Algorithm 6.2: Evolutionary Algorithm for Robust Regression

```
Input: Data set, objective function of desired estimator
Output: Estimated parameters $\hat{\beta}_0, ..., \hat{\beta}_p$
1  Select $(g_1, \ldots, g_n) \in G$ uniformly at random, constituting the initial population
2  Compute a unique hyperplane $H$ defined by $\hat{\beta}_0, ..., \hat{\beta}_p$ from $(g_1, \ldots, g_n)$
3  Determine the objective value $f$ for the chosen estimator
4  Conduct one of the following adaptions chosen uniformly at random:
5      Point mutation: Randomly select $g_i, g_j \in (g_1, \ldots, g_n)$ with $g_i \neq g_j$ and
       exchange their values to obtain $(g'_1, \ldots, g'_n)$.
6      Hyperplane mutation: Let $r_i$ be the residuals defined by $\hat{\beta}_0, ..., \hat{\beta}_p$.
       Randomly select $k > p + 1$ and compute a set $I$ of $p + 1$ indices such that

           $\{r_1^2, \ldots, r_n^2\}_{(k-p)} \leq \min\{r_i^2; i \in I\} \leq \max\{r_i^2; i \in I\} \leq \{r_1^2, \ldots, r_n^2\}_{(k)}$ .

       Set $g'_i \in (g'_1, \ldots, g'_n)$ to 1 if and only if $i \in I$.
7  Compute a unique hyperplane $H'$ defined by $\beta'_0, \ldots, \beta'_p$ from $(g'_1, \ldots, g'_n)$
8  Determine the objective value $f'$ for the chosen estimator
9  if $f' \leq f$ then set $(g_1, \ldots, g_n) = (g'_1, \ldots, g'_n)$, $H = H'$, $\hat{\beta}_i = \beta'_i$ and $f = f'$
10 if none of {elapsed time,conducted adaptions,consecutive adaptions without better
   fitness} exceeds its predetermined maximum then go to 4
11 return $\hat{\beta}_0, ..., \hat{\beta}_p$
```

It is helpful to also use a resampling adaption that selects a completely new individual $(g'_1, \ldots, g'_n) \in G$ uniformly at random. We included it equiprobable, but the algorithm is typically faster when not using it as often as the other adaptions.

We omitted LMS and LTA in the description because of their similarity to LQS and LTS, respectively. The main difference of our algorithm to PROGRESS and FAST-LTS is that we have a continuous process changing subsamples instead of drawing a fixed number of subsamples. Thus, we are able to use the information of good candidate solutions in a better way. Only using the point mutation would lead to staying in local optima. The hyperplane mutation redeems this disadvantage in potentially moving far away from local optima, but still using information contained in the solution.

The question how to compute a unique hyperplane from a subset of size $p+1$ remains. As a first step, we compute the hyperplane H through the subset of data points. As we again assume general position, this hyperplane is unique. The second step depends on the estimator and is described in more detail in the following. A third optional step is to adjust the intercept of the unique hyperplane by doing an LTS/LQS/LQD univariate estimate on the residuals defining the objective value (Rousseeuw and Hubert, 1997). We include this step in our algorithm.

6.2.1 Computing a Unique Hyperplane for LTS

Rousseeuw and Van Driessen (2006) show that the following procedure guarantees an improvement in the objective value of the LTS estimate H:

1. Determine the h_p points with the smallest squared residuals with regard to H.
2. Compute the least squares fit $\hat{\beta}_{\text{LS}}$ on these h_p points which defines the unique hyperplane.

6.2.2 Computing a Unique Hyperplane for LQS

The situation for LQS is more complex. The following procedure also guarantees an improvement in objective value (Stromberg, 1993):

1. Determine the h_p points with the lowest squared residuals with regard to H.
2. Compute a a minimax fit estimation

$$\hat{\beta}_{\text{minimax}} = \min_{\beta_0,\ldots,\beta_p} \max\{r_1\left(\beta_0,\ldots,\beta_p\right)^2,\ldots,r_{h_p}\left(\beta_0,\ldots,\beta_p\right)^2\}$$

on these h_p points where $r_1,\ldots,r_{h_p}$ denote the residuals for the chosen h_p points.

One possibility to compute such a fit is based on computing least squares fits on all $\binom{h_p}{p+2}$ possible subsamples of size $p+2$ (Stromberg, 1993). For large h_p or p this is clearly prohibitive. A feasible possibility to compute a unique hyperplane is to do a least squares fit on a subset of the data uniquely defined by H. We again propose to use least squares on the h_p points with the smallest squared residuals with regard to H.

This often leads to an improved objective value, but in contrast to LTS regression the improvement is not guaranteed. Thus we only choose the new hyperplane instead of H if it leads to a better objective value. A third possibility to improve the objective value is to do weighted least squares on the data points with the lowest squared residuals with regard to H and give higher weight to the data points with higher residuals.

6.2.3 Computing a Unique Hyperplane for LQD

The similarity of LQD to LQS allows us to use the following procedure to obtain a guaranteed better objective value:

1. Determine the set of points $Z = \{(x_i - x_j, y_i - y_j)\}$ such that the corresponding absolute residual differences $|r_i - r_j|$ with regard to H are among the h_p smallest.
2. Compute a minimax fit estimation on Z.

It is easy to see that this procedure guarantees a better objective value, because Croux and Rousseeuw (1992) showed that it is possible to compute the LQD by computing an LQS on the data set of differences $\{(x_i - x_j, y_i - y_j)\,; 1 \leq i < j \leq n\}$. This is however, due to a typically larger h_p than in LQS estimation, even more prohibitive than in case of the LQS for large h_p or p. Again, a good alternative is to use least squares fits on subsets of the data (this time on the according data sets of differences).

6.3 Comparison

We compare our algorithm with results from PROGRESS for LQS and results from FAST-LTS for LTS. We omit the LQD, because—as we have seen—a transformation to LQS exists and to our knowledge, no implementation of LQD algorithms for high dimensional data exists. PROGRESS is implemented in the function `lqs` of the R (R Development Core Team, 2008) package *MASS* (Venables and Ripley, 2002) and FAST-LTS is implemented in the function `ltsReg` of the R package *robustbase* (Todorov *et al.*, 2007). Our own algorithms are implemented in the function `robreg.evol` of the R package *RFreak* (Nunkesser, 2008).

To compare the algorithms, we simulate data with $n = 500$ data points for $p = 1, \ldots, 30$ from two different models. In the first model, the independent regressors are normally distributed. In the second model, they stem from a uniformly distributed random design on the interval $(0, 1)$. The first model is given by

$$Y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i \;\; i = 1, \ldots, n$$

where β_0 is an intercept term and $\varepsilon_i \sim \mathcal{N}(0, 1)$ are random errors. The parameter β_0 is set to 0, while $\beta_1, \ldots \beta_p$ equal 2. We add 40% outliers to the data in choosing two disjunct subsamples of size 20%. We add 3 to the explanatory data in the first

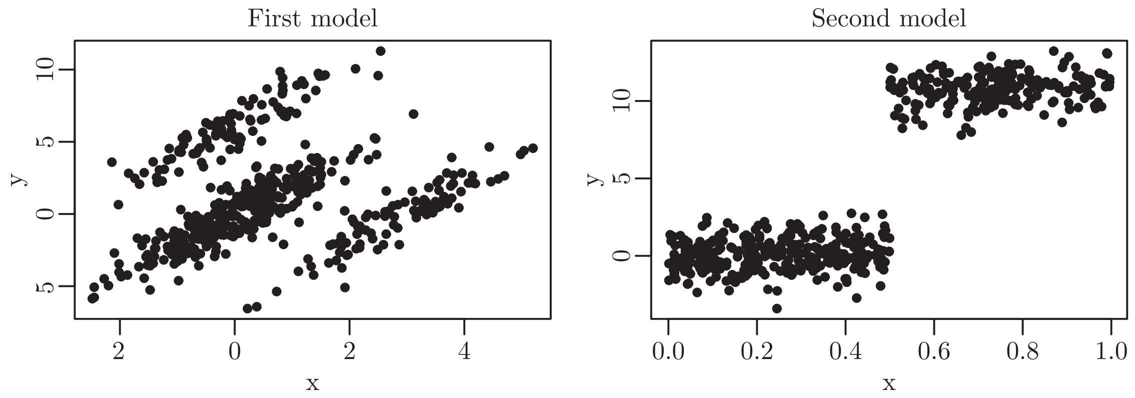


Figure 6.1: *Example of simulated data for $p = 1$.*

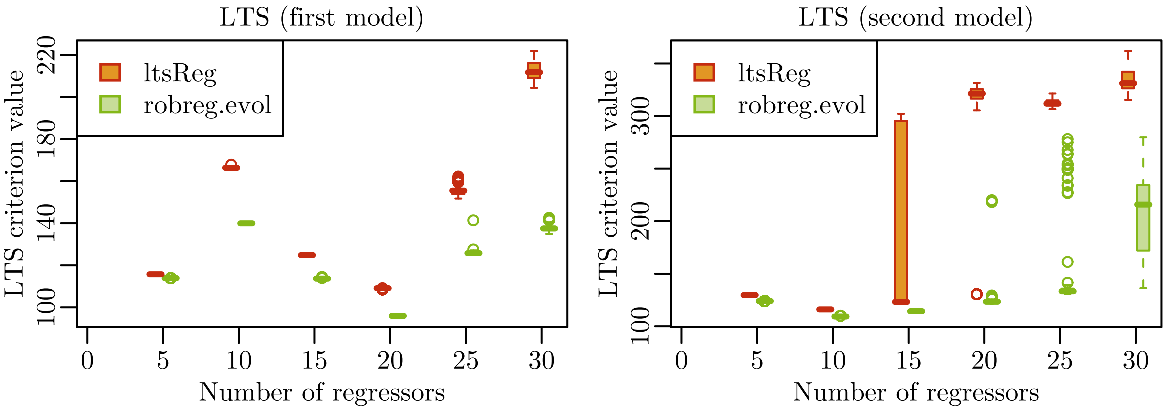


Figure 6.2: *Comparison of `robreg.evol` with `ltsReg`.*

subsample and 6 to the response in the second subsample, generating additive outliers in the explanatory and the response variables, respectively.

The second model contains a structural change. The parameter β_1 is 1, while $\beta_2, \ldots \beta_p$ are 0. Thus, the corresponding regressors add only noise to the problem. The structural change is in the intercept, which is

$$\beta_0 = \begin{cases} 0, & \text{if } x_{i1} \leq \{x_{j1}; 1 \leq j \leq 500\}_{300:500} \\ 10, & \text{if } x_{i1} > \{x_{j1}; 1 \leq j \leq 500\}_{300:500} \end{cases} .$$

A good robust estimate of β should give β_0 close to 0 and the slopes as above. Figure 6.1 shows an example of two simulated data sets based on these models with $p = 1$.

To compare the algorithms honestly, we measure the runtime `lqs` and `ltsReg`, respectively, need for the computation and give our algorithm exactly the same amount of time. Figure 6.2 shows the results for LTS and Figure 6.3 shows the results for LMS.

In nearly all conducted runs, our algorithm achieves better results than the established algorithms `lqs` and `ltsReg`. Another observation is that the structural change

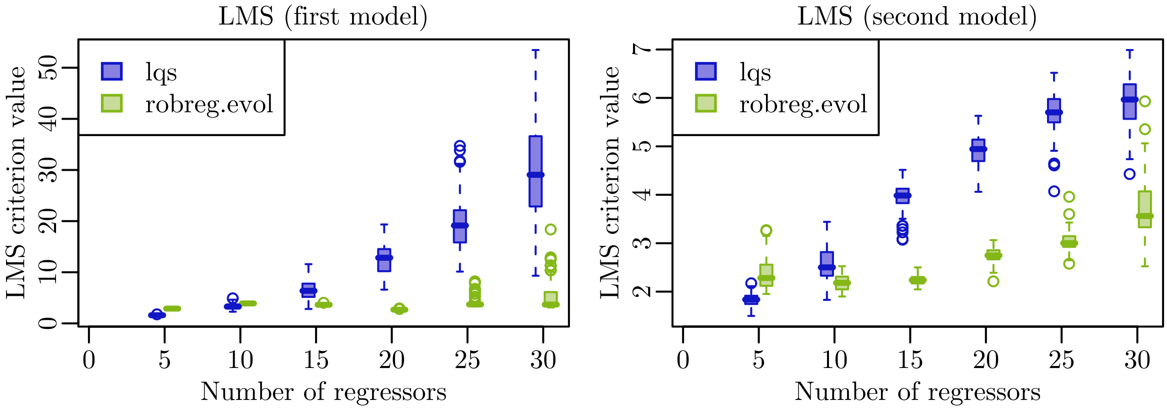


Figure 6.3: *Comparison of* `robreg.evol` *with* `lqs`*.*

in the second data set affects the LTS estimation of `ltsReg` heavily for $p \geq 15$ while the effect on `robreg.evol` is far less.

Further experiments show that the gap between our algorithm and the established ones widens more, if we allow the algorithms to spend even more time. All in all, `robreg.evol` seems to be better suited for the considered data situations.

Part III

Genetic Association Studies

7 Preliminaries

The aim of *genetic association studies* is to identify genetic factors that may contribute to a medical condition. We consider *case-control studies*, in which patients who have the medical condition (the *cases*) are compared with subjects who do not have the condition but are otherwise similar (the *controls*). The genetic factor we consider is *genetic variation*. The most common and also often considered most important genetic variation is the *single nucleotide polymorphism* (SNP, pronounced snip).

7.1 Single Nucleotide Polymorphisms

Schwender *et al.* (2006) give an overview of the necessary basics of SNP studies. We take up the concepts presented by Schwender *et al.* (2006) needed here.

The nucleus of almost every human cell comprises the complete human *genome*, which is a blueprint for all cellular structures and activities in the human body. The genome consists of *chromosome* pairs (one chromosome from the mother, one from the father). Each chromosome is a huge chain of two intertwined strands of *deoxyribonucleic acid* (DNA), structured as a *double helix*. The DNA strands are sequences of *nucleotides*, where each nucleotide consists of a phosphate group, a sugar, and a nitrogenous base. The four different nitrogenous bases *adenine* (A), *thymine* (T), *cytosine* (C), and *guanine* (G) build four different nucleotides. The nitrogenous bases on the two DNA strands are paired via hydrogen bounds. A is paired with T and C is paired with G. Thus, one sequence of letters in $\{A, T, C, G\}$ suffices to describe the two strands of DNA. Figure 7.1 shows an example of a DNA sequence according to Drew *et al.* (1981) and created with UCSF Chimera (Pettersen *et al.*, 2004). Additionally, Figure 7.1 shows a representation of the same DNA sequence in the layout used by Alberts *et al.* (2002).

Less than 1% of the human DNA differs between individuals. In absolute terms, these are still millions of base pair positions at which different bases can occur. Each of the forms a DNA segment can take is called an *allele*. Alleles occurring in more than 1% of the population are called *polymorphisms*. Looking at a fixed base pair position or *locus*, a polymorphism at this specified locus is called a *single nucleotide polymorphism*, because it is a polymorphism at a single nucleotide position. In an analysis concerning the genotype of individuals, we consider the chromosome pairs of an individual. A SNP has typically two alleles, the *major allele* occurring in the majority of the population

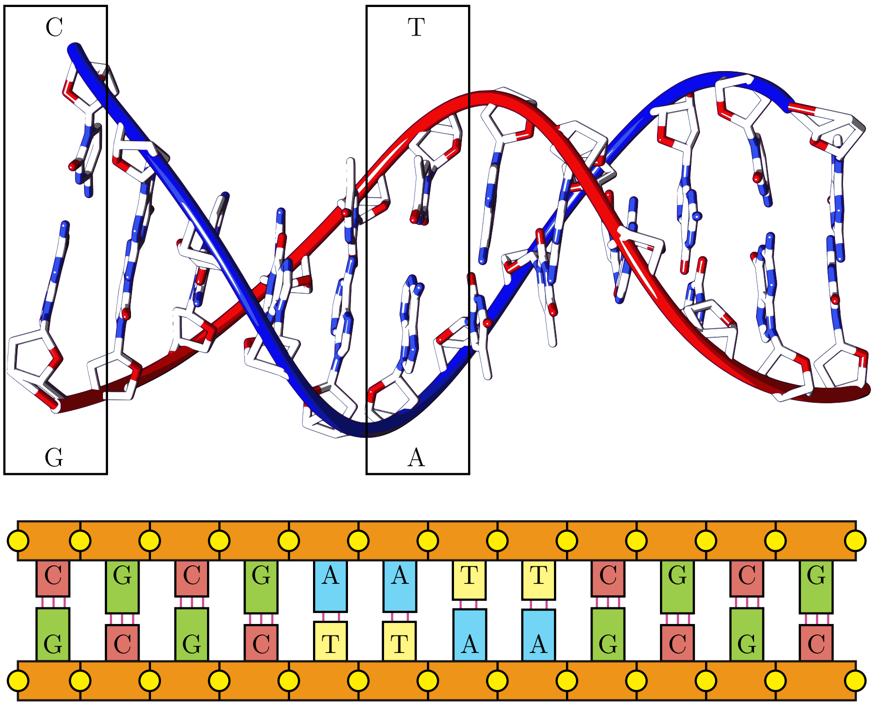


FIGURE 7.1: *DNA sequence described by CGCGAATTCGCG.*

and the *minor allele* (often denoted by A and a). As we consider chromosome pairs, a SNP in our analysis can take three forms: AA (*homozygous reference*), Aa/aA (*heterozygous variant*), aa (*homozygous variant*). Figure 7.2 shows an illustration of the three different SNP forms.

Note, that there is a difference between the SNP forms Aa and aA (one has the minor allele in the father chromosome, the other form has the minor allele in the mother chromosome). However, in the measuring methods underlying the data we work on, Aa and aA cannot be distinguished. Thus, we are only able to consider three different SNP forms. Our approach for genetic association studies on SNP data is to use evolutionary computation. Many of the classification problems underlying case-control studies are NP-hard (depending on the prediction model used) and thus motivate the use of heuristics. In addition, the sheer amount of data in genetic studies impedes exact algorithms. The algorithm we propose in the following uses *genetic programming* (GP).

7.2 Genetic Programming

Genetic programming is an evolutionary computation (see Section 6.1) concept by Koza (1992). When we adopt a unified view of evolutionary computation like the view

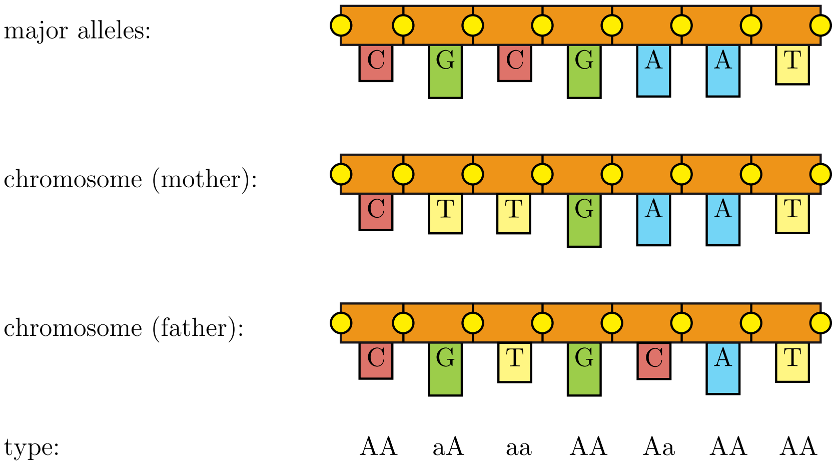


FIGURE 7.2: *The three different SNP forms AA, Aa/aA, and aa.*

of De Jong (2006), genetic programming may also be described by Algorithm 6.1. The main difference to evolutionary algorithms as used in Section 6.2 is the genotype search space. The idea of genetic programming is to apply the Darwin–Wallace principle of natural selection to obtain computer programs by evolution. Computer programs in genetic programming are described by *symbolic expressions*. The set of possible symbolic expressions constitutes the genotype search space. It is typically described by all symbolic expressions that can be recursively composed of functions from a function set F and terminals from a terminal set T. Consider as an example $F = \{\wedge, \vee, \neg\}$ and $T = \{x_1, x_2\}$. The genotype search space can then be interpreted as all Boolean expressions that may be formed with the variables x_1 and x_2 and the operations $\wedge$, $\vee$, and $\neg$.

Some part of the comparison of heuristics may be done by *hypothesis testing*.

7.3 Hypothesis Testing

In principle, it is possible to analyze the stochastic process underlying an evolutionary computation algorithm theoretically. Unfortunately, the class of problems that may be analyzed with state-of-the-art techniques is limited (see e. g. Oliveto *et al.*, 2007 for a survey on theoretical results for combinatorial optimization algorithms). Where a theoretical analysis is not yet possible, empirical results may help. In order to obtain statistically valid results, *hypothesis testing* is proposed e. g. by Bartz-Beielstein (2006) and Zitzler *et al.* (2008).

Hypothesis testing (see e. g. Mood *et al.*, 1974) is a *statistical inference* method. Statistical inference is concerned with inferring information on some collection of elements from samples. In our case, we want to infer information on the performance of variants of a particular algorithm from observations made in algorithm runs. In hypothesis testing, the information we would like to infer is, whether or not a *hypothesis* about the algorithm variants should be accepted. The statistical inference we would like to make here (if it is true) is that one algorithm variant outperforms other variants. When testing hypotheses, we instead assume the *null hypothesis* H_0 that the algorithm variants produce samples drawn from the same distribution or from distributions with the same mean value, i. e. that no variant outperforms the other significantly. The probability of obtaining a result at least as extreme as the one observed, assuming H_0 is true, is called *p-value* and computed using an inferential statistical test. A p-value lower than a chosen significance level α signifies, that the null hypothesis can be rejected in favor of an *alternative hypothesis* H_1, at a significance level of α. Here, we mostly define the alternative hypothesis one-sided, i. e. the alternative hypothesis states that the result of an algorithm variant produces results coming from a "better" distribution.

For results from stochastic optimizers, we typically use *nonparametric tests* (see e. g. Conover, 1999), i. e. tests that only make weak assumption about the distributions producing the samples. This is a safe choice, because stronger distribution assumptions cannot be theoretically justified for stochastic optimizer outputs in general. The appropriate test to compare the results of two algorithm variants in our scenario is the *Wilcoxon signed rank test.*

The *Wilcoxon signed rank test* works on the absolute differences of pairs of algorithm outputs measurable as real numbers, i. e. results two different algorithms variants output for the same data set. Our null hypothesis is that these differences are symmetric about 0, our alternative hypothesis is that there is a location shift in favor of the algorithm variant assumed to be better. In the test, the cases with zero difference are eliminated from consideration and the remaining absolute differences are then ranked from lowest to highest, with tied ranks included where appropriate. These rank values get a negative sign, if and only if the original difference was negative. The positive and negative ranks are then summed. The higher the absolute sum, the higher the evidence, that the null hypothesis should be rejected. As we consider one-sided tests, we have to specify if a large negative or a large positive sum shows that an algorithm variant is better. Table 7.1 shows an example for the Wilcoxon signed rank test.

To compare more than two algorithm variants, we use the *Friedman rank sum test.* The Friedman rank sum test works on the ranks of blocks of algorithm variant outputs for the same data set. These ranks are summed for each algorithm variant. Our null hypothesis is, that the differences between these rank sums are symmetric about 0. The higher the difference in rank sums, the higher the evidence, that the null hypothesis should be rejected. Table 7.2 shows an example for the Friedman rank sum test.

When conducting these tests, we will often only state the alternative hypothesis in an informal way, implying that one of the algorithm variant outcomes contains better results and report the p-value.

Output A	Output B	$A - B$	$\lvert A - B\rvert$	rank	signed rank
78	78	0	0	-	-
24	24	0	0	-	-
64	62	2	2	1	1
45	48	-3	3	2	-2
64	68	-4	4	3.5	-3.5
52	56	-4	4	3.5	-3.5
30	25	5	5	5	5
50	44	6	6	6	6
64	56	8	8	7	7
50	40	10	10	8.5	8.5
78	68	10	10	8.5	8.5
22	36	-14	14	10	-10
84	68	16	16	11	11
40	20	20	20	12	12
90	58	32	32	13	13
72	32	40	40	14	14
					$\sum \cdot = 67$

TABLE 7.1: *Example for the Wilcoxon signed rank test*

Output A	Output B	Output C	block rank of A	block rank of B	block rank of C
9.0	7.0	6.0	3	2	1
9.5	6.5	8.0	3	1	2
5.0	7.0	4.0	2	3	1
7.5	7.5	6.0	2.5	2.5	1
9.5	5.0	7.0	3	1	2
7.5	8.0	6.5	2	3	1
8.0	6.0	6.0	3	1.5	1.5
7.0	6.5	4.0	3	2	1
8.5	7.0	6.5	3	2	1
6.0	7.0	3.0	2	3	1
			$\sum \cdot = 26.5$	$\sum \cdot = 21.0$	$\sum \cdot = 12.5$

TABLE 7.2: *Example for the Friedman rank sum test*

8 Genetic Programming for Association Studies

One of the major goals of association studies is to identify SNPs and SNP interactions that lead to a higher disease risk. Since individual SNPs typically only have a slight to moderate effect—in particular when considering more complex diseases—the focus is on the detection of interactions (Garte, 2001; Culverhouse *et al.*, 2002), i. e. the effect a combination of SNPs has. The search for such interacting SNPs is additionally impeded by the facts that the interactions are usually of a high order and that they are explanatory for relatively small subgroups of the patients (Pharoah *et al.*, 2004).

Various methods have been suggested for and applied to genotype data to identify SNP interactions. These procedures span from exhaustive searches based on e. g. multiple testing approaches (Marchini *et al.*, 2005; Goodman *et al.*, 2006; Ritchie *et al.*, 2001) to methods based on discrimination procedures (e. g. Lunetta *et al.*, 2004). For overviews on such approaches, see Heidema *et al.* (2006) and Hoh and Ott (2003).

One of the most promising methods is logic regression (Ruczinski *et al.*, 2003), an adaptive classification and regression procedure that tries to identify Boolean combinations of binary variables associated with the response (e. g. the case-control status). In several comparisons with other regression or discrimination approaches, logic regression has shown a good performance in its application to SNP data (Kooperberg *et al.*, 2001; Witte and Fijal, 2001; Ruczinski *et al.*, 2004; Schwender, 2007). Moreover, logic regression can be employed for detecting interactions and quantifying their importance (Kooperberg and Ruczinski, 2005; Schwender and Ickstadt, 2008).

8.1 Boolean Concept Learning

Similar to logic regression, the procedure we propose also uses logic expressions as predictors. Our intention is to classify with logic expressions.

From a computer scientist's point of view, classification with logic expressions is identical to Boolean concept learning (see e. g. Valiant, 1984). We intend to learn a concept that produces output (examples) in $B = \{0, 1\}$ from inputs from B^n. More precisely, in case-control genetic association studies on SNP data, we intend to understand a procedure that produces output in $\{\text{case}, \text{control}\}$ (encoded by $B = \{0, 1\}$) from inputs in $\{AA, aA/Aa, aa\}^n$ (encoded by $P^n = \{0, 1, 2\}^n$). In order to generalize our procedure for other data than SNP data, we set $P = \{0, \ldots, p - 1\}$. Our prediction models for this concept are so called *multiple-valued input, binary-valued output functions* f, which are a mapping

$$f : P^n \rightarrow B \ .$$

In order to use Boolean concept learning and the advantages of a Boolean algebra, we do not search for such functions directly with multi-valued variables but use a mapping to Boolean variables. To our knowledge, there are three approaches to map multi-valued input variables to Boolean variables, leading to a Boolean algebra and an easier interpretation of the functions. The approaches differ in generality and the number of generated literals. The most general approach was suggested by Rudell and Sangiovanni-Vincentelli (1987) and defines

$$x^S := \begin{cases} 1, & \text{if } x \in S \\ 0, & \text{otherwise} \end{cases} \tag{8.1}$$

and its complement

$$\bar{x}^S := \begin{cases} 0, & \text{if } x \in S \\ 1, & \text{otherwise} \end{cases} , \tag{8.2}$$

for a set $S \subseteq P$ of input values leading to 2^p distinct *positive literals* and 2^p distinct *negative literals.* Su and Sarris (1972) define

$$x^{a,b} := \begin{cases} 1, & \text{if } a \leq x \leq b \\ 0, & \text{otherwise} \end{cases} \tag{8.3}$$

and its complement

$$\bar{x}^{a,b} := \begin{cases} 0, & \text{if } a \leq x \leq b \\ 1, & \text{otherwise} \end{cases} , \tag{8.4}$$

for $a, b \in P$ leading to $p(p-1)/2$ distinct positive literals and $p(p-1)/2$ distinct negative literals. Finally, Dussault *et al.* (1976) restrict (8.3) and (8.4) to literals $x^a = x^{a,b}$ where $a = b$ and their complements, leading to p distinct positive literals and p distinct negative literals. We will use this approach, because it leads to the smallest genotype search space.

To additionally downsize the search space, only polynomials of the chosen literals are allowed (also called functions in *disjunctive normal form* or DNF). The following definition is based on Wegener (1987).

Definition 8.1.

1. A *monomial* m is a product (conjunction) of some literals. The *length* of m is equal to the number of literals of m.
2. A *polynomial* p is a sum (disjunction) of monomials. The *length* of p denoted by $\text{length}(p)$ is equal to the sum of the lengths of all monomials m which are summed up by p.
3. A polynomial p *computes* $f : D \subset P^n \rightarrow \{0,1\}$ if $p(x) = f(x)$ for $x \in D$. It is a *minimal polynomial* for f, if p computes f and no polynomial computing f has smaller length than p.

An additional benefit of restricting the search to polynomials is that polynomials are easier to interpret and may even reveal biologically meaningful information in case-control association studies. Depending on the problem at hand, it may also be convenient to allow only a subset of the literals, e. g. if there are biological reasons not to consider certain literals.

In Definition 8.1 we defined that a polynomial p computes a partial function f if p produces the same output in the domain of f. The problem of finding a minimal polynomial computing a partial function is the first we will consider in the presentation of our algorithm in the following section.

8.2 Genetic Programming Algorithm

The described problem of finding a minimal polynomial formally is the following:

Problem 8.1 (Minimum Polynomial)**.**

Given: Truth-table of a partial function $f : D \subset \{0,1\}^n \rightarrow \{0,1\}$.

Goal: Find a minimal polynomial computing f.

The original problem to develop models distinguishing between cases and controls is a learning problem. Thus, we will allow solutions to Problem 8.1 to contradict parts of the given truth-table. Otherwise, the prediction model would surely overfit. The reason to first consider the stated version of Problem 8.1 is twofold. First, this more rigid case allows us to prove that the problem of finding minimal polynomials is NP-hard (Lukas and Czort, 1999, show this by applying a result of Haussler, 1988). Second, Problem 8.1 is also an interesting problem by itself and we will consider it more closely in Section 8.5.

For association studies however, we have to allow contradictions to the given truth-table.

Definition 8.2. Let the truth-table of a partial function $f : D \subset P^n \to \{0, 1\}$ be given. Further, let p be a polynomial consisting only of literals of the form x^a and $\bar{x}^a$ with $a \in P$.

1. The *number of cases predicted correctly* by p is defined by

$$\mathrm{cases}(p) := |\{x \in D \mid p(x) = f(x) = 1\}| \ ,$$

the *number of specified cases* by

$$\mathrm{cases}(f) := |\{x \in D \mid f(x) = 1\}| \ .$$

2. The *number of controls predicted correctly* by p is defined by

$$\mathrm{controls}(p) := |\{x \in D \mid p(x) = f(x) = 0\}| \ ,$$

the *number of specified controls* by

$$\mathrm{controls}(f) := |\{x \in D \mid f(x) = 0\}| \ .$$

3. The *number of observations predicted correctly* by p is defined by

$$\mathrm{observations}(p) := \mathrm{cases}(p) + \mathrm{controls}(p) \ ,$$

the *number of specified observations* by

$$\mathrm{observations}(f) := |D| \ .$$

4. The *misclassification rate* (MCR) of p is defined by

$$\mathrm{mcr}(p) := 1 - \frac{\mathrm{observations}(p)}{\mathrm{observations}(f)} \ .$$

Allowing contradictions leads to a multicriterial problem.

Problem 8.2.

Given: Truth-table of a partial function $f : D \subset P^n \to \{0, 1\}$.

Goal: Find a polynomial computing f that consists of literals of the form x^a and $\bar{x}^a$ with $a \in P$, has short length and a low MCR.

In the context of multi-objective optimization, an individual *dominates* another individual, if at least one objective has a superior value and none an inferior. An individual is *pareto optimal*, if it is not dominated by another individual. Thus, we seek to find pareto optimal individuals. The GP approach proposed for this is described by Algorithm 8.1 (GPAS). Figure 8.1 shows the adaptions used in GPAS.

Algorithm 8.1: Genetic Programming for Association Studies (GPAS)

```
   Input: Data set, $i \in \{1, 2\}$ determining which fitness function to use
   Output: Set of polynomials
 1 Create an initial random population composed of two individuals each of which
   consists of one randomly selected literal
 2 Perform the following steps on the current generation
 3 begin
 4     Select all individuals in the population for reproduction, and draw seven of
       the individuals uniformly at random with replacement
 5     Conduct each of the following adaptions to one (mutations) or two (crossover)
       of the seven randomly selected individuals
 6         Crossover: Combine one of the two chosen individuals with one randomly
           chosen monomial from the other individual
 7         Literal insertion: Insert a new literal
 8         Literal deletion: Delete a literal
 9         Literal replacement: Replace a literal by a new literal
10         Monomial insertion: Insert a new literal as a new monomial
11         Monomial deletion: Delete a monomial
12     Evaluate the fitness of the adapted and reproduced individuals with fitness
       function $f_i$ that maps an individual $p$ to a triple:

           $f_1(p) := (\mathrm{cases}(p), \mathrm{controls}(p), \mathrm{length}(p))$
           $f_2(p) := (\mathrm{mcr}(p), \mathrm{controls}(p), \mathrm{length}(p))$

13     Select all adapted and reproduced individuals that are not dominated for the
       next generation.
14 end
15 if none of {elapsed time, conducted adaptions, consecutive adaptions without better
   fitness} exceeds its predetermined maximum then go to 3
16 return the final population
```

Using polynomials as the genotype has the additional benefit that the adaptions in one generation are possible in amortized constant time, when we store the polynomials as trees and the children in this tree in dynamic arrays. The major computational part of the fitness evaluation is to determine the number of cases and controls classified correctly by the logic expression. For fast fitness computation, we additionally store a bitset in each node of the tree representing the polynomial. The bitset consists of as many bits as there are observations in the data set, and the ith bit is true if the polynomial is true for the SNP values of the ith observation.

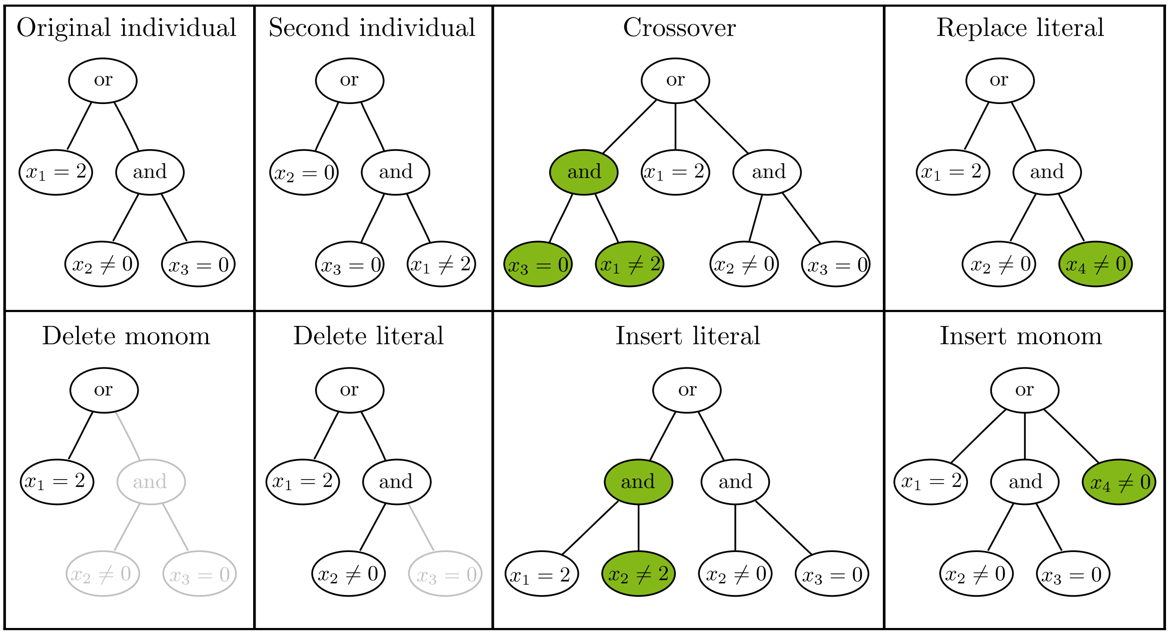


FIGURE 8.1: *Example for the crossover and the different mutations used in GPAS (with the notation $x = a$ and $x \neq a$ instead of x^a and $\bar{x}^a$).*

The bitsets of the literals are initially computed for all possible literals. If a monomial is changed during a mutation operation the bitset of the monomial is recomputed using the bitsets of its literals. The computation is sped up, since the bitsets of the other monomials remain unchanged and can be reused to compute the bitset of the whole polynomial. In addition, bitsets are compact and allow fast logic operations. For example, one logic operation of the bitset of the whole polynomial with the bitset describing the case-control status suffices to compute the number of cases and controls predicted correctly.

Algorithm 8.1 contains two different fitness functions. To elucidate the difference between f_1 and f_2 consider, e. g., two individuals a and b with the same length for a data set with the same number of cases and controls. Suppose, that a predicts 50% of the cases and 90% of the controls correctly and b predicts 89% of the cases and 50% of the controls correctly. When we use f_2, a dominates b, while it does not dominate b when we use f_1. Thus, f_1 treats cases and controls equally, while f_2 prefers models that contradict with few controls. If our task is to discriminate between cases and controls, f_1 is the appropriate fitness function. If we want to search SNP interactions that explain subsets of the cases with few contradictions in the controls like Pharoah *et al.* 2004 suggest, f_2 is better suited. To aid the detection of such interactions, we additionally devise a visualization of the final population.

8.2.1 Visualization for Interaction Search

The visualization we present displays the interactions in the final population in a tree. Thus, we are able to see many different interactions at a glance. To obtain this visualization (for an example of a resulting tree see Figure 8.4) we proceed as described in Algorithm 8.2.

Algorithm 8.2: Construct Interaction Tree

```
Input: Population $P$
Output: Interaction tree
Compute the set $M$ of all monomials occurring in $P$
Create a tree $T$ consisting of its root node $t$.
SearchMostCommonLiteral ($M, t, T$)
return $T$
```

Procedure `SearchMostCommonLiteral(`*set of monomials M, tree node t, tree T*`)`

```
Search for the most common literal $\ell$ in $M$
Determine the set $M_\ell$ of monomials in $M$ containing $\ell$
Set $M_{\overline{\ell}} := M \setminus M_\ell$
Delete $\ell$ from all monomials in $M_\ell$
Construct a node $t_\ell$ containing information about $\ell$
Set $t_\ell$ to be a new child of $t$ in $T$
if $M_\ell \neq \emptyset$ then SearchMostCommonLiteral ($M_\ell, t_\ell, T$)
if $M_{\overline{\ell}} \neq \emptyset$ then SearchMostCommonLiteral ($M_{\overline{\ell}}, t, T$)
```

We additionally store information in the tree nodes on how often the resulting interactions and partial interactions occur, and on how many observations they explain.

In the remainder of Section 8.2, we investigate some decisions in the design of GPAS, for example the application of crossover.

8.2.2 The Role of Crossover

The usage of crossover is discussed controversially (see e. g. Banzhaf *et al.*, 1998). To analyze the merit of crossover for our algorithm, we conduct hypothesis testing on simulated data. To simulate data, we use the R package *scrime* (Schwender and Fritsch, 2008), which is intended to mimic a process from nature. We generate data sets consisting of m inputs in $\{0, 1, 2\}$ on $n \geq 10$ variables $X_1, \ldots, X_n$. The probabilities for a 0, 1, and 2 are 0.5625, 0.375, and 0.0625, respectively. This corresponds to a minor allele frequency of 0.25. The output y is then randomly drawn from a Bernoulli distribution with mean $\mathrm{Prob}\big(Y = 1\big)$, where

$$\mathrm{logit}\big(\mathrm{Prob}(Y = 1)\big) \;=\; -0.5 + 1.5\big(X_3^0 X_9^0 X_{10}^0\big) + 1.5\big(\bar{X}_6^0 X_7^0\big) \tag{8.5}$$

such that the probability for being a case is 0.924 if for an input both logic expressions are true, and 0.731 if one of them is true. This probability is still 0.378 if neither of the two is true.

We run Algorithm 8.1 with fitness function f_1 and a variant without crossover on 100 data sets built according to (8.5) until they create the 10000th individual, i. e. both algorithms create the same number of individuals. Afterwards, we test with a Wilcoxon signed rank sum test, if the best individuals in the last populations of the algorithm with the crossover operation (measured by MCR) are significantly better than without crossover, i. e. if the crossover operation leads faster to good results.

Thus, we conduct a one-sided test of the alternative hypothesis that the algorithm variant with crossover produces better individuals in terms of the MCR. In the following, we state such alternative hypotheses together with the computed p-value of the hypothesis test.

Hypothesis 8.1 (p-value $1.45 \cdot 10^{-8}$)**.** *GPAS with crossover obtains better results after creating 10000 individuals than GPAS without crossover on data built according to* (8.5).

To analyze if crossover is generally necessary to attain better solutions, we repeat the experiment on 100 new data sets until the 1000000th individual is created. This is long enough to reach stagnation in the parts of the final population typically containing the best individual.

Hypothesis 8.2 (p-value $8.34 \cdot 10^{-9}$)**.** *GPAS with crossover obtains better results after creating 1000000 individuals than GPAS without crossover on data built according to* (8.5).

The computed p-values are far below the typical choice of the significance level $\alpha = 0.05$. Therefore, we should reject the corresponding null hypotheses that the results with crossover are not better and instead assume that crossover is helpful.

8.2.3 The Role of Literal Replacement

Similar to the crossover operation, one might ask, if the mutation replacing a literal by another literal is necessary for the algorithm, because an insertion and deletion mutation could lead to the same individual. To answer this question, we conduct the same experiments as for the crossover operation again for this mutation on new data sets.

Hypothesis 8.3 (p-value 0.002)**.** *GPAS with literal replacement obtains better results after creating 10000 individuals than GPAS without literal replacement on data built according to* (8.5).

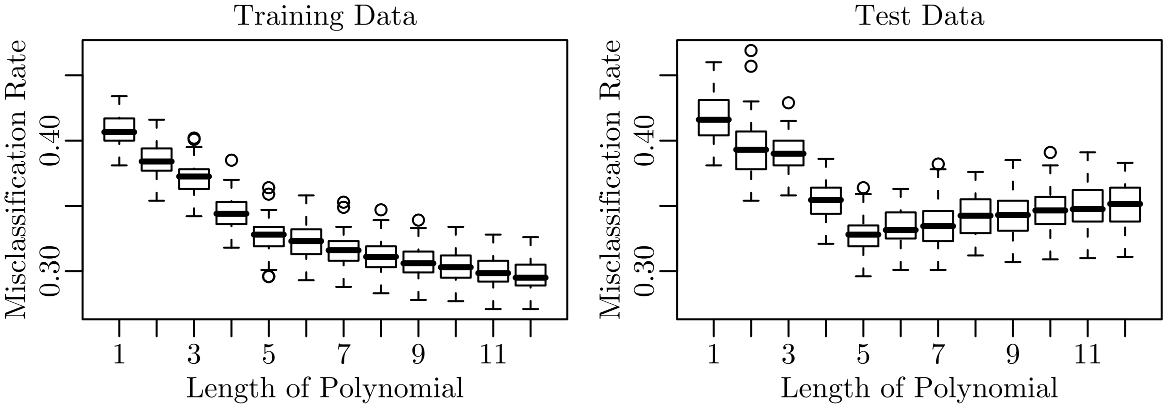


FIGURE 8.2: *MCRs for the best individuals of different sizes in 100 training and 100 test runs.*

Hypothesis 8.4 (p-value 0.02)**.** *GPAS with literal replacement obtains better results after creating 1000000 individuals than GPAS without literal replacement on data built according to* (8.5)*.*

The computed p-values are also below the typical choice of the significance level $\alpha = 0.05$. Therefore, we conclude that literal replacement is also helpful.

8.3 Automated Rules to Select the Best Individual

Problem 8.2 contains multiple objectives. Thus, solutions to Problem 8.2 are in most cases a set of individuals not dominating each other. It is possible to inspect them manually and to pick the most interesting ones by hand. Since the population may be very big, it is helpful to have an automated selection instead. We present automated rules intended to select the individual that best reflects the underlying concept we want to learn. To illustrate this, we look at the problem on data simulated according to (8.5).

The main problem in determining the best individual of the final population is illustrated in Figure 8.2. The figure shows the result of 100 training runs on data sets built according to (8.5). The best individual, i. e. the individual with the smallest MCR for each length between 1 and 12 is then tested on a test data set, which is also built according to (8.5). We see that—not surprisingly—the longest individuals perform best on the training data. We also see that the individuals with the true model size 5 perform best on the test data. Our task here is to find automated rules that conclude from a training run result, that the true model size is indeed 5. Such rules help to evade overfitting of the data, i. e. they avoid choosing a longer individual in the training run with worse generalization and prediction properties. A first apparent idea is to only consider points on the *convex hull* of the point set P consisting of the

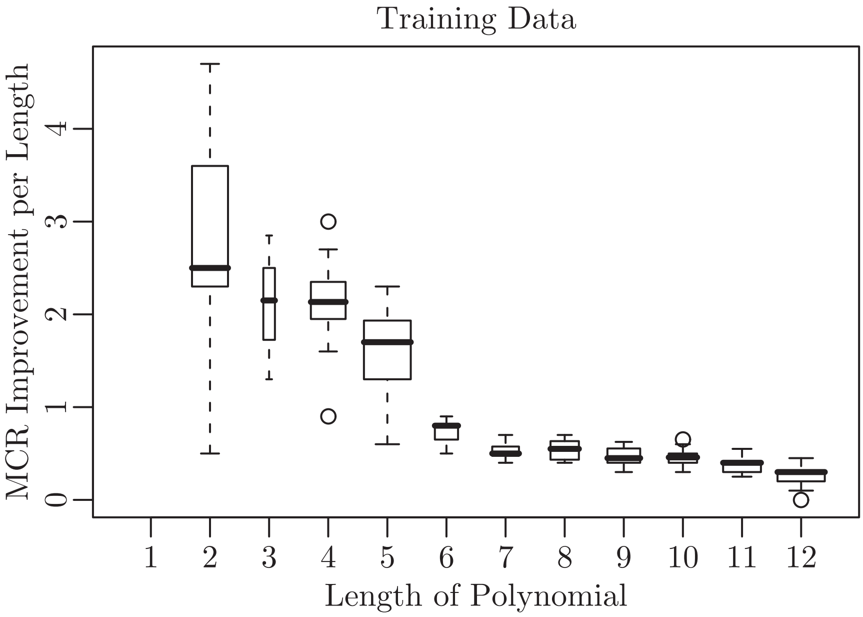


FIGURE 8.3: *MCRs improvement per length for individuals represented by points on the convex hull.*

points $(\mathrm{length}(p), \mathrm{mcr}(p)) \in \mathbb{N} \times \mathbb{R}$ for every polynomial p in the final population of a training run.

Definition 8.3. A subset S of the plane is called *convex*, if and only if for any pair of points $p, q \in S$ the line segment $\{\tau p + (1-\tau)q \mid 0 \le \tau \le 1\}$ is completely contained in S. The *convex hull* of a set S is the smallest convex set that contains S.

Computing the convex hull of n points in the plane needs time $O(n \log n)$ (see e. g. de Berg *et al.*, 2008 for details). When we only consider points on the convex hull, we exclude individuals with a gain in MCR that is relatively too small. For the remaining points we now consider the slope between any two points lying next to each other, i. e. the improvement in MCR per length when considering the longer individual. More precisely, for two points lying next to each other representing polynomials p_1 and p_2 with $\mathrm{mcr}(p_1) > \mathrm{mcr}(p_2)$ we consider

$$\frac{\mathrm{mcr}(p_1) - \mathrm{mcr}(p_2)}{\mathrm{length}(p_2) - \mathrm{length}(p_1)} \,. \tag{8.6}$$

The resulting values are depicted in Figure 8.3. The figure illustrates, that the choice of length 5 is a much more apparent choice after these considerations.

On this basis, we try a simple *threshold rule*. We accept the largest individual guaranteeing a higher relative MCR improvement than 1. In 92% of the runs, this rule chooses the correct model size. One may argue, that using a threshold poses too much adaption to the data situation. To investigate this, we try the threshold rule on true model sizes between 1 and 12 again with 100 runs per size. The rule still determines the best model size in 70.41% of the runs. In 14.42% it misses the correct size by one and in 15.17% of the runs it misses the correct size by more than one. In conclusion, the automated rule is apt for the considered data situations. For different data situations it may be necessary to replace the constant threshold by a threshold function depending on the length of the individual. Another idea would be to use a gap statistic as proposed by Tibshirani *et al.* (2001) to detect gaps between data clusters.

8.4 Results on Real and Simulated Data

Apart from data simulated according to (8.5), we use GPAS for the analysis of two different real data sets described in the following. All analyses are conducted on a Pentium 4 CPU with 2.56 GHz and 1024 MB of RAM.

8.4.1 GENICA

The GENICA study is an age-matched and population-based case-control study carried out by the Interdisciplinary Study Group on **G**ene **EN**vironment **I**nteraction and Breast **CA**ncer in Germany, a joint initiative of researchers dedicated to the identification of genetic and environmental factors associated with sporadic breast cancer. Cases and controls have been recruited in the greater Bonn, Germany, region. Apart from exogenous risk factors such as reproduction variables, hormone variables and life style factors, the genotypes of about 100 polymorphisms have been assessed from these women (for details on the GENICA study, see Justenhoven *et al.*, 2004).

Here, the focus is on a subset of the genotype data from the GENICA study. More precisely, data of 1258 women (609 cases and 649 controls) and 63 SNPs are available for the analysis. Since a small number of observations show a large number of missing values, we remove all women with more than five missing values leading to a total of 1191 observations (561 cases and 630 controls). The remaining missing values are replaced SNP-wise by random draws from the marginal distribution.

8.4.2 HapMap

The goals of the International HapMap Project (The International HapMap Consortium, 2003) are the development of a haplotype map of the human genome and the comparison of genetic variations of individuals from different populations. To achieve this goal, millions of SNPs have been genotyped for each of 270 people from four different populations.

Here, the SNP data of 45 unrelated Han Chinese from Beijing and 45 unrelated Japanese from Tokyo are considered. We focus on the BRLMM genotypes (Bayesian Robust Linear Model with Mahalanobis distance; see Affymetrix Inc., 2006) of the 262264 SNPs measured by an Affymetrix GeneChip® 250K Nsp microarray. All SNPs showing one or more missing genotypes (54400 SNPs), for which not all three genotypes are observed (75481 SNPs), or that have a minor allele frequency less than or equal to 0.1 (10609 SNPs) are excluded in this order from the analysis leading to a data set composed of the genotypes of 121774 SNPs and 90 individuals.

8.4.3 Identification of Interesting SNP Interactions

In association studies concerned with sporadic breast cancer it is assumed that not individual SNPs but combinations of many SNPs have high impact on the cancer risk, and that each of these interactions is a risk factor for a particular (relatively small) subgroup of patients (Pharoah *et al.*, 2004). In the analysis of the GENICA data set, we are thus interested in identifying high-order interactions explaining several ten cases, but only a few controls.

Here, we constrain each individual in Algorithm 8.1 to consist of a maximum of two monomials. As $\overline{\mathrm{SNP}}_i^0$ codes a dominant effect of SNP_i, and SNP_i^2 a recessive effect, we restrict the set of literals used to these two literals and their respective complements.

In this application of GPAS to the GENICA data set, we gather the individuals of 50 independent runs each of which stops after 500000 generations, which takes about ten minutes. From the resulting 49564 individuals, the tree visualization is constructed with Algorithm 8.2. An excerpt from this tree is shown in Figure 8.4.

Each path from the root to an inner node or leaf represents an interaction occurring in the final population. The first line in each node consists of the number of monomials containing the corresponding interaction and the percentage of monomials consisting of the ancestral interaction that also contain the literal represented by the node, where this literal is displayed in the second line. The third line shows the number of cases and controls explained by the corresponding interaction. For example, the eight marked literals form an interaction that explains (is true for) 81 cases and only 12 controls and is contained in 404 of the individuals.

Figure 8.4 also reveals that the most interesting SNP interactions all contain the interaction $\mathrm{ERCC2}_{6540}^0\overline{\mathrm{ERCC2}}_{18880}^0$ (for better readability, we write SNP names like ERCC2_18880 as $\mathrm{ERCC2}_{18880}$). This two-way interaction has already been found by Justenhoven *et al.* (2004) and by Schwender and Ickstadt (2008), but they were not able to identify interactions of higher orders with better odds ratios.

For comparison, GPAS is again applied to the GENICA data set using random assignments of the case-control status to the women. In this case, all detected individuals show ratios of explained cases to explained controls that are smaller than the ratios of comparable interactions found in the original application. For example, the individual that is best comparable with the interaction that is marked in Figure 8.4

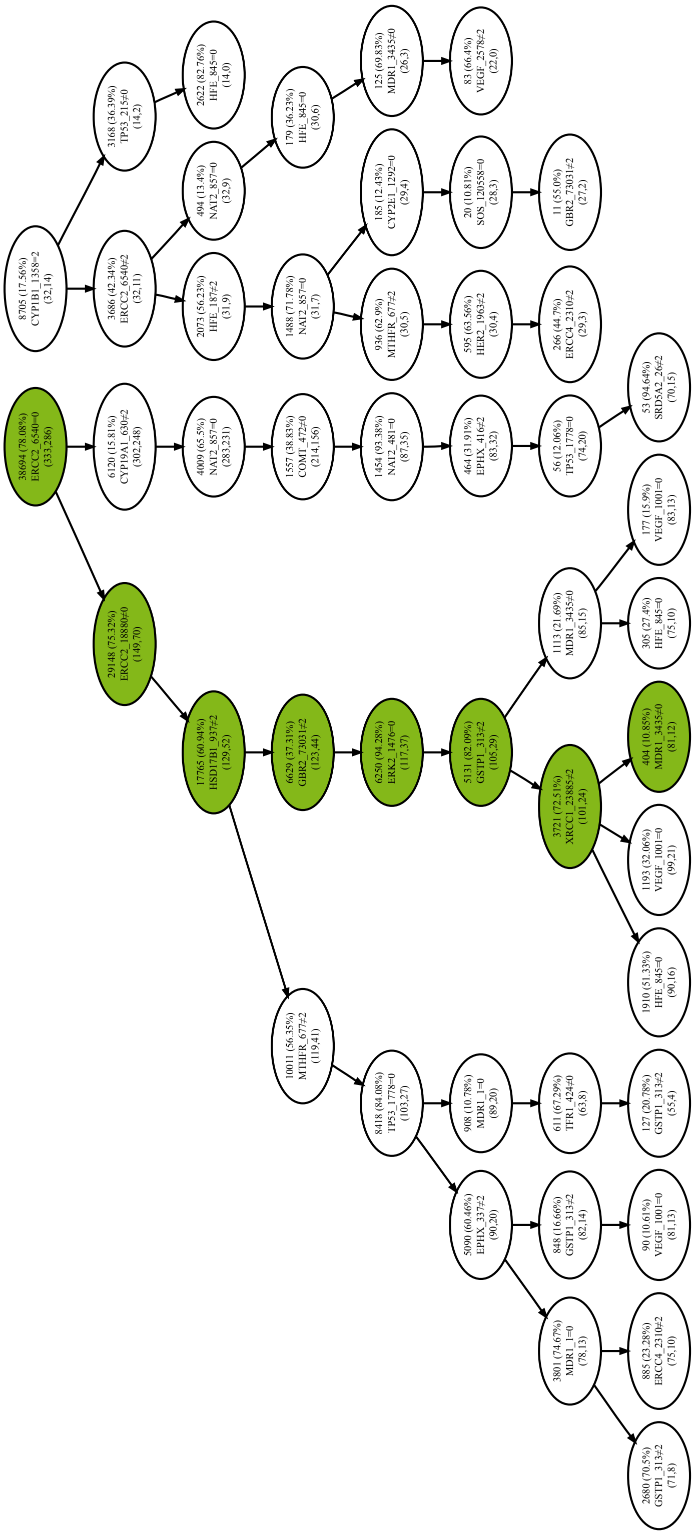

8705 (17.56%) CYP1B1_1358=2 (32,14)
3168 (36.39%) TP53_215≠0 (14,2)
2622 (82.76%) HFE_845=0 (14,0)
3686 (42.34%) ERCC2_6540≠2 (32,11)
494 (13.4%) NAT2_857=0 (32,9)
179 (36.23%) HFE_845=0 (30,6)
125 (69.83%) MDR1_3435≠0 (26,3)
83 (66.4%) VEGF_2578≠2 (22,0)
2073 (56.23%) HFE_187≠2 (31,9)
1488 (71.78%) NAT2_857=0 (31,7)
185 (12.43%) CYP2E1_1292=0 (29,4)
20 (10.81%) SOS_120558=0 (28,3)
11 (55.0%) GBR2_73031≠2 (27,2)
936 (62.9%) MTHFR_677≠2 (30,5)
595 (63.56%) HER2_1963≠2 (30,4)
266 (44.7%) ERCC4_2310≠2 (29,3)
38694 (78.08%) ERCC2_6540=0 (333,286)
6120 (15.81%) CYP19A1_630≠2 (302,248)
4009 (65.5%) NAT2_857=0 (283,231)
1557 (38.83%) COMT_472≠0 (214,156)
1454 (93.38%) NAT2_481=0 (87,35)
464 (31.91%) EPHX_416≠2 (83,32)
56 (12.06%) TP53_1778=0 (74,20)
53 (94.64%) SRD5A2_26≠2 (70,15)
29148 (75.32%) ERCC2_18880≠0 (149,70)
17765 (60.94%) HSD17B1_937≠2 (129,52)
6629 (37.31%) GBR2_73031≠2 (123,44)
6250 (94.28%) ERK2_1476=0 (117,37)
5131 (82.09%) GSTP1_313≠2 (105,29)
1113 (21.69%) MDR1_3435≠0 (85,15)
177 (15.9%) VEGF_1001=0 (83,13)
305 (27.4%) HFE_845=0 (75,10)
3721 (72.51%) XRCC1_23885≠2 (101,24)
404 (10.85%) MDR1_3435≠0 (81,12)
1193 (32.06%) VEGF_1001=0 (99,21)
1910 (51.33%) HFE_845=0 (90,16)
10011 (56.35%) MTHFR_677≠2 (119,41)
8418 (84.08%) TP53_1778=0 (103,27)
908 (10.78%) MDR1_1=0 (89,20)
611 (67.29%) TFR1_424≠0 (63,8)
127 (20.78%) GSTP1_313≠2 (55,4)
5090 (60.46%) EPHX_337≠2 (90,20)
848 (16.66%) GSTP1_313≠2 (82,14)
90 (10.61%) VEGF_1001=0 (81,13)
3801 (74.67%) MDR1_1=0 (78,13)
885 (23.28%) ERCC4_2310≠2 (75,10)
2680 (70.5%) GSTP1_313≠2 (71,8)

FIGURE 8.4: *Excerpt from a tree visualization on GENICA (with the notation $x = a$ and $x \neq a$ instead of x^a and $\bar{x}^a$).*

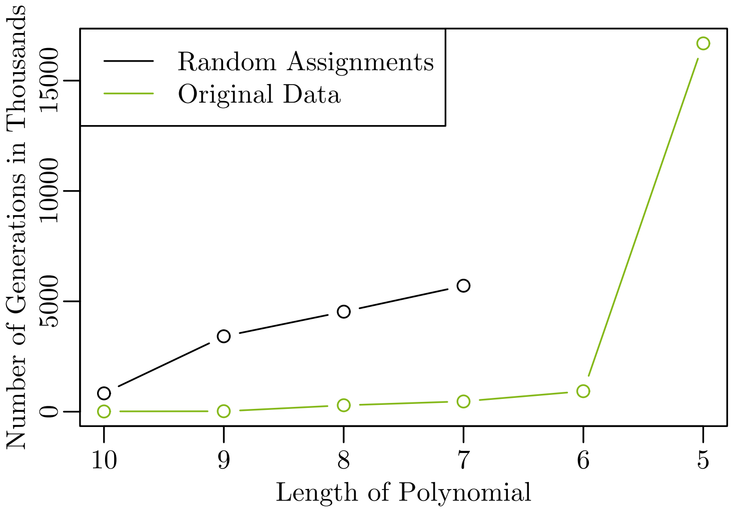


FIGURE 8.5: *Number of generations (in thousands) in which individuals of certain lengths predicting all observations correctly are found in the application of Algorithm 8.1 to the HapMap data set using the real ethnicity and a random group assignment.*

and explains 81 cases and 12 controls is a logic expression that is true for 89 cases and 30 controls.

To examine if the exclusion of SNP_i^1 and its complement has a large influence on the detection of interesting interactions, we also apply GPAS to the GENICA data set using the complete set of literals. In this analysis, some of the literals in the identified monomials are indeed of this type. However, these literals have mostly only a small effect, or they are equivalent to one of SNP_i^0, $\overline{\text{SNP}_i^0}$, SNP_i^2, and $\overline{\text{SNP}_i^2}$, i. e. we obtain the same or nearly the same results if we replace SNP_i^1 and $\overline{\text{SNP}_i^1}$ by one of SNP_i^0, $\overline{\text{SNP}_i^0}$, SNP_i^2, and $\overline{\text{SNP}_i^2}$.

To exemplify that GPAS is not restricted to data sets consisting of several ten to a few hundred SNPs, but can also be applied to data from whole genome studies, we apply GPAS to the subset of the HapMap data set described in Section 8.4.2. As it might be possible that individual SNPs have a large influence in this example, we do not restrict the number of monomials in an individual. Furthermore, we only run GPAS once but without a termination criterion. All other settings remain unchanged compared to the analysis of the GENICA data set.

After running for nine minutes, GPAS detects an individual composed of ten literals in generation 13683 that can be used to distinguish between the Japanese and the Han Chinese unambiguously.

This individual can still be optimized by reducing the length of the polynomial. Shortly after detecting this individual, GPAS finds individuals down to length six (see Figure 8.5), and finally in generation 16691641 an individual composed of five literals,

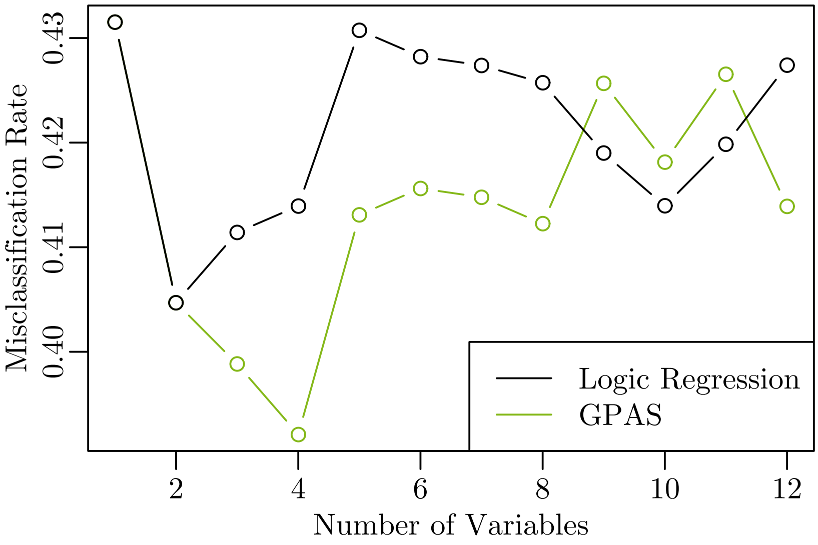


FIGURE 8.6: *Misclassification rates of GPAS and logic regression in their applications to the GENICA data set with restricted numbers of variables in the models.*

where each of these individuals predict all observations correctly. The final individual is

$$\text{SNP\_A}^2_{2104109} \vee \overline{\text{SNP\_A}}^2_{2181939} \text{SNP\_A}^0_{2148128} \overline{\text{SNP\_A}}^0_{2226436} \vee \text{SNP\_A}^2_{2132375} \ .$$

For comparison, GPAS is applied to the HapMap data set using random group assignments. Not surprisingly, the run on the data set with random group assignments also leads to perfect separations of the two groups because of the low number of observations in the data. However, the detected logic expressions are composed of more than five literals, and it takes much longer to detect these individuals (for an example of the results of such an application, see Figure 8.5). Thus, we may conclude that our algorithm detects structure in this large data set although many perfect separations of the two groups exist because of the low number of observations.

8.4.4 Discrimination

To examine how the misclassification rate depends on the number of variables in the model, GPAS is applied to the GENICA data set considering individuals composed of differing numbers of literals. For each number of variables considered, we let GPAS run for 10000 generations, which takes about one minute for each run. For comparison, the GENICA data set is also analyzed using logic regression (Ruczinski *et al.*, 2003), where the number of variables allowed is constrained in the different applications. In Figure 8.6, the resulting misclassification rates estimated by *ten-fold cross-validation* are displayed. Cross-validation is a technique that is used to uncover too strong

		GENICA	HapMap	Simulation
GPAS	Mean	**0.392**	**0.011**	**0.329**
	St. Dev.	0.047	0.034	0.018
Logic Regression	Mean	0.405	0.144	0.342
	St. Dev.	0.049	0.103	0.022
CART	Mean	0.429	0.356	0.371
	St. Dev.	0.034	0.101	0.015
Bagging	Mean	0.453	0.022	0.382
	St. Dev.	0.031	0.044	0.018
Random Forests	Mean	0.450	**0.011**	0.379
	St. Dev.	0.021	0.034	0.018

TABLE 8.1: *Means and standard deviations of the misclassification rates of the applications of several discrimination methods to the GENICA, the HapMap and the simulated data sets.*

adaptation to the data. In k-fold cross-validation, the original data is partitioned into k subsamples. The cross-validation process is then repeated k times with each of the k subsamples as *test data* for an algorithm run on the combined remaining $k-1$ subsamples (*training data*). Thus, the algorithm runs on the training data and tests its results on the test data. We report the mean MCR on the ten test data results. Figure 8.6 shows that the misclassification rates of Algorithm 8.1 and logic regression are equal if the number of literals is less than 3. This is due to the fact that both use $\mathrm{ERCC2}^0_{6540}$ or $\mathrm{ERCC2}^0_{6540}\overline{\mathrm{ERCC2}^0_{18880}}$, respectively, as classification rule in any of the respective iterations of the cross-validation. However, the misclassification rate of Algorithm 8.1 becomes smaller than the one of logic regression if the models are allowed to be composed of three to eight variables.

For a comparison of GPAS with further tree-based discrimination methods, CART (Breiman *et al.*, 1984), Bagging (Breiman, 1996) and Random Forests (Breiman, 2001) are applied to the GENICA data set, where the parameters of the latter two procedures are optimized over several values.

In Table 8.1, the misclassification rates of these applications are summarized. This table reveals that GPAS leads to less misclassifications than the other discrimination procedures.

For the application of these discrimination methods to the HapMap data set, the number of variables has to be reduced to a size that these approaches can handle. On the computer we run our experiments on, only our algorithm works on the full data set. We therefore use the Significance Analysis of Microarrays (SAM) (Tusher *et al.*, 2001) adapted for categorical data (Schwender, 2005) to reduce the number of SNPs from 121774 to 157. All discrimination methods are then applied to this subset of SNPs, and the misclassification is estimated by nine-fold cross-validation, where each of the nine subsets is composed of five randomly chosen Han Chinese and five randomly chosen Japanese.

Since for each of the training sets several models might exist that predict all training observations correctly, we additionally use *bootstrap aggregating* to stabilize the discrimination. In this case, we take 100 *bootstrap samples* of each training data set. A bootstrap sample is a uniformly drawn sample with replacement of the same size as the data set sampled from. Thus, we get 100 results for one training run. To discriminate between Han Chinese and Japanese, we take for each observation the result of the majority of the 100 results. These runs are also stopped after 10000 generations, which takes about twelve minutes for one training (consisting of 100 runs due to the use of bootstrap aggregating).

As Table 8.1 shows, both our algorithm and Random Forests only misclassify one observation, whereas the discrimination methods that use a single model as classification rule, i. e. CART and logic regression, show a comparatively high misclassification rate.

Furthermore, the five discrimination methods are applied to 50 simulated data sets built according to (8.5), where each of these data sets is used once as training set and once as test set. (The classification rule trained on data set 1 is tested on data set 2, the rule trained on data set 2 is tested on data set 3, and so on.) As Table 8.1 reveals, GPAS again shows a misclassification rate that is smaller than the ones of the four other discrimination procedures, and that comes very close to the actual misclassification rate of 32.6%.

8.4.5 Learning the 11-multiplexer

Problem 8.2 describes the problem behind concept learning of multi-valued input, binary-valued output functions. This problem is obviously not limited to genetic association studies. Thus, we look at logic circuit data here. Algorithm 8.1 searches for single output functions, which complicates finding hard to solve benchmark instances from VLSI design (typically containing multiple outputs, see e. g. Brayton *et al.*, 1984). Typical benchmark instances for Boolean concept learners like the multiplexer (Koza, 1992) or the parity function (Koza, 1994) are for multiple reasons also inadequate. First, there is no apparent meaningful variant with multi-valued input and binary-valued output. Second, parity has a DNF of exponential size and Algorithm 8.1 only searches for DNFs. Nevertheless, we include results on the multiplexer function as a binary-valued input function, because it comes closest to our requirements. Here, we consider the 11-multiplexer.

Definition 8.4. The *11-multiplexer* $\mathrm{MUX11} : B^{11} \to B$ is defined as

$$\mathrm{MUX11}(a_2, a_1, a_0, d_7, \ldots, d_0) := d_{a_2 2^2 + a_1 2^1 + a_0 2^0} \ .$$

m	64	128	256	512	1024
MCR	28.12	8.59	0.4	0.0	0.0

Table 8.2: *Mean MCR when learning MUX11.*

Algorithm 8.1 needs only a few seconds to find the DNF of MUX11 on the complete truth table. To test the ability of Algorithm 8.1 to learn MUX11, we determine the MCRs of runs on sampled training data sets with sizes $\{2^6, \dots, 2^{10}\}$. We draw 100 samples for each size and report the mean MCR on the complete truth table of MUX11 in Table 8.2. The results indicate that GPAS is also useful for concept learning of logic circuits.

8.5 Logic Minimization

So far, we considered the application of Algorithm 8.1 to case-control genetic association studies and Boolean concept learning of a multiplexer. In Section 8.2, we encountered the similar problem of logic minimization on incompletely specified truth tables (Problem 8.1) which is our topic here. In fact, in Section 8.4.5 we already encountered a small example for logic minimization with Algorithm 8.1 when we searched for the DNF of MUX11 on the complete truth table of MUX11. However, Algorithm 8.1 is able to deal with multi-valued inputs. This is why we consider the multi-valued version of Problem 8.1 here.

Problem 8.3 (Minimum Polynomial)**.**

Given: Truth-table of a partial function $f : D \subset P^n \rightarrow \{0,1\}$.

Goal: Find a minimal polynomial computing f.

The major difference between concept learning and logic minimization is that logic minimization requires to find functions explaining the given truth table completely, while in a case-control study with a complicated underlying process (which is certainly the case in genetic association studies) this would inevitably lead to overfitting, i. e. functions that are not able to predict for further inputs. So clearly, standard logic minimization approaches cannot hope to compete with our algorithm on genetic association studies. But it is an interesting question, if our algorithm is able to compete with logic minimization approaches on problems with an underlying logical process.

The standard logic minimization tool for this purpose is Espresso MV (Rudell and Sangiovanni-Vincentelli, 1987). For the Boolean case of logic minimization on incompletely specified truth tables, there is also a GP algorithm by Droste (1997) using ordered binary decision diagrams (OBDDs). Kristensen and Miltersen (2006) show that finding small OBDDs for incompletely specified truth tables is NP-hard. This indicates another reason, why logic minimizers are inapt for genetic association studies. For binary-valued logic minimization on completely specified truth tables and for Boolean concept learning various evolutionary approaches exist (see e. g. Banzhaf *et al.*, 1998).

8.5.1 Choice of the Fitness Function for Logic Minimization

Algorithm 8.1 contains the fitness function f_1, which we may also use for logic minimization. While there are many intuitive reasons to use three objectives and therefore large populations when an underlying natural process is assumed, less objectives may also work when minimizing logic circuits. Thus, we also consider the following fitness functions for a partial function f, a polynomial p, and a length restriction $\ell_{\max}$, i. e. the maximum value allowed for $\text{length}(p)$:

$$\begin{aligned} f_3(p) &:= (\text{observations}(p), \text{length}(p)) \\ f_4(p) &:= \frac{\text{cases}(p)}{\text{cases}(f)} + \frac{\text{controls}(p)}{\text{controls}(f)} - \frac{\text{length}(p)}{\ell_{\max} \cdot \text{observations}(f)} \end{aligned}$$

Note, that the last term in f_4 prevents an individual of length $\ell + 1$ that is not better than an individual of length ℓ in the remaining term of f_4 from being accepted. Similar to (8.5), we construct data sets consisting of m three-valued inputs on $n \geq 10$ variables $X_1, \ldots, X_n$. This time, we draw 0, 1, and 2 with equal probability and evaluate the polynomial

$$X_3^0 X_9^0 X_{10}^0 \vee \bar{X}_6^0 X_7^0$$

of length 5. We conduct 100 runs of variants of Algorithm 8.1 with f_1, f_3, and f_4 with $n = 50$ and $m = 1000$ until the algorithm variant stagnates for 10000 consecutive generations. In these runs, the original algorithm with f_1 delivers better results than the variants, i. e. shorter polynomials. As a first testing step for this finding, we conduct a Friedman test on the results, which delivers a p-value of $2.2 \cdot 10^{-16}$. Thus, we may assume that the algorithm variants do not perform equally well. To clarify if f_1 really outperforms f_3 and f_4 we conduct multiple Wilcoxon signed rank sum tests. When we use multiple testing, the significance of the test result changes. To factor this in, we use *Bonferroni correction*, i. e. multiply the resulting p-value by the number of tests.

Hypothesis 8.5 (corrected p-value for both tests $7.4 \cdot 10^{-15}$). *For the task of logic minimization on the considered data situation, fitness function f_1 outperforms fitness functions f_3 and f_4.*

	$m = 100$	1000	10000	25000	50000
$n = 10$	yes	yes	yes	yes	yes
20	yes	yes	yes	yes	no
30	yes	yes	yes	no	no
40	no	no	no	no	no
50	no	no	no	no	no

TABLE 8.3: *Situations where Espresso MV terminated.*

8.5.2 Results on Simulated Logic Circuit Data

As mentioned before, the standard multi-valued logic minimization tool is Espresso MV (Rudell and Sangiovanni-Vincentelli, 1987). To compare the performance of Algorithm 8.1 and Espresso MV, we construct data sets with the process described in the previous section for all 25 combinations of $m \in \{100, 1000, 10000, 25000, 50000\}$ and $n \in \{10, 20, 30, 40, 50\}$. On these data sets, we run our algorithm and Espresso MV each for a maximum of 1 hour. When the algorithms terminate within that time, both are able to compute solutions consisting of the minimal possible number of monomials, i. e. 2. But Espresso MV does not compute a solution in the given time span for larger m and n (see Table 8.3) and for $n \geq 40$ it even did not terminate after 16 hours. Algorithm 8.1 always finds the optimal solution, i. e. the true model, which leads to the conclusion that the application as a concept learner is also successful on these data sets.

It is an interesting question, if the success of our genetic programming algorithm transfers to standard GP (Koza, 1992), i. e. if the success is solely based on the evolutionary approach. Koza (1992) argues that standard GP produces nonrandom results, i. e. delivers better results than blind random search on functions like MUX11. On the other hand, Koza (1992) states that there are Boolean functions that are too hard to learn or minimize for standard GP. To elucidate how our approach compares to standard GP, we run standard GP with the function set $F = \{\wedge, \vee, \neg, \mathrm{IF}\}$ and a terminal set T comprising the same literals as used in Algorithm 8.1 on the 25 data sets. The result of the runs is that standard GP is not able to compute a minimal solution in any of the runs in the given time span. Hence, Algorithm 8.1 drastically outperforms standard GP on the considered data situations.

These first results already show that our algorithm is successful in situations the standard approach cannot cope with. Algorithm 8.1 works well in these examples, because the underlying function has a small DNF (although larger DNFs would not help Espresso MV either). Therefore, we look at the performance on larger DNFs in the following. We reconsider the example with $n = 50$ and $m = 1000$ but this time, we generate data sets with more literals. Strong imbalances in the relation between the number of monomials and the size of the monomials lead to either too much cases

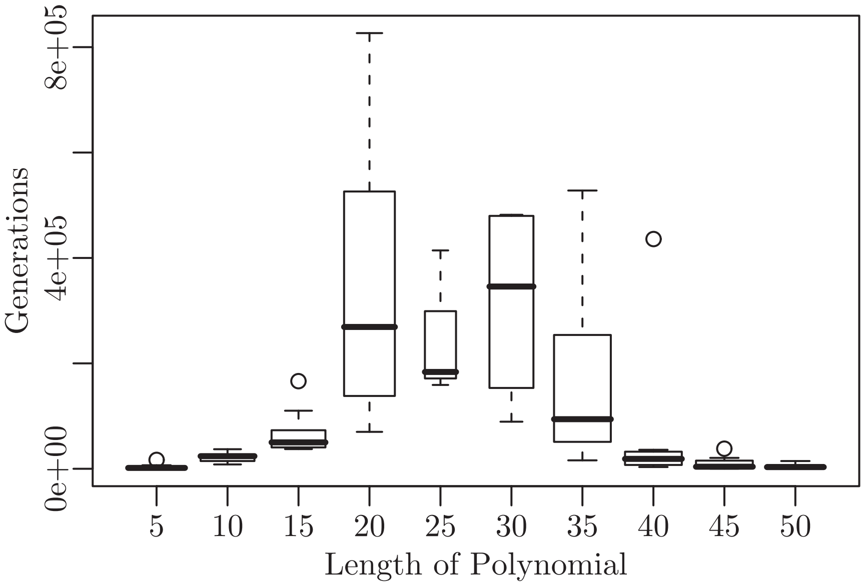


Figure 8.7: *Generations needed to find an optimal solution for underlying DNFs of different sizes. Thinner boxes indicate that not all runs were successful.*

or too much controls in the generated data. To cover a reasonable amount of different DNF lengths, we consider DNFs consisting of five monomials. For these DNFs we consider monomial sizes in $\{1, 2, \dots, 10\}$, again for a maximum running time of one hour per run and report the generation the optimal solution is found.

The result, depicted in Figure 8.7, indicates that up to DNF size 25 it gets harder for the algorithm to find optimal solutions and gets easier again after size 25. Note, that the width of the boxes is proportional to the square root of the number of successful runs and that the box for size 25 is the thinnest, i. e. for this size many runs did not find an optimal solution in the given time span. It is not surprising that it gets easier for the algorithm again for large sizes, because of the values of n and m, i. e. for large sizes of the DNF, the relation between 0s and 1s in the output gets imbalanced and easier functions than originally intended are sufficient to explain the data. When considering $m \in \{100, 1000, 10000, 25000, 50000\}$ output values out of 3^n possible ones this is not surprising and typical for highly inspecified truth tables. Nevertheless, we see that Algorithm 8.1 shows good performance when used for logic minimization in situations with a high amount of inspecified truth table values.

Part IV

Concluding Part

9 Conclusions

The introduction of this thesis started with a quote from Shamos (1976):

> From the viewpoint of applied computational complexity, statistics is a gold mine, for it provides a rich and extensive source of unanalyzed algorithms and computational procedures.

The interface between computer science and statistics offers fruitful exchange and interesting problems. In this thesis, we picked computational problems from regression and classification—in particular robust regression and classification in genetic association studies—and tackled them from a computer scientist's perspective.

The result is a number of new algorithms (see Table 9.1 for an overview) and insights into the treated problems that are useful for further work in the field. The new algorithms improve and complement existing ones and contribute to the respective communities. Many of the presented algorithms are available in the statistics software package R and ready to be used (see Table 9.2 for an overview). The evolutionary computation algorithms are implemented in an extendible framework and allow an easy implementation of evolutionary approaches for different problems.

Apart from the algorithms, much of the methodology used here is applicable to other problems. Other robust regression methods may be tackled with similar techniques, giving a prospect of future work. In particular, geometric duality and the use of decision problems are useful for exact algorithms. The evolutionary approach is useful for heuristics which we often need for high dimensional data because of the NP-hardness of many robust regression methods.

The use of evolutionary computation is already a growing field and especially the ideas for our genetic programming algorithm are extendible in the future. Many classification and learnings problem are of a similar kind and amenable to our approach. With logic minimization, a first example of a different application field was given. In the light of the generality of the problem, many more applications are conceivable. In addition to new application fields, the extension of the ideas to more general functions than $f : \{0, \ldots, p-1\} \to \{0, 1\}$ is of high interest. In the next and concluding section, we give a more detailed outlook of the possible extension to $f : \{0, \ldots, p-1\} \to \{0, \ldots, q\}$.

A very promising field for the application of similar genetic programming algorithms is clustering, where already some evolutionary computation approaches exist (see e. g. Chen and Wang, 2005; Hruschka *et al.*, 2006).

Problem	Main contributions
Q_n	Online algorithm with good running time for many data situations
S_n	Online algorithm with update time $O\left(n\right)$
LQD in $\mathbb{R}^2$	Upper runtime bounds of time $O(n^2 \log^2 n)$ and expected time $O(n^2 \log n)$ Randomized algorithm with expected runtime $O(n^2 \log^2 n)$ Approximation algorithm with runtime $O\left(n^2 \log n \left(\log n - \log\log\left(1+\varepsilon\right)\right)\right)$
Robust regression in high dimensional data	Evolutionary Algorithm for LMS, LQS, LTS, LTA, and LQD.
Case-control genetic association studies	GP algorithm for interaction search and discrimination Automated rule to select good solutions
Logic Minimization	GP algorithm

TABLE 9.1: *Summary of main contributions*

Algorithm	R Package	Function
Algorithm 5.2 Approximation algorithm for the LQD in $\mathbb{R}^2$	robfilter	`lqd.filter`
Algorithm 6.2 Evolutionary algorithm for robust regression	RFreak	`robreg.evol`
Algorithm 8.1 Genetic programming for association studies	RFreak	`GPASDiscrimination` `GPASInteractions`

TABLE 9.2: *Algorithms available in R*

10 Classifying with Decision Diagrams

The HapMap data we encountered in Section 8.4.2 originally contained genetic data on four ethnies, not just two. Apart from case-control studies, the observable traits investigated in genetic association studies often have more than two categories. It is therefore of high interest, to extend Algorithm 8.1 to multi-valued responses. Unfortunately, multi-valued polynomials have certain disadvantages, e. g. they are hard to interpret in a biological meaningful way. Therefore, we propose to use decision diagrams instead for multi-valued responses (see e. g. Wegener, 2000).

Definition 10.1 (Wegener, 2000)**.** A Binary Decision Diagram (BDD) on the variable set $X_n = \{x_1, \dots, x_n\}$ consists of a directed acyclic graph $G = (V, E)$ whose inner nodes (non-sink nodes) have outdegree 2 and a labeling of the nodes and edges. The inner nodes get labels from X_n and the sinks get labels from $\{0, 1\}$. For each inner node, one of the outgoing edges gets the label 0 and the other one gets the label 1. In a BDD on X_n, each node v represents a Boolean function $f_v \in B_n$ defined in the following way:

The computation of $f_v(a)$, $a \in \{0, 1\}^n$, starts at v. At nodes labeled by x_i, the outgoing edge labeled by a_i is chosen. Then $f_v(a)$ is equal to the label of the sink which is reached on the considered path.

A straightforward generalization for multi-valued responses are multiterminal decision diagrams, which allow the sinks to be labeled by more than two different values. In the intended application, we additionally have to deal with multi-valued input data, for example SNP data. We may use the transformation to a Boolean algebra encountered in Section 8.1 and label the BDD nodes with literals

$$x^b := \begin{cases} 1, & \text{if } x = b \\ 0, & \text{otherwise} \end{cases}$$

and their complements. Another possibility is to use multivalued decision diagrams whose inner nodes have outdegree 3 and edge labeling $\{0, 1, 2\}$ in case of SNP data. Figure 10.1 shows an example of a multivalued multiterminal decision diagram that determines the relation between two inputs $x, y \in \{0, 1, 2\}^2$. It computes whether $x < y$, $x > y$, or $x = y$ by alternately reading parts of x and y.

We define computation, prediction, and misclassification similar to Definition 8.2 and obtain a problem analogous to Problem 8.2.

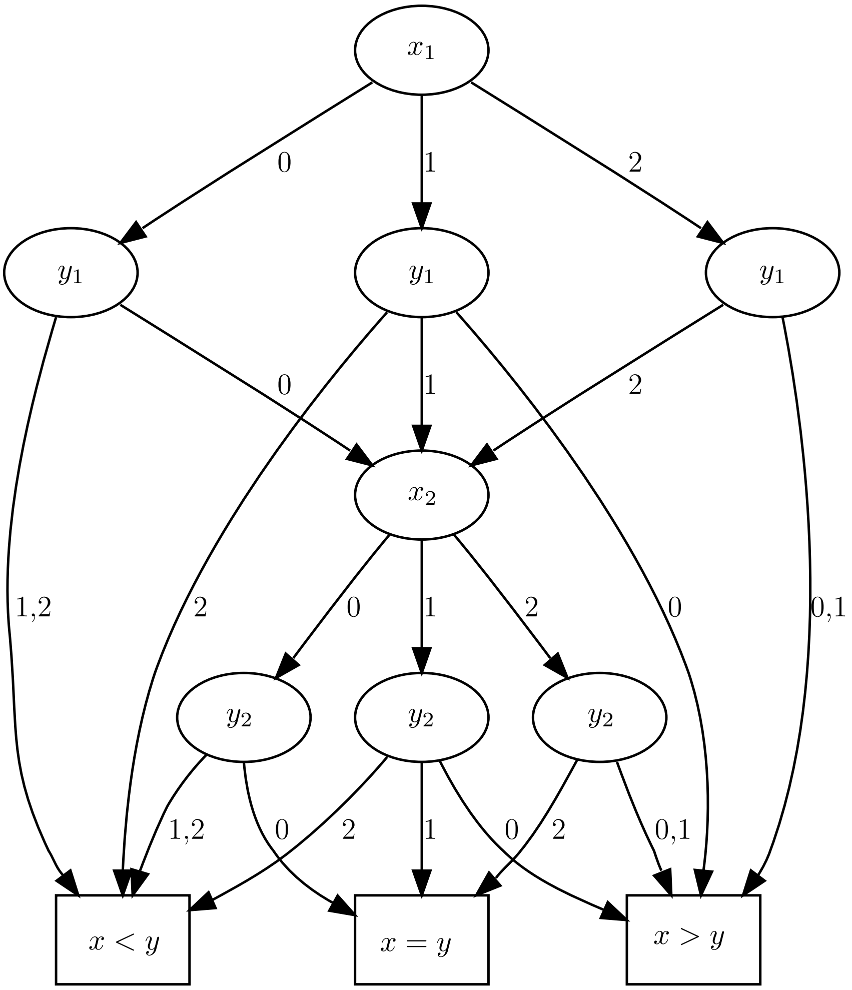

x_1
0
1
2
y_1
y_1
y_1
0
1
2
x_2
1,2
2
0
1
2
0
0,1
y_2
y_2
y_2
1,2
0
2
1
0
2
0,1
$x < y$
$x = y$
$x > y$

FIGURE 10.1: *Multivalued multiterminal decision diagram determining the relation between two inputs* $x, y \in \{0, 1, 2\}^2$.

Definition 10.2.

1. A multiterminal decision diagram d with root r computes $g : D \subset \{0, \dots, p\} \to \{0, \dots, q\}$ if $f_r(x) = g(x)$ for $x \in D$.
2. The *number of observations predicted correctly* by d is defined by

$$\text{observations}(d) := |\{x \in D \mid f_r(x) = g(x)\}| \ ,$$

the *number of specified observations* by

$$\text{observations}(g) := |D| \ .$$

3. The *misclassification rate* (MCR) of d is defined by

$$\text{mcr}(d) := 1 - \frac{\text{observations}(d)}{\text{observations}(g)} \ .$$

Problem 10.1.

Given: Truth-table of a partial function $f : D \subset \{0, \dots, p-1\}^n \to \{0, \dots, q\}$.

Goal: Find a multiterminal decision diagram computing f that has a small number of nodes and a low MCR.

It seems advisable to also use multiobjective fitness functions for Problem 10.1.

10.1 Ordered Decision Diagrams

Decision Diagrams like we introduced them have a lot of representational power but many typical operations are hard to perform. Bryant (1986) introduced *variable orderings* as a restriction that allows more efficient operations.

Definition 10.3. A *variable ordering* π on $X_n = \{x_1, \dots, x_n\}$ is a permutation on the index set $\{0, \dots, n\}$. The position of x_i in the π-ordered list of variables is $\pi\,(i)$. An *ordered decision diagram* is a decision diagram, where the sequence of tests on a path is restricted by the variable ordering π, i. e., if an edge leads from an x_i-node to an x_j-node, then $\pi(i) < \pi(j)$.

Using a variable ordering has the advantage of more efficient operations and a smaller search space, but the disadvantage of having to control violations of the variable ordering.

10.2 Ideas for Adaptions

Apart from the genotype search space and the fitness function, the used adaptions are the most important choice in designing a genetic programming algorithm for decision diagrams. We cannot use exactly the same adaptions as in Figure 8.1, because changes like deleting or inserting nodes need more care in decision diagrams. For example, when deleting a node we have to redirect the ingoing edges. When inserting a node we have to split an edge and decide where the outgoing edges should lead. Redirecting edges itself poses an interesting local adaption.

We also have to take more care of the adaptions when we use a variable ordering. Droste (1997) proposes two crossover operations, that respect the variable ordering. The *path crossover* randomly chooses an edge e_1 to a node v_1 in decision diagram D_1 and a node v_2 in decision diagram D_2 with $\pi(v_1) \leq \pi(v_2)$. The subtree starting at v_2 is then inserted into D_1 as the new target of e_1. The *complete crossover* instead replaces the subtree starting at v_1 with the subtree starting at v_2 and redirects all ingoing edges of v_1 to v_2.

Wegener (2000) also describes how to generate ordered binary decision diagrams (OBDDs) with genetic programming. The mutation Wegener (2000) uses takes the given truth-table into account. In a first step, an OBDD that computes 1 for a random number of truth-table rows and 0 else is constructed. In the second step, this OBDD is used to mutate an OBDD by conducting an *XOR-synthesis*. The result is, that the mutated OBDD computes the opposite than before on the randomly chosen truth-table rows.

All in all, a genetic programming algorithm using decision diagrams and the ideas used for Algorithm 8.1 seems promising for association studies with multi-valued responses.

Bibliography

Adel'son-Vel'skiĭ, G. M. and Landis, E. M. (1962). An algorithm for the organization of information. *Soviet Mathematics Doklady*, **3**, 1259–1263.

Affymetrix Inc. (2006). BRLMM: An improved genotype calling method for the GeneChip Human Mapping 500k array set. Technical report, Affymetrix Inc., Santa Clara, Calif.

Agarwal, P. K. and Sharir, M. (1998). Efficient algorithms for geometric optimization. *ACM Computing Surveys*, **30**(4), 412–458. `http://doi.acm.org/10.1145/299917.299918`.

Agulló, J. (2002). An exchange algorithm for computing the least quartile difference estimator. *Metrika*, **55**, 3–16. `http://dx.doi.org/10.1007/s001840200182`.

Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Walter, P. (2002). *Molecular Biology of the Cell*. Garland Science, New York, fourth edition.

Banzhaf, W., Francone, F. D., Keller, R. E., and Nordin, P. (1998). *Genetic Programming: An Introduction*. Morgan Kaufmann Publishers Inc., San Francisco.

Baragona, R., Battaglia, F., and Cucina, D. (2004). Fitting piecewise linear threshold autoregressive models by means of genetic algorithms. *Computational Statistics & Data Analysis*, **47**(2), 277–295. `http://dx.doi.org/10.1016/j.csda.2003.11.003`.

Barreto, H. and Maharry, D. (2006). Least median of squares and regression through the origin. *Computational Statistics & Data Analysis*, **50**, 1391–1397. `http://dx.doi.org/10.1016/j.csda.2005.01.005`.

Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation*. Natural Computing Series. Springer-Verlag, Berlin, Heidelberg.

Bernholt, T. (2005). Robust estimators are hard to compute. Technical Report 52/2005, SFB 475, Universität Dortmund.

Bernholt, T. (2006). *Effiziente Algorithmen und Komplexität in der robusten Statistik*. Ph.D. thesis, Universität Dortmund.

Bespamyatnikh, S. N. (1998). An optimal algorithm for closest-pair maintenance. *Discrete and Computational Geometry*, **19**(2), 175–195. `http://dx.doi.org/10.1007/PL00009340`.

Blum, M., Floyd, R. W., Pratt, V. R., Rivest, R. L., and Tarjan, R. E. (1973). Time bounds for selection. *Journal of Computer and System Sciences*, **7**(4), 448–461. `http://dx.doi.org/10.1016/S0022-0000(73)80033-9`.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, **31**(3), 307–327. `http://dx.doi.org/10.1016/0304-4076(86)90063-1`.

Brayton, R. K., Sangiovanni-Vincentelli, A. L., McMullen, C. T., and Hachtel, G. D. (1984). *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, Mass.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, **24**(2), 123–140. `http://dx.doi.org/10.1023/A:1018054314350`.

Breiman, L. (2001). Random Forests. *Machine Learning*, **45**(1), 5–32. `http://dx.doi.org/10.1023/A:1010933404324`.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth, Belmont, Calif.

Brockwell, P. J. and Davis, R. A. (2002). *Introduction to Time Series and Forecasting*. Springer, New York, second edition.

Brownlees, C. T. and Gallo, G. M. (2006). Financial econometric analysis at ultra-high frequency: Data handling concerns. *Computational Statistics & Data Analysis*, **51**(4), 2232–2245. `http://dx.doi.org/10.1016/j.csda.2006.09.030`.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, **35**(8), 677–691. `http://dx.doi.org/10.1109/TC.1986.1676819`.

Brys, G., Hubert, M., and Struyf, A. (2004). A robust measure of skewness. *Journal of Computational & Graphical Statistics*, **13**(4), 996–1017. `http://dx.doi.org/10.1198/106186004X12632`.

Chan, T. M. (1999). Geometric applications of a randomized optimization technique. *Discrete and Computational Geometry*, **22**(4), 547–567. `http://dx.doi.org/10.1007/PL00009478`.

Chazelle, B., Guibas, L. J., and Lee, D. T. (1985). The power of geometric duality. *BIT*, **25**(1), 76–90. `http://dx.doi.org/10.1007/BF01934990`.

Chen, E. and Wang, F. (2005). Dynamic clustering using multi-objective evolutionary algorithm. In *Computational Intelligence and Security*, volume 3801 of *Lecture Notes in Computer Science*, pages 73–80, Berlin, Heidelberg. Springer-Verlag. `http://dx.doi.org/10.1007/11596448_10`.

Cole, R., Sharir, M., and Yap, C. K. (1987). On k-hulls and related problems. *SIAM Journal on Computing*, **16**(1), 61–77. `http://dx.doi.org/10.1137/0216005`.

Conover, W. J. (1999). *Practical Nonparametric Statistics*. John Wiley & Sons Inc., New York, third edition.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., second edition.

Croux, C. and Rousseeuw, P. J. (1992). Time-efficient algorithms for two highly robust estimators of scale. *Computational Statistics*, **1**, 411–428.

Croux, C., Rousseeuw, P. J., and Hössjer, O. (1994). Generalized s-estimators. *Journal of the American Statistical Association*, **89**, 1271–1281.

Culverhouse, R., Suarez, B. K., Lin, J., and Reich, T. (2002). A perspective on epistasis: Limits of models displaying no main effect. *The American Journal of Human Genetics*, **70**(2), 461–471. `http://dx.doi.org/10.1086/338759`.

de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, third edition.

De Jong, K. A. (2006). *Evolutionary Computation: A Unified Approach*. The MIT Press, Cambridge, Mass.

Donoho, D. and Huber, P. (1983). The notion of breakdown point. In P. Bickel, K. Doksum, and J. Hodges, Jr., editors, *A Festschrift for Erich L. Lehmann*, pages 157–184. Wadsworth, Belmont, Calif.

Drew, H. R., Wing, R. M., Takano, T., Broka, C., Tanaka, S., Itakura, K., and Dickerson, R. E. (1981). Structure of a b-dna dodecamer: conformation and dynamics. *Proceedings of the National Academy of Sciences of the United States of America*, **78**(4), 2179–2183.

Droste, S. (1997). Efficient genetic programming for finding good generalizing Boolean functions. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzo, H. Iba, and R. L. Riolo, editors, *Proceedings of the Second Annual Conference on Genetic Programming*, pages 82–87, San Francisco, Calif. Morgan Kaufmann Publishers, Inc.

Dryden, I. and Walker, G. (1999). Highly resistant regression and object matching. *Biometrics*, **55**(3), 820–825. `http://dx.doi.org/10.1111/j.0006-341X.1999.00820.x`.

Dussault, J., Metze, G., and Krieger, M. (1976). A multivalued switching algebra with boolean properties. In *Proceedings of the Sixth International Symposium on Multiple-valued Logic*, pages 68–73, Los Alamitos, Calif. IEEE Computer Society Press.

Edelsbrunner, H. and Mücke, E. P. (1990). Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, **9**(1), 66–104. `http://doi.acm.org/10.1145/77635.77639`.

Edelsbrunner, H. and Souvaine, D. (1990). Computing least median of squares regression and guided topological sweep. *Journal of the American Statistical Association*, **85**, 115–119.

Fried, R. (2007). On the robust detection of edges in time series filtering. *Computational Statistics & Data Analysis*, **52**(2), 1063–1074. `http://dx.doi.org/10.1016/j.csda.2007.06.011`.

Fried, R. and Schettlinger, K. (2008). *robfilter: Robust Time Series Filters*. R package version 2.3.

Fried, R., Bernholt, T., and Gather, U. (2006). Repeated median and hybrid filters. *Computational Statistics & Data Analysis*, **50**(9), 2313–2338. `http://dx.doi.org/10.1016/j.csda.2004.12.013`.

Garte, S. (2001). Metabolic susceptibility genes as cancer risk factors: Time for a reassessment? *Cancer Epidemiology Biomarkers & Prevention*, **10**, 1233–1237.

Gather, U. and Fried, R. (2003). Robust scale estimation for local linear temporal trends. *Tatra Mountains Mathematical Publications*, **26**, 87–101.

Gelper, S., Schettlinger, K., Croux, C., and Gather, U. (2009). Robust online scale estimation in time series: A regression-free approach. *Journal of Statistical Planning and Inference*, **139**(2), 335–349. `http://dx.doi.org/10.1016/j.jspi.2008.04.018`.

Gentle, J. E., Härdle, W., and Mori, Y., editors (2004). *Handbook of Computational Statistics - Concepts and Methods*. Springer, New York.

Giloni, A. and Padberg, M. (2004). The finite sample breakdown point of ℓ_1-regression. *SIAM Journal on Optimization*, **14**(4), 1028–1042. `http://dx.doi.org/10.1137/S1052623403424156`.

Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, **101**(23), 215–220.

Goodman, J. E., Mechanic, L. E., Luke, B. T., Ambs, S., Chanock, S., and Harris, C. C. (2006). Exploring SNP-SNP interactions and colon cancer risk using polymorphism interaction analysis. *International Journal of Cancer*, **118**(7), 1790–1797. `http://dx.doi.org/10.1002/ijc.21523`.

Hampel, F. R. (1974). The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, **69**(346), 383–393.

Har-Peled, S. (1998). Constructing cuttings in theory and practice. In *SCG '98: Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, pages 327–336, New York. ACM Press. `http://doi.acm.org/10.1145/276884.276921`.

Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and valiant's learning framework. *Artificial Inelligence*, **36**(2), 177–221. `http://dx.doi.org/10.1016/0004-3702(88)90002-1`.

Hawkins, D. M. (1993). The feasible set algorithm for least median of squares regression. *Computational Statistics & Data Analysis*, **16**(1), 81–101. `http://dx.doi.org/10.1016/0167-9473(93)90246-P`.

Hawkins, D. M. and Olive, D. J. (1999). Improved feasible solution algorithms for high breakdown estimation. *Computational Statistics & Data Analysis*, **30**(1), 1–11. `http://dx.doi.org/10.1016/S0167-9473(98)00082-6`.

Heidema, G. A., Boer, J. M. A., Nagelkerke, N., Mariman, E. C. M., van de A, D. L., and Feskens, E. J. M. (2006). The challenge for genetic epidemiologists: How to analyze large numbers of SNPs in relation to complex diseases. *BMC Genetics*, **7**(23). `http://dx.doi.org/10.1186/1471-2156-7-23`.

Hodges, Jr., J. (1967). Efficiency in normal samples and tolerance of extreme values for some estimators of location. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume I, pages 163–186, Berkeley, Los Angeles. University of California Press. `http://projecteuclid.org/euclid.bsmsp/1200512985`.

Hodges, Jr., J. L. and Lehmann, E. L. (1963). Estimates of location based on rank tests. *The Annals of Mathematical Statistics*, **34**(2), 598–611. `http://projecteuclid.org/euclid.aoms/1177704172`.

Hoh, J. and Ott, J. (2003). Mathematical multi-locus approaches to localizing complex human trait genes. *Nature Reviews Genetics*, **4**, 701–709. `http://dx.doi.org/10.1038/nrg1155`.

Hössjer, O. (1994). Rank-based estimates in the linear model with high breakdown point. *Journal of the American Statistical Association*, **89**(425), 149–158.

Hruschka, E. R., Campello, R. J., and de Castro, L. N. (2006). Evolving clusters in gene-expression data. *Information Sciences*, **176**(13), 1898–1927. `http://dx.doi.org/10.1016/j.ins.2005.07.015`.

Johnson, D. B. and Kashdan, S. D. (1978). Lower bounds for selection in $X + Y$ and other multisets. *Journal of the ACM*, **25**(4), 556–570. `http://doi.acm.org/10.1145/322092.322097`.

Johnson, D. B. and Mizoguchi, T. (1978). Selecting the kth element in $x + y$ and $x_1 + x_2 + ... + x_m$. *SIAM Journal on Computing*, **7**(2), 147–153. `http://dx.doi.org/10.1137/0207013`.

Johnstone, I. M. and Velleman, P. F. (1985). The resistant line and related regression methods. *Journal of the American Statistical Association*, **80**(392), 1041–1054.

Justenhoven, C., Hamann, U., Pesch, B., Harth, V., Rabstein, S., Baisch, C., Vollmert, C., Illig, T., Ko, Y., Brüning, T., and Brauch, H. (2004). ERCC2 genotypes and a corresponding haplotype are linked with breast cancer risk in a German population. *Cancer Epidemiology, Biomarkers and Prevention*, **13**(12), 2059–2064.

Knuth, D. E. (1973). *The Art of Computer Programming Vol. 3: Sorting and Searching*. Addison-Wesley Publishing Company, Reading, Mass.

Kooperberg, C. and Ruczinski, I. (2005). Identifying interacting SNPs using Monte Carlo logic regression. *Genetic Epidemiology*, **28**(2), 157–170. `http://dx.doi.org/10.1002/gepi.20042`.

Kooperberg, C., Ruczinski, I., LeBlanc, M., and Hsu, L. (2001). Sequence analysis using logic regression. *Genetic Epidemiology*, **21**(Suppl 1), S626–S631.

Koza, J. R. (1992). *Genetic Programming*. The MIT Press, Cambridge, Mass.

Koza, J. R. (1994). *Genetic Programming II*. The MIT Press, Cambridge. Mass.

Kristensen, J. T. and Miltersen, P. B. (2006). Finding small OBDDs for incompletely specified truth tables is hard. In *Proceedings of COCOON 2006*, volume 4112 of *Lecture Notes in Computer Science*, pages 489–496, Berlin, Heidelberg. Springer-Verlag. `http://dx.doi.org/10.1007/11809678_51`.

Lukas, S. and Czort, A. (1999). The complexity of minimizing disjunctive normal formulas. Technical report, Department of Computer Science, University of Aarhus.

Lunetta, K. L., Hayward, L. B., Segal, J., and van Eerdewegh, P. (2004). Screening large-scale association study data: exploiting interactions using random forests. *BMC Genetics*, **5**(32). `http://dx.doi.org/10.1186/1471-2156-5-32`.

Ma, Y. and Genton, M. G. (2000). Highly robust estimation of the autocovariance function. *Journal of Time Series Analysis*, **21**(6), 663–684. `http://dx.doi.org/10.1111/1467-9892.00203`.

Marchini, J., Donnely, P., and Cardon, R. C. (2005). Genome-wide strategies for detecting multiple loci that influence complex diseases. *Nature Genetics*, **37**, 413–417. `http://dx.doi.org/10.1038/ng1537`.

Mebane, Jr., W. R. and Sekhon, J. (2004). Robust etimation and outlier detection for overdispersed multinomial models of count data. *American Journal of Political Science*, **48**(2), 391–410. `http://dx.doi.org/10.1111/j.0092-5853.2004.00077.x`.

Megiddo, N. (1979). Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, **4**(4), 414–424. `http://dx.doi.org/10.1287/moor.4.4.414`.

Mercurio, D. and Spokoiny, V. (2004). Statistical inference for time-inhomogeneous volatility models. *The Annals of Statistics*, **32**(2), 577–602. `http://dx.doi.org/10.1214/009053604000000102`.

Meyer, M. C. (2003). An evolutionary algorithm with applications to statistics. *Journal of Computational & Graphical Statistics*, **12**(2), 265–281. `http://dx.doi.org/10.1198/1061860031699`.

Mood, A. M., Graybill, F. A., and Boes, D. C. (1974). *Introduction to the Theory of Statistics*. McGraw-Hill Book Company, New York, third edition.

Muthukrishnan, S. (2005). Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, **1**(2). `http://dx.doi.org/10.1561/0400000002`.

Nunkesser, R. (2008). Rfreak–an r package for evolutionary computation. Technical Report 12/2008, SFB 475, Technische Universität Dortmund.

Oliveto, P. S., He, J., and Yao, X. (2007). Computational complexity analysis of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, **4**(3), 281–293. `http://dx.doi.org/10.1007/s11633-007-0281-3`.

Pettersen, E. F., Goddard, T. D., Huang, C. C., Couch, G. S., Greenblatt, D. M., Meng, E. C., and Ferrin, T. E. (2004). Ucsf chimera–a visualization system for exploratory research and analysis. *Journal on Computational Chemistry*, **25**(13), 1605–1612. `http://dx.doi.org/10.1002/jcc.20084`.

Pharoah, P. D., Dunning, A. M., Ponder, B. A., and Easton, D. F. (2004). Association studies for finding cancer-susceptibility genetic variants. *Nature Reviews Cancer*, **4**(11), 850–860. `http://dx.doi.org/10.1038/nrc1476`.

Plackett, R. L. (1972). The discovery of the method of least squares. *Biometrika*, **59**, 239–251. `http://dx.doi.org/10.1093/biomet/59.2.239`.

Portnoy, S. and Koenker, R. (1997). The gaussian hare and the laplacian tortoise: Computability of squared-error versus absolute-error estimators. *Statistical Science*, **12**(4), 279–300. `http://dx.doi.org/10.1214/ss/1030037960`.

Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry: An Introduction*. Springer, New York.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ritchie, M. D., Hahn, L. W., Roodi, N., Bailey, L. R., Dupont, W. D., Parl, F. F., and Moore, J. H. (2001). Multifactor-dimensionality reduction reveals high-order interactions among estrogen-metabolism genes in sporadic breast cancer. *American Journal of Human Genetics*, **69**(1), 138–147. `http://dx.doi.org/10.1086/321276`.

Roos, T. and Widmayer, P. (1994). k-violation linear programming. *Information Processing Letters*, **52**(2), 109–114. `http://dx.doi.org/10.1016/0020-0190(94)00134-0`.

Rousseeuw, P. and Hubert, M. (1997). Recent developments in PROGRESS. In Y. Dodge, editor, *L_1-Statistical Procedures and Related Topics*, volume 31 of *Lecture Notes-Monograph Series*, pages 201–214, Beachwood, Ohio. Institute of Mathematical Statistics. `http://dx.doi.org/10.1214/lnms/1215454138`.

Rousseeuw, P. J. (1984). Least median of squares regression. *Journal of the American Statistical Association*, **79**, 871–880.

Rousseeuw, P. J. and Basset Jr., G. W. (1991). Robustness of the p-subset algorithm for regression with high breakdown point. In W. Stahel and S. Weisberg, editors, *Directions in Robust Statistics and Diagnostics, Part II*, volume 34 of *The IMA Volumes in Mathematics and its Applications*, pages 185–194. Springer, New-York.

Rousseeuw, P. J. and Croux, C. (1993). Alternatives to the median absolute deviation. *Journal of the American Statistical Association*, **88**(424), 1273–1283.

Rousseeuw, P. J. and Leroy, A. M. (1987). *Robust Regression and Outlier Detection*. John Wiley & Sons Inc., New York.

Rousseeuw, P. J. and Van Driessen, K. (2006). Computing lts regression for large data sets. *Data Mining and Knowledge Discovery*, **12**(1), 29–45. `http://dx.doi.org/10.1007/s10618-005-0024-4`.

Ruczinski, I., Kooperberg, C., and LeBlanc, M. (2003). Logic regression. *Journal of Computational and Graphical Statistics*, **12**, 475–511. `http://dx.doi.org/10.1198/1061860032238`.

Ruczinski, I., Kooperberg, C., and LeBlanc, M. (2004). Exploring interactions in high-dimensional genomic data: An overview of logic regression, with applications. *Journal of Multivariate Analysis*, **90**(1), 178–195. `http://dx.doi.org/10.1016/j.jmva.2004.02.010`.

Rudell, R. L. and Sangiovanni-Vincentelli, A. (1987). Multiple-valued minimization for PLA optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **6**(5), 727–750. `http://dx.doi.org/10.1109/TCAD.1987.1270318`.

Schwender, H. (2005). Modifying microarray analysis methods for categorical data – SAM and PAM for SNPs. In C. Weihs and W. Gaul, editors, *Classification – The Ubiquitous Challenge*, pages 370–377, Berlin, Heidelberg. Springer-Verlag. `http://dx.doi.org/10.1007/3-540-28084-7_42`.

Schwender, H. (2007). *Statistical analysis of genotype and gene expression data*. Ph.D. thesis, Universität Dortmund.

Schwender, H. and Fritsch, A. (2008). *scrime: Analysis of High-Dimensional Categorical Data such as SNP Data*. R package version 1.0.0.

Schwender, H. and Ickstadt, K. (2008). Identification of SNP interactions using logic regression. *Biostatistics*, **9**(1), 187–198. `http://dx.doi.org/10.1093/biostatistics/kxm024`.

Schwender, H., Rabstein, S., and Ickstadt, K. (2006). Do you speak genomish? *Chance*, **19**(3), 3–8.

Shamos, M. I. (1976). Geometry and statistics: Problems at the interface. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 251–280. Academic Press, New York.

Smid, M. (1991). Maintaining the minimal distance of a point set in less than linear time. *Algorithms Review*, **2**, 33–44.

Souvaine, D. L. and Steele, J. M. (1987). Time- and space-efficient algorithms for least median of squares regression. *Journal of the American Statistical Association*, **82**(399), 794–801.

Stigler, S. M. (1981). Gauss and the invention of least squares. *The Annals of Statistics*, **9**(3), 465–474. `http://dx.doi.org/10.1214/aos/1176345451`.

Stromberg, A. J. (1993). Computing the exact least median of squares estimate and stability diagnostics in multiple linear regression. *SIAM Journal on Scientific Computing*, **14**(6), 1289–1299. `http://dx.doi.org/10.1137/0914076`.

Su, S. Y. H. and Sarris, A. A. (1972). The relationship between multivalued switching algebra and boolean algebra under different definitions of complement. *IEEE Transactions on Computers*, **21**(5), 479–485. `http://dx.doi.org/10.1109/T-C.1972.223544`.

The International HapMap Consortium (2003). The International HapMap Project. *Nature*, **426**, 789–796.

Tibshirani, R., Walther, G., and Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **63**(2), 411–423. `http://dx.doi.org/10.1111/1467-9868.00293`.

Todorov, V., Ruckstuhl, A., Salibian-Barrera, M., and Maechler, M. (2007). *robustbase: Basic Robust Statistics*. R package version 0.2-8.

Tusher, V., Tibshirani, R., and Chu, G. (2001). Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Sciences of the United States of America*, **98**(9), 5116–5124. `http://dx.doi.org/10.1073/pnas.091062498`.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM Computing Surveys*, **27**(11), 1134–1142. `http://doi.acm.org/10.1145/1968.1972`.

van Oostrum, R. and Veltkamp, R. C. (2002). Parametric search made practical. In *SCG '02: Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, pages 1–9, New York. ACM Press. `http://doi.acm.org/10.1145/513400.513401`.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition.

Wegener, I. (1987). *The Complexity of Boolean Functions*. John Wiley & Sons Inc., New York.

Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams*. SIAM, Philadelphia.

Witte, J. S. and Fijal, B. A. (2001). Introduction: Analysis of sequence data and population structure. *Genetic Epidemiology*, **21**(Suppl 1), S600–S601.

Zitzler, E., Knowles, J., and Thiele, L. (2008). Quality assessment of pareto set approximations. In J. Branke, K. Deb, K. Miettinen, and R. Slowinski, editors, *Multiobjective Optimization*, number 5252 in Lecture Notes in Computer Science, chapter 14, pages 373–404. Springer-Verlag, Berlin, Heidelberg. `http://dx.doi.org/10.1007/978-3-540-88908-3_14`.

List of Algorithms

4.1 Sketch of the algorithm of Johnson and Mizoguchi (1978) 24
4.2 Moving window selection in $X + Y$ 25
4.3 `Insert`(x_{ins}, sample X, sample Y, buffer B) (for Q_n) 26
4.4 `Delete`(x_{del}, sample X, sample Y, buffer B) (for Q_n) 27
4.5 Moving window computation of S_n 35
4.6 `Insert`(x_{ins}, sample X, m) (for S_n) 36
4.7 `Delete`(x_{del}, sample X, m) (for S_n) 37
5.1 `SearchLocalSolution`(set of lines L, value r, level parameter k) 46
5.2 Approximate minimum of k-level 47
5.3 `NoOfIntersectionsBetween`(set of lines L, intercept r_1, intercept r_2) . . 49
5.4 Randomized algorithm for minimum of k-level 50
6.1 Basic Evolutionary Algorithm 55
6.2 Evolutionary Algorithm for Robust Regression 56
8.1 Genetic Programming for Association Studies (GPAS) 73
8.2 Construct Interaction Tree 75
8.3 `SearchMostCommonLiteral`(set of monomials M, tree node t, tree T) . . 75

List of Figures

3.1 Bounded change of the sample median after the insertion of $n-1$ observations. 12
3.2 Linear regression example. 15
3.3 Data on international phone calls from Belgium with LS and LAD fit and data on light intensity and temperature of the star cluster CYG OB1 with LAD and LMS fit. 16
3.4 Estimation by standard deviation and by MAD with outliers. 17

4.1 Heart activity measured by ECG. 20
4.2 Regions in the matrix M with definitely smaller or certainly greater values than $x_{(i)} + y_{(i)}$. 23
4.3 Example for the pointer structure. 26
4.4 Simulated GARCH(1,1) series and estimated volatilities. 29
4.5 Time series with piecewise constant variance and estimated volatilities. 29
4.6 Time series with piecewise constant level plus additive noise generated from an AR(1) model and estimated volatilities. 30
4.7 Daily returns of Bayer AG stocks between January 4, 1960 and July 17, 2007 and estimated volatilities. 30
4.8 Running time needed for the online analyses of the four considered data situations in comparison to the offline algorithm. 31
4.9 Positions of the buffer B in the matrix M for the four data sets. 32
4.10 Illustration of $\mathcal{L}_1$, $\mathcal{G}_1$, $\mathcal{L}_2$, and $\mathcal{G}_2$. 34

5.1 Nine points and a line in primal space and the corresponding dual space. 41
5.2 An example for the mapping of four primal points to twelve dual lines. 43
5.3 The idea behind `SearchLocalSolution`. 46
5.4 The basic idea behind the approximation and the randomized algorithm. 48
5.5 Boxplots of the running time in seconds. 52

6.1 Example of simulated data for $p = 1$. 59
6.2 Comparison of `robreg.evol` with `ltsReg`. 59
6.3 Comparison of `robreg.evol` with `lqs`. 60

7.1 DNA sequence described by CGCGAATTCGCG. 64
7.2 The three different SNP forms AA, Aa/aA, and aa. 65

8.1 Example for the crossover and the different mutations used in GPAS. . 74
8.2 MCRs for the best individuals of different sizes. 77
8.3 MCRs improvement per length for individuals represented by points on the convex hull. 78
8.4 Excerpt from a tree visualization on GENICA. 81
8.5 Number of generations in which individuals of certain lengths predicting all observations correctly are found. 82
8.6 Misclassification rates of GPAS and logic regression in their applications to the GENICA data set with restricted numbers of variables. 83
8.7 Generations needed to find an optimal solution for underlying DNFs of different sizes. 89

10.1 Multivalued multiterminal decision diagram determining the relation between two inputs $x, y \in \{0, 1, 2\}^2$. 96

List of Tables

2.1 Overview of underlying publications and the contribution of the author 7

7.1 Example for the Wilcoxon signed rank test 67
7.2 Example for the Friedman rank sum test 67

8.1 Means and standard deviations of the misclassification rates of the applications of several discrimination methods to the GENICA, the HapMap and the simulated data sets. 84
8.2 Mean MCR when learning MUX11. 86
8.3 Situations where Espresso MV terminated. 88

9.1 Summary of main contributions 94
9.2 Algorithms available in R 94

Index

11-multiplexer, 85

A, *see* adenine
adenine, 63
alarm systems, 20
algorithms
  approximation algorithm for least quartile difference, 47–49
  approximation algorithm for minimum of k-level, 47–49
  basic evolutionary algorithm, 55
  evolutionary algorithm for robust regression, 56
  moving window computation of S_n, 35
  moving window selection in $X + Y$, 25
  randomized algorithm for least quartile difference, 49–51
  randomized algorithm for minimum of k-level, 49–51
  sketch of the algorithm of Johnson and Mizoguchi, 24
allele, 63
  major allele, 63
  minor allele, 64
alternative hypothesis, 66
AR(1), 30
asymptotic notation, 13
AVL trees, 22

BDD, *see* Binary Decision Diagram
Binary Decision Diagram, 95
Boolean concept learning, 69
bootstrap aggregating, 85
bootstrap sample, 85
boxplots, 52
breakdown, 11
breakdown point, 11
  contamination breakdown point, 11
  finite sample breakdown point, 11
  of least absolute deviations regression, 15
  of least median of squares, 16
  of least quantile of squares, 53
  of least quartile difference, 39
  of least squares regression, 15
  of least trimmed squares, 53
  of least trimmed sum of absolute values, 53
  of median absolute deviation, 17
  of Q_n, 19
  of S_n, 19
  of sample mean, 11
  of sample median, 12
  of sample standard deviation, 17
  replacement breakdown point, 12

C, *see* cytosine
case, 63
case-control studies, 63
chromosome, 63
closed half-space, *see* half-space
complexity, *see* computation time
computation time
  of least median of squares, 40
  of least median of squares through the origin, 45

  of least quantile of squares through the origin, 45
  of least quartile difference, 40, 44
  of median, 13, 19
  of median of two sets, 33
  of median of $X + Y$, 23
  of minimum of k-level, 45
  of Q_n, 19, 22, 27
  of S_n, 19, 33, 38
  of selection in $X + Y$, 22
construct interaction tree, 75
contamination breakdown point, *see* breakdown point
control, 63
convex, 78
convex hull, 77, 78
cross-validation, 84
crossover, 54
cuttings, 45
cytosine, 63

data streams, 20
`Delete` (for Q_n), 27
`Delete` (for S_n), 37
deoxyribonucleic acid, 63
disjunctive normal form, *see* polynomial
DNA, *see* deoxyribonucleic acid
DNF, *see* polynomial
dominate, 72
double helix, 63
dual space, 40
duality transform, 40
  for least quartile difference, 42

efficiency, 13
  Gaussian efficiency, 13
  of least median of squares, 39
  of least quantile of squares, 53
  of least quartile difference, 39
  of least trimmed squares, 54
  of median absolute deviation, 17
  of Q_n, 19
  of S_n, 19
  of sample mean, 13
  of sample median, 13
Espresso MV, 86
estimation of location, 11–12
  Hodges-Lehmann estimator, 22, 23
  sample mean, 11
  sample median, 12
estimation of scale, 16–38
  median absolute deviation, 17
  Q_n, 19
  S_n, 19
  sample standard deviation, 16
evolutionary algorithms, *see* evolutionary computation
evolutionary computation, 54–55, 64–65
  evolutionary algorithms, 54–55
  genetic programming, 64–65
exact fit property, 53
example, 69
explanatory variable, 13

FAST-LTS, 54
feasible solution algorithm, 54
finite sample breakdown point, *see* breakdown point
fitness, 54
Friedman rank sum test, 66
FSA, *see* feasible solution algorithm

G, *see* guanine
GARCH$(1, 1)$, 28
Gaussian efficiency, *see* efficiency
general position, 16, 42
generation, 55
genetic association studies, 63
genetic programming, *see* evolutionary computation
genetic programming for association studies, 73
genetic variation, 63

genome, 63
genotype, 54
geometric duality, 40–41
  applied to least quartile difference, 42
GP, *see* evolutionary computation
GPAS, *see* genetic programming for association studies
guanine, 63

half-space, 41
heterozygous variant, 64
Hodges-Lehmann estimator, 22
homozygous reference, 64
homozygous variant, 64
hypothesis, 66
hypothesis testing, 65–66

individual, 54
`Insert` (for Q_n), 26
`Insert` (for S_n), 36
inversion, 49
inversion table, 49

k-fold cross-validation, *see* cross-validation
k-level, 42
  approximation algorithm for minimum, 47–49
  computation time of minimum, 45
  minimum point of, 42
  randomized algorithm for minimum, 49–51
k-violation linear programming, *see* k-level

LAD, *see* least absolute deviations
least absolute deviations, 15
  breakdown point of, 15
least median of squares, 15, 53–60
  breakdown point of, 16
  computation time of, 40
  efficiency of, 39
  evolutionary algorithm for, 56
  through the origin, 45
least quantile of squares, 40, 53–60
  breakdown point of, 53
  efficiency of, 53
  evolutionary algorithm for, 56
  through the origin, 45
least quartile difference, 39–60
  approximation algorithm for, 47–49
  breakdown point of, 39
  computation time of, 40, 44
  dual problem, 42
  duality transform, 42
  efficiency of, 39
  evolutionary algorithm for, 56
  randomized algorithm for, 49–51
least squares, 14
  breakdown point of, 15
least trimmed squares, 53–60
  breakdown point of, 53
  efficiency of, 54
  evolutionary algorithm for, 56
least trimmed sum of absolute values, 53–60
  breakdown point of, 53
  evolutionary algorithm for, 56
linear model, 13
linear regression, 13–16
  least absolute deviation, 15
  least median of squares, 15
  least quantile of squares, 40
  least quartile difference, 39
  least squares, 14
  least trimmed squares, 53
  least trimmed sum of absolute values, 53
  through the origin, 45
literal, 70
LMS, *see* least median of squares
locus, 63
lower half-space, *see* half-space
LQD, *see* least quartile difference
LQS, *see* least quantile of squares

`lqs`, 58
LS, *see* least squares
LTA, *see* least trimmed sum of absolute values
LTS, *see* least trimmed squares
`ltsReg`, 58

MAD, *see* median absolute deviation
MASS, 58
MCR, *see* misclassification rate
mean, 11
  breakdown point of, 11
  efficiency of, 13
  sample mean, 11
medcouple, 23
median, 12
  breakdown point of, 12
  computation time of, 13, 19, 23, 33
  efficiency of, 13
  of two sets, 33
  of $X + Y$, 23
  sample median, 11, 12
median absolute deviation, 17
  breakdown point of, 17
  efficiency of, 17
minimal polynomial, *see* polynomial
minimum polynomial, 71, 86
misclassification rate, 72, 97
monomial, 71
  length of, 71
multiterminal decision diagram, 95
mutation, 54

nonparametric tests, 66
`NoOfIntersectionsBetween`, 49
nucleotides, 63
null hypothesis, 66

OBDD, *see* ordered binary decision diagram
online computation, 19
  of Q_n, 24–28
  of S_n, 34–38
open half-space, *see* half-space
order statistic, 11, 12
ordered binary decision diagram, 98
ordered decision diagram, 97

p-value, 66
parametric search, 44, 45
pareto optimal, 72
phenotype, 54
plane sweep algorithm, 45
polymorphisms, 63
polynomial, 71
  length of, 71
  minimal polynomial, 71
population, 54
primal space, 40
procedures
  `Delete` (for Q_n), 27
  `Delete` (for S_n), 37
  `Insert` (for Q_n), 26
  `Insert` (for S_n), 36
  `NoOfIntersectionsBetween`, 49
  `SearchLocalSolution`, 46
  `SearchMostCommonLiteral`, 75
PROGRESS, 54

Q_n, 19, 20, 22–32
  breakdown point of, 19
  computation time of, 19, 22, 27
  efficiency of, 19
  offline computation of, 23
  online computation of, 24–28
quartile, 52

R, 51
recombination, *see* crossover
regression through the origin, *see* linear regression
repeated median, 34
replacement breakdown point, *see* breakdown point
residual, 14
residual difference, 39

response variable, 13
returns, 31
RFreak, 58
robfilter, 51
`robreg.evol`, 58
robustbase, 58

S_n, 19, 33–38
  breakdown point of, 19
  computation time of, 19, 33, 38
  efficiency of, 19
  offline computation of, 33
  online computation of, 34–38
SAM, *see* significance analysis of microarrays
sample mean, *see* mean
sample median, *see* median
sample standard deviation, 16
  breakdown point of, 17
`SearchLocalSolution`, 46
`SearchMostCommonLiteral`, 75
selection in $X + Y$, 22
  computation time of, 22
signal, 20
significance analysis of microarrays, 84
single nucleotide polymorphism, 63–64
SNP, *see* single nucleotide polymorphism
standard GP, 88
statistical inference, 65
stopping criterion, 55
subset algorithm, 39
sweep line, 45
symbolic expression, 65

T, *see* thymine
test data, 84
threshold rule, 78
thymine, 63
time series, 20
training data, 84

update, 21
upper half-space, *see* half-space

variable ordering, 97
vertical line, 45
volatility, 21

whiskers, 52
Wilcoxon signed rank test, 66