Benchmark Experiments

A Tool for Analyzing Statistical Learning Algorithms

Manuel J. A. Eugster

München 2011

Benchmark Experiments

A Tool for Analyzing Statistical Learning Algorithms

Dissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
am Institut für Statistik
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

Vorgelegt von
Manuel J. A. Eugster
am 25. Januar 2011
in München

Erstgutachter:	Prof. Dr. Friedrich Leisch, LMU München
Zweitgutachter:	Prof. Dr. Achim Zeileis, LFU Innsbruck
Rigorosum:	16. März 2011

Abstract

Benchmark experiments nowadays are the method of choice to evaluate learning algorithms in most research fields with applications related to statistical learning. Benchmark experiments are an empirical tool to analyze statistical learning algorithms on one or more data sets: to compare a set of algorithms, to find the best hyperparameters for an algorithm, or to make a sensitivity analysis of an algorithm. In the main part, this dissertation focus on the comparison of candidate algorithms and introduces a comprehensive toolbox for analyzing such benchmark experiments. A systematic approach is introduced – from exploratory analyses with specialized visualizations (static and interactive) via formal investigations and their interpretation as preference relations through to a consensus order of the algorithms, based on one or more performance measures and data sets. The performance of learning algorithms is determined by data set characteristics, this is common knowledge. Not exactly known is the concrete relationship between characteristics and algorithms. A formal framework on top of benchmark experiments is presented for investigation on this relationship. Furthermore, benchmark experiments are commonly treated as fixed-sample experiments, but their nature is sequential. First thoughts on a sequential framework are presented and its advantages are discussed. Finally, this main part of the dissertation is concluded with a discussion on future research topics in the field of benchmark experiments.

The second part of the dissertation is concerned with archetypal analysis. Archetypal analysis has the aim to represent observations in a data set as convex combinations of a few extremal points. This is used as an analysis approach for benchmark experiments – the identification and interpretation of the extreme performances of candidate algorithms. In turn, benchmark experiments are used to analyze the general framework for archetypal analyses worked out in this second part of the dissertation. Using its generalizability, the weighted and robust archetypal problems are introduced and solved; and in the outlook a generalization towards prototypes is discussed.

The two freely available R packages – benchmark and archetypes – make the introduced methods generally applicable.

Zusammenfassung

Benchmark Experimente können heutzutage als das Standardwerkzeug zur Evaluierung von Lernalgorithmen bezeichnet werden; sie werden in nahezu allen Forschungsbereichen mit Anwendungen im Statistischen Lernen angewandt. Dieses empirische Werkzeug ermöglicht unterschiedlichste Untersuchungen von Lernalgorithmen auf einem oder mehreren Datensätzen: der Vergleich einer Menge von Lernalgorithmen, das Finden der besten Hyperparameter für einen Algorithmus, oder eine Sensitivitätsanalyse eines Algorithmus. Fokus des Hauptteils dieser Dissertation liegt auf dem Vergleich mehreren Algorithmen, und es wird ein umfassender Werkzeugkasten zur Analyse vorgestellt. Die Arbeit führt eine systematische Vorgehensweise ein – ausgehend von explorativen Untersuchungen mit spezialisierten Visualisierungen (statisch und interaktiv), über formale Auswertungen und deren Interpretation als Präferenzrelation, bis hin zu einer Konsensusordnung der Lernalgorithmen basierend auf einem oder mehreren Performanzmassen und Datensätzen. Die Performanz von Algorithmen wird von den Eigenschaften eines Datensatzes bestimmt, das ist weitgehend bekannt. Nicht genau bekannt ist jedoch der konkrete Zusammenhang zwischen den Datensatzeigenschaften und den Algorithmen. Aufbauend auf Benchmark Experimenten wird eine Methodik zur Untersuchung solcher Zusammenhänge vorgestellt. Des Weiteren werden Benchmark Experimente als Experimente mit fixierter Anzahl von Replikationen gesehen – Ihre Natur ist jedoch sequentiell. Es werden erste Gedanken zu einer sequentiellen Ausführung vorgestellt und die möglichen Vorteile diskutiert. Abschluss des Hauptteils dieser Dissertation bildet eine Diskussion über mögliche zukünftige Forschungsthemen im Bereich von Benchmark Experimenten.

Der zweite Teil der Dissertation beschäftigt sich mit der Archetypenanalyse. Archetypenanalyse repräsentiert Beobachtungen eines Datensatzes als Konvexkombinationen einiger weniger Extrempunkte. Dieses Konzept wird als eine mögliche Analyse von Benchmark Experimenten vorgestellt – das Finden und das Interpretieren extremen Performanzen der Lernalgorithmen. Im Gegenzug werden Benchmark Experimente verwendet, um die flexible Methodik zu untersuchen, welche in diesem Teil der Dissertation herausgearbeitet wird. Die Flexibilität erlaubt das einfache Erweitern des Archetypenproblems. Die konkreten Erweiterungen und Lösungen von gewichteten und robusten Archetypen werden präsentiert; und im Ausblick wird eine Verallgemeinerung in Richtung Prototypen diskutiert.

Die beiden frei verfügbaren R Pakete – benchmark und archetypes – stellen die vorgestellten Methoden dieser Dissertation allgemein zur Verfügung.

Danksagung

> Was heute noch wie ein Märchen klingt, kann morgen Wirklichkeit sein. Hier ist ein Märchen von übermorgen: ...
>
> *(Raumpatrouille Orion)*

In den vergangenen vier Jahren hat sich meine Dissertation oft wie ein Märchen angefühlt – viele Menschen haben Anteil daran, dass es nicht dabei geblieben ist.

Besonders danken möchte ich Fritz Leisch, mein Doktorvater, der mir diese Dissertation ermöglicht hat; dem ich von Wien nach München folgen konnte und der mir immer als wissenschaftlicher Mentor zur Seite stand, ohne mir Schranken in der Ausrichtung meiner Interessen und Forschungen vorzugeben. Bedanken möchte ich mich auch bei Torsten Hothorn, der viel mit mir zusammengearbeitet hat und in den vergangenen vier Jahren ein wichtiger Gesprächspartner für mich war. Beide haben bei mir einen prägenden Eindruck bezüglich Wissenschaft, Forschung und der universitären Arbeit hinterlassen.

Weiters danke ich Carolin Strobl für die Zusammenarbeit an einem Manuskript das Teil dieser Dissertation ist. Danke auch an Achim Zeileis, dessen Paket einen wichtigen Teil zu diesem Manuskript beisteuert und der dankenswerterweise als Zweitgutachter dieser Dissertation fungiert.

Ich bedanke mich ganz herzlich beim Institut für Statistik, welches einen österreichischen Informatiker aufs freundlichste aufgenommen und integriert hat. Allen voran natürlich Sebastian Kaiser – ohne ihn hätte ich mich nie und nimmer in diesem Dschungel zurechtgefunden. Danke auch an Fabian Scheipl und Nora Fenske für das Beantworten meiner vielen vielen Fragen.

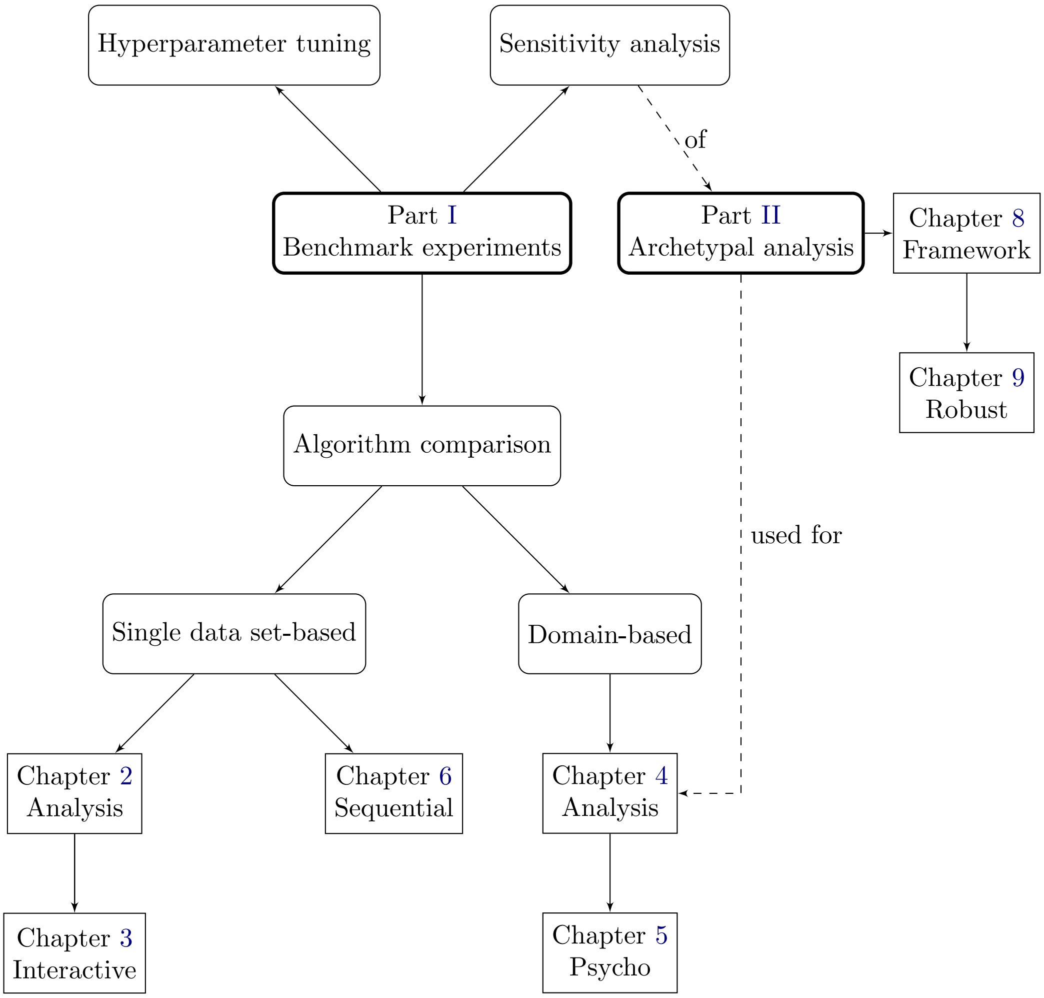
All das wäre nicht möglich ohne die Unterstützung meiner Familie. Meine Eltern Hartwig und Maria und meine Schwester Carola, die seit je her ohne Bedenken hinter all meinen Entscheidungen stehen und mich bedingungslos unterstützen. Widmen möchte ich diese Arbeit meinem Sohn Jakob und meiner Freundin Sarah . . .

> "*Jakob, du hast die zähen letzten Tage dieser Dissertation wieder spannend und aufregend gemacht.*"
>
> "*Sarah, ohne dich wären das hier nur leere Seiten –* ***High Five!***"

Manuel J. A. Eugster München, Januar 2011

Contents

Scope of this work



This dissertation is concerned with benchmark experiments as a tool for analyzing statistical learning algorithms. In general, benchmarking is the process of comparing individual objects which compete in a specific field of activity. Such a comparison is based on numbers computed by performance measures. The benchmark experiment can be then seen as the process where the performances of the objects are assessed. Nowadays benchmarking and benchmark experiments (the terms are often used interchangeably) are widely used in many diverse areas. Popular areas are for example business management (see Camp, 1993, as one of the first publications on benchmarking in this field) and computer science (Jain, 1991, is the classical book on measuring and comparing computer performances). In this dissertation, the objects are statistical learning algorithms, the fields of activity are learning tasks (represented by data sets or data generating processes) and the process is an empirical experiment.

Statistical learning algorithms are algorithms which solve a defined learning task, i.e., which learn structures of interest of a given data set. Learning tasks can be roughly categorized as either supervised or unsupervised (Hastie et al., 2009). In supervised learning, the goal is to predict a response based on an a set of input variables; in unsupervised learning, the goal is to describe associations of input variables. Typical supervised learning tasks are classification and regression; unsupervised learning tasks are for example cluster and archetypal analysis. Statistics and machine learning provide a wide variety of algorithms to solve such learning tasks, but in most cases it is not possible to describe the general behavior of an algorithm for a given learning task analytically. Benchmark experiments provide an empirical alternative. Their principal goal is the assessment of an algorithm's quality with respect to certain performance measures under specific conditions. Hothorn et al. (2005) define a theoretical framework for benchmark experiments; this work uses their formalism and framework as fundament and extends it to a comprehensive toolbox for benchmark experiments.

Basis of each learning task is a data set $\mathfrak{L} = \{z_1, \dots, z_N\}$ with N observations drawn from a (known or unknown) data generating process DGP, written as $\mathfrak{L} \sim DGP$. Candidate algorithm a is a potential problem solver; and $a(\,\cdot \mid \mathfrak{L})$ is the fitted model based on the data set $\mathfrak{L}$. The function $p(a, \mathfrak{L})$ assesses the performance of interest of the fitted model $a(\,\cdot \mid \mathfrak{L})$, that is the performance of algorithm a based on data set $\mathfrak{L}$. Common performance measures are misclassification error in classification tasks or mean squared error in regression tasks; but computation time or memory consumption can be of interest as well. Furthermore, specifically designed performance measure can used to investigate individual characteristics of an algorithm (like estimated parameters or number of iterations until convergence). Since $\mathfrak{L}$ is randomly drawn from DGP, $p(a, \mathfrak{L})$ is a random variable as well and its variability is induced by the variability of the data generating process. In order to estimate the variability, B independent and identically distributed data sets are drawn from DGP and B performances $p(a, \mathfrak{L}^b)$ are measured ($b = 1, \dots, B$). This general framework allows to investigate different aspects of an algorithm. Depending on which settings of the benchmark experiment are seen as the variables of change (i.e., the independent variables) different problems are investigated. The fully specified benchmark experiment can be written as

$$p(a_{\theta_1}, \mathfrak{L}^b) \text{ with } \mathfrak{L}^b \sim DGP_{\theta_2},$$

with θ_1 the hyperparameters (or tuning parameters) of algorithm a, and θ_2 the parameters of the data generating process DGP. Possible problems are then:

Hyperparameter tuning: Investigating a set of hyperparameters θ_1 for an algorithm a on a fixed data generating process DGP.

Sensitivity analysis: Investigating algorithm a when the structure of a data generating process θ_2 is changed.

Algorithm comparison: Comparing a set of candidate algorithms a_k ($k = 1, \dots, K$) on (1) a single data generating process or (2) a set of data generating processes.

This classification conforms to the taxonomy of statistical questions given by Dietterich (1998, cf. Figure 1). For all problems, the inspection of more than one performance measure $p_j(a_{\theta_1}, \mathfrak{L}^b)$ ($j = 1, \dots, J$) allows multicriteria (or multiobjective) investigations. Of course, several problems can be treated in one structured benchmark experiment. Note that the problems of hyperparameter tuning and algorithm comparison are very similar. Both compare a set of models – in the first case the models are fitted by one algorithm with different hyperparameters, while in the second case the models are fitted by a set of algorithms (with given hyperparameters). Furthermore, the individual fitting procedures can contain hyperparameter tuning to find the optimal hyperparameters for the given data set in an algorithm comparison problem. However, in all problem cases a huge amount of benchmark data is generated which has to be analyzed in a statistically correct way.

In this dissertation we focus on the analysis of benchmark experiments used for algorithm comparisons. However, most of the introduced methods are directly usable for hyperparameter tuning as well. Benchmark experiments for sensitivity analyses are used to analyze an unsupervised learning algorithm, archetypal analysis, which in turn is then used to analyze benchmark experiments. Archetypal analysis (first addressed by Cutler and Breiman, 1994) has the aim to represent observations in a multivariate data set as convex combinations of a few, not necessarily observed, extremal points (archetypes). We use this to identify the extreme performances of candidate algorithms. Also, the analysis of the archetypal algorithm serves as an example for a performance measure specifically designed to investigate the algorithm's robustness.

In detail, this dissertation consists of two parts – Part I discusses benchmark experiments with focus on algorithm comparison and Part II discusses archetypal analysis. The individual parts are organized as follows (the chart on page xv illustrates the content).

In Part I, Chapter 1 defines the formal benchmark experiment framework and introduces needed methodology like resampling methods. Chapter 2 focuses on the analysis of single data set-based benchmark experiments. Note that the correct term would be single data generating process-based benchmark experiments. However, in most (real-world) benchmark experiments the data generating process DGP is unknown and a data set $\mathfrak{L}$ determines all information about it; therefore we use these two terms interchangeable if not explicitly specified to simplify readability. In this chapter we present a systematic approach from exploratory analyses with specialized visualizations via formal investigations and their interpretation as preference relations through to a consensus order of the candidate algorithms based on a set of performance measures. To simplify exploratory analyses of a single data-set based benchmark experiments, Chapter 3 introduces an interactive analysis approach. In Chapter 4 we extend the single data set-based approach to a joint analysis for a collection of data sets, a so called problem domain. Specialized visualization methods allow for easy exploration of the huge amount of benchmark data. We use archetypal analysis (see Part II) to characterize the archetypal performances of candidate algorithms within a problem domain. And, analogous to single data sets, we introduce an approach to analyze a domain based on formal inference procedures. This allows, among other things, to compute a statistically correct order relation of the candidate algorithms for a problem domain. It is well known (and empirically shown in the exemplar benchmark experiments) that characteristics of data sets have an influence on the performance of algorithms. In Chapter 5 we develop a formal framework to determine the influence of data set characteristics on the performance of learning algorithms. Up to here, we treated benchmark experiments as fixed-sample experiments (B, the number of drawn data sets, is "somehow" defined). In Chapter 6 we show that the nature of benchmark experiments is actually sequential. We provide first thoughts on the advantages of taking this into account – namely, controlling B – and discuss monitoring and decision-making aspects using sequential and adaptive designs. Part I of this work is concluded with Chapter 7 where we summarize the main findings. We close with a discussion of future work in the field of benchmark experiments.

In Part II, Chapter 8 introduces the concept of archetypal analysis; it defines the concrete problem, lays out the theoretical foundations and shows the classical algorithm. The chapter defines a methodological (and computational) framework which allows an easy generalization of the archetypal problem. In Chapter 9 we take advantage of this framework and define a weighted and a robust algorithm. Archetypal analysis approximates the convex hull of a data set, therefore outliers have a great influence on the solution. We adapt the original archetypes estimator to be a robust estimator and present the corresponding fitting algorithm. To investigate the algorithm's robustness we define specialized performance measures and perform a benchmark experiment for sensitivity analysis. This part of the dissertation is concluded with an outlook on the generalization of archetypes in Chapter 10.

Finally, Appendix A provides the computational details of this dissertation. Two R packages (R Development Core Team, 2010) make the introduced methodology applicable. Section A.1 provides details on the benchmark package; Section A.2 provides details on the archetypes package. Information for replicating the analyses of this dissertation are given in this chapter as well.

Parts of this dissertation are based on published manuscripts, manuscripts which are currently under review, and freely available software packages:

Part I:

Chapter 2: Eugster and Leisch (2008). Bench plot and mixed effects models: first steps toward a comprehensive benchmark analysis toolbox. *Compstat 2008—Proceedings in Computational Statistics*, pages 299–306, 2008.
Eugster, Hothorn, and Leisch (2010a). Exploratory and inferential analysis of benchmark experiments. Under review, 2010.

Chapter 3: Eugster and Leisch (2010). Exploratory analysis of benchmark experiments – an interactive approach. Accepted for publication in *Computational Statistics*, 2010.

Chapter 4: Eugster, Hothorn, and Leisch (2010b). Domain-based benchmark experiments: exploratory and inferential analysis. Under review, 2010.

Chapter 5: Eugster, Leisch, and Strobl (2010c). (Psycho-)analysis of benchmark experiments – a formal framework for investigating the relationship between data sets and learning algorithms. Under review, 2010.

Part II:

Chapter 8: Eugster and Leisch (2009). From Spider-Man to Hero – archetypal analysis in R. *Journal of Statistical Software*, 30(8), pages 1–23, 2009.

Chapter 9: Eugster and Leisch (2011). Weighted and robust archetypal analysis. *Computational Statistics and Data Analysis*, 55(3), pages 1215–1225, 2011.

Appendix A:

Section A.1: Eugster (2011). benchmark: Benchmark Experiments Toolbox. R package version 0.3-2. `http://cran.r-project.org/package=benchmark`.

Section A.2: Eugster (2010). archetypes: Archetypal Analysis. R package version 2.0-2. `http://cran.r-project.org/package=archetypes`.

Part I.

Benchmark experiments

Chapter 1.

Introduction

In statistical learning benchmark experiments are empirical investigations with the aim of comparing and ranking algorithms with respect to certain performance measures. New benchmark experiments are published on almost a daily basis; it is the method of choice to evaluate new learning algorithms in most research fields with applications related to learning algorithms. Selected examples of recently published empirical studies are: Caruana and Niculescu-Mizil (2006), comparing ten supervised learning algorithms using eight performance measures on eleven binary classification problems of different application domains; Martens et al. (2007), comparing support vector machines with three other algorithms as credit scoring models; and Huang et al. (2009), comparing three classification algorithms on genetic data. All three studies have in common that their comparisons and conclusions are based on simple summary statistics of the estimated performances (point estimates like mean or median). Even though these three publications are not randomly selected, they represent a prevalent way of analyzing benchmark experiments. Apparently, only looking at the heavily compacted summary statistics ignores a lot of interesting and primarily important information on the benchmark experiment.

The assessment of the performance of learning algorithms has been addressed in many publications in the last four decades. The estimation of the generalization error using cross-validation started with the pioneering work of Stone (1974). Hand (1986) and McLachlan (1987) are outlines of the developments in the first few years. The article “Ten more years of error rate research” by Schiavo and Hand (2000) is an update of the two surveys and reviews the further improvements in error rate research until the year 2000. Now, another decade later it is common practice to use cross-validation or resampling techniques to derive an estimator for the generalization error. However, surprisingly few publications are available on how to evaluate benchmark experiments beyond the computation of point estimates. One early exception is Dietterich (1998) who suggests 5×2 cross-validation with a paired t-test to test two algorithms for their significant difference. Newer exceptions are Demsar (2006) with an extension by Garcia and Herrera (2008), who use common statistical tests to compare classification algorithms over multiple data sets; and Hornik and Meyer (2007) who derive consensus rankings of the learning algorithm based on their performances. In consequence of the absence of evaluation methods, the goal of this dissertation is to present a toolbox of methods which enables a comprehensive and statistically correct analysis of benchmark experiments.

From our point of view the benchmarking process consists of three hierarchical levels: (1) In the Setup level the design and its elements of the benchmark experiment are defined, i.e., data sets, candidate algorithms, performance measures and suitable resampling strategies are declared. (2) In the Execution level the defined setup is executed. Here, computational aspects play a major role; an important example is the parallel computation of the experiment on different computers. (3) And in the Analysis level the computed raw performance measures are analyzed using common and specialized statistical methods. This dissertation is mainly concerned with the analysis level; in what the derivation of a statistically correct order of the candidate algorithms is a major objective. But certainly, the design of a benchmark experiment and its analysis are related – in fact, the chosen design determines the possible analyses methods. A prominent example is that k-fold cross-validation violates the assumption of independent observations (Nadeau and Bengio, 2003), therefore most common statistical methods are not appropriate. We follow the design framework defined by Hothorn et al. (2005) which allows to use standard statistical methods; Section 1.1 introduces the framework detail. An essential point of benchmark experiments and the generalization performance of learning algorithms is the data basis available for the learning task. On this account, Section 1.2 defines in detail the terms “data generating processes”, “data sets”, and “resampling”. Section 1.3 concludes this introductory chapter with an outline of the remaining chapters on the analysis of benchmark experiments.

1.1. Theoretical framework

Hothorn et al. (2005) introduce a theoretical framework for inference problems in benchmark experiments based on well defined distributions of performance measures. They show that standard statistical test procedures can be used to investigate hypothesis of interests. We already sketched parts of the formalism in the section on the scope of this work (see page xvii). We now introduce the framework in detail, which is stepwise extended in the remaining chapters.

The basic design elements of benchmark experiments are the candidate algorithms, a data generating process, a performance measure of interest, and the number of replications. In each replication, a learning sample is drawn from the data generating process. The algorithms are fitted on the learning sample and validated according to the specified performance measure (possibly using a corresponding test sample).

Benchmark experiments are defined as follow (based on Hothorn et al., 2005): Given is a data generating process DGP. We draw $b = 1, \dots, B$ independent and identically distributed learning samples of size n:

$$\begin{aligned} \mathfrak{L}^1 &= \{z_1^1, \dots, z_n^1\} \sim DGP \\ &\vdots \\ \mathfrak{L}^B &= \{z_1^B, \dots, z_n^B\} \sim DGP \end{aligned}$$

We assume that there are $K > 1$ candidate algorithms a_k $(k = 1, \dots, K)$ available for the solution of the underlying learning task. For each algorithm a_k, $a_k(\,\cdot \mid \mathfrak{L}^b)$ is the fitted model based on a learning sample $\mathfrak{L}^b$ $(b = 1, \dots, B)$. This model conditions on the randomly drawn learning sample $\mathfrak{L}^b$ from the data generating process DGP. Therefore, the model itself has a distribution $\mathcal{A}_k$ on the model space of a_k which again depends on the data generating process DGP:

$$a_k(\,\cdot \mid \mathfrak{L}^b) \sim \mathcal{A}_k(DGP),\ k = 1, \dots, K$$

Strictly speaking, the fitted model also depends on the hyperparameters θ_1 of the algorithm (as formalized on page xvii). For algorithm comparison, however, it is reasonable to require that the fitting procedure incorporates both hyperparameter tuning as well as the final model fitting itself. Furthermore, note that we use $a_k(\,\cdot \mid \mathfrak{L}^b)$ as a general abstraction of a fitted model. It encapsulates all its functional details (like fitted parameters, design matrices, or prediction functions). The concrete performance measure of interest determines which details of a fitted model are used.

The performance of the candidate algorithm a_k when provided with the learning samples $\mathfrak{L}^b$ is measured by a scalar function $p(\cdot)$:

$$p_{bk} = p(a_k, \mathfrak{L}^b) \sim \mathcal{P}_k(DGP)$$

The p_{bk} are samples drawn from the distribution $\mathcal{P}_k(DGP)$ of the performance measure of the algorithm a_k on the data generating process DGP. The scalar function $p(\cdot)$ is freely definable to represent the performance of interest as a number. Common measures are the misclassification error in classification tasks, the mean squared error in regression tasks, or the Rand index in clustering tasks. Performances related to the computational process like running time or memory consumption of the fitting or prediction procedures can be of interest as well. Furthermore, the function $p(\cdot)$ can be totally specialized and map individual characteristics of an algorithm to numbers. An example is the performance measure we defined to measure and compare the robustness of the archetypal analysis algorithms in Part II of this dissertation.

This work focus on the comparison and ranking of algorithms suitable for supervised learning tasks, primarily classification and regression problems. (See Dolnicar and Leisch, 2010, on how to deal with benchmarking for clustering tasks using the same theoretical framework.) An essential aspect in supervised learning tasks is the estimation of the performance of an algorithm on future observations – the so-called generalization performance. On this account we provide an in-depth discussion on the general structure of common performance measures used in such learning tasks.

In supervised learning each observation $z \in \mathfrak{L}^b$ is of the form $z = (y, x)$ where y denotes the response variable and x describes a vector of input variables (note that for readability we omit the subscript $i = 1, \ldots, n$ for x, y, and z). The aim of a supervised learning task is to construct a prediction function $\hat{y} = a_k(x \mid \mathfrak{L}^b)$ which, based on the input variables x, provides us with information about the unknown response y. The discrepancy between the true response y and the predicted response $\hat{y}$ for an arbitrary observation $z \in \mathfrak{L}^b$ is measured by a scalar loss function $l(y, \hat{y})$. The performance measure is then defined by some functional μ of the loss function's distribution over all observations of learning sample $\mathfrak{L}^b$:

$$p_{bk} = p(a_k, \mathfrak{L}^b) = \mu(l(y, a_k(x \mid \mathfrak{L}^b))) \sim \mathcal{P}_k(DGP)$$

Typical loss functions for classification are the misclassification and the deviance (or cross-entropy). The misclassification error for directly predicted class labels is

$$l(y, \hat{y}) = I(y \neq \hat{y}),$$

with $I(\cdot)$ the indicator function. The deviance for predicted class probabilities $\hat{y}_g$ ($g = 1, \ldots, G$ different classes) is

$$l(y, \hat{y}) = -2 \times \text{log-likelihood} = -2 \sum_{g=1}^{G} I(y = g) \log \hat{y}_g.$$

The absolute error and the squared error are common loss functions for regression. Both measure the difference between the true and the predicted value; in case of the squared error this difference incurs quadratic:

$$l(y, \hat{y}) = (y - \hat{y})^2$$

Reasonable choices for the functional μ are the expectation and the median (in association with absolute loss).

Now, p_{bk} are samples drawn from the theoretical performance distribution $P_k(DGP)$ of algorithm a_k on the data generating process DGP. In most cases we are not able to determine the theoretical performance distribution analytically and we have to approximate it empirically. The learning performance is an obvious first approximation:

$$\hat{p}_{bk} = p(a_k, \mathfrak{L}^b) = \hat{\mu}_{\mathfrak{L}}(l(y, a_k(x \mid \mathfrak{L}^b))) \sim \hat{\mathcal{P}}_k(DGP)$$

$\hat{\mu}_{\mathfrak{L}}$ denotes the empirical functional of the loss function's distribution over all observations of the corresponding learning sample. $\hat{\mathcal{P}}_k$ denotes in this case the algorithm's learning distribution function of the performance measure evaluated using $\mathfrak{L}^b$ ($b = 1, \ldots, B$). Unfortunately learning performance is not a good estimate of the generalization error – as commonly known it would reward overfitting (see, for example, Hastie et al., 2009). An analytic way to approximate the generalization performance is to estimate the optimism of the learning performance and add it. The Akaike and Bayesian information criteria, and the minimum description length approach are examples for such methods (see Hastie et al., 2009, for a discussion on these "in-sample" generalization performances). However, an analytic approximation is not always possible and approaches based on independent test samples are the primary way of approximating the theoretical performance distribution of an algorithm ("extra-sample" generalization performance in the sense of Hastie et al., 2009).

Suppose that independent test samples $\mathfrak{T}^b$ with sufficient numbers of observations are drawn from the data generating process DGP. An estimation of the generalization performance of algorithm a_k learned on learning sample $\mathfrak{L}^b$ is then

$$\hat{p}_{bk} = \hat{p}(a_k, \mathfrak{L}^b) = \mu_{\mathfrak{T}}(l(y, a_k(x \mid \mathfrak{L}^b))) \sim \hat{\mathcal{P}}_k(DGP).$$

$\mu_{\mathfrak{T}}$ is the functional of the loss function's distribution over all observations of the corresponding test sample $\mathfrak{T}^b$. $\hat{\mathcal{P}}_k$ denotes the algorithm's distribution function of the performance measure evaluated using $\mathfrak{T}^b$. Dependent on the data situation the approximation of $\mathcal{P}_k$ by $\hat{\mathcal{P}}_k$ is of different quality; the next section covers common data situations and their consequences in benchmark experiments.

1.2. Data generating process

In the theoretical discussion on the estimation of the generalization performance, we assume that we know the data generating process DGP and can draw sufficient observations. In practical situations, however, this knowledge can be incomplete. Then, the present data situation determines the data generating process and, consequently, the empirical performance distributions $\hat{\mathcal{P}}_k$ ($k = 1, \ldots, K$). In this section we discuss what Hothorn et al. (2005) call a simulation problem (the data generating process is known) and a real-world problem (only one finite data set from the data generating process is available).

We consider a data generating process DGP to be some distribution function where each drawn observation z is distributed according to it. The statistical and machine learning literature then names three samples drawn from such a DGP (e.g., Bishop, 1995; Hastie et al., 2009): a learning sample, a validation sample, and a test sample. The learning sample is used to fit the algorithms; the validation sample is used for hyperparameter tuning; and the test sample is used for assessment of the generalization performances of the algorithms with the final hyperparameters. It is difficult to give a general rule on how to choose the number of observations in each of the three samples; Hastie et al. (2009) state that a typical ratio is that validation and test samples are 50% of the learning sample. In benchmark experiments for algorithm comparison the validation sample is used in an implicit step of the fitting procedure – and in fact is "just" another benchmark experiment for hyperparameter tuning (cf. page xvii) – therefore, we do not consider it in more detail.

If the data generating process is known, a new learning sample $\mathfrak{L}^b$ and a new test sample $\mathfrak{T}^b$ are drawn in each replication $b = 1, \dots, B$. The resulting empirical performance distribution $\hat{\mathcal{P}}_k$ approximates the theoretical performance distribution $\mathcal{P}_k$ with arbitrary precision (by drawing samples with more and more observations). For practical reasons, $\mathfrak{T}^b$ can be also set to one fixed test sample $\mathfrak{T}$, given that $\mathfrak{T}$ is large enough so that resulting deviations are insignificant small.

In most practical applications no precise knowledge about the data generating process is available. Instead a data set $\mathfrak{L} = \{z_1, \dots, z_N\} \sim DGP$ of size N is all the information we have. In this case we use resampling methods to mimic the data generating process. In statistical literature, a variety of resampling methods are available; we focus on the most common ones – bootstrapping (non-parametric) and subsampling – and present strategies to draw learning samples $\mathfrak{L}^b$ and corresponding test samples $\mathfrak{T}^b$:

Bootstrapping: Efron (1979) introduces the bootstrap procedure. A learning sample $\mathfrak{L}^b$ is defined by drawing $n = N$ observations with replacement from the original data set $\mathfrak{L}$ of size N. This means that $\mathfrak{L}^b$ is drawn from the empirical distribution function of $\mathfrak{L}$ and, consequently, the observations are independent and identically distributed.

Subsampling: The subsampling procedure as a valid procedure is mainly traced back to Politis and Romano (1994). A learning sample $\mathfrak{L}^b$ is defined by drawing $n \ll N$ observations without replacement from the original data set $\mathfrak{L}$. $\mathfrak{L}^b$ are then themselves independent and identically distributed samples of a smaller size from the true unknown distribution of the original data set.

For both strategies the corresponding test samples $\mathfrak{T}^b$ can be defined in terms of the out-of-bag observations $\mathfrak{T}^b = \mathfrak{L} \setminus \mathfrak{L}^b$. Out-of-bag observations as test samples lead to non-independent observations of the performance measures but their correlation vanishes as N tends to infinity. Another strategy is to define the test samples $\mathfrak{T}^b$ as a newly drawn bootstrap sample in each replication. Furthermore, note that k-fold cross-validation fits into this framework as well: B is set to k, and the learning and test samples are defined in terms of the k subsamples of the data set $\mathfrak{L}$. But because of the apparently independent samples it is not used in this dissertation.

The different resampling methods have different impacts on the approximation quality of the theoretical performance distribution $\mathcal{P}_k$ by the computed empirical performance distribution $\hat{\mathcal{P}}_k$. Efron (1983) and Efron (1986) are two early publications investigating different error rates systematically. Two of their general findings are that "*cross-validation gives a nearly unbiased estimate of the true error, but often with unacceptable high variability*" and "*the bootstrap gives an estimate of the true error rate with low variability, but with a possible large downward bias*".

1.3. Overview

We now have defined the theoretical foundations of this dissertation. The remaining chapters extend the framework to cover further aspects – leading to a comprehensive benchmark experiments toolbox.

We start with single data set-based benchmark experiments. Simplest benchmark experiment is the comparison of K candidate algorithms on 1 data set according to 1 performance measure using B replications. The execution of such a setup results in $1 \times B \times K \times 1$ raw performance measures $\hat{p}_{bk}$, i.e., K empirical performance distributions $\hat{\mathcal{P}}_k$. In Chapter 2 we propose exploratory data analysis tools and formal inference procedures (parametric and non-parametric) to compare these performance distributions and, among other things, to establish a preference relation. Furthermore, Chapter 2 extends the framework to allow J performance measures $p_j(\cdot)$, $j = 1, \dots, J$. This means we estimate $1 \times B \times K \times J$ raw performance measures $\hat{p}_{bkj}$, i.e., $K \times J$ empirical performance distributions $\hat{\mathcal{P}}_{kj}$. We then propose to establish a preference relation for each performance measure and to aggregate the relations to one global preference relation using consensus methods.

Visualization can help a lot to understand the data created in benchmark experiments. Interactive visualizations can help gain further insights even more; they enable to generate new questions and hypotheses from benchmark data (e.g., from $\hat{p}_{bkj}$) unseen with their static equivalents. In Chapter 3 we extend the exploratory data analysis tools presented in Chapter 2 with interactivity and discuss the advantages.

In Chapter 4 we investigate domain-based benchmark experiments. A problem domain in the sense of this dissertation is a collection of data sets $\mathfrak{L}_m$. A single data set-based experiment is executed for each of the $m = 1, \dots, M$ data sets; the result are $M \times B \times K \times J$ raw performance measures $\hat{p}_{mbkj}$, i.e., $M \times K \times J$

empirical performance distributions $\hat{\mathcal{P}}_{mkj}$. Usually the analysis is done for each data set separately. Chapter 4 extends this single data set-based approach to a joint analysis for the complete collection of data sets. We present specialized visualization methods for easy exploration of the huge amount of benchmark data. Archetypal analysis is used to describe extreme performances within a problem domain. And, we present a parametric method based on mixed-effects models for a formal statistical analysis of the complete problem domain.

It is common knowledge that certain characteristics of data sets – such as linear separability or observation size – determine the performance of learning algorithms. Chapter 5 proposes a formal framework for investigations on this relationship within a problem domain. To realize the interaction between data sets and algorithms, the data sets (in fact their learning samples $\mathfrak{L}_m^b$) are characterized using statistical and information-theoretic measures. The characterizations are juxtaposed to pairwise preferences based on the raw performance measures $\hat{p}_{mbkj}$ (for $J = 1$). The framework then allows to determine the performance ranking of the candidate algorithms on groups of data sets with similar characteristics by means of recursive partitioning Bradley-Terry models.

Chapter 6 provides first thoughts on sequential/adaptive benchmarking. In the previous chapters we used formal inference procedures to compare the candidate algorithms based on the $M \times B \times K \times J$ performance estimates $\hat{p}_{mbkj}$. Now, the framework strictly defines each step of a benchmark experiment – from its setup to its final preference relation of the candidate algorithms – but says nothing about the number of replications B. So, in most benchmark experiments B is a “freely chosen” number and the experiments are considered as fixed-sample experiments. The nature of benchmark experiments, however, is sequential and Chapter 6 elaborates the advantages taking this into account. We use case studies of typical benchmark scenarios to discuss monitoring and decision making aspects.

Each chapter concludes with a summary and an outlook for further improvements concerning the chapter’s topic. The complete benchmark experiment part of this dissertation is concluded in Chapter 7 with thoughts on new directions for benchmarking research. As a short-time goal we want to systematically investigate the effect of different resampling methods on data set characteristics and, consequently, algorithm performances. As a long-time goal we contemplate a grammar of benchmarking embedded in the theory of experimental designs.

The R package benchmark implements most of the methods introduced in this dissertation's part. In Appendix A.1 we explain the design and the concept of the package; and show how to reproduce most of the application examples shown in the dissertation.

Chapter 2.

Analysis of benchmark experiments

This chapter covers the analysis of single data set-based benchmark experiments. It introduces a systematic four step approach to analyze the result of an executed setup as described in Section 1 – with the major goal of a statistically correct order of the candidate algorithms. The four steps are:

1. Visual analysis using (specialized) exploratory tools to formulate hypotheses of interests.

2. Formal investigation of these hypotheses using common statistical inference procedures.

3. Interpretation of the analysis results as preferences, i.e., mathematical (order) relations.

4. Combination of various preferences (based on different performance measures) using consensus methods.

The chapter structure is based on the analysis of real benchmark experiments. This means, we use one exemplar real-world benchmark experiment throughout the chapter; the methods are introduced "from simple to complex"; and for each method its result on the exemplar benchmark experiment is discussed immediately – which often leads to the introduction of further analysis methods. This course of action allows to present new methodology for the analysis of benchmark experiments packaged within an application-oriented guide for practitioner.

We start the analysis of benchmark experiments in Section 2.1 with the "common analysis approach" used by a lot of publications (three exemplar references are named in Chapter 1). The problems of this superficial analysis are discussed and do motivate our proposed systematic approach. Following the principles of exploratory data analysis we start with exploratory tools to formulate hypotheses of interest in Section 2.2. Drawbacks of basic plots are discussed and a newly developed specialized benchmark experiment plot is introduced. Section 2.3 introduces formal tests to inspect hypotheses of interest. The design of benchmark experiments is a random block design, therefore common parametric and non-parametric tests can be applied. In this chapter we propose the Friedman test and Wilcoxon-Nemenyi-McDonald-Thompson test as non-parametric methods and model the experiment using the parametric mixed-effects models. In this context, the problem of significance versus relevance is discussed as well. In further consequence we interpret the analysis results – point estimates or pairwise test results – as preferences, i.e., mathematical (order) relations, which is discussed in Section 2.4. So far, all presented methods cover one aspect of a benchmark experiment, i.e., they result in a statistically correct order concerning one performance measure of the candidate algorithms. In practical situations one is often interested in different behaviors of the candidate algorithms (multicriteria or multiobjective optimization) and more than one performance measure are computed (for example prediction error and computation time). Therefore, suitable methods to combine different preferences are needed; Section 2.5 discusses consensus decision-making methods which allow such combinations. The chapter is concluded with a summary and an outlook for further developments in Section 2.6.

Example. An exemplar benchmark experiment demonstrates our methods throughout the chapter. Note that the primary goal of this example is to illustrate general interesting aspects of benchmark experiments; we are not necessarily using up-to-date classifiers. Data set and candidate algorithms are arbitrarily replaceable.

The learning problem is the binary classification problem `mnk3` from the UCI Machine Learning repository (Asuncion and Newman, 2007). It consists of 6 nominal attributes and 554 observations. The used candidate algorithms are linear discriminant analysis (`lda`, purple 🟪), k-nearest neighbor classifier (`knn`, yellow 🟨), classification trees (`rpart`, red 🟥), support vector machines (`svm`, blue 🟦), neural networks (`nnet`, green 🟩); see all, e.g., Venables and Ripley (2002); Hastie et al. (2009). As representative of ensemble and bagging methods, respectively, we use random forests (`rf`, orange 🟧; Breiman, 2001). Misclassification error is used as performance measure; the number of bootstrap samples B is 250. The execution of this benchmark experiment results in 250 misclassification measures per candidate algorithm. These measures are the estimated empirical misclassification distributions $\hat{\mathcal{P}}_k(\texttt{mnk3})$ of each candidate algorithm on data set `mnk3`. They build the basis for the comparison of the algorithms right up to the arrangement of an order relation. Details on the concrete computation (software, algorithms, hyperparameter tuning) are available in the Appendix A.

2.1. Common analysis

Common analyses of benchmark experiments consist of the comparison of the empirical performance measure distributions based on some summary statistics (point estimations): algorithm a_k is better than algorithm $a_{k'}$ iff $\phi(\hat{\mathcal{P}}_k) < \phi(\hat{\mathcal{P}}_{k'})$. ϕ is a scalar functional and for example a measure of central tendency, statistical dispersion, or shape. Depending on the functional, different conclusions on the algorithms' performances are possible, i.e., conclusions concerning their behavior in best- ($\phi = \text{Min}$), worst- ($\phi = \text{Max}$) and average- ($\phi = \text{Mean or Median}$) case scenarios.

In some cases, confidence intervals are calculated to indicate the significance of differences. Efron and Tibshirani (1993), for example, provide different methods for bootstrap confidence intervals; simplest estimates are the percentile intervals $[l, u]$ with l the $B * \alpha$ and u the $B * (1 - \alpha)$ percentile.

Example (cont.). Table 2.1 shows the most established summary statistics for performance estimates. Looking at the average-case scenario based on the mean performance values (column Mean), the order of the candidate algorithms is:

$$\texttt{svm} < \texttt{rpart} < \texttt{rf} < \texttt{nnet} < \texttt{knn} < \texttt{lda}$$

However, the corresponding confidence intervals of all algorithms intersect, i.e., their performance differences are not significant (Figure 2.1). This indicates that no strict total order $<$ or equivalence relation $=$ can be defined. In Section 2.4 we discuss how to express this circumstance.

Using the maximal performance values (column Max) one can apply the minimax rule for minimizing the maximum possible loss:

$$\texttt{svm} < \texttt{lda} = \texttt{rpart} < \texttt{rf} < \texttt{nnet} < \texttt{knn}$$

$\phi =$	Mean	SD	Median	Max
`lda`	0.0352	0.0094	0.0350	0.0561
`knn`	0.0344	0.0118	0.0340	0.0707
`nnet`	0.0293	0.0123	0.0273	0.0631
`rf`	0.0197	0.0117	0.0185	0.0567
`rpart`	0.0116	0.0080	0.0100	0.0561
`svm`	0.0110	0.0059	0.0100	0.0340

Table 2.1.: Performance estimations based on common summary statistics ϕ: based on the B bootstrap samples, the mean, standard deviance (SD), median and maximum (Max) values of the empirical misclassification distributions are calculated.

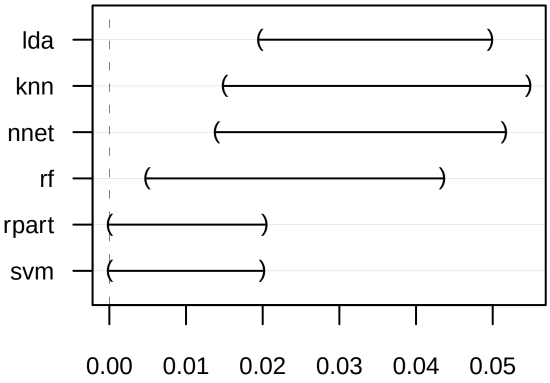


Figure 2.1.: Visualization of the bootstrap percentile intervals: the difference in the performances of two algorithms is significant, if their intervals do not intersect.

One can imagine (and we will also see later on; for example, in Figure 2.2c) that outlier performances are quite likely. To make the minimax-order more robust against outliers, only the mth-worst performance values are used; specifying m, for example, according to the 95% quantiles of the performance distributions (a more sophisticated approach is the calculation of confidence intervals using bootstrapping). With $m = 12$, the mth-worst values are

`lda`	`knn`	`nnet`	`rf`	`rpart`	`svm`
0.0495	0.0545	0.051	0.0432	0.0202	0.02

and the corresponding minimax-order is:

$$\texttt{svm} < \texttt{rpart} < \texttt{rf} < \texttt{lda} < \texttt{nnet} < \texttt{knn}$$

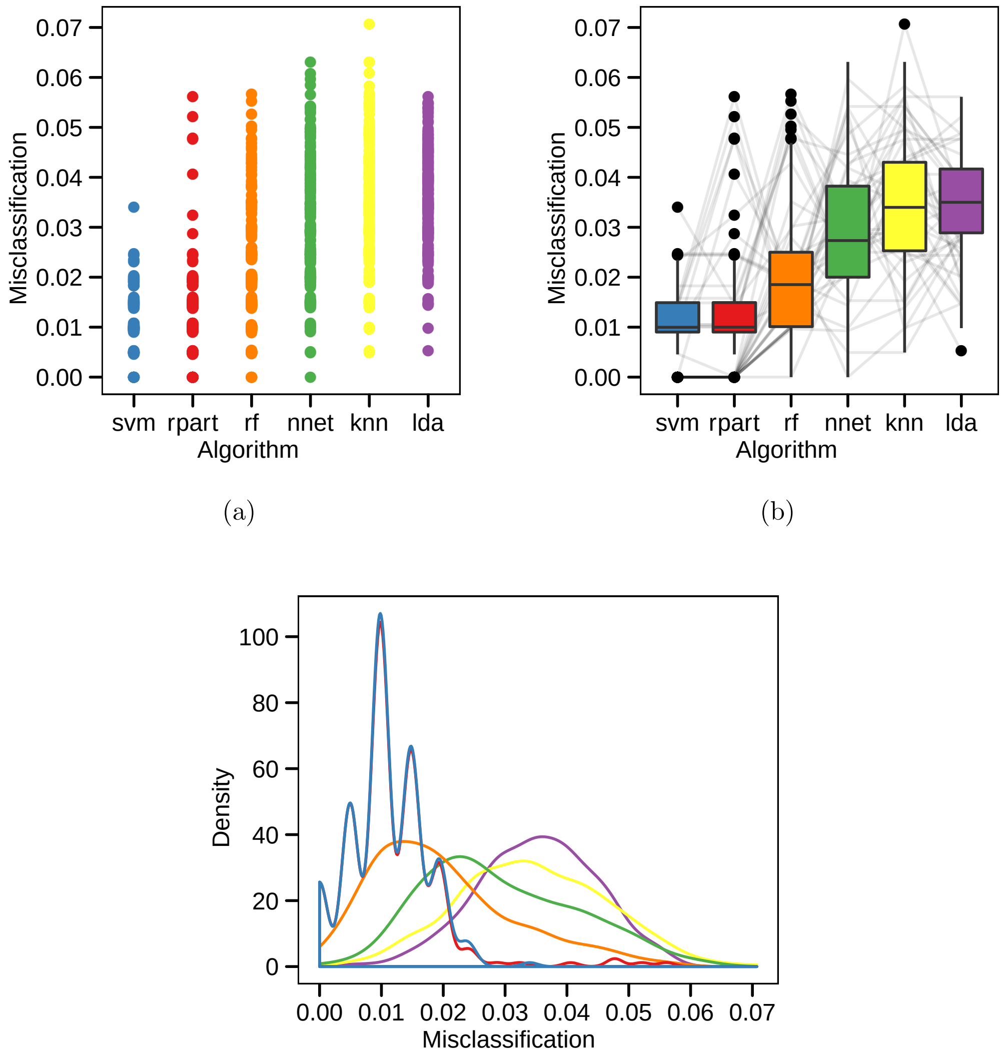
2.2. Exploratory analysis

In many cases, analyses based on the heavily compacted numbers of Table 2.1 are the only source for an algorithms' ranking. But in doing so, one loses a lot of interesting and important information about the experiment. Therefore, the first step of our proposed systematic approach is the usage of visualizations. Strip plots (with the algorithms on the x-axis and their performances on the y-axis, represented with a dot for each benchmark replication) allow the examination of the raw performances distributions. Box plots summarize the performance distributions and allow the identification of outliers. Histograms and density estimates as estimations of the unobservable underlying performance density function.

Example (cont.). Figures 2.2a, 2.2b and 2.2c show the strip, box and density plot, respectively. Looking at the strip and density plots, it can be seen that the distributions for `svm` and `rpart` are not only skewed, but also multimodal. The algorithms often get stuck in local minima. The box plot supports the assumption of skewed distributions as both medians are approximately equal to the first quantiles. `rf` shows similar patterns to `svm` and `rpart` at the lower end, but the median is near to the middle of the box. `nnet` seems to be slightly skewed, all other algorithms seems to be unimodal.

The figures also allow a first impression of the overall order of the algorithms. `svm` and `rpart` basically have the same performance, the small differences in their mean performance are mostly caused by a few outliers. Their performances define the lower misclassification range from 0 to 0.02. In this range, all algorithms find local minima with similar performance values – they present the same patterns. `rf`, `nnet`, `knn` and `lda` scatter over the complete range, whereas `knn` and `lda` perform similar. `nnet` scatters most; it has outliers close to the best global and also results near to the worst global performance. The worst performance is defined by `knn`.

One massive problem of the strip plot is the overlapping of dots. For example, the impression of the order between `lda` and `knn`: It seems that `lda` is better than `knn`, but if we take a look at the summary statistics in Table 2.1, we see that the mean and median performances of `knn` are slightly better. Additionally, the standard strip plot claims the independence of the bootstrap samples. Indeed we know that, for example, `svm` and `rpart` perform similar over all benchmark replications, but we do not know their ranking per bench-



(a)

(b)

(c)

Figure 2.2.: Basic plots; for better comparability are the algorithms sorted according to their mean performance. (a) Strip plot: the performance of each algorithm on each benchmark replication is shown as a dot. (b) Box plot: the performance of each algorithm is aggregated by the five-number summary with outliers identified. (c) Density plot: approximation of the empirical performance distribution functions of each algorithm.

mark replication (i.e., which algorithm is on which rank and how often). One possibility to avoid the first problem, is the usage of box and density plots. As we can see in Figure 2.2b and 2.2c, the impression of the order is correct. Another possibility is to jitter the dot plots, i.e., adding some random noise to the data. But they do not solve the problem of the claimed independence. The benchmark experiment plot (beplot) was developed to overcome these limitations and to get a better understanding of benchmark experiments.

Benchmark experiment plot. Instead of random jittering, we use the ranks of the algorithms on each learning sample to horizontally stretch out the dots. For each benchmark replication, the algorithms are ordered according to their performance value: r_{bk} denotes the rank of $\hat{p}_{bk}$ in the joint ranking of $\hat{p}_{b1}, \ldots, \hat{p}_{bK}$, ties are broken at random. We draw separate dot plots for each rank. This can be seen as creating a podium with K places, and having separate dot plots for each podium place. The following pseudo code outlines the calculation of the benchmark experiment plot podium:

```
Input: $\hat{p}_{bk}$ = matrix of performance values with $K$ columns and $B$ rows;
Output: $w^k_{bk}$ = list of $K$ podium places: each place is a matrix with $K$
        columns and $B$ rows;
for $b = 1 \ldots B$ do
    for $k = 1 \ldots K$ do
        $r_{bk}$ = rank of $\hat{p}_{bk}$ in the joint ranking of $\hat{p}_{b1}, \ldots, \hat{p}_{bK}$, ties are broken at
        random;
        $w^{r_{bk}}_{bk} = \hat{p}_{bk}$;
```

Additionally, a bar plot is shown for each podium to overcome the overlapping of the dots and to show the proportion of the algorithm in the specific podium place.

The dots, i.e., the performance measures, in the displayed plots are not independent from each other because all algorithms are evaluated on each learning sample. This dependency can be displayed by extending the benchmark experiment plot and connecting all dots corresponding to one learning sample with a line, resulting in a modified version of a parallel coordinates plot.

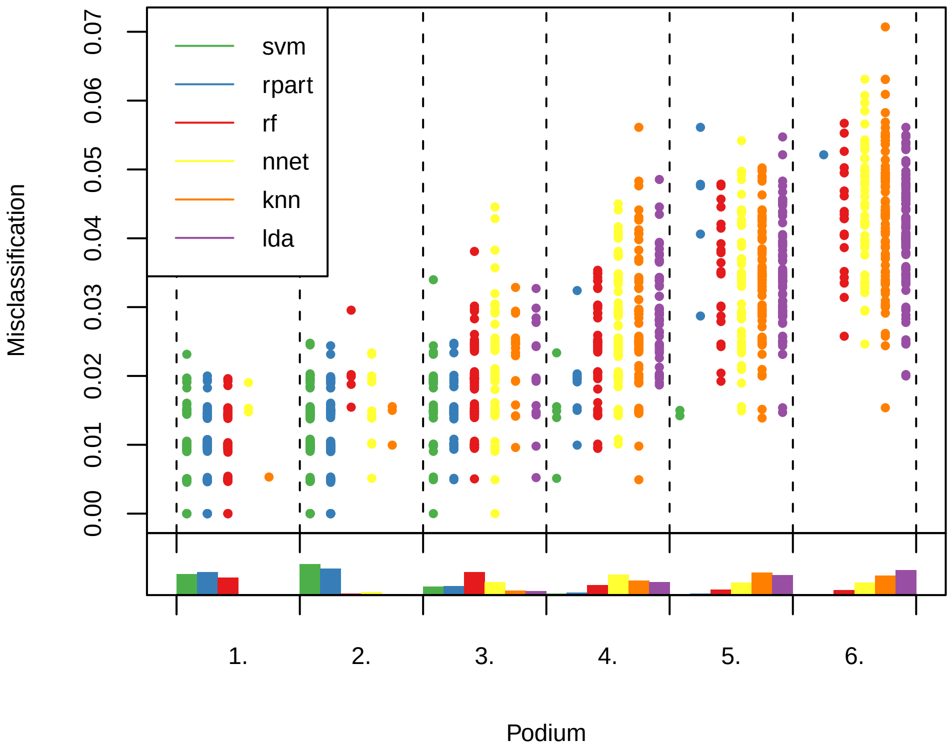


Figure 2.3.: Benchmark experiment plot of the example: the x-axis is a podium with 6 places. For each benchmark replication, the algorithms are sorted according to their performance values and a dot is drawn on the corresponding place. To visualize the count of an algorithm on a specific position, a bar plot is shown for each of podium places.

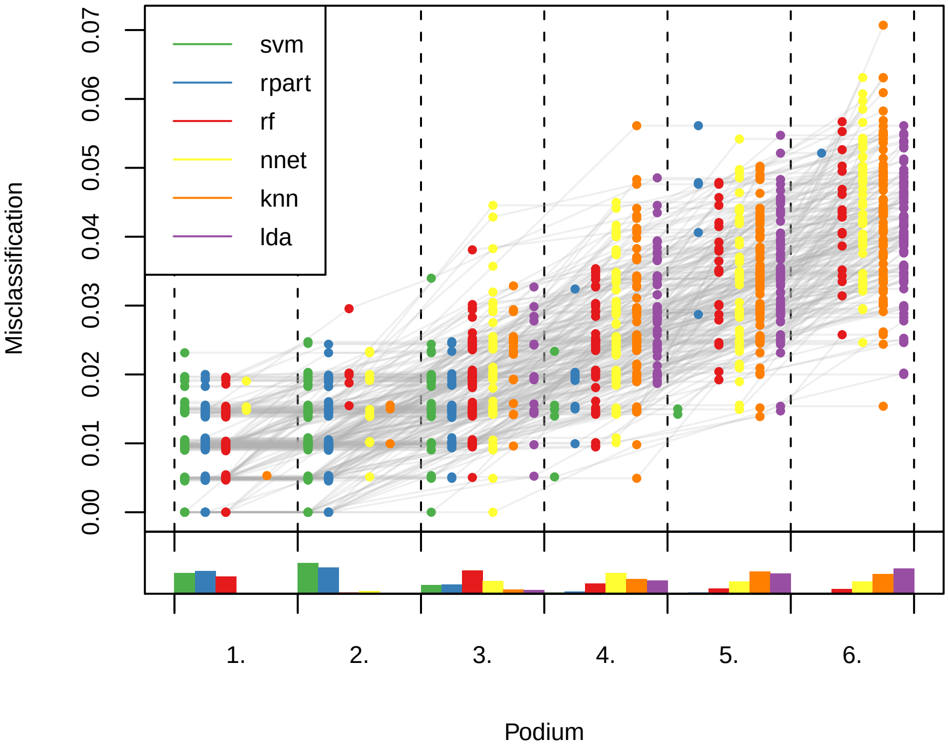


Figure 2.4.: Benchmark experiment plot: the beplot is extended to represent the dependency of the dots of one bootstrap sample with a line between them. We use transparency (alpha shading) to overcome the problem of overlapping lines.

Example (cont.). Figure 2.3 shows the benchmark experiment plot. It supports the assumption (given by Table 2.1 and Figure 2.2) that `svm` and `rpart` perform similar, whereby `rpart` has some results on the lower places. Moreover, we see that `rf` is equally often on place 1, but has only a few second and most third places. This is an aspect that is impossible to infer from the marginal distributions of the performance measures alone (whether displayed by dot plots, box plots, histograms, or density estimates). An example where the impression of similar performances is not supported is given in the exemplar benchmark experiment used in Eugster and Leisch (2008). `nnet` has results in all places, but it is clearly the algorithm with most fourth places. `knn` and `lda` perform similar in most cases; whereas `knn` has some performances in the second place.

Figure 2.4 shows the benchmark experiment plot with connected dots. Here, correlations between algorithm performances (parallel vs. crossing lines) are uncovered. For example, `svm`, `rpart` and `rf` find the same local minima on almost all learning samples. Another interesting fact between those algorithms is that whenever `rpart` or `svm` perform best, another algorithm performs equal (nearly nothing but parallel lines); whereas if `rf` performs best it is really the best algorithm (a lot of crossing lines). Further interesting analyses would explore the characteristics of these learning samples and see if there are any noticeable characteristics which lead to this circumstance. Chapter 5 presents a formal framework for investigating the relationship between data sets and candidate algorithms. The analysis of such coherences is simplified with an interactive version of the benchmark experiment plot with brushing techniques; Chapter 3 presents a prototype implementation.

2.3. Statistical inference

The visual analysis of benchmark experiments gives first impressions of the overall order of the algorithms (among other things). Now, to verify these impressions and to derive a statistically correct order and ranking we need formal tools – statistical inference and primarily the testing of hypothesis provides them.

The design of a benchmark experiment is a random block design. This type of experiment has two classification factors: the experimental one, for which we want to determine systematic differences, and the blocking one, which represents a known source of variability. In terms of benchmark experiments, the experimental factor is the set of algorithms and the blocking factor is a sample from the learning samples of the data. The random block design is basically modeled by

$$\begin{gathered} p_{bk} = \kappa_0 + \kappa_k + \beta_b + \epsilon_{bk}, \\ b = 1, \dots, B, \; k = 1, \dots (K-1), \end{gathered}$$

with the set of candidate algorithms modeled as κ_k, and the sampling modeled as β_b. κ_0 is the intercept (the reference algorithm) and expresses a baseline performance; the κ_k are then the individual differences from this basic performance. We investigate the null hypothesis of no algorithm differences,

$$\begin{aligned} H_0 &: \; \kappa_1 = \cdots = \kappa_{K-1} = 0, \\ H_A &: \; \exists k : \; \kappa_k \neq 0. \end{aligned}$$

Below we discuss appropriate non-parametric and parametric procedures to test this null hypothesis. The concrete choice for one of the procedures is based on the properties of the distributions.

2.3.1. Non-parametric procedures

Non-parametric (or distribution-free) tests are procedures which make no or few assumptions about the probability distributions of the data (in contrast to parametric tests; a general introduction gives, e.g., Hollander and Wolfe, 1999). In benchmark experiments, where we draw observations from the unknown real performance distributions of the algorithms, assumptions often seem to be violated; for example, the assumption of normality in case of `svm` and `rpart` in Figure 2.2c. The Friedman test and the Wilcoxon-Nemenyi-McDonald-Thompson test are presented to determine the null hypothesis of no algorithm differences in exactly that case.

Friedman test. A global test of whether there are any differences between the algorithms at all, can be performed using the Friedman test (e.g., Demsar, 2006; Hollander and Wolfe, 1999). This test takes the assumptions $\sum_{b=1}^{B} \beta_b = 0$, $\sum_{k=0}^{K-1} \kappa_k = 0$, ϵ_{bk} are mutually independent, and each one comes from the same continuous distribution.

The Friedman procedure uses the ranks r_{bk} already defined in the section on the benchmark experiment plot, but ties are averaged. Set $R_k = \sum_{b=1}^{B} r_{bk}$, $R_{\cdot k} = \frac{R_k}{B}$, $R_{\cdot\cdot} = \frac{K+1}{2}$ and compute the test statistic

$$S = \frac{12B}{K(K+1)} \sum_{k=1}^{K} (R_{\cdot k} - R_{\cdot\cdot})^2.$$

When H_0 is true, the statistic S has an asymptotic (B tending to infinity) χ^2 distribution based on $K-1$ degrees of freedom. We reject H_0, for a given significance level α, if $S \geq \chi^2_{K-1,\alpha}$. The distribution of the test statistic under the null hypothesis clearly depends on the (mostly) unknown distribution of the data and thus is (mostly) unknown as well. Hence we use permutation tests, where the unknown null distribution is replaced by the conditional null distribution, i.e., the distribution of the test statistic given the observed data (Hothorn et al., 2006).

Example (cont.). In case of our exemplar benchmark experiment the exploratory analysis indicates that there are differences between the algorithms. Using this test, we can formally support that indication. The measures R_k have been calculated as:

`lda`	`knn`	`nnet`	`rf`	`rpart`	`svm`
R_1	R_2	R_3	R_4	R_5	R_6
1260	1233	1062.5	748	482	464.5

The test statistic S is 787.1201 and the p-value is $< 2.2 \times 10^{-16}$ (numerically zero). The test rejects the null hypothesis for all meaningful significance levels. We proceed with a test looking at all pairwise comparisons to find the algorithms which actually differ.

Wilcoxon-Nemenyi-McDonald-Thompson test. This test is based on the Friedman's within-blocks ranks and is designed to make decisions about individual differences between pairs of treatments, i.e., algorithms (Hollander and Wolfe, 1999). The assumptions are the same as for the Friedman test.

Given the R_k, the procedure calculates the $K(K-1)/2$ differences $R_k - R_{k'}$, $k = 1, \ldots (K-1)$, $k' = k, \ldots, K$. At an experiment-wise error rate α, the test then reaches its pairwise decisions – corresponding to each algorithm pair $(\kappa_k, \kappa_{k'})$ – by the criterion

$$\text{Decide} \begin{cases} \kappa_k \neq \kappa_{k'} & \text{if } |R_k - R_{k'}v| \geq r_\alpha, \\ \kappa_k = \kappa_{k'} & \text{otherwise.} \end{cases}$$

The constant r_α is chosen to make the experiment-wise error rate equal to α. As before, the test is used in a permutation procedure (Hothorn et al., 2006).

Example (cont.). The measures $R_k - R_{k'}$ haven been calculated as:

		`nnet`	`svm`	`rpart`	`knn`	`rf`
		R_6	R_5	R_4	R_3	R_2
`lda`	R_1	−198	−796	−778	−27	−512
`rf`	R_2	−314	−284	−266	−485	
`knn`	R_3	−170	−768	−751		
`rpart`	R_4	−580	−18			
`svm`	R_5	−598				

The corresponding p-values are:

		`nnet`	`svm`	`rpart`	`knn`	`rf`
		R_6	R_5	R_4	R_3	R_2
`lda`	R_1	0	0	0	0.9858	0
`rf`	R_2	0	0	0	0	
`knn`	R_3	0	0	0		
`rpart`	R_4	0	0.9981			
`svm`	R_5	0				

The null hypothesis of no difference is rejected for all pairs of candidate algorithms, except (`rpart`, `svm`) and (`lda`, `knn`).

2.3.2. Parametric procedures

Non-parametric tests give answers to the hypothesis of interest but the underlying random block design is not really modeled, i.e., no estimates for the parameters κ_k, β_b and ϵ_{bk} are calculated. At the time of writing, we are not aware of any non-parametric method available for estimating the parameters, however, parametric procedures are. Looking again at `svm` and `rpart` in Figure 2.2c – their distributions can be seen as asymptotically normal distributed. Furthermore, we are able to draw as many random samples B from the performance distribution as required and therefore can rely on the asymptotic normal and large sample theory, respectively. A major advantage compared to the non-parametric methods is the calculation of simultaneous confidence intervals which enables the controlling of the experiment-wise error (Hothorn et al., 2008).

Mixed-effects model. An adequate parametric modeling of the random block design is done using mixed-effects models (e.g., Pinheiro and Bates, 2000). κ_k is a fixed, β_b a random effect; the assumptions are $\beta_b \sim N(0, \sigma_b^2)$ and $\epsilon_{bk} \sim N(0, \sigma^2)$. Hence, we estimate only one parameter σ_b^2 for the effect of the data set. A modeling, by contrast, with the effect of the data set as main effect, would have lead to B parameters.

The most common method to fit linear mixed-effects models is to estimate the "variance components" by the optimization of the restricted maximum likelihood (REML) through EM iterations or through Newton-Raphson iterations (see e.g. Pinheiro and Bates, 2000). The results are the estimated parameters: the variances of the random effects $\hat{\sigma}^2$ (for ϵ_{bk}) and $\hat{\sigma}_b^2$ (for β_b); the performance of the baseline algorithm $\hat{\kappa}_0$ and the performance differences $\hat{\kappa}_k$ $(k = 1, \ldots, (K-1))$ between the baseline algorithm and the remaining algorithms. In addition to model the design and to estimate the parameters, this procedure allows to test the hypothesis of no algorithm differences as well. A global test, whether there are any differences between the algorithms which do not come from the sampling, is performed with ANOVA and the F-test. Pairwise comparisons, i.e., which algorithms actually differ, are done using Tukey contrasts. In addition to the test decisions (the adjusted p-values) this approach allows the calculation of simultaneous confidence intervals to control the experiment-wise error (we refer to Hothorn et al., 2008, for a detailed explanation).

Example (cont.). The estimates for the parameters have been calculated as

$$\hat{\sigma}_b = 0.0037, \hat{\sigma} = 0.0094$$

and

`lda`	Δ`knn`	Δ`nnet`
$\hat{\kappa}_0$	$\hat{\kappa}_1$	$\hat{\kappa}_2$
0.03522	-0.00078	-0.0059

Δ`rf`	Δ`rpart`	Δ`svm`
$\hat{\kappa}_3$	$\hat{\kappa}_4$	$\hat{\kappa}_5$
-0.01549	-0.02359	-0.02427

where Δ denotes that the estimated value is the difference between the baseline algorithm `lda` and the corresponding algorithm. Model diagnostic is available in the source code for replicating the analysis (see Appendix A).

The global test rejects the null hypothesis that all algorithms have the same performance with a p-value $< 2.2 \times 10^{-16}$. Figure 2.5 shows the 95% family-wise confidence intervals calculated for the pairwise comparisons. The differences between (`lda`, `knn`) and (`rpart`, `svm`) are not significant, the corresponding confidence intervals intersect zero. Note that the result is equivalent to the non-parametric Wilcoxon-Nemenyi-McDonald-Thompson test, and can be taken as an indication for the validity of the assumptions made when applying the linear mixed-effects model.

2.3.3. Significance versus relevance

Statistical significance does not imply a practically relevant discrepancy. As commonly known, the degree of significance can be affected by drawing more or fewer samples. A possibility to control this characteristic of benchmark experiments is to define and to quantify "how large a significant difference has to be to be relevant". General test methods which consider relevance are equivalence tests (e.g., Wellek, 2003). Here, the null hypothesis states that

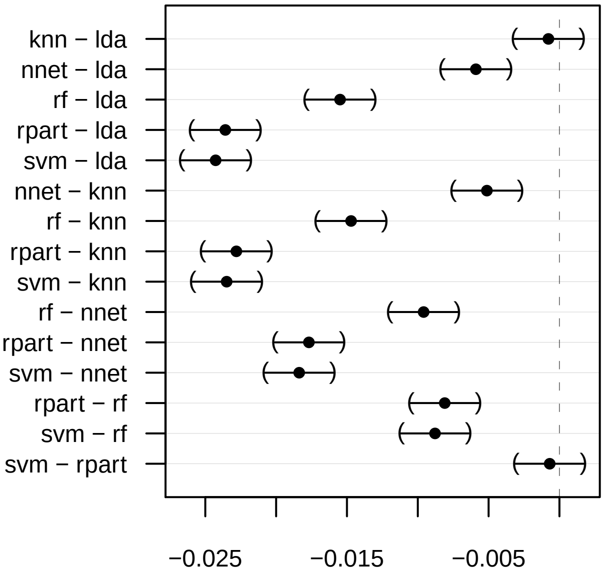


Figure 2.5.: Simultaneous 95% confidence intervals for multiple comparisons of means using Tukey contrasts based on the mixed-effects model of the example experiment.

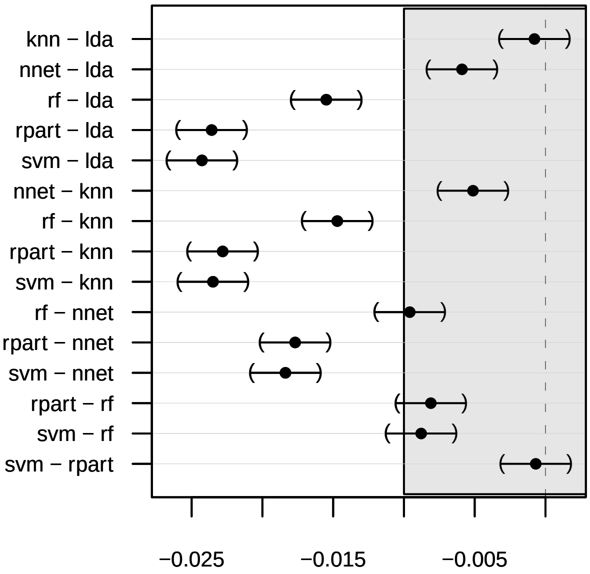


Figure 2.6.: Simultaneous 95% confidence intervals for multiple comparisons extended with the area of equivalence gray-highlighted between $[-0.01, 0.01]$.

the groups differ more than a tolerably small amount. The two one-sided test (TOST), the most basic form of equivalence tests, compares two groups:

$$\begin{aligned} H_0 &: \ \Delta < \Delta_1 \vee \Delta > \Delta_2, \\ H_A &: \ \Delta \in [\Delta_1, \Delta_2] \end{aligned}$$

Δ denotes the difference of the two groups, i.e., the algorithms, and $[\Delta_1, \Delta_2]$ denotes the area of equivalence (zone of non-relevance). The null hypothesis is rejected if the $(1 - \alpha) \cdot 100\%$ confidence interval for the difference between the two groups is completely contained in the area of equivalence. Thus, if the null hypothesis is rejected, the groups differ less than the tolerably small amount $|\Delta_1 - \Delta_2|$; one can state that the groups are similar, a possibly significant difference is not relevant. Of course, the definition of the area of equivalence contains the subjective view of the practitioner; normally, it is based on some domain-specific knowledge.

Example (cont.). Let $[-0.01, 0.01]$ be the area of equivalence. Figure 2.6 shows the 95% family-wise confidence intervals with this area gray-highlighted. All confidence intervals which are completely inside the area are classified as algorithms with non-relevant differences: (`lda`, `knn`), (`nnet`, `lda`), (`nnet`, `knn`) and (`rpart`, `svm`).

2.4. Preference relations

A major goal of benchmark experiments is the determination of an order or ranking of the candidate algorithms. We propose to interpret the analysis results as the data set's preferences as to the candidate algorithms. The notion of preference has a central role in many disciplines of social sciences, like economics or psychology. The general idea is to model choices between alternatives using the formal framework of mathematical relations. In economics, for example, the idea is to model preferences of consumers using such relations. In this section we introduce the concepts and notions of preferences and relations needed in case of benchmark experiments. General references are Davey and Priestley (2002) on relations and Luce and Raiffa (1957) on preferences.

The framework of preferences defines two fundamental binary relations: (1) the strict preference $\prec$ which stands for "better", and (2) the indifference preference

$\sim$ which stands for "equal in value to". In case of benchmark experiments the domain of both relations is the set of candidate algorithms $\{a_k \mid k = 1, \dots, K\}$. So, $a_k \prec a_{k'}$ means that algorithm a_k performs better than $a_{k'}$ – the data set prefers algorithm a_k to $a_{k'}$. $a_k \sim a_{k'}$ means that algorithm a_k performs equally to $a_{k'}$ – the data set has no explicit choice between the two algorithms. In benchmark experiments the following properties of the two relations are fulfilled ($k, k' = 1, \dots, K$):

$\sim$: reflexive ($a_k \sim a_k$) and symmetric ($a_k \sim a_{k'} \Rightarrow a_{k'} \sim a_k$).

$\prec$: irreflexive ($a_k \not\prec a_k$) and asymmetric ($a_k \prec a_{k'} \Rightarrow a_{k'} \not\prec a_k$).

Furthermore, completeness ($a_k \prec a_{k'} \vee a_k \sim a_{k'} \vee a_{k'} \prec a_k$) is met as well. The set of binary preferences concerning benchmark experiments are presented as chain of relations using the logical and operator $\wedge$; for example $a_k \prec a_{k'} \wedge a_{k'} \sim a_k$. As notational convention, chains of relations can be contracted; for example $a_k \prec a_{k'} \sim a_k$ abbreviates $a_k \prec a_{k'} \wedge a_{k'} \sim a_k$ (requires transitivity, i.e., $a_k \prec a_{k'} \wedge a_{k'} \prec a_k \Rightarrow a_k \prec a_k$).

Various methods presented in the previous sections provide results interpretable as a data sets' preference to the candidate algorithms. They are roughly divided into pairwise comparisons based on point estimations (for example Mean and Max in Section 2.1) or statistical tests (pairwise tests in Section 2.3). However, all have in common that their pairwise comparisons induce a mathematical relation R which we interpret as preference relation; depending on the kind of relation, "better" or "equal in value to", we set $(a_k \, R \, a_{k'}) \Rightarrow a_k \prec a_{k'}$ or $(a_k \, R \, a_{k'}) \Rightarrow a_k \sim a_{k'}$. The properties of the preference relation (in addition to the ones described above) are defined by the properties of the relation R. In the following we discuss the different induced preference relations, their properties and their interpretation.

2.4.1. Point estimation based preferences

Comparisons based on point estimations induce strict total order relations $<$ (because point estimations are elements of the set of real numbers). Such a relation is transitive ($a_k < a_{k'} \wedge a_{k'} < a_k \Rightarrow a_k < a_k$), trichotomous ($\forall a_k, a_{k'} : a_k < a_{k'} \vee a_k = a_{k'} \vee a_{k'} < a_k$) and a strict weak order, where the associated equivalence is the equality $=$. The induced preference relation (with $< \Rightarrow \prec$ and $= \Rightarrow \sim$) inherits this properties.

Example (cont.). In the example of Section 2.1 we already induced order relations and chains of order relations; now we can embed these relations into the framework of preferences. For example, `mnk3`'s chain of preference for the average-case ($\phi =$ Mean based) is

$$\texttt{svm} \prec \texttt{rpart} \prec \texttt{rf} \prec \texttt{nnet} \prec \texttt{knn} \prec \texttt{lda},$$

and the chain of preference for the worst-case scenario ($\phi =$ Max based) is

$$\texttt{svm} \prec \texttt{lda} \sim \texttt{rpart} \prec \texttt{rf} \prec \texttt{nnet} \prec \texttt{knn}.$$

However, the interpretation of Figure 2.1 as a preference needs the definition of a relation which takes statistical significance into account.

2.4.2. Statistical test based preferences

Decisions based on statistical tests provide two information: (1) whether there is a significant (or relevant) difference between two algorithms (test statistic or p-value) and if there is one, (2) which algorithm is better (direction of the test statistic). Using this information we can define ($k, k' = 1, \dots, K$):

$a_k = a_{k'}$ iff their difference is not significant (relevant). This relation is reflexive and symmetric.

$a_k < a_{k'}$ iff a_k is significantly (or relevantly) better than $a_{k'}$. This relation is irreflexive and asymmetric.

Now, this enables to define the data set's preferences based on statistical tests analogous to preferences based on point estimates (i.e., $< \Rightarrow \prec$ and $= \Rightarrow \sim$).

Example (cont.). In Figure 2.1 all confidence intervals intersect, i.e., their performance differences are not significant. So, based on bootstrap percentile intervals, `mnk3`'s chain of preference is:

$$\texttt{svm} \sim \texttt{rpart} \sim \texttt{rf} \sim \texttt{nnet} \sim \texttt{knn} \sim \texttt{lda}$$

If we are interested in an "equal in value to" preference relation, the result of, for example, the Wilcoxon-Nemenyi-McDonald-Thompson test (for all meaningful significance levels) induces a preference relation with the incidence matrix:

	lda	rf	knn	rpart	svm	nnet
lda	1	0	1	0	0	0
rf	0	1	0	0	0	0
knn	1	0	1	0	0	0
rpart	0	0	0	1	1	0
svm	0	0	0	1	1	0
nnet	0	0	0	0	0	1

In this concrete case the relation is transitive as well (as we can simply validate). So, this is an equivalence relation and the preferences of the data set are partitioned into several equivalence classes: (knn, lda), (nnet), (rf), (rpart, svm).

Using all available information (p-value and direction), the pairwise comparisons based, for example, on mixed-effects models (Figure 2.5 where left or right from the zero line indicates which algorithm is better) induces a relation with the following incidence matrix:

	lda	rf	knn	rpart	svm	nnet
lda	0	0	0	0	0	0
rf	1	0	1	0	0	1
knn	0	0	0	0	0	0
rpart	1	1	1	0	0	1
svm	1	1	1	0	0	1
nnet	1	0	1	0	0	0

This specific relation is per definition irreflexive and asymmetric and, as we can check, transitive and negatively transitive ($a_k \not\prec a_{k'} \wedge a_{k'} \not\prec a_{k''} \Rightarrow a_k \not\prec a_{k''}$). This is called a strict weak order and defines equivalence classes with an order between these classes. Figure 2.7a shows its visualization using a Hasse diagram. The diagram is read from bottom to top, algorithms at the same height are elements of the same equivalence class. mnk3's chain of preference is:

$$\texttt{rpart} \sim \texttt{svm} \prec \texttt{rf} \prec \texttt{nnet} \prec \texttt{knn} \sim \texttt{lda}$$

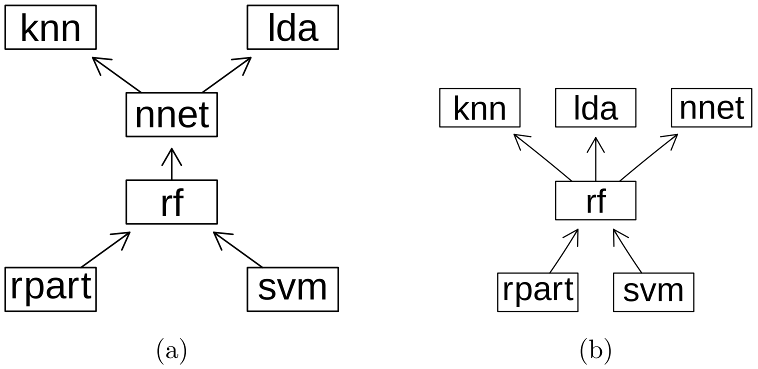


Figure 2.7.: Hasse diagrams representing the preferences given by the pairwise comparisons based on the mixed-effects models, (a) with significant differences, and (b) with relevant differences.

Involving the relevance of differences (Figure 2.6) leads to a strict weak order as well; the Hasse diagram is shown in Figure 2.7b and the chain of preference is:

$$\texttt{rpart} \sim \texttt{svm} \prec \texttt{rf} \prec \texttt{nnet} \sim \texttt{knn} \sim \texttt{lda}$$

The difference between `nnet` and the already existing equivalence class (`lda`, `knn`) is not relevant anymore, now they form one combined equivalence class.

Transitivity. In the exemplar benchmark experiment the resulting preference relations are transitive – this is not true for benchmark experiments in general. One common scenario which violates transitivity is that of no significant differences between algorithms a_k and $a_{k'}$ as well as $a_{k'}$ and $a_{k''}$ but a significant difference between a_k and $a_{k''}$. Transitivity (or intransitivity) depends on many factors, e.g., the number of bootstrap samples B and the shape of the performance distributions (e.g., uni- or multimodal) – both factors are transitivity-friendly in case of the exemplar benchmark experiment. Without transitivity, the establishment of an order is not possible (and no representation as abbreviated chain of relations), however, further analyses are possible just as well. So the next step is to look at preferences based on different behaviors (performance measures) of the candidate algorithms and combine them in a suitable way.

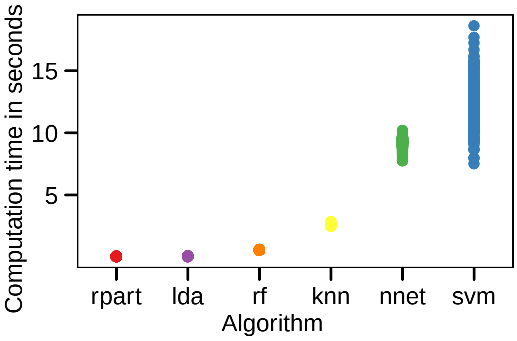


Figure 2.8.: Computation time of the candidate algorithms in seconds. The benchmark experiment replication on a workstation with a AMD Sempron 3400+ (2.00 gigahertz) processor and 1 gigabyte main memory.

2.5. Preference combination

In practical situations one can be interested in the evaluation of the candidate algorithms with respect to more than one performance measure. This problem is known as multicriteria or multiobjective optimization and literature provides various approaches for solving such decision problems (see, e.g., Ehrgott, 2005). Here, we propose the consensus decision-making to combine preferences based on different performance measures.

In Chapter 1 the setup of a benchmark experiment is defined with one performance measure. We extend the setup and assume that there are $j = 1, \dots, J$ performance measures computed by scalar functions $p_j(\cdot)$:

$$p_{bkj} = p_j(a_k, \mathfrak{L}^b) \sim \mathcal{P}_{kj} = \mathcal{P}_{kj}(DGP)$$

In further consequence J empirical performance distributions $\hat{\mathcal{P}}_{kj}(DGP)$ are estimated, and an analysis analogous to sections 2.1, 2.2 and 2.3 is done for each one. The result is an ensemble of J' preference relations $\mathcal{R} = \{R_1, \dots, R_{J'}\}$. J' relations because more than one relation can be established from one performance measure; see, for example, the mean and the worst-case relations based on the misclassification error. The individual preferences describe the behavior of the candidate algorithms according to one criterion at a time, this section discusses a method to combine the individual preferences to one global, multicriteria optimized, preference relation.

Example (cont.). Second performance measure of the exemplar benchmark experiment is the computation time, defined as the sum of the algorithm's fit and predict time. Figure 2.8 shows a strip plot; the chain of preference based on the mean performances (a point estimation based preference) is:

$$\texttt{rpart} \prec \texttt{lda} \prec \texttt{rf} \prec \texttt{knn} \prec \texttt{nnet} \prec \texttt{svm}$$

In summary the analysis of the exemplar benchmark experiment leads to the following ensemble $\mathcal{R} = \{R_m, R_w, R_c\}$ of interesting preference relations:

1. Based on the misclassification and the pairwise comparisons using a mixed-effects model (computed in Section 2.3):
   $R_m = \texttt{rpart} \sim \texttt{svm} \prec \texttt{rf} \prec \texttt{nnet} \prec \texttt{knn} \sim \texttt{lda}$

2. Based on the misclassification and the minimax rule (computed in Section 2.1):
   $R_w = \texttt{svm} \prec \texttt{lda} \sim \texttt{rpart} \prec \texttt{rf} \prec \texttt{nnet} \prec \texttt{knn}$

3. Based on the mean computation time (computed in Section 2.5):
   $R_c = \texttt{rpart} \prec \texttt{lda} \prec \texttt{rf} \prec \texttt{knn} \prec \texttt{nnet} \prec \texttt{svm}$

The interpretation of the ensemble is: Algorithm `svm` has one of the best misclassification rates, the best worst-case performance, but the worst computation time performance. On the other side, `rpart` has the best misclassification rate too, is second on the worst-case performance, and has the best computation time. The obvious questions are now – what is the overall order of the candidate algorithms and, especially in real applications, which algorithm to choose for further applications? Consensus methods provide a formal approach to answer such questions.

2.5.1. Consensus decision-making

The aggregation of different performance measures can be seen as the aggregation of members' opinions in a group discussing a specific topic. A formal system of group decision-making which is appropriate for our needs is consensus decision-making (see, e.g. Farquharson, 1969). Two widely accepted consensus approaches are the Borda count and Condorcet methods. The Borda count ranks the algorithms according to their total numbers of wins across all paired comparisons. Condorcet methods, on the other hand, only ranks an algorithm in front of another algorithm if the individual wins exceeds the number of losses. Both methods fail for criteria which are assumed to be reasonable requirements of fair voting and decision-making methods (the criteria are defined by Arrow's impossibility theorem, see, Arrow, 1963; Nurmi, 1988). Consequently, the properties of the resulting relations are not clearly defined and further (automatic) processing can be complicated. For this reason we suggest the application of optimization consensus for which Hornik and Meyer (2007) present an exact solver over a suitable ("user" chosen) class of relations.

Optimization consensus. The basis is an ensemble of relations $\mathcal{R} = \{R_1, \ldots, R_{J'}\}$. Now, this approach formalizes the natural idea of describing consensus relations as the ones which optimally represent the ensemble of relations $\mathcal{R}$. It is defined as the minimization problem

$$\sum_{R \in \mathcal{R}} w_R \cdot d(R_{con}, R) \Rightarrow min_{R_{con} \in \mathcal{C}},$$

where w_R is a weighting factor, $\mathcal{C}$ a suitable ("user" chosen) class of possible relations, and d a suitable distance measure.

Kemeny and Snell (1972) show that for order relations there is only one unique distance measure d which satisfies axioms natural for preference relations (see their publication for details). The symmetric difference distance d_Δ is defined as the cardinality of the symmetric difference of the relations, or equivalently, the number of pairs of objects being in exactly one of the two relations ($\oplus$ denotes the logical XOR operator):

$$\begin{aligned} d_\Delta(R_1, R_2) = \#\{(a_k, a_{k'}) \mid \\ & (a_k, a_{k'}) \in R_1 \;\oplus\; (a_k, a_{k'}) \in R_2, \\ & k, k' = 1, \ldots, K\} \end{aligned}$$

The definition of the class $\mathcal{C}$ of possible relations allows to control the properties of the resulting consensus relation. For example, $\mathcal{C} = \{\text{class of partial orders}\}$ results in a consensus relation which is reflexive, antisymmetric, and transitive; $\mathcal{C} = \{\text{class of linear orders}\}$ results in a consensus relation which is complete as well.

Solving the minimization problem based on d_Δ can be done by integer linear programming. Hornik and Meyer (2007) reformulate the problem as a binary program (for the relation incidences) and present an exact solver. The concrete formulation and the solving of this NP complete problem is beyond the scope of this chapter and we refer to the original publication for details.

Example (cont.). The interesting ensemble of relation is $\mathcal{R} = \{R_m, R_w, R_c\}$. The partial order consensus is then

$$R_{con} = \texttt{rpart} \sim \texttt{svm} \prec \texttt{lda} \prec \texttt{rf} \prec \texttt{nnet} \prec \texttt{knn}.$$

So, given the three relations based on different behaviors of the algorithms, `rpart` and `svm` are equivalent; and we can go with both in real applications.

The weights w_R in the definition of the minimization problem enable us to express importance. If, for example, the worst-case scenario is important for a real application we weight the relation R_w higher than the others; and computation time only plays a minor role, we weight the relation R_c lower than the others: $w_{R_m} = 1$, $w_{R_w} = 1.5$, $w_{R_c} = 0.2$. The partial order consensus is then computed as

$$R_{con} = \texttt{svm} \prec \texttt{lda} \sim \texttt{rpart} \prec \texttt{rf} \prec \texttt{nnet} \prec \texttt{knn}.$$

2.6. Summary

This chapter introduces a systematic four step approach to analyze single data set-based benchmark experiments. The four steps are: (1) Visual analysis using basic plots and the specialized benchmark experiment plot (beplot) which allows a "look inside" the benchmark experiment. (2) Inferential analysis using formal procedures like non-parametric Friedman-based tests and parametric mixed-effects models to investigate hypotheses of interests. The advantage of the parametric approach is the calculation of simultaneous confidence intervals, which enables a simple handling of the significance versus relevance problem. (3) Interpretation of the computed pairwise results, on basis of statistical tests or point estimates, as preference relations. (4) Combination of more than one performance measure for a multiobjective treatment of the candidate algorithms using the optimization consensus method.

A natural way for future enhancements is the extension to benchmark experiments based on specific domains of interest, i.e., sets of data sets. Exploratory and inferential tools are needed to handle such experiments and infer knowledge about the behavior of candidate algorithms for a domain of learning problems. Chapter 4 discusses domain-based benchmark experiments.

Further future work contains investigations in all three (abstract) levels of the benchmarking process: Setup, Execution and Analysis. For the Setup level we want to develop a grammar of benchmarking which formalizes the specification of benchmark experiments. In the Execution layer we want to use sequential testing to reduce computation time (Chapter 6), if possible. In the Analysis layer other modeling mechanisms like mixture and hierarchical models may be interesting. Furthermore we want to examine the methodology of benchmarking and see if different elements within the benchmark process causes different results, e.g., is there a significant difference between results of parametric tests against non-parametric tests.

Chapter 3.

Interactive analysis of benchmark experiments

Interactive data visualization has a long tradition within statistics and data analysis. Since the 1960s visualization systems for exploratory data analysis (EDA) with interactivity are developed; see, e.g., Cook and Swayne (2007) for the history of statistical data visualization. Closely related to the visualization of benchmark experiments is exploratory model analysis (EMA), which refers to methods and procedures for exploring the space of models: Unwin et al. (2003) introduce an interactive approach using parallel coordinates for different types of useful plots; and Wickham (2007) extends this approach by describing different levels of data, computed after fitting the models.

This chapter tries to build a bridge between the process of creating a set of models (the benchmark experiment) and further analyses of models using methods of EMA or further analyses of data sets using methods of EDA. In Section 3.1 we introduce the benchmark experiment space by describing the different components a benchmark experiment accumulates and how they are linked together. To illustrate our ideas an example is introduced which is used throughout this chapter. Section 3.2 recapitulates the static version of the benchmark experiment plot (beplot introduced in Section 2.2) and then describes the concept of interactive benchmark analysis using the beplot as central point for the “navigation through” the benchmark experiment space. We present the concept using a prototype implementation within one concrete software environment, but of course the concept is realizable in any software environment which provides extensible interactive graphics. Section 3.3 then illustrates the usage of the interactivity by answering exemplary questions which may arise during the analysis of benchmark experiments. The figures in this example have been left as they appear on screen, i.e., there is no labeling of a plot as any such information can be obtained by directly querying the graphic. It is always difficult to show the full power of interactive software on paper, so all resources are provided as electronic supplements; see Appendix A (note the remarks on the prototype implementation in Section 3.2). Section 3.4 concludes the chapter with an outlook for further developments.

3.1. Benchmark experiment space

A benchmark experiment consists of the following components (this partly anticipates the following chapters): (1) $k = 1, \dots, K$ candidate algorithms a_k we are interested in comparing on (2) $m = 1, \dots, M$ data sets $\mathfrak{L}_m$ according to (3) $j = 1, \dots, J$ performance measures $p(\cdot)$; a (4) resampling strategy to draw $b = 1, \dots, B$ learning samples and corresponding test samples from each data set. In this chapter, we use the bootstrap to obtain learning samples and the out-of-bag observations as test samples.

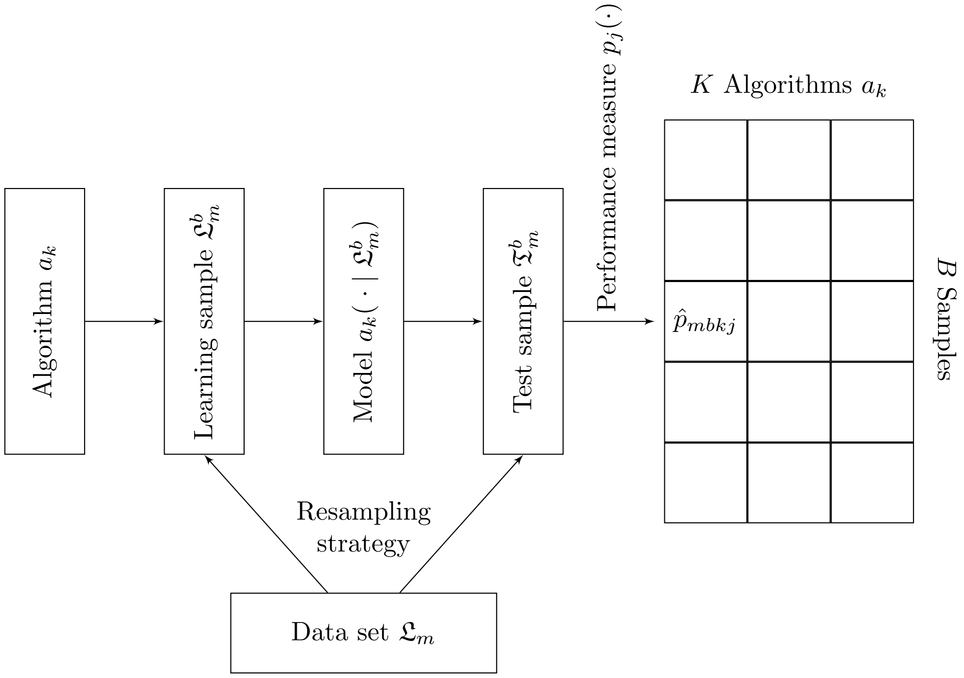

Algorithm a_k
Learning sample $\mathfrak{L}_m^b$
Model $a_k(\cdot \mid \mathfrak{L}_m^b)$
Test sample $\mathfrak{T}_m^b$
Performance measure $p_j(\cdot)$
K Algorithms a_k
$\hat{p}_{mbkj}$
B Samples
Resampling strategy
Data set $\mathfrak{L}_m$

Figure 3.1.: The benchmark experiment components and their relations behind a single performance value. Using the interactive visualization environment, these components can be studied using methods of exploratory data and model analysis (EDA and EMA).

The execution of such a benchmark experiment leads to $M \times B$ learning samples $\mathfrak{L}_m^b$ and test samples $\mathfrak{T}_m^b$; $M \times B \times K$ models $a_k(\,\cdot \mid \mathfrak{L}_m^b)$; and $M \times B \times K \times J$ performance estimates $\hat{p}_{mbkj}$. Therefore, the benchmark experiment space we want to explore consists of the four dimensional benchmark experiment result plus the related components. Figure 3.1 illustrates the relations for one specific performance value $\hat{p}_{mbkj}$ calculated in a benchmark experiment with three algorithms, five learning and test samples, one data set and one performance measure ($K = 3, B = 5, M = 1, J = 1$): a resampling strategy draws a learning and a test sample from a data set; the algorithm creates a model based on the learning sample, and the model's performance according to the performance measure is estimated on the test sample. The extension of the benchmark experiment with more than one performance measure and more than one data set leads to an extension of Figure 3.1 in the third and fourth dimension respectively. Note that all these relations contribute to the final value $\hat{p}_{mbkj}$ and therefore are interesting in the analysis of the benchmark experiment.

Interactive exploratory data analysis provides a simple but powerful way to visualize parts of this high dimensional and complex benchmark experiment space. Each component has one or more visual representations and interaction concepts such as selection, linking and brushing are used to display the relations between them. This chapter presents one way of visualizing some of the components' linking, but of course lots of other ways are possible. A sophisticated gedankenexperiment is the benchmarking of different implementations of the same method with respect to the computation time of different parts (code profiling): the algorithms are represented as source code linked to the different performance measures which are represented as, for example, dot plot; and an interactive action on the dot plot leads to the highlighting of the corresponding source code part.

Exemplar benchmark experiment. The benchmark experiment we use for illustration is a regression problem and constructed as follows: the data set is `cars` (McNeil, 1977), where the speed of cars and the distances taken to stop are provided for cars common in the 1920s. To make our demonstration more interesting, we conducted a new experiment and recorded the speed and stopping distance of a currently common car: 30 mph and 8 ft stopping distance; see Figure 3.2 for the full data set. The candidate algorithms used (with corresponding R functions in parenthesis) are linear regression (`lm`), robust linear regression (`rlm`), cubic smoothing spline (`smooth.spline`, abbreviation is `spl`) and local polynomial regression (`loess`); see all, e.g., Venables and Rip-

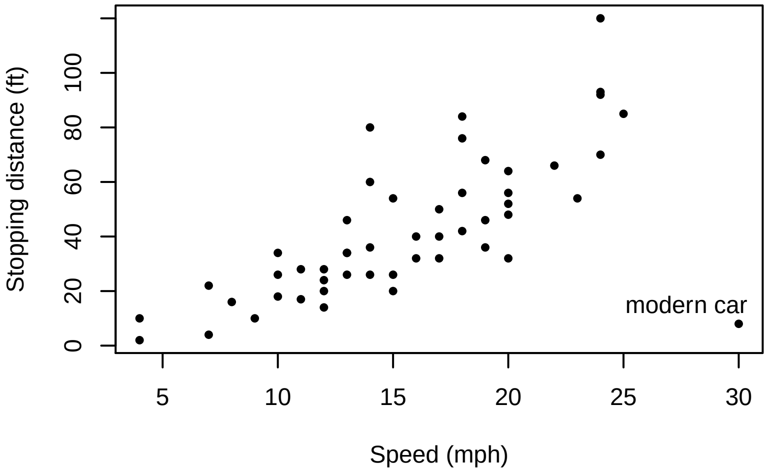


Figure 3.2.: The `cars` data set (McNeil, 1977), which give the speed of cars and the distances taken to stop recorded in the 1920s. The original data set is extended with the data of a currently common car: 30 mph and 8 ft stopping distance.

ley (2002). The performance measures of interest are the prediction mean squared error and the computation time of the model fitting process. $B = 100$ bootstrap samples are drawn as learning samples with the corresponding out-of-bootstrap samples as test samples.

3.2. Interactive environment

The base software for the prototype implementation we rely on is the R statistical environment (R Development Core Team, 2010) with the iPlots package (Urbanek and Wichtrey, 2008), which provides high interaction statistical graphics directly accessible within R. This enables a mixture of command-line driven and interactive analysis which is highly useful for the analysis of benchmark experiments, as we will see later on. iPlots offers a wide variety of plots which all support interactive concepts, such as querying and linked highlighting. In this chapter we explain only the concepts we need, for a full introduction we refer to Urbanek and Theus (2003). The icp package (Gouberman and Urbanek, 2008) extends iPlots and allows the creation of new, fully interactive plots using only pure R code. This environment enables the rapid implementation of the interactive benchmark experiment plot concept and provides enough basic plots to visualize and explore the benchmark experiment space.

At the time of writing the prototype implementation, iPlots 2.0 together with icp appeared to be the most promising environment to realize the concept. With this in mind, we extended the icp package with new functionality (in cooperation with the authors of icp). Unfortunately the development of iPlots 2.0 and icp has been frozen and iPlots is in the process of a complete rewrite (see Urbanek, 2009); currently there is no stable version available. Hence at the time of this writing we are only in the position to present the concept rather than a final implementation of the interactive benchmark experiment plot. By the time a stable and extensible iPlots 3.0 version is available, we will port the prototype and release it in the benchmark package (Appendix A.1). In the meantime, all software needed to reproduce the results in this chapter are available on the homepage of the author of this dissertation.

3.2.1. Benchmark experiment plot

The benchmark experiment plot is a visualization of the benchmark experiment result, more precisely of the $1 \times B \times K \times 1$ part of it. The plot tries to visualize the behavior of the algorithms on the drawn learning and test samples according to one specific performance measure on one data set. Figure 3.3 shows a static version created for the exemplar benchmark experiment. The x-axis is divided into K partitions, where K is the number of candidate algorithms; we call this a "podium". For each bootstrap sample, the algorithms are ranked according to their performance values (with ties broken randomly) and a dot is plotted in the partition corresponding to the algorithm's rank (best performance to the left, worst to the right). Thus a partition corresponds to a rank and not to an algorithm. Within each partition, a further separation occurs: the algorithms are ordered according to their mean performance so that their results can be clearly seen. For example, notice that the dots for the `rlm` algorithm occur in the leftmost position within each partition, and that there is only a single `rlm` dot in the rightmost partition, which tells us that `rlm` had the highest value of MSE for only one bootstrap sample. The bar plots under each podium place show the number of values for each algorithm within each podium place. This is a quick way to note again that `rlm` has many points in the first two podium places and very few in the last two.

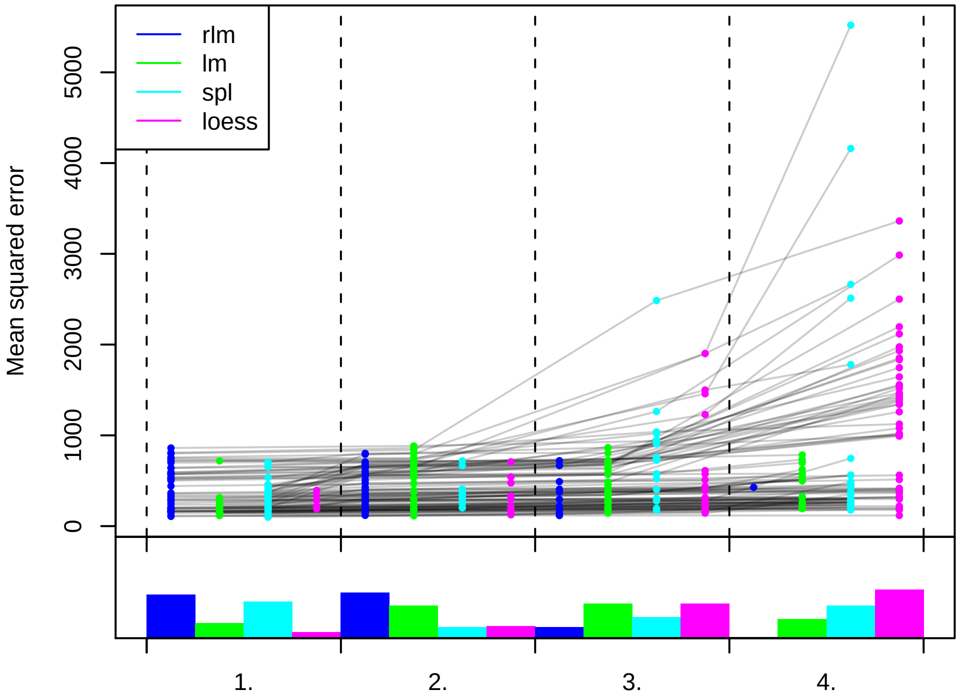
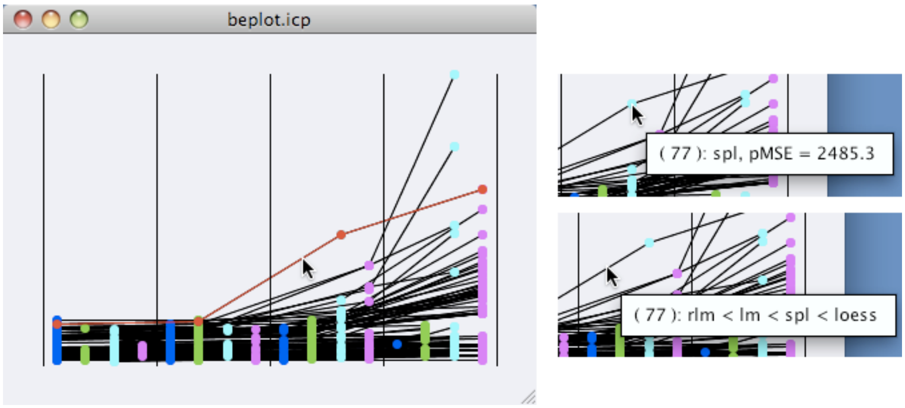


Figure 3.3.: Static version of the benchmark experiment plot; the displayed performance measure is the prediction mean squared error. The algorithms' order in each podium place is (from left to right): `rlm` – blue, `lm` – green, `spl` – cyan, `loess` – magenta. The colors and positions are fixed throughout the chapter; note that the colors are selected to be compatible with the interactive environment.

To further enhance the plot, the dots for each bootstrap sample are connected to indicate their dependency. The lines are colored black and we use transparency (alpha shading) to overcome the problem of overdrawing lines. For example, notice that a lot of lines between the leftmost and second leftmost position are horizontal, which tells us that in these cases the corresponding algorithms perform equally and they are randomly ordered.

3.2.2. Interactive benchmark experiment plot

The interactive version of the benchmark experiment plot only consists of the upper part of the static version, as the bar plots are just another visualization of the same data and can be shown in a separate bar plot linked with the beplot. There are two graphic elements representing interesting content: (1) a dot represents a model by performance measure; (2) a line represents a benchmark replication. A mouse event on any dot or line fires an event which calls an R function. The R function can be anything you choose; here we use them to highlight graphic elements, to show numeric information, and to communicate with other plots. This functionality of individual R functions reacting to an event has been added by us and is not available in the official icp package.



(77): spl, pMSE = 2485.3

(77): rlm < lm < spl < loess

Figure 3.4.: Possible interactions with the interactive bench plot. Mouse-click (left window) on a line highlights the line and all other linked objects. *Control*-mouse-over a dot (right upper window clip) displays information about the model; *Control*-mouse-over a line (right lower window clip) displays information about the benchmark replication.

All in all, the user interface of the benchmark experiment plot prototype strictly follows the user interface convention defined by iPlots 2.0, the interactions are performed using the same keyboard shortcuts. The most important interactions with their default behavior are:

Mouse-click: Highlight the object at the mouse position and all other linked objects (see Figure 3.4, left window).

Control-mouse-over: Show a tooltip with information.

- The default information for a dot is the name and the performance value of the algorithm (Figure 3.4, upper-right window clip).
- The default information for a line is the bootstrap identifier and the order of the algorithms, as this is sometimes hard to determine with all the lines (Figure 3.4, lower-right window clip).

Other functionalities are accessible using the context menu, e.g., hide the lines or step forward and backward through different versions of randomly broken ties.

The benchmark experiment plot, as already mentioned, is a visualization of the $1 \times B \times K \times 1$ part of the benchmark experiment result. Interactivity and the concept of linking now enables an easy way to view more than one performance measure: open one beplot for each performance measure and through the linking, highlighting one object highlights the same object in all other benchmark experiment plots. In the next section we demonstrate this amongst others by means of the exemplar benchmark experiment.

3.3. Interactive analysis

A common exploratory analysis of a benchmark experiment consists of the usage of various basic and specialized plots, shown in Section 2.2. Among others, the benchmark experiment plot of the prediction error (Figure 3.3) is produced. This plot raises, for example, the questions “which one of the $B = 100$ learning samples was used to train the worst `loess` model?” (highlighted in Figure 3.4) or “why is this single blue dot there at the fourth podium place?”. These questions are not easily answerable with the static version, but with the interactive version of the plot.

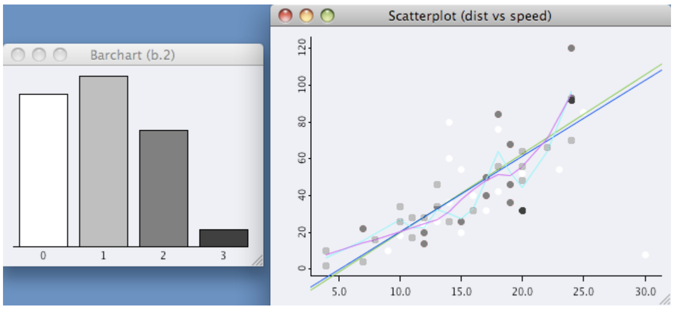


Figure 3.5.: Visualizations of benchmark experiment space components for the benchmark replication highlighted in Figure 3.4: frequency of the observations in the learning sample and the corresponding test sample by means of a bar chart and in combination with a scatter plot showing the data set and the models.

First, we have to decide which additional visual representations of which benchmark experiment space components are useful for this specific analysis: (1) the raw data are visualized using a scatter plot (for higher-dimensional problems one can use a scatter plot matrix and parallel coordinates to show the raw data, or use projection methods). For a specific benchmark replication, (2) the learning sample and the test sample are visualized using a bar chart, with color brushing linked to the data scatter plot, which shows the frequency of the observations in the learning sample (therefore, zero means that the observation is in the test sample); and (3) the models are represented as lines within the data scatter plot. The general color coding is the following: any aspect of a model, e.g., the model's performance in the benchmark experiment plot or the representation of the model within the data, is represented with the color already used in the static benchmark experiment plot (`rlm` – blue 🟦, `lm` – green 🟩, `spl` – cyan 🟦, `loess` – magenta 🟪); highlighted objects are colored in red. All other colors are explained when they appear.

Figure 3.5 shows details of the benchmark replication highlighted in Figure 3.4. By means of the bar chart we see that the test sample consists of 17 observations (white bar); and the learning sample is formed by the 34 remaining observations, whereas 19 observations are drawn once (light gray), 13 observations are drawn twice (middle gray) and 2 observations are drawn thrice (dark gray). In combination with the scatter plot, the frequency per observation and the effect on the models (weighting of the observations) is observable. For example, notice that `spl` and `loess` end with a strong slope because the corresponding observations with high stopping distance are two and three times available. An interesting aspect is, that the "modern car" observation is not

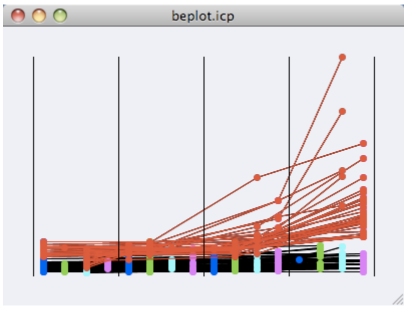
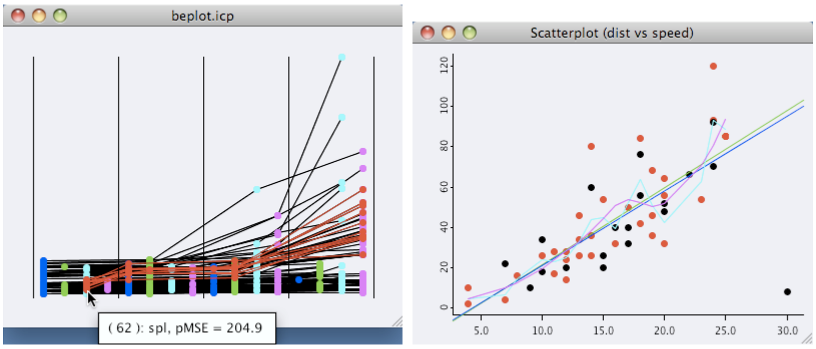


Figure 3.6.: All benchmark replications highlighted where the “modern car” observation is not used in the learning sample; they are all in the upper level of the performance range.

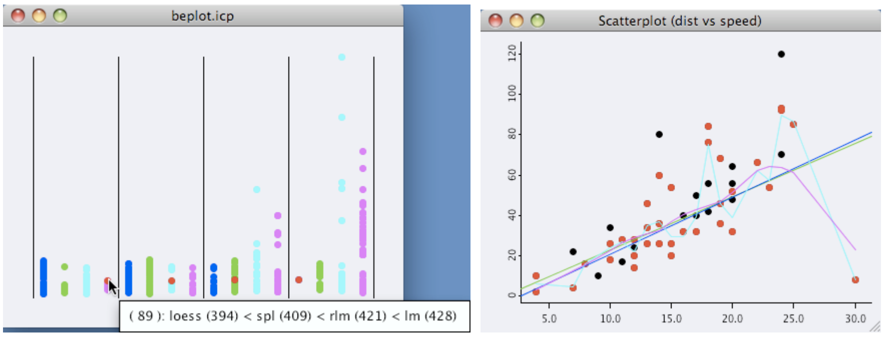
used for learning (its color is white) and the performance of the models are in the upper level of the performance range (as we can see in the benchmark experiment plot). To check this coherence, Figure 3.6 shows the benchmark experiment plot with all benchmark replications highlighted where the “modern car” observation is not in the learning sample – it looks as if the assumed coherence applies.



(62): spl, pMSE = 204.9

Figure 3.7.: All benchmark replications highlighted where the “modern car” observation is not used in the learning sample and the smoothing spline algorithm has a lower performance than all other algorithms in the first partition (left window). Details for the replication with lowest performance of this selection (right window).

The highlighting in Figure 3.6 uncovers that in benchmark replications where the “modern car” observation is not in the learning sample, the smoothing spline algorithm (cyan dots on the third position in the leftmost partition) has the lowest prediction mean squared error. For further investigations on this fact, we refine the highlighting and only select benchmark replications where `spl` has a lower performance than all other models, see Figure 3.7 (left



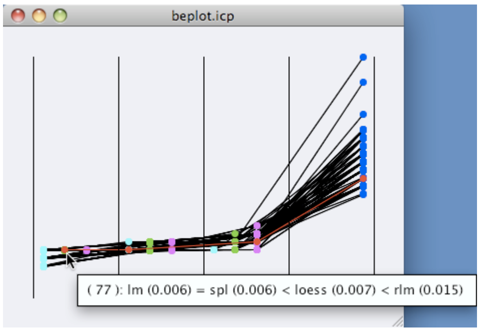
(89): loess (394) < spl (409) < rlm (421) < lm (428)

Figure 3.8.: The benchmark experiment plot with hidden lines and a tooltip with information about the replication where the robust linear regression is in the last partition (left window); and the details (right window).

window). Out of this selection, benchmark replication 62 is the one with the lowest performance; Figure 3.7 (right window) shows its details. In this case the scatter plot displays whether an observation is in any learning sample of the selected benchmark replications (then it is selected, i.e. red colored) or not (not selected, i.e. black colored). The rightmost observation in the learning sample forces the `spl` model (cyan line) to end with a negative slope. And this allows a good prediction of the "modern car" observation compared to all other models: `spl` = 19.1, `lm` = 95.1, `rlm` = 98.2 and `loess` = 192.5. The obvious assumption is that this rightmost observation is in either learning sample of the highlighted benchmark replications – which turns out to be true.

Another aspect which attracts attention in Figure 3.5 is the single blue dot (a `rlm` model) in the rightmost partition. The relatively low value indicates that all models perform well on this benchmark replication, and probably `rlm` is on the last place because of the randomly broken ties. For further investigations we hide the lines and highlight the dots of this specific benchmark replication, see Figure 3.8 (left window). The highlighted dots give the impression that they lie on a straight line, but a tooltip which shows the order and the rounded mean squared error reveals that there are small differences between the model's performances. Figure 3.8 (right window) shows details and allows an explanation: the "modern car" observation is in the learning sample and has a high impact (leverage factor) on the smoothing spline (cyan line) and local polynomial regression (magenta line) models.

The concept of linking enables an easy way to look into more than one performance measure, and investigate their relations. This can be interesting if a single performance measure does not effectively distinguish algorithms



(77): lm (0.006) = spl (0.006) < loess (0.007) < rlm (0.015)

Figure 3.9.: Benchmark experiment plot of the second performance measure, the computation time of the model fitting process; this plot is linked with the benchmark experiment plot in Figure 3.4.

one from another; in that case, a second measure can be used to calculate a global order (see Section 2.5 for theoretical details). In our exemplar benchmark experiment the computation time of the model fitting process is used as second performance measure; Figure 3.9 shows the benchmark experiment plot. This benchmark experiment plot is linked to the benchmark experiment plot in Figure 3.5 (which shows the prediction mean squared error), therefore the same benchmark replication is highlighted. In this specific replication, `lm` (green, second position in each partition) and `spl` (cyan, first position) consume equivalent computation time, `loess` (magenta, third position) consumes nearly as much and `rlm` (blue, fourth position) consumes twice as much. Generally, this plots shows that there is not much difference between `lm` and `loess`, even though the `loess` algorithm fits some (simple) linear models during the model fitting procedure. This indicates that the `loess` implementation is highly optimized.

3.4. Summary

This chapter shows how to integrate interactivity into the process of the exploratory analysis of benchmark experiments with one data set. The interactive version of the benchmark experiment plot allows a rapid exploration of the benchmark experiment result, and of the related components with the concept of linking. We propose that this software should be used as the first step in a chain of exploratory data and model analysis procedures and methods. In that sense, it is a bit like preprocessing the data to highlight the interesting questions.

The presented method is usable for benchmark experiments with one data set or, in case of benchmark experiment with more than one data set, for the individual analysis per data set; this means, analyzing each $m \times B \times K \times J$ part of the benchmark experiment result with its related components individually ($m = 1, \ldots, M$). Linking benchmark experiments plots of two or more different parts is only of limited use because the linked elements do not really relate to each other (except the algorithm components). For example, notice that a benchmark replication b on data set $\mathfrak{L}_m$ has nothing to do with the benchmark replication b on data set $\mathfrak{L}_{m'}$. Chapter 4 introduces some specialized static visualizations for these kind of benchmark experiments; future steps towards a comprehensive interactive benchmark analysis toolbox contains the implementation of their interactive versions.

Chapter 4.

Analysis of domain-based benchmark experiments

In the previous chapters we introduced methods to analyze single data set-based benchmark experiments. Exploratory and inferential methods are used to compare the distributions and to finally set up mathematical (order) relations between the algorithms. The foundations for such a *systematic modus operandi* are defined in Chapter 1; there we review the theoretical framework defined by Hothorn et al. (2005) for inference problems in benchmark experiments. The framework allows to use standard statistical test procedures to test for differences in the performances. The practical toolbox is provided in Chapter 2; there we introduce various analysis methods and define a systematic four step approach from exploratory analyses via formal investigations through to the algorithms' orders based on a set of performance measures.

Modern computer technologies like parallel, grid, and cloud computing (i.e., technologies subsumed by the term High-Performance Computing; see, for example, Hager and Wellein, 2010) now enable researchers to compare sets of candidate algorithms on sets of data sets within a reasonable amount of time. Especially in the Machine Learning community, services like MLcomp (Abernethy and Liang, 2010) and MLdata (Henschel et al., 2010), which provide technical frameworks for computing performances of learning algorithms on a wide range of data sets recently gained popularity. Of course there is no algorithm which is able to outperform all others for all possible data sets, but it still makes a lot of sense to order algorithms for specific problem domains. The typical application scenarios for the latter being which algorithm to deploy in a specific application, or comparing a newly developed algorithm with other algorithms on a well-known domain.

A problem domain in the sense of this dissertation is a collection of data sets. For a benchmark experiment the complete domain or a sample from the domain is available. Note that such a domain might even be indefinitely large, e.g., the domain of all fMRI images of human brains. Naturally, domain-based benchmark experiments produce a "large bunch of raw numbers" and sophisticated analysis methods are needed; in fact, automatisms are required as inspection by hand is not possible any more. This motivation is related to Meta-Learning – predicting algorithm performances for unseen data sets (see for example Pfahringer and Bensusan, 2000; Vilalta and Drissi, 2002). However, we are interested in learning about the algorithms' behaviors on the given problem domain.

This chapter is organized as follows: In Section 4.1 we first extend the theoretical framework of benchmark experiments for more than one data set and review how the (local) single data set-based benchmark experiments have been done. Given the computation of local results for each data set of the domain, Section 4.2 introduces visualization methods to present the results in their entirety. In Section 4.3 we take the design of the domain-based benchmark experiments into account and model it using mixed-effects models. This enables an analysis of the domain based on formal statistical inference. Section 4.4 presents a different approach; in the original sense of benchmarking, we determine extreme performances using archetypal analysis and use them as benchmarks to compare with. In Section 4.5 we demonstrate the methods on two problem domains: The UCI domain (Section 4.5.1) as a well-known domain; useful, for example, when one is developing a new algorithm. The Grasshopper domain (Section 4.5.2) as a black-box domain, where we simply want to find the best learning algorithm for predicting whether a grasshopper species is present or absent in a specific territory. Section 4.6 concludes the article with a summary and future work. All computational details are provided in the Appendix A.

4.1. Design of experiments

The design elements of benchmark experiments are the candidate algorithms, the data sets, the learning samples (and corresponding validation samples) drawn with a resampling scheme from each data set, and the performance measures of interest. In each replication the algorithms are fitted on a learning sample and validated according to the specified performance measures (probably on corresponding validation samples).

Now, in real-world applications we are often interested in the algorithms' performances (e.g., misclassification and computation time) within a domain of problems (e.g., the domain of patients' data in a clinic). A domain is specified with a collection of data sets. In detail, for the candidate algorithm a_k we are interested in the $j = 1, \dots, J$ performance distributions on the $m = 1, \dots, M$ data sets which define the problem domain $\mathfrak{D} = \{\mathfrak{L}_1, \dots, \mathfrak{L}_M\}$:

$$p_{mbkj} = p_j(a_k, \mathfrak{L}_m^b) \sim \mathcal{P}_{mkj} = \mathcal{P}_{kj}(\mathfrak{L}_m)$$

The p_{mbkj} are samples drawn from the jth performance distribution $\mathcal{P}_{kj}(\mathfrak{L}_m)$ of the algorithm a_k on the data set $\mathfrak{L}_m$. Analogously as above the performance is measured on a validation sample, i.e., $\hat{p}_{mbkj}$ is computed and the empirical performance distribution $\hat{\mathcal{P}}_{kj}(\mathfrak{L}_m)$ is estimated.

The execution of a benchmark experiment results in $M \times B \times K \times J$ raw performance measures, i.e., $M \times K \times J$ empirical performance distributions $\hat{\mathcal{P}}_{mkj}$. This allows to analyze a multitude of questions with a wide variety of methods – for example: computing an order of the algorithms based on some simple summary statistics from the empirical performance distributions; or more sophisticated, testing hypotheses of interest by modeling the performance measure distributions and using statistical inference. Additionally, each question can be answered on different scopes, i.e., locally for each data set, or globally for the domain. For the remainder of this chapter we assume that the following systematic stepwise approach defined in Chapter 2 has been executed for each given data set $\mathfrak{L}_m$:

1. Compare candidate algorithms: The candidate algorithms are pairwise compared based on their empirical performance distributions $\hat{\mathcal{P}}_{mkj}$ by simple summary statistics or statistical tests (parametric or non-parametric); this results in J' comparisons.

   Example: The algorithms `svm`, `rpart`, and `rf` are compared; the pairwise comparisons according to their misclassification errors are $\{\texttt{svm} \prec \texttt{rf}, \texttt{rpart} \prec \texttt{rf}, \texttt{svm} \sim \texttt{rpart}\}$ (based on a statistical test), and according to their computation times $\{\texttt{rpart} \prec \texttt{rf}, \texttt{rf} \prec \texttt{svm}, \texttt{rpart} \prec \texttt{svm}\}$ (based on the mean statistics).

2. Compute performance relations: The J' comparisons are interpreted as an ensemble of relations $\mathcal{R}_m = \{R_1, \dots, R_J\}$. Each R_j represents the relation of the K algorithms with respect to a specific performance measure and the data set's preference as to the candidate algorithms.

Example (cont.): The data set's preferences are $R_1 = \texttt{svm} \sim \texttt{rpart} \prec \texttt{rf}$ in case of the misclassification error and $R_2 = \texttt{rpart} \prec \texttt{rf} \prec \texttt{svm}$ in case of the computation time.

3. Aggregate performance order relations: The ensemble $\mathcal{R}_m$ is aggregated to, for example, a linear or partial order $\bar{R}_m$ of the candidate algorithms. As a suitable class of aggregation methods we use consensus rankings. The individual relations R_j can be weighted to express the importance of the corresponding performance measure.

   Example (cont.): The linear order with the weights $w_1 = 1$ and $w_2 = 0.2$ (i.e., computation time is much less important than the misclassification error) is then $\texttt{rpart} \prec \texttt{svm} \prec \texttt{rf}$.

These data of different aggregation levels are available for each data set $\mathfrak{L}_m$ of the problem domain $\mathfrak{D}$. The obvious extension of the local approach to compute a domain-based order relation is the further aggregation of the data sets' algorithm orders (following Hornik and Meyer, 2007):

4. Aggregate local order relations: The domain specific algorithms' order relation $\bar{R}$ is computed by aggregating the ensemble of consensus relations $\mathcal{R} = \{\bar{R}_1, \ldots, \bar{R}_M\}$ using consensus methods.

This approach allows the automatic computation of a statistically correct domain-based order of the algorithms. But the "strong" aggregation to relations does not allow statements on the problem domain to a greater extent. In the following we introduce methods to visualize and to model the problem domain based on the individual benchmark experiment results. On the one hand these methods provide support for the global order $\bar{R}$, on the other hand they uncover structural interrelations of the problem domain $\mathfrak{D}$.

4.2. Visualizations of the domain

A benchmark experiment results in $M \times K \times J$ estimated performance distributions $\hat{\mathcal{P}}_{mkj}$. The simplest visualizations are basic plots which summarize the distributions, like strip charts, box plots, and density plots, conditioned by the domain's data sets. So called Trellis plots (Becker et al., 1996) allow a relative comparison of the algorithms within and across the set of data sets.

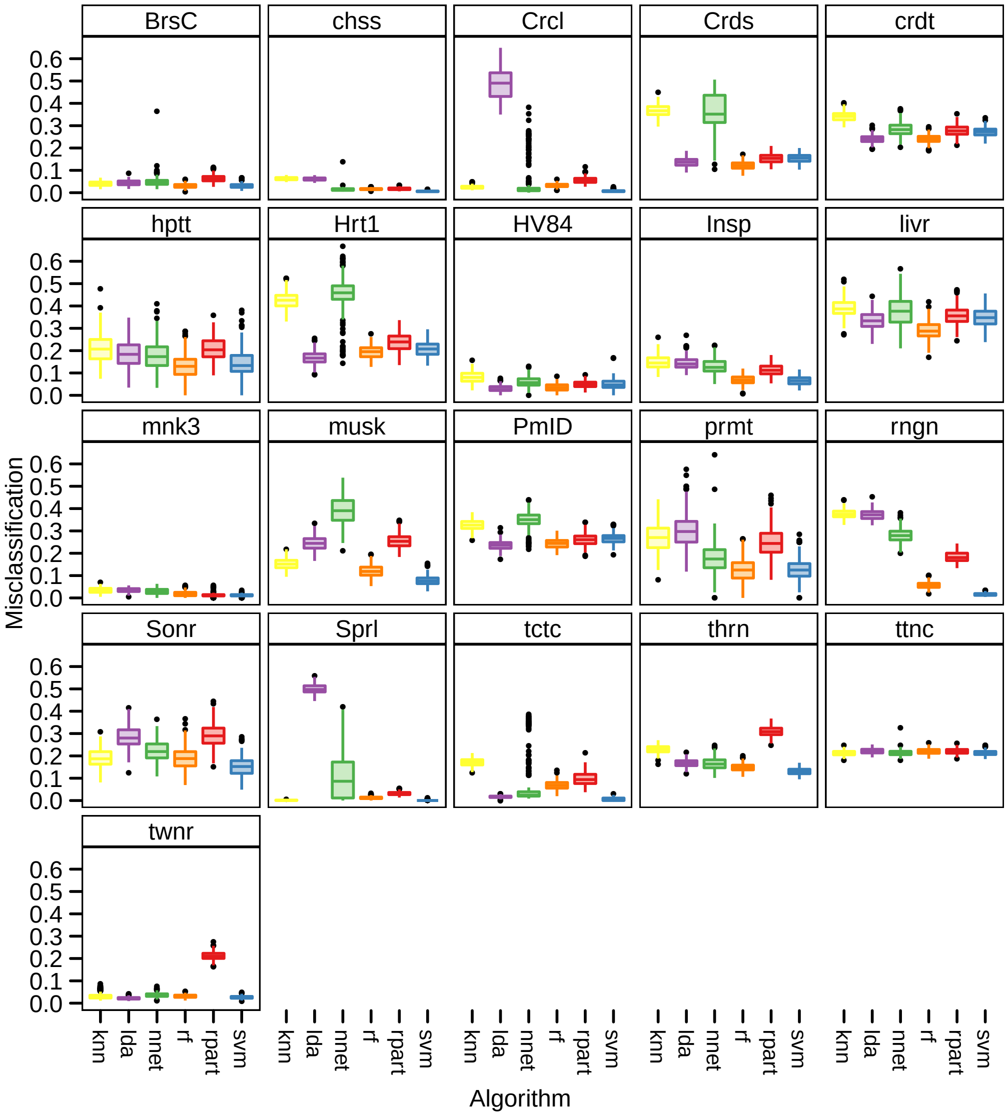


Figure 4.1.: Trellis graphic with box plot panels. The plot shows the misclassification error of the UCI domain benchmark experiment described in Section 4.5.1.

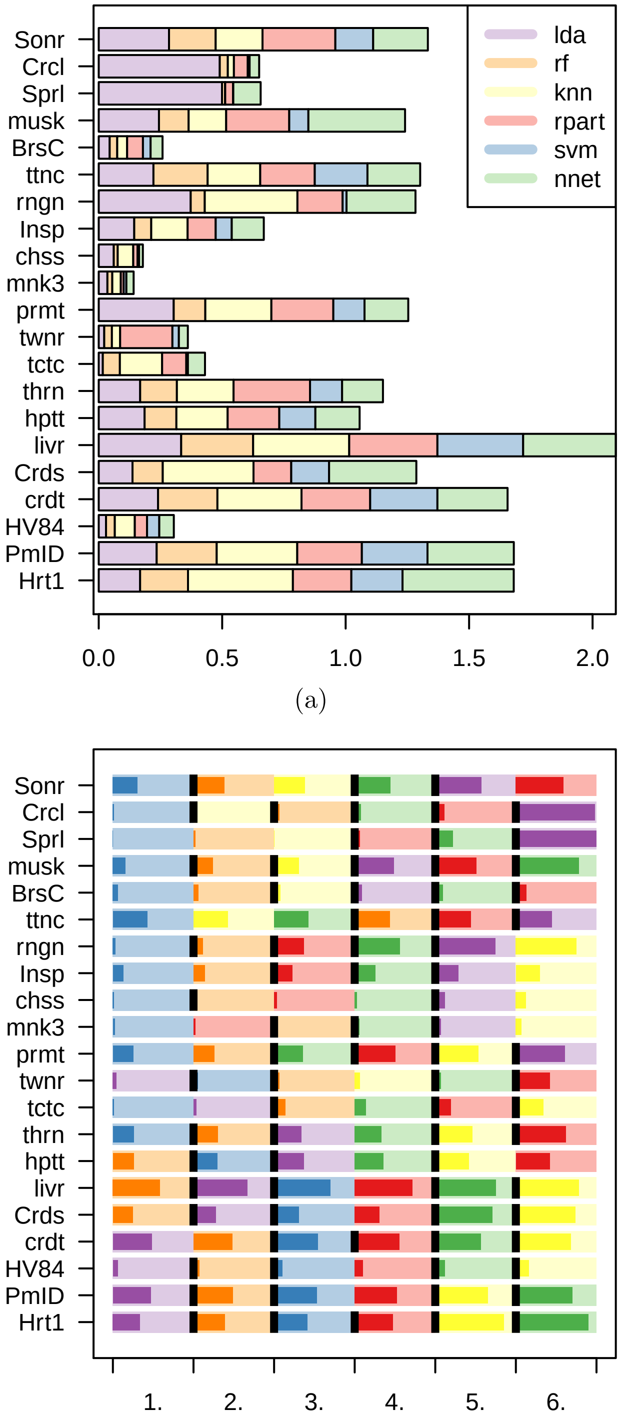
Figure 4.1 shows a Trellis plot with box plots of six algorithms' misclassification errors (`knn`, `lda`, `nnet`, `rf`, `rpart`, and `svm`) on a problem domain defined by 21 UCI data sets (Section 4.5.1 provides the experiment details). Using this visualization we see that there are data sets in this domain which are equally "hard" for all candidate algorithms, like `ttnc`, `mnk3` or `BrsC`; while the algorithms on other data sets perform much more heterogeneous, like on `prmnt` and `rngn`. From an algorithm's view, `lda` for example, has the highest misclassification error of the problem domain on data sets `Crcl` and `Sprl` (which are circular data). Moreover, whenever `lda` solves a problem well, other algorithms perform equally.

Further basic visualizations allowing relative comparisons of the estimated performance distributions $\hat{\mathcal{P}}_{mkj}$ based on descriptive statistics are stacked bar plots, spine plots and mosaic plots. In all visualizations one axis contains the data sets and the other the stacked performance measure (either raw or relative). Figure 4.2a exemplarily shows the stacked bar plot of the UCI domain's mean misclassification errors (the order of the data sets is explained below). Notice, for example, that for the candidate algorithms the data set `mnk3` is on average much "less hard" to solve than `livr`. This plot is an indicator for a learning problem's complexity; if all candidate algorithms solve the problem well, it is probably an easy one. (Figure 4.2b is explained further down.)

Now, in addition to the empirical performance distributions $\hat{\mathcal{P}}_{mkj}$, the pairwise comparisons, the resulting set of relations $\mathcal{R}_m$, and the locally aggregated orders $\bar{R}_m$ are available. To incorporate these aggregated information into the visualizations we use the distance measure already introduced and used in Section 2.5. The symmetric difference distance d_Δ is defined as the cardinality of the relations' symmetric difference, or equivalently, the number of pairs of objects being in exactly one of the two relations R_1, R_1 ($\oplus$ denotes the logical XOR operator):

$$\begin{aligned} d_\Delta(R_1, R_2) = \#\{(a_k, a_{k'}) \mid \\ & (a_k, a_{k'}) \in R_1 \;\oplus\; (a_k, a_{k'}) \in R_2, \\ & k, k' = 1, \ldots, K\} \end{aligned}$$

Computing all pairwise distances for the relations R_m ($m = 1, \ldots, M$) results in a symmetric $M \times M$ distance matrix D representing the distances of the domain $\mathfrak{D}$ based on the candidate algorithms' performances. An obvious way to analyze D is to hierarchically cluster it. Because detecting truly similar data sets within a domain is most interesting (in our point of view), we propose



(a)

(b)

Figure 4.2.: Visualizations of the candidate algorithms' misclassification errors on the UCI domain: (a) stacked bar plot; (b) benchmark summary plot (legend omitted).

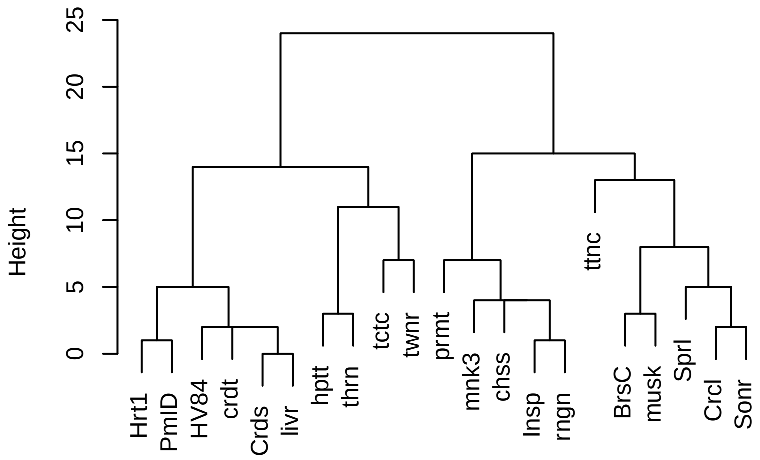


Figure 4.3.: Dendrogram showing the clustering of the UCI domain's data sets based on their candidate algorithms' performance relations.

to use agglomerative hierarchical clustering with complete linkage (see, e.g., Hastie et al., 2009). Figure 4.3 shows the corresponding dendrogram for the UCI domain's relation $\mathcal{R} = \{R_1, \dots, R_{21}\}$ based on the algorithms' misclassification errors. `Crcl` and `Sonr` for example, are in one cluster – this means that the candidate algorithms are in similar relations. Note that the raw performance measures are not involved anymore. Therefore, it is hard to see these similarities in basic visualizations like the stacked bar plot (Figure 4.2a), even if the data sets are ordered according to the data sets' linear order determined by the hierarchical clustering.

Benchmark summary plot. The benchmark summary plot (bsplot) overcomes these difficulties by adapting the stacked bar plot and incorporating a set of relations $\mathcal{R}$. Each bar uses the total width, and is evenly divided into as many partitions as candidate algorithms. The partitions are assigned to the candidate algorithms and their order is determined by the corresponding (linear or partial) relation $\bar{R}_m$. A descriptive statistic of the corresponding empirical performance distribution $\hat{\mathcal{P}}_{mkj}$ of interest is then plotted as bar from the bottom up in each partition; the values are scaled in a way that the domain's worst performance fills the partition. Color coding is used to simplify interpretation – partition backgrounds with light, performance bars with dark colors. Moreover, the relations $\bar{R}_m$ are visualized using borders between partitions. So, if there is for example a significant difference in the performance of two candidate algorithms, a (black) border is shown, otherwise no border is shown. The bars are sorted according to a linear order of the distance matrix D; just like the one computed by the hierarchical clustering. The axis representing the data sets is equivalent to the stacked bar plot, the other axis is a podium for the candidate algorithms. Obviously, this plot only works in case of linear orders. Visually interpreted, the aggregated global consensus relation $\bar{R}$ is the aggregation of the light colors over the data set axis.

Figure 4.2b shows the UCI domain's bsplot. In comparison with the stacked bar plot (Figure 4.2a) the individual benchmark experiment results are now more clearly visible. For example, `svm` (blue) has the most first places – 13 times (6 times exclusively), and is never worse than a third place. `lda` (purple) is the algorithm with the highest misclassification (on data set `Crcl`) and `knn` (yellow) is the algorithm with the most last places. Based on `lda` the domain splits into two clusters, one where it performs well (i.e., a set of linearly separable data sets) and one where not (the non-linearly separable data sets). `rf` (orange) also performs well within this domain, while `nnet` (green) is in most cases of medium performance.

Benchmark summary graph. In the UCI problem domain the resulting relations are all transitive, this is not generally true for relations based on statistical tests (see Chapter 2), therefore the benchmark summary graph (bsgraph) enables a general visualization. The domain $\mathfrak{D}$ is represented by a complete weighted graph K_M with M vertices for the domain's data sets and $M(M-1)/2$ edges. The edges' weights are defined by the distance matrix D. The graph's layout follows a spring model and is computed using the Kamada-Kawai algorithm (see, e.g., Gansner and North, 2000, for a description and software implementation). The layouted graph is then visualized with additional information available from the individual benchmark experiments. Our implementation shows the data sets' winner algorithms by filling the nodes with the corresponding colors; if there is no unique winner algorithm for a data set the node is unfilled. The widths and colors of the edges correspond to the distances, i.e., the shorter the distance the wider and darker the edge. Our implementation allows showing only a subset of edges corresponding to a subset of distances to make the visualization more clear.

Figure 4.4 shows the UCI domain's bsgraph with edges visible which correspond to tenth smallest distance. Here, for example, it is clearly visible that subset A of the domain's data sets has similar algorithm performances (although only the winners are visualized). It is also visible that the domain splits into two sub-domains: sub-domain B where the algorithm `svm` (blue) performs best, and sub-domain C where the algorithms `rf` (orange) and `lda` (purple) perform best. In case of unfilled nodes the dominant sub-domain algorithms are always winner as well together with other algorithms (compare with Figure 4.2b).

The benchmark summary graph defines the basis for more complex visualizations. One future work is an interactive version along the lines of the gcExplorer – an interactive exploration tool of neighbor gene clusters represented as graphs (Scharl and Leisch, 2009, cf. Figure 1). This tool enables the access to the nodes complex information using interactivity; the same idea can be used for the benchmark summary graph. This then allows interactive analysis of domain-based benchmark experiments analogous to the interactive analysis of single data set-based benchmark experiments presented in Chapter 3. Furthermore, looking at the introduced visualizations raises the question "why" do some candidate algorithms perform similar on some data sets and not on others – which data set characteristics affect the algorithms' performances and lead to such clusters as seen in Figure 4.4? We investigate this question in Chapter 5, where we introduce a formal framework based on (recursively partitioning) Bradley-Terry models (the most widely used method to study preferences in psychology and related disciplines) for automatic detection of important data set characteristics and their joint effects on the algorithms' performances in potentially complex interactions.

4.3. Models of the domain

The analysis methods introduced so far – the aggregation of local relations to a domain-based order relation and the visualization methods – rely on locally (data set-based) computed results. In this section we take the design of domain-based benchmark experiments into account and model the $M \times K \times J$ estimated performance distributions $\hat{\mathcal{P}}_{mkj}$ for $J = 1$ accordingly. This enables a domain's analysis based on formal statistical inference.

A domain-based benchmark experiment with one performance measure of interest is a type of experiment with two experimental factors (the candidate algorithms and the domain's data sets), their interactions, and blocking factors at two levels (the blocking per data set and the replications within each data set). It is written

$$p_{mbk} = \kappa_k + \beta_m + \beta_{mk} + \beta_{mb} + \epsilon_{mbk} \tag{4.1}$$

with $m = 1, \dots, M$, $b = 1, \dots, B$, and $k = 1, \dots, K$. κ_k represents the algorithms' mean performances, β_m the mean performances on the domain's data sets, β_{mk} the interactions between data sets and algorithms, β_{mb} the effect of the subsampling within the data sets, and ϵ_{mbk} the systematic error.

Mixed-effects models. Linear mixed-effects models are the appropriate tool to estimate the parameters described in Formula 4.1. Mixed-effects models incorporate both fixed effects, which are parameters associated with an entire population or with certain repeatable levels of experimental factors, and random effects, which are associated with individual experimental or blocking units drawn at random from a population (Pinheiro and Bates, 2000). The candidate algorithms' effect κ_k is modeled as fixed effect, the data sets' effect β_m as random effect (as the data sets can be seen as randomly drawn from the domain they define). Furthermore, β_{mk}, β_{mb} and ϵ_{mbk} are defined as random effects as well. The random effects follow $\beta_m \sim N(0, \sigma_1^2)$, $\beta_{mk} \sim N(0, \sigma_2^2)$, $\beta_{mb} \sim N(0, \sigma_3^2)$, and $\epsilon_{mbk} \sim N(0, \sigma^2)$. Analogous to single data set-based benchmark experiments, we can rely on the asymptotic normal and large sample theory (see Chapter 2).

The most common method to fit linear mixed-effects models is to estimate the "variance components" by the optimization of the restricted maximum likelihood (REML) through EM iterations or through Newton-Raphson iterations (see Pinheiro and Bates, 2000). The results are the estimated parameters: the variances $\hat{\sigma}_1^2$, $\hat{\sigma}_2^2$, $\hat{\sigma}_3^2$, and $\hat{\sigma}^2$ of the random effects; and the K fixed effects. The model allows the following interpretation – of course conditional on the domain $\mathfrak{D}$ – for an algorithm a_k and a data set $\mathfrak{L}_m$: $\hat{\kappa}_k$ is the algorithm's mean performance, $\hat{\beta}_m$ is the data set's mean complexity, and $\hat{\beta}_{mk}$ is the algorithm's performance difference from its mean performance conditional on the data set (coll., "how does the algorithm like the data set").

The parametric approach of mixed-effects models allows statistical inference, in particular hypothesis testing, as well. The most common null hypothesis of interest is "no difference between algorithms". A global test, whether there are any differences between the algorithms which do not come from the "randomly drawn data sets" or the sampling is the F-test. Pairwise comparisons, i.e., which algorithms actually differ, can be done using Tukey contrasts. The calculation of simultaneous confidence intervals enables controlling the experiment-wise error rate (we refer to Hothorn et al., 2008, for a detailed explanation).

Figure 4.5a shows simultaneous 95% confidence intervals for the algorithms' misclassification error based on a linear mixed-effects model. Two algorithms are significantly different if the corresponding confidence interval does not contain the zero. The confidence intervals are large because of the very heterogeneous algorithm performances over the data sets (cf. Figure 4.2b; Section 4.5.1 describes the result in detail). Now, statistical significance does not imply a practically relevant difference. As commonly known, the degree of significance can be affected by drawing more or less samples. A possibility to control this characteristic of benchmark experiments is to define and quantify how large a significant difference has to be to be relevant. Let $[\Delta_1, \Delta_2]$ be the area of equivalence (zone of non-relevance). The null hypothesis is rejected if the $(1 - \alpha) * 100\%$ confidence interval is completely contained in the area of equivalence (equivalence tests are the general method which consider relevance; see, for example, Wellek, 2003). Figure 4.5b shows the UCI domain's pairwise comparisons with $[-0.10, 0.10]$ as the area of equivalence. For example, the difference between `rpart` and `rf` is significant (the interval does not contain the zero) but is not relevant (the area of equivalence completely contains the interval); the difference between `svm` and `lda` is significant and relevant. Of course, the definition of the area of equivalence contains the subjective view of the practitioner – normally, it is based on domain-specific knowledge.

Finally, the pairwise comparisons of candidate algorithms' significant or relevant differences allow to establish a preference relation based on the performance measure for the domain $\mathfrak{D}$. From now on, the analysis of domain-based benchmark experiments proceeds analogously to the analysis of single data set-based benchmark experiments. Chapter 2 introduce the concept of preference relations based on statistical tests and their combination using consensus methods. Now, the methodology introduced above makes it possible to model the experiment for each performance measure; i.e., to fit J linear mixed-effects models, to compute the significant or relevant pairwise comparisons, and to establish a preference relation R_j for each performance measure ($j = 1, \ldots, J$). Consensus methods aggregate the J preference relations to single domain-based order relation of the candidate algorithms. This can be seen as a multicriteria or multiobjective optimization and allows, for example, to select the best candidate algorithm with respect to a set of performance measures for the given domain $\mathfrak{D}$.

4.4. Archetypes of the domain

In this section we present a different approach to analyze the performances $\hat{p}_{mbkj}$ of a set of candidate algorithms a_k on a domain $\mathfrak{D}$. In general, comparing objects often means comparing with a "best" or "worst" object, i.e., comparing with an extreme object (the benchmark). In the simplest case of one performance measure describing the objects, the benchmarks are the univariate minimum or maximum values. However, often more than one performances are measured for each object; and then there are no uniquely defined extremes. On this account, one has to define multivariate extreme values – and one possible extreme values are archetypes. Heavlin (2006) and Porzio et al. (2008) use archetypes to estimate benchmarks in different fields: Heavlin (2006) in case of computer performances; Porzio et al. (2008) in case of a top 200 world university ranking.

Archetypal analysis has the aim to find "pure types", the archetypes, within a set defined in a specific context. In statistics, archetypal analysis is first introduced by Cutler and Breiman (1994). Part II of this dissertation gives a detailed introduction into archetypal analysis; its theoretical foundations, the detailed optimization problem, and the algorithm to solve it. In this section we roughly describe the concept: For a given data set X, the goal of archetypal analysis is to find a number of archetypes Z which are (1) convex combinations of the data set's observations, and (2) the observations are convex combinations of these archetypes. These two constraints lead to the fact that the determined archetypes are located on the convex hull of the data set; and therefore are good candidates for extreme values (i.e., benchmarks). The optimization problem is to find the two coefficient matrices α and β which minimize the residual sum of squares

$$\mathrm{RSS} = \|X - \alpha Z^\top\|_2 \text{ with } Z = X^\top \beta$$

subject to the convex combination constraints of the coefficient matrices α and β. We refer to Section 8.1 (Formula 8.1 ff.) for a detailed explanation on the problem's components.

In case of the benchmark experiments in this dissertation we have $M \times B \times K \times J$ performance measures $\hat{p}_{mbkj}$. In order to investigate them with archetypal analysis we compile a tabular data matrix X which contains the performance measures apposite to the kind of archetypes we are interested in. Here, we focus on $J = 1$ performance measures and take the performances on the M data sets as variables of interest for each of the B replications of the K algorithms:

$$\begin{array}{r|ccc|}
\multicolumn{1}{r}{} & \multicolumn{3}{c}{M \text{ data sets}} \\ \cline{2-4}
\begin{array}{l} B \text{ samples from} \\ \text{algorithm } a_1 \end{array} \left\{ \begin{array}{c} \\ \\ \\ \end{array} \right. & \begin{array}{l} \hat{p}_{1111} \\ \vdots \\ \hat{p}_{1B11} \end{array} & \begin{array}{c} \cdots \\ \ddots \\ \cdots \end{array} & \begin{array}{l} \hat{p}_{M111} \\ \vdots \\ \hat{p}_{MB11} \end{array} \\
 & \vdots & \ddots & \vdots \\
\begin{array}{l} B \text{ samples from} \\ \text{algorithm } a_K \end{array} \left\{ \begin{array}{c} \\ \\ \\ \end{array} \right. & \begin{array}{l} \hat{p}_{11K1} \\ \vdots \\ \hat{p}_{1BK1} \end{array} & \begin{array}{c} \cdots \\ \ddots \\ \cdots \end{array} & \begin{array}{l} \hat{p}_{M1K1} \\ \vdots \\ \hat{p}_{MBK1} \end{array} \\ \cline{2-4}
\end{array}$$

Of course, the tabular data matrix can be compiled in many ways, and each one enables a different interpretation. Examples for other useful compilations are: only using the mean or median performances of the algorithms; or using more than one performance measure (standardized) for each data set. Regardless of how the tabular data matrix is compiled, the archetypes Z define the extreme performances of the domain $\mathfrak{D}$ conditional on this data matrix (and consequently on the sets of algorithms, performance measures, and data sets). Furthermore, the coefficient matrix α provides an interpretable assignment of the observations (i.e., algorithms) to the archetypes (i.e., extreme performances).

Figure 4.6 shows an archetypes solution in case of the UCI domain example. The correct number of archetypes is determined by running the algorithm from two to ten archetypes and using the "elbow criterion" on the RSS. Here, we present the four archetypes solution (which is the number where the first elbow occurs); note that the supplementary material (Appendix A) provides information to see the corresponding scree plot and to investigate all other solutions. Figure 4.6a presents a parallel coordinates plot of the data matrix X with color coding of the lines to indicate the algorithms, and the four archetypes A1 (red), A2 (green), A3 (blue), and A4 (cyan). Archetype A1 (red) is clearly the "minimum", it is low for all data sets. The remaining three archetypes represent archetypal performances which are low and high on different data sets. No archetype is available in this solution which is a clear "maximum" (i.e., high on all data sets). Beside that, note that on data set `ttnc` all four archetypes have similar values; on this data set all algorithms perform similar (cf. Figure 4.2).

Figure 4.6b shows a parallel coordinates plot of the coefficient matrix α. This means that four axes represent the four archetypes on a range $[0, 1]$, and each line visualizes the archetypes' contributions to the approximation of each individual performance (observation). Again, the lines are colored according to the algorithms to show correlations between the archetypes and the algorithms. Archetype A1 mostly contributes to the performances of the algorithms `svm` (blue) and `rf` (orange). As archetype A1 is the "minimum" archetype, this indicates that `svm` and `rf` tend to be the algorithms with low misclassification errors within this domain. Archetype A2 mostly contributes to the algorithms `knn` (yellow) and `nnet` (green). Archetypes A3 and A4 contribute to `lda` (purple) and `rpart` (red) respectively. In this example, the pooling of two algorithms by one archetype – `svm` and `rf` by A1, `knn` and `nnet` by A2 – indicates their similar performance within this domain. This corresponds with the final analysis of the UCI domain in Section 4.5.1 (see for example Figure 4.7b).

4.5. Benchmarking UCI and Grasshopper domains

We present two domain-based benchmark experiments – one for each application scenario we sketch in the introduction. The UCI domain serves as a domain for the scenario when comparing a newly developed algorithm with other well-known algorithms on a well-known domain. We already used the UCI domain in the previous sections to illustrate the presented methods and we now give more details on this benchmark experiment and complete the analysis. The Grasshopper domain serves as domain where we want to find the best candidate algorithm for predicting whether a grasshopper species is present or absent in a specific territory. The algorithm is then used as a prediction component in an enterprise application software system.

4.5.1. UCI domain

The UCI domain is defined by 21 data sets binary classification problems available from Asuncion and Newman (2007). We are interested in the behavior of the six common learning algorithms linear discriminant analysis (`lda`, purple), k-nearest neighbor classifiers, (`knn`, yellow), classification trees (`rpart`, red), support vector machines (`svm`, blue), neural networks (`nnet`, green), and random forests (rf, orange); see all, for example, Hastie et al. (2009). The benchmark experiment is defined with $B = 250$ replications, bootstrapping as resampling scheme to generate the learning samples $\mathfrak{L}^b$, and the out-of-bag scheme for the corresponding validation samples $\mathfrak{T}^b$. Misclassification on the validation samples is the performance measure of interest. A benchmark experiment is executed and analyzed on each data set according to the local systematic stepwise approach (Steps 1-3) given in the beginning of Section 4.1 (and defined in Chapter 2). The results are $21 \times 6 \times 1$ estimated performance distributions $\hat{P}^j_{mk}$, the corresponding pairwise comparisons based on mixed-effects models and test decisions for a given $\alpha = 0.05$, and the resulting preference relations $\mathcal{R} = \{R_1, \dots, R_{21}\}$. Note that we present selected results, the complete results are available in the supplemental material (see Appendix A on computational details).

The Trellis plot in Figure 4.1 shows the box plots of the estimated performance distributions. Table 4.1 lists the resulting preference relations R_m; in this benchmark experiment all relations are transitive, therefore the listed chains of preferences can be built ($a_k \sim a_{k'}$ indicates no significant difference, $a_k \prec a_{k'}$ indicates a significantly better performance of a_k). The domain-based linear order relation $\bar{R}$ computed by the consensus method (Step 4) is:

$$\texttt{svm} \prec \texttt{rf} \prec \texttt{lda} \prec \texttt{rpart} \prec \texttt{nnet} \prec \texttt{knn}$$

This order coincides with the impression given by the bsplot's light colors in Figure 4.2b: svm (blue) has the most first places, rf (orange) the most second and some first places, lda (purple) has some first places, rpart (red) and nnet (green) share the middle places, and knn (yellow) has the most last places.

Computing the linear mixed-effects model leads to a model with the estimated candidate algorithm effects:

lda $\hat{\kappa}_1$	knn $\hat{\kappa}_2$	nnet $\hat{\kappa}_3$	rf $\hat{\kappa}_4$	rpart $\hat{\kappa}_5$	svm $\hat{\kappa}_6$
0.2011	0.1948	0.1885	0.1144	0.1750	0.1100

lda has the worst, svm the best mean performance. Data set mnk3 has the lowest and data set livr the highest estimated performance effect (i.e., complexity) among the domain's data sets:

	$\hat{\beta}_m$	$\hat{\beta}_{mk}$					
		`lda`	`knn`	`nnet`	`rf`	`rpart`	`svm`
`mnk3`:	-0.0943	-0.0722	-0.0669	-0.0658	0.0009	-0.0696	-0.005
`livr`:	0.1693	-0.0326	0.0253	0.0152	0.0109	0.0175	0.0652

For data set `mnk3`, all algorithms except `rf` perform better than their mean performance; for `livr` only `lda`. These estimated parameters conform with the performance visualizations in figures 4.1 and 4.2.

Figure 4.5 shows the (a) significant and (b) relevant pairwise comparisons. There is, for example, a significant difference between `svm` and `rpart` in favor of `svm` and no significant difference between `svm` and `rf`. The interpretation of the pairwise significant differences results in the incidence matrix shown in Figure 4.7a. The corresponding relation is no linear or partial order relation (as we can verify). However, plotting only the asymmetric part of its transitive reduction as Hasse diagram enables a visualization and an interpretation of the relation (Hornik and Meyer, 2010) – Figure 4.7b shows this Hasse diagram, nodes are ordered bottom-up. For the UCI domain and based on the mixed-effects model analysis we can state that `rf` and `svm` are better than `knn`, `nnet`, and `rpart`. In case of `lda` this analysis allows no conclusion. This result corresponds with the global linear order relation $\bar{R}$ computed by the aggregation of the individual preference relations. And also with the archetypal analysis of the domain done in Section 4.4.

4.5.2. Grasshopper domain

In this application example we are interested in finding the best algorithm among the candidate algorithm as a prediction component of an enterprise application software system. The domain is the domain of grasshopper species in Bavaria (Germany), the task is to learn whether a species is present or absent in a specific territory.

The data were extracted from three resources. The grasshopper species data are available in the "Bavarian Grasshopper Atlas" (Schlumprecht and Waeber, 2003). In this atlas, Bavaria is partitioned into quadrants of about 40km^2. Presence or absence of each species it is registered for each quadrant. The territory data consist of climate and land usage variables. The climate variables are available from the WorldClim project (Hijmans et al., 2005) in a 1km^2

resolution. These 19 bioclimate (metric) variables describe for example the seasonal trends and extreme values for temperature and rainfall. The data are primary collected between 1960 and 1990. The land usage variables are available from the CORINE LandCover project CLC 2000 (Federal Environment Agency, 2004). Based on satellite images the territory is partitioned into its land usage in a 100m^2 resolution using FRAGSTAT 3.3 (McGarigal et al., 2002). These 20 (metric and categorical) land usage variables describe the percentage of, for example, forest, town and traffic (we binarized a variable if not enough metric values are available). The climate and land usage variables are averaged for each quadrant for which the grasshopper data are available. Additionally, the Gauss-Krüger coordinates and the altitude are available for each quadrant. We use the standardized altitude but omit the coordinates as the candidate algorithms are not able to estimate spatial autocorrelation and heterogeneity. Now, to define the domain, we understand each grasshopper species as individual data set. The quadrants where a species is present are positively classified; as negatively classified quadrants we draw random samples from the remaining ones. If enough remaining quadrants are available we create a balanced classification problem, otherwise we use all remaining quadrants. We only use data sets with more than 300 positively classified quadrants – so, the Grasshopper domain is finally defined by 33 data sets.

The candidate algorithms of interest are linear discriminant analysis (`lda`, purple), k-nearest neighbor classifiers, (`knn`, yellow), classification trees (`rpart`, red), support vector machines (`svm`, blue), naive Bayes classifier (`nb`, brown), and random forests (`rf`, orange); see all, for example, Hastie et al. (2009). The benchmark experiment is defined with $B = 100$ replications, bootstrapping as resampling scheme to generate the learning samples $\mathfrak{L}^b$, and the out-of-bag scheme for the corresponding validation samples $\mathfrak{T}^b$. Misclassification on the validation samples is the performance measure of interest. Note that we presents selected results, the complete results are available in the supplemental material (see Appendix A on computational details).

Figure 4.8 shows the Trellis plot with box plots for the six algorithms' misclassification errors. We see that for most data sets the relative order of the candidate algorithms seems to be similar, but that the individual data sets are differently "hard" to solve. The locally computed preference relations $\mathcal{R} = \{R_1, \ldots, R_{33}\}$ (using mixed-effects models; see Section 2.3) contains non-transitive relations; therefore, a visualization using the benchmark summary plot is not possible. Now, one possibility is to plot the asymmetric part of the transitive reduction (like in Figure 4.7b) for each of the 33 relations in a Trellis plot. However, such a plot is very hard to read and the benchmark summary graph provides a simpler visualization (albeit with less information). Figure 4.9a shows the bsgraph with the six smallest distance levels visible. The nodes show the color of the algorithm with the minimum median misclassification error. We see that for most data sets `rf` (orange) is the best algorithm. The nodes' cross-linking indicates that the relations do not differ much in general. The algorithms follow the general order pattern even for data sets where this plot indicates a big difference, for example the `NEUS` data set (cf. Figure 4.8).

A consensus aggregation of $\mathcal{R}$ results in the following linear order:

$$\texttt{rf} \prec \texttt{lda} \prec \texttt{svm} \prec \texttt{rpart} \prec \texttt{nb} \prec \texttt{knn}$$

This order confirms the exploratory analysis. To formally verify this order we compute the domain-based linear mixed-effects model and the resulting pairwise comparisons. Figure 4.9b shows the corresponding simultaneous 95% confidence intervals and the resulting order is:

$$\texttt{rf} \prec \texttt{lda} \prec \texttt{svm} \prec \texttt{rpart} \prec \texttt{nb} \sim \texttt{knn}$$

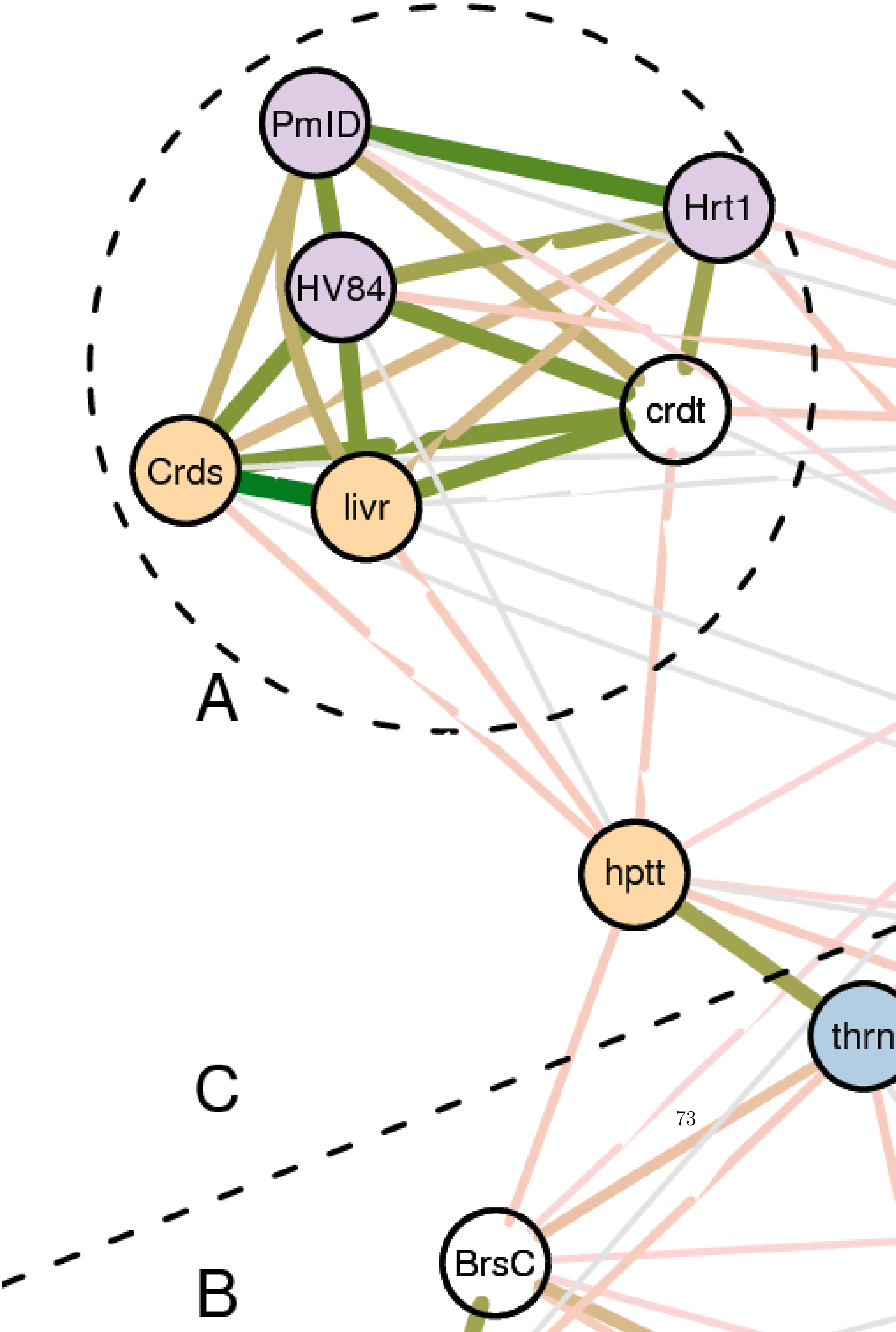
All three analyses, exploratory, consensus-, and mixed-effect model-based, lead to the same conclusion: the random forest learning algorithm is the best algorithm (according to the misclassification error) for the Grasshopper domain.

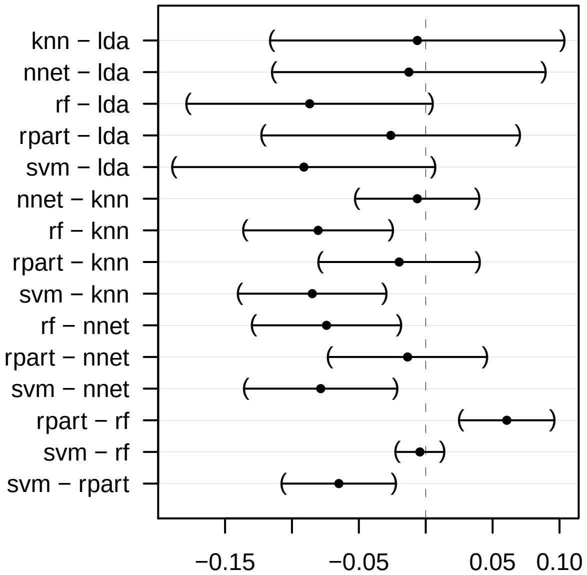
4.6. Summary

The great many of published benchmark experiments show that this method is the primary choice to evaluate learning algorithms. Chapter 1 defines the theoretical framework for inference problems in benchmark experiments. Chapter 2 introduces the practical toolbox with a systematic four step approach from exploratory analysis via formal investigations through to a preference relation of the algorithms. Now, the present chapter extends the framework theoretically and practically from single data set-based benchmark experiments to domain-based (set of data sets) benchmark experiments.

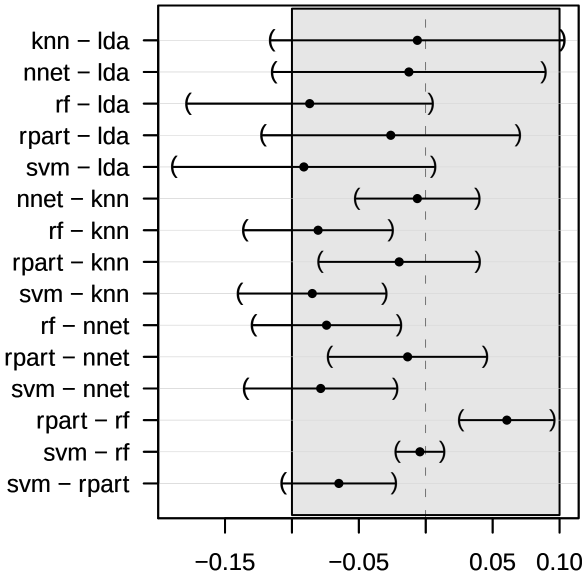
Given the computation of local – single data set-based – benchmark experiment results for each data set of the problem domain, the chapter introduces two specialized visualization methods. The benchmark summary plot (bsplot) is an adaption of the stacked bar plot. It allows the visualization of statistics of the algorithms' estimated performance distributions incorporated with the data sets' preference relations. This plot only works in case of linear or partial order relations, while the benchmark summary graph (bsgraph) enables a general visualization. The problem domain is represented by a complete weighted graph with vertices for the domain's data sets. The edges' weights are defined by the pairwise symmetric difference distances. The layouted graph is visualized with additional information from the local benchmark experiments which allows to find patterns within the problem domain.

An analysis of the domain based on formal statistical inference is enabled by taking the experiment design – two experimental factors, their interactions, and blocking factors at two levels – into account. We use linear mixed effects models to estimate the parameters where the algorithms are defined as fixed effects, all others as random effects. The estimated model allows the interpretation – conditional on the domain – of the algorithms' mean performances, the data sets' mean complexities and how suitable an algorithm for a data set is. Furthermore, testing hypotheses of interest is possible as well. A global test of the most common hypothesis of "no difference between the algorithms" can be performed with an F-test, a pairwise comparison can be performed using Tukey contrasts. The definition of an area of equivalence allows to incorporate practical relevance instead of statistical significance. Finally, the pairwise comparisons establish a preference relation of the candidate algorithms based on the domain-based benchmark experiment. Archetypal analysis provides a different view of benchmark experiments. Archetypal performances are computed and used as data driven extreme values (i.e., benchmarks). The archetype coefficients of the individual performances then allow an interpretation of the similarity according to the determined benchmark performances. The two exemplar domain-based benchmark experiments show the proposed methods are sound, i.e., that all computed results conform with each other.


PmID
Hrt1
HV84
crdt
Crds
livr
A
hptt
thrn
C
BrsC
B

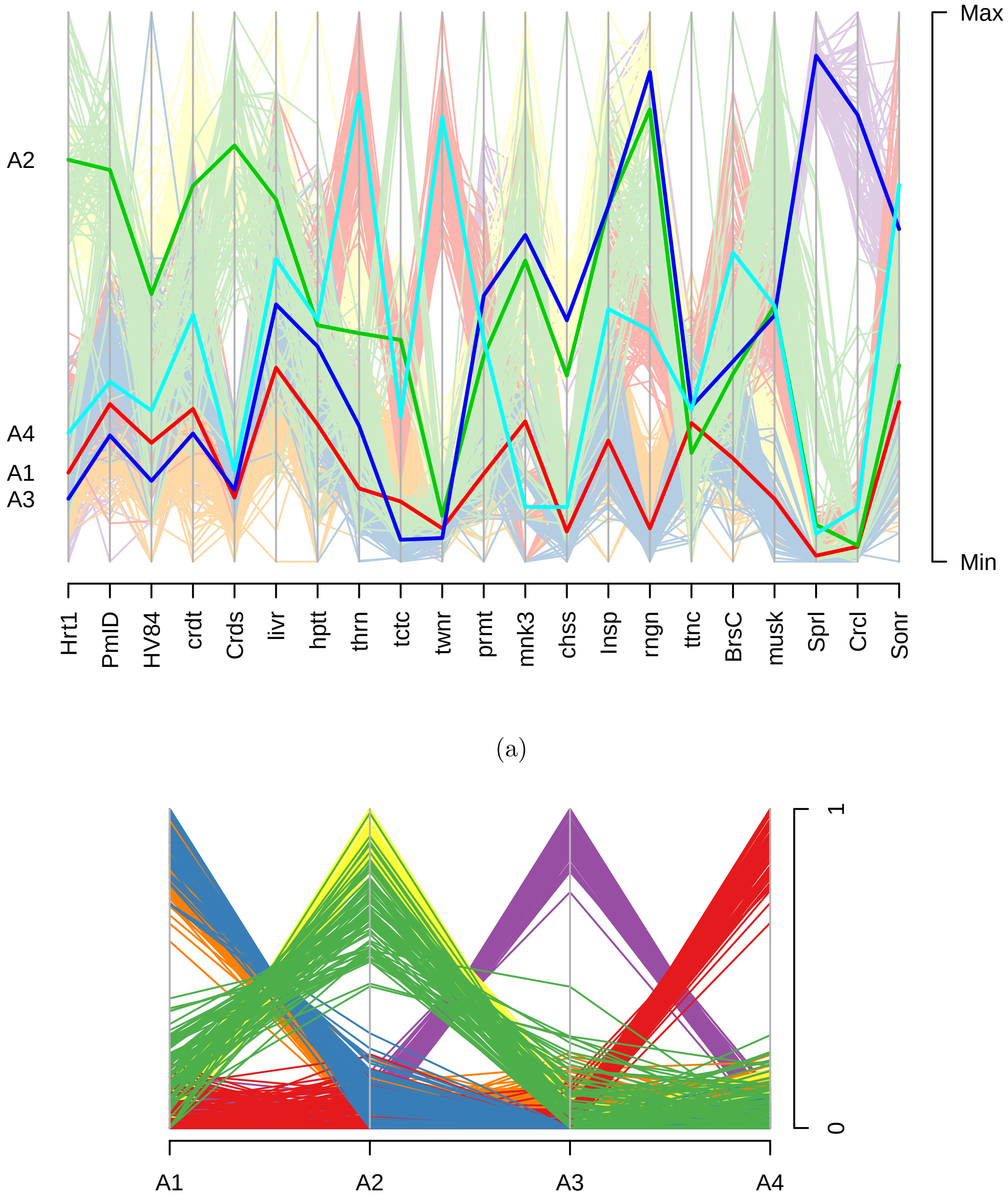


(a)



(b)

Figure 4.5.: The UCI domain's simultaneous 95% confidence intervals for multiple (a) significant and (b) relevant comparisons for a fitted linear mixed-effects model on the algorithms' misclassification errors.



(a)

(b)

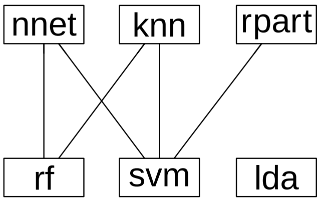
Figure 4.6.: The four misclassification error archetypes of the UCI domain: parallel coordinates plots of (a) the data matrix X and the archetypes and (b) the coefficient matrix α. In both plots the lines are colored according to the candidate algorithms.

$R_{\texttt{BrsC}}$:	$\texttt{rf} \sim \texttt{svm} \prec \texttt{knn} \prec \texttt{lda} \prec \texttt{nnet} \prec \texttt{rpart}$
$R_{\texttt{Crds}}$:	$\texttt{rf} \prec \texttt{lda} \prec \texttt{rpart} \sim \texttt{svm} \prec \texttt{nnet} \prec \texttt{knn}$
$R_{\texttt{chss}}$:	$\texttt{svm} \prec \texttt{nnet} \sim \texttt{rf} \sim \texttt{rpart} \prec \texttt{knn} \sim \texttt{lda}$
$R_{\texttt{Crcl}}$:	$\texttt{svm} \prec \texttt{knn} \prec \texttt{rf} \prec \texttt{nnet} \prec \texttt{rpart} \prec \texttt{lda}$
$R_{\texttt{crdt}}$:	$\texttt{lda} \sim \texttt{rf} \prec \texttt{svm} \prec \texttt{rpart} \prec \texttt{nnet} \prec \texttt{knn}$
$R_{\texttt{Hrt1}}$:	$\texttt{lda} \prec \texttt{rf} \prec \texttt{svm} \prec \texttt{rpart} \prec \texttt{knn} \prec \texttt{nnet}$
$R_{\texttt{hptt}}$:	$\texttt{rf} \prec \texttt{svm} \prec \texttt{lda} \sim \texttt{nnet} \prec \texttt{knn} \sim \texttt{rpart}$
$R_{\texttt{HV84}}$:	$\texttt{lda} \prec \texttt{rf} \prec \texttt{rpart} \sim \texttt{svm} \prec \texttt{nnet} \prec \texttt{knn}$
$R_{\texttt{Insp}}$:	$\texttt{rf} \sim \texttt{svm} \prec \texttt{rpart} \prec \texttt{nnet} \prec \texttt{knn} \sim \texttt{lda}$
$R_{\texttt{livr}}$:	$\texttt{rf} \prec \texttt{lda} \prec \texttt{rpart} \sim \texttt{svm} \prec \texttt{nnet} \prec \texttt{knn}$
$R_{\texttt{mnk3}}$:	$\texttt{rpart} \sim \texttt{svm} \prec \texttt{rf} \prec \texttt{nnet} \prec \texttt{knn} \sim \texttt{lda}$
$R_{\texttt{musk}}$:	$\texttt{svm} \prec \texttt{rf} \prec \texttt{knn} \prec \texttt{lda} \prec \texttt{rpart} \prec \texttt{nnet}$
$R_{\texttt{PmID}}$:	$\texttt{lda} \prec \texttt{rf} \prec \texttt{rpart} \sim \texttt{svm} \prec \texttt{knn} \prec \texttt{nnet}$
$R_{\texttt{prmt}}$:	$\texttt{rf} \sim \texttt{svm} \prec \texttt{nnet} \prec \texttt{rpart} \prec \texttt{knn} \prec \texttt{lda}$
$R_{\texttt{rngn}}$:	$\texttt{svm} \prec \texttt{rf} \prec \texttt{rpart} \prec \texttt{nnet} \prec \texttt{knn} \sim \texttt{lda}$
$R_{\texttt{Sonr}}$:	$\texttt{svm} \prec \texttt{knn} \sim \texttt{rf} \prec \texttt{nnet} \prec \texttt{lda} \sim \texttt{rpart}$
$R_{\texttt{Sprl}}$:	$\texttt{knn} \sim \texttt{rf} \sim \texttt{svm} \prec \texttt{rpart} \prec \texttt{nnet} \prec \texttt{lda}$
$R_{\texttt{thrn}}$:	$\texttt{svm} \prec \texttt{rf} \prec \texttt{lda} \sim \texttt{nnet} \prec \texttt{knn} \prec \texttt{rpart}$
$R_{\texttt{tctc}}$:	$\texttt{lda} \sim \texttt{svm} \prec \texttt{nnet} \sim \texttt{rf} \prec \texttt{rpart} \prec \texttt{knn}$
$R_{\texttt{ttnc}}$:	$\texttt{knn} \sim \texttt{nnet} \sim \texttt{svm} \prec \texttt{rf} \prec \texttt{rpart} \prec \texttt{lda}$
$R_{\texttt{twnr}}$:	$\texttt{lda} \prec \texttt{svm} \prec \texttt{knn} \sim \texttt{rf} \prec \texttt{nnet} \prec \texttt{rpart}$

Table 4.1.: UCI domain's chains of preference relations $\mathcal{R} = \{R_1, \ldots, R_{21}\}$.

	knn	lda	nnet	rf	rpart	svm
knn	0	0	0	0	0	0
lda	0	0	0	0	0	0
nnet	0	0	0	0	0	0
rf	1	0	1	0	0	0
rpart	0	0	0	0	0	0
svm	1	0	1	0	1	0

(a)



(b)

Figure 4.7.: Interpretation of the pairwise significant differences, Figure 4.5a, as preference relation: (a) incidence matrix, (b) the corresponding Hasse diagram (nodes are ordered bottom-up).

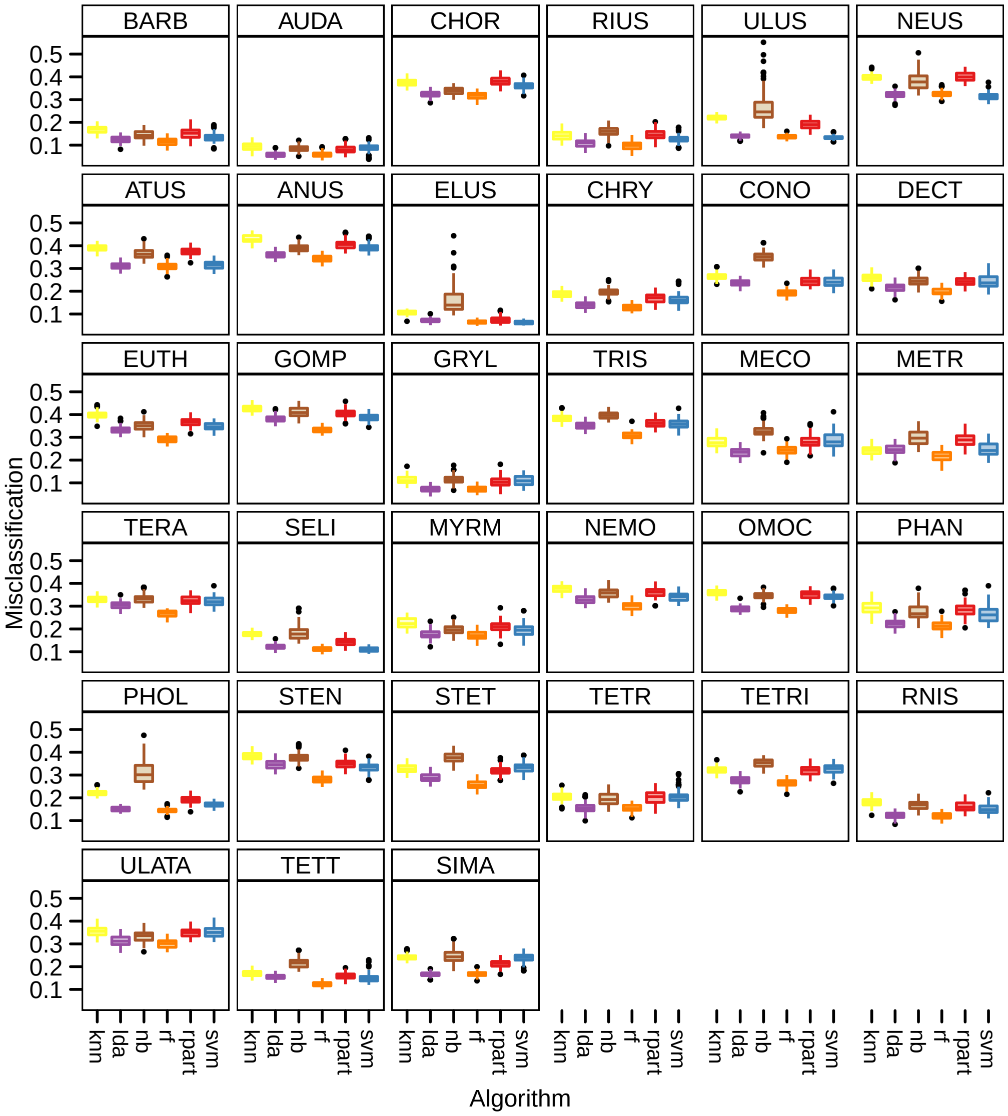
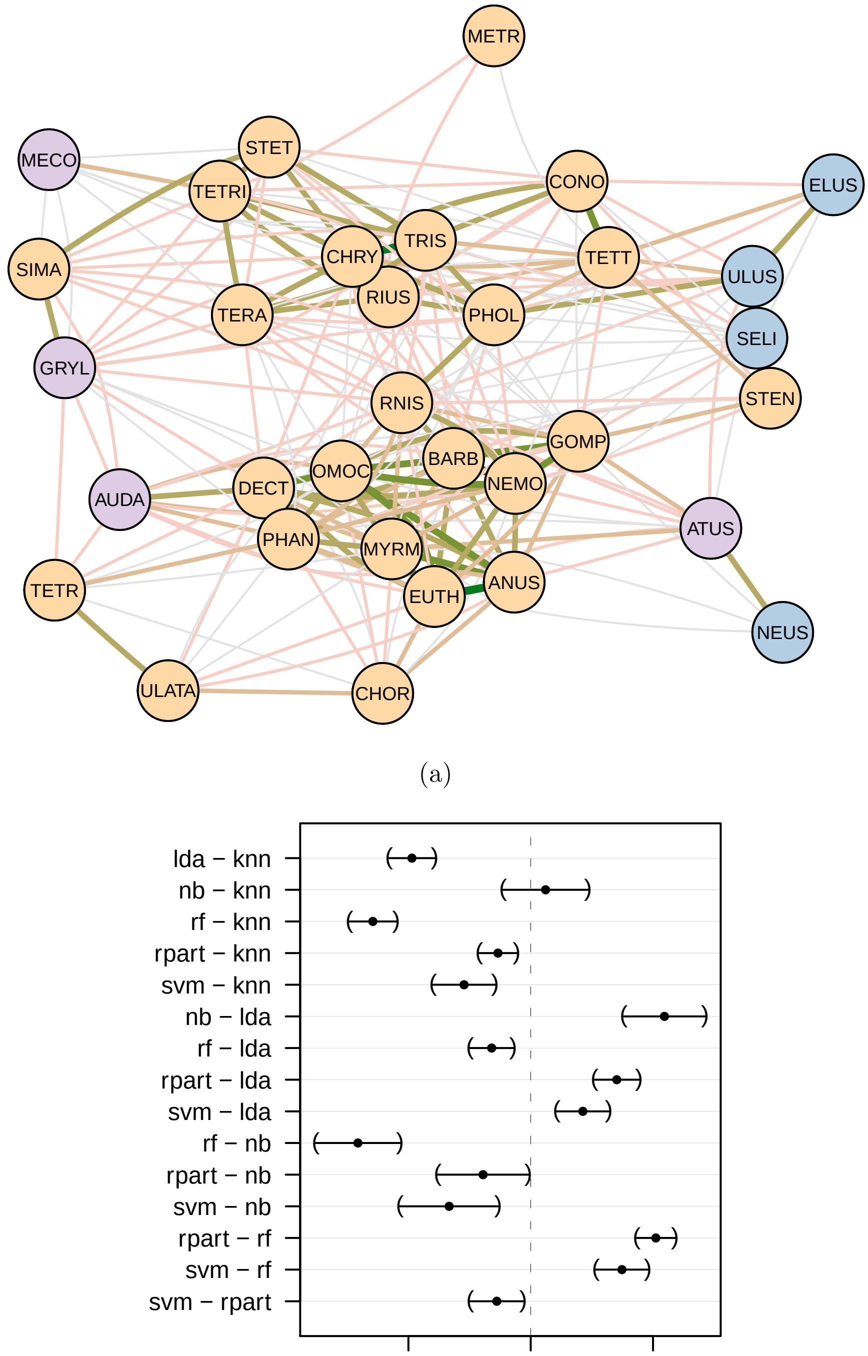


Figure 4.8.: Trellis graphic with box plot of the candidate algorithms' misclassification error on the Grasshopper domain. Each data set is one grasshopper species.



(a)

(b)

Figure 4.9.: (a) The Grasshopper domain's benchmark summary graph; the color of the nodes indicate the algorithm with the minimum median misclassification error. (b) The Grasshopper domain's simultaneous 95% confidence intervals for multiple significant comparisons for a fitted linear mixed-effects model on the algorithms' misclassification errors.

Chapter 5.

(Psycho)-Analysis of benchmark experiments

In machine and statistical learning, benchmark experiments are empirical investigations with the aim of comparing and ranking learning algorithms with respect to certain performance measures. In particular, performance is investigated on a collection of data sets, e.g., from the UCI Machine Learning Repository (Asuncion and Newman, 2007). It is well known that the characteristics of the data sets have an influence on the performance of the algorithms – almost every publication that proposes a new algorithm presents its performance on data sets in relation to different characteristics (even though often only the number of observations and attributes vary). Nonetheless, in most publications differences of the data sets are noted but not used for further well-founded analyses; perhaps the best known study is STATLOG by King et al. (1995), newer ones are e.g. Lim et al. (2000) and Caruana et al. (2008). An approach incorporating both algorithms and data sets was suggested by Kalousis et al. (2004), who investigate the relations between learning algorithms and data sets by means of clustering the algorithms on one hand and the data sets on the other hand based on the performance measures. These cluster results (a large number of graphics) are then manually interpreted to find relations. The present article is an enhancement of their approach and provides an automated framework where each step of the relation finding process is based on sound statistical methodology.

In psychology and related disciplines the pairwise comparative choice model suggested by Bradley and Terry (1952) is the most widely used method to study preferences of subjects (e.g. consumers or patients) on some objects (e.g. a set of chocolate bars or different pain therapies). The preference scaling of a group of subjects may not be homogeneous, but different groups of subjects with certain characteristics may show different preference scalings. A newly developed semi-parametric approach for recursive partitioning of Bradley-Terry models (Strobl et al., 2010) takes this circumstance into account – it identifies groups of subjects with homogeneous preference scalings in a data-driven and statistically correct way. This approach is an extension of the classical algorithms for classification and regression trees (CART) (Breiman et al., 1984; Quinlan, 1993) and results in a tree where the subjects are divided into groups according to their characteristics, and in each terminal leaf a Bradley-Terry model shows the preference scaling within this group, as described in detail in the next section.

The use of Bradley-Terry models has also been suggested for deriving consensus rankings from benchmark studies (Hornik and Meyer, 2007). However, in order to utilize the information inherent in different characteristics of the data sets, here we suggest to apply the advanced approach of recursive partitioning of Bradley-Terry models in the analysis of benchmark studies. In this framework, the data sets are the subjects and the algorithms are the objects. The interpretation is equivalent to the interpretation of the preference relations in Section 2.4: a data set expresses its preferences for an algorithm by the performance of the algorithm. In other words, the algorithm that performs best on a certain data set is considered to be most preferred by the data set. Using statistical and information-theoretic measures to characterize the data sets, the approach of recursive partitioning of Bradley-Terry models enables us to determine the influence of data set characteristics on the performance of the algorithms. It provides a framework to either investigate the influence exploratory or test particular hypothesis of interest. This chapter defines all parts of the framework and, as a first step, presents its prospects for the exploratory analysis.

The chapter introduces all related methods in detail: Section 5.1 reviews the needed benchmark experiment notation. In Section 5.2 we define a sound and flexible framework for data set characterization and introduce a common set of data set characteristics. Finally, in Section 5.3 we outline the principle of model-based recursive partitioning and its generalization to Bradley-Terry models. A toy example is used to demonstrate each part of the framework. Section 5.4 then applies the proposed method to a real-world example based on classification problems from the well-known UCI Machine Learning Repository. The chapter is concluded with a summary and an outlook in Section 5.5. All computational details are provided in the Appendix A.

5.1. Benchmark experiments

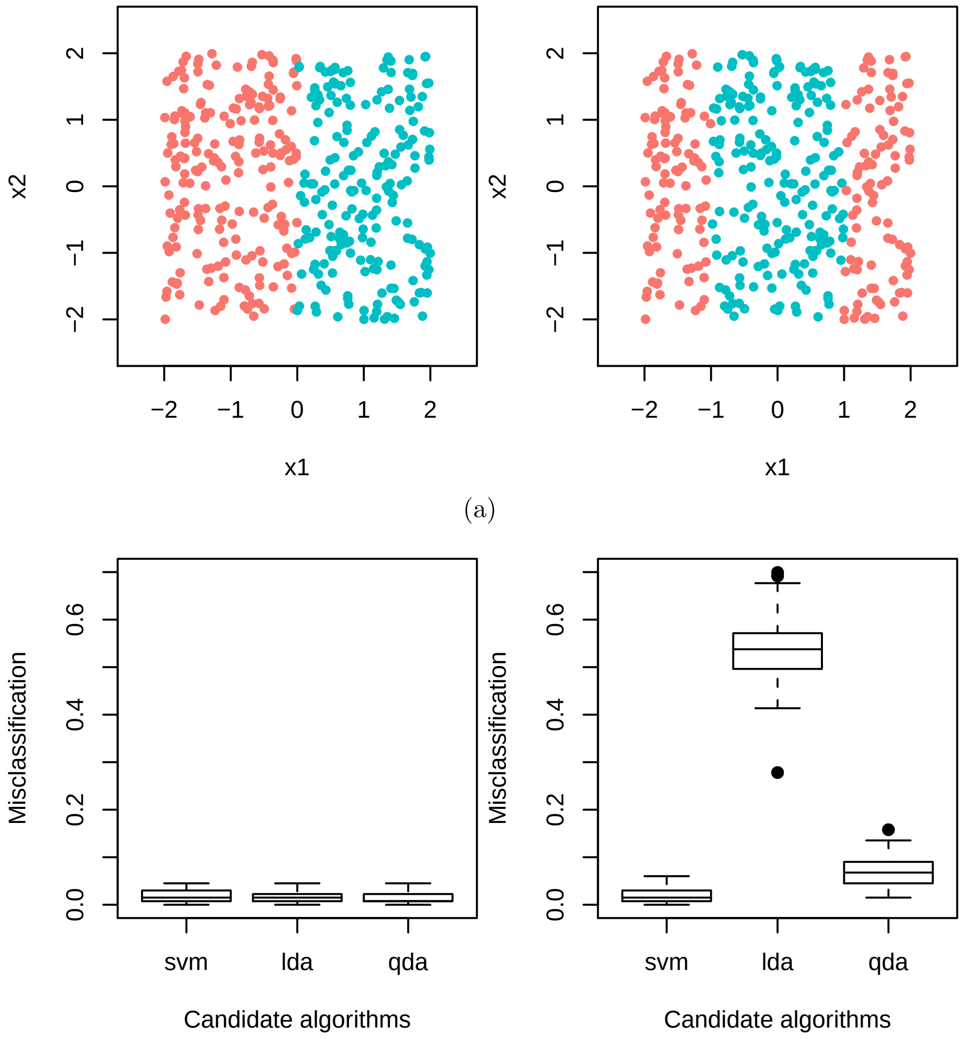
Basis of this framework are dombain-based benchmark experiments as defined in Chapter 2 and Chapter 4. We estimate the empirical performance distributions $\hat{\mathcal{P}}_k(\mathfrak{L}_m)$ of K candidate algorithms a_k according to one performance measure $p(\cdot)$ on a specific problem domain. This domain is specified by a collection of data sets $\mathfrak{L}_1, \ldots, \mathfrak{L}_M$. A benchmark experiment is executed on each data set $\mathfrak{L}_m$ $(m = 1, \ldots, M)$ and an order of the algorithms is calculated. However, the performance rankings of the candidate algorithms will vary over the different data sets in all realistic benchmark scenarios and one common assumption is that the performance depends on particular characteristics of the data sets. This chapter proposes an approach to answer the question which data set characteristics make the algorithms perform differently on certain data sets.

Exemplar benchmark experiment. For illustration purposes, an artificial toy example is used. It consists of two 2-dimensional classification problems with 400 observations in each case: `ds1` is linearly separable, `ds2` is not linearly separable; Figure 5.1a shows scatter plots of the data. The algorithms of interest are support vector machines (`svm`), linear discriminant analysis (`lda`) and quadratic discriminant analysis (`qda`) (see e.g. Hastie et al., 2009, and Appendix A for computational details).

In case of the toy example ($\mathfrak{L}_1$ = `ds1`, $\mathfrak{L}_2$ = `ds2`, $B = 100$ with 2/3-subsampling, i.e. $n = 267$, as resampling scheme and p being the misclassification error), Figure 5.1b shows a box plot of the misclassification error: As expected, a_1 = `svm` and a_2 = `qda` solve both problems very well; a_3 = `lda` solves the linearly separable problem very well, but has huge problems with the non-linearly separable one. The goal is now to provide a method which detects that `lda` has problems with data set `ds2` because of the non-linearly separable feature space.

5.2. Data set characterization

The question why certain algorithms perform better on particular data sets than others requires a possibility to describe the data sets. One common ap-



(a)

(b)

Figure 5.1.: (a) Exemplary classification problems and (b) performance measures of the candidate algorithms: (left) `ds1` where the feature space is linearly separable; (right) `ds2` with non-linearly separable feature space.

proach is to extract statistical and informative measures from the data sets; the STATLOG project by King et al. (1995) was the first one which defined such a set of structural characteristics. Newer approaches, e.g., Kalousis and Hilario (2000) and de Souto et al. (2008), extend this set of data set characteristics and use them in terms of meta-learning – an approach to learn which algorithm is best suited for an unknown learning problem (see, e.g. Vilalta and Drissi, 2002).

Given some user-specified characteristics, data set characterization can be seen as a two-step process: (1) map each data set into its individual characterization space; (2) reduce the individual characterization spaces into one common characterization space where all data sets are comparable, i.e., a metric can be defined. More formal, let $\mathcal{L}$ be the space of all data sets and $\mathfrak{L} \in \mathcal{L}$. The function

$$\mathrm{map}\colon \mathcal{L} \to \mathbb{R}^* \quad \text{with} \quad \mathfrak{L} \mapsto x^*$$

computes one specific characteristic of a data set. $\mathbb{R}^*$ indicates that the dimension of the vector x^* can depend on the data set. For example, computing the skewness of each continuous input variable results in a vector with dimension equal to the number of continuous input variables of the given data set; on the other hand, computing the number of observations results in one number for every data set. The dimension of x^* can even be zero if, for example, a data set has no continuous input variables so that the characteristic is missing. This first step does not guarantee that all data sets are comparable, therefore another function $\mathrm{red}(\cdot)$ is defined as

$$\mathrm{red}\colon \mathbb{R}^* \to \mathbb{R}^d \quad \text{with} \quad x^* \mapsto x^d$$

which reduces the dimension of characteristic vector x^* to dimension d identical for all data sets. Examples for such reduction functions are: the mean or a quantile (for which $d = 1$) or a histogram representation (for which d is chosen according to the number of bins) for characteristics like the skewness, that provide a value for each continuous variable in the data set; or the identity function (for which $d = 1$) for characteristics like the number of observations, that provide exactly one value for each data set in the first place.

Characteristic	Description	ds1	ds2
obs.n	number of observations	400	400
var.n	number of variables	2	2
nvar.n	number of nominal variables	0	0
nvar.entropy	mean nominal variable entropy		
nvar.bin	number of binary variables	0	0
cvar.n	number of continuous variables	2	2
cvar.mac	mean multiple attribute correlation	0.06	0.06
cvar.skew	mean skewness	-0.08	-0.08
cvar.kurt	mean kurtosis	-1.20	-1.20
resp.cl	number of response classes	2	2
resp.entropy	mean response entropy	5.93	5.93
i2r.fcc	**first canonical correlation**	**0.86**	**0.04**
i2r.frac1	variation from first linear discriminant	1.00	1.00
i2r.mi	mean mutual information		
i2r.envar	equivalent number of variables		
i2r.nsratio	noise to signal ratio		

Table 5.1.: Description of the characteristics used in this chapter and its realization for the linearly separable (ds1) and non-linearly separable (ds2) classification problems.

A data set characterization then consists of a set of characteristics $\{(\mathrm{map}_1, \mathrm{red}_1), \ldots, (\mathrm{map}_T, \mathrm{red}_T)\}$, and for a given data set $\mathfrak{L}$, its characterization is the vector $c = (c_1, \ldots, c_T)$ with

$$c_t = \mathrm{red}_t(\mathrm{map}_t(\mathfrak{L})),\ t = 1, \ldots, T.$$

This framework allows a sound and flexible definition of data set characterization (and a simple implementation in software). Common characteristics, like those defined in King et al. (1995), Kalousis and Hilario (2000) and de Souto et al. (2008) can be formulated in terms of this map/reduce framework. As already noted, the STATLOG project defined the first characteristics which are broadly established nowadays. To simplify matters we use most of their characteristics together with some additional ones; Table 5.1 provides a list of typical data set characteristics (for a detailed description we refer to the original paper). With respect to our notation, map_t, $t = 1, \ldots, T$, corresponds to the different characteristics in Table 5.1 and red_t was chosen to be the mean for all characteristics. Columns ds1 and ds2 of Table 5.1 show the characterization of the two data sets in case of the toy example. As the data sets are

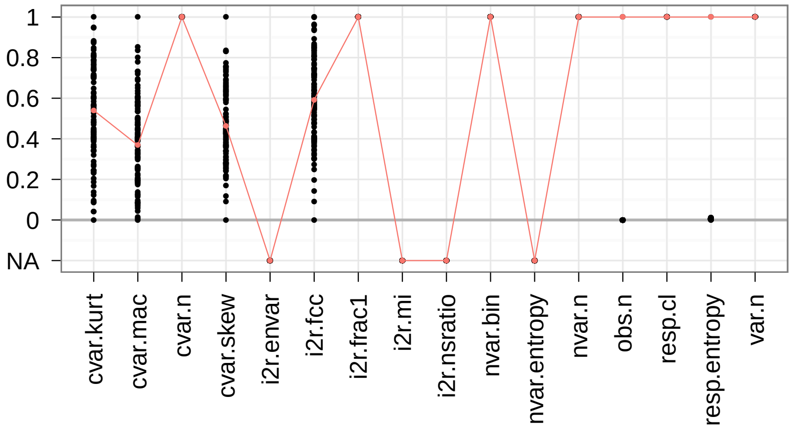


Figure 5.2.: Relative variation of the characterizations of the 100 drawn samples in case of `ds1`. The red line marks the characterization of the original data set; NA means that this characteristic is not available on this data set.

constructed in way that they match in all characteristics except the linearly separability, the first canonical correlation (i2r.fcc) is the only characteristic that differs; the first canonical correlation ranges between $[0, 1]$, whereas 1 means linearly separable and 0 not linearly separable.

Now, extending the benchmark experiment framework with the calculation of data set characteristics allows us to determine the influence of data set characteristics on the performance of algorithms: for each sample b drawn from the original data set m, the benchmark experiment provides (1) the performance of the candidate algorithms p_{mbk} ($k = 1, \dots, K$), and (2) the characterization of the sample $c_{mb} = (c_{mb1}, \dots, c_{mbT})$ with $c_{mbt} = red_t(map_t(\mathfrak{L}^b_m))$. Note that some characteristics could vary between samples drawn from the same data set, while others definitely stay constant. For example, the mean response entropy (resp.entropy) could vary depending on how many observations are drawn from each class, whereas the number of variables (var.n) always stays constant. Figure 5.2 shows the relative variation of the characterization of the 100 drawn samples in case of `ds1`.

The result of such an experiment is a collection of tuples of the performance of each algorithm on each learning sample and the characterizations of each sample. The corresponding data matrix is of the form of a tabular with $B \cdot M$ rows (for B samples drawn from M original data sets) and $K+T$ columns (for the performances measures of K algorithms and the T data set characteristics):

$$\begin{array}{l|ccc|ccc|}
 & \multicolumn{3}{c}{K \text{ algorithms}} & \multicolumn{3}{c}{T \text{ characteristics}} \\ \cline{2-7}
 & p_{111} & \cdots & p_{11K} & c_{111} & \cdots & c_{11T} \\
B \text{ samples from data set } \mathfrak{L}_1 & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
 & p_{1B1} & \cdots & p_{1BK} & c_{1B1} & \cdots & c_{1BT} \\
 & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
 & p_{M11} & \cdots & p_{M1K} & c_{M11} & \cdots & c_{M1T} \\
B \text{ samples from data set } \mathfrak{L}_M & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
 & p_{MB1} & \cdots & p_{MBK} & c_{MB1} & \cdots & c_{MBT} \\ \cline{2-7}
\end{array}$$

5.3. Preference scaling

The basic model for preference scaling, that is used here for comparing the performance of different candidate algorithms on a variety of data sets, is the Bradley-Terry model (Bradley and Terry, 1952). Here, we consider the formulation by Critchlow and Fligner (1991), that allows ties so that each comparison of the performance of two algorithms has three possible outcomes: (1) the first algorithm wins, (2) the second algorithm wins, or (3) both algorithms perform equally (i.e., a tie).

Here, we conduct paired comparisons of the performance measures for the K algorithms: For K algorithms there are $\frac{K\cdot(K-1)}{2}$ paired comparisons. Accordingly, the left part of the data matrix illustrated above is now replaced by a $B \cdot M \times \frac{K\cdot(K-1)}{2}$ table with an entry for each paired comparison (columns) on each data set (rows):

$$\begin{array}{l|ccc|}
 & \multicolumn{3}{c}{K \cdot (K-1)/2 \text{ comparisons}} \\ \cline{2-4}
 & R(p_{111}, p_{112}) & \cdots & R(p_{11K-1}, p_{11K}) \\
B \text{ samples from data set } \mathfrak{L}_1 & \vdots & \ddots & \vdots \\
 & R(p_{1B1}, p_{1B2}) & \cdots & R(p_{1BK-1}, p_{1BK}) \\
 & \vdots & \ddots & \vdots \\
 & R(p_{M11}, p_{M12}) & \cdots & R(p_{M1K-1}, p_{M1K}) \\
B \text{ samples from data set } \mathfrak{L}_M & \vdots & \ddots & \vdots \\
 & R(p_{MB1}, p_{MB2}) & \cdots & R(p_{MBK-1}, p_{MBK}) \\ \cline{2-4}
\end{array}$$

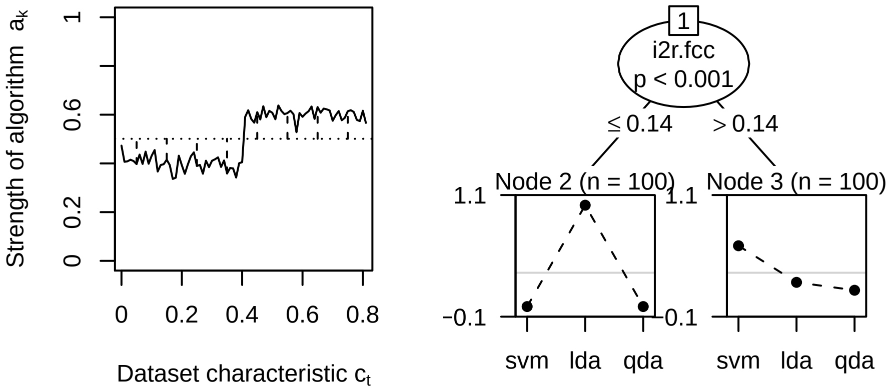
The entries of the table are the relations $R(p_{mbk}, p_{mbk'})$ describing one of the outcomes (1), (2) or (3) of the comparison of algorithms k and k' on sample b drawn from data set m. Since this new data matrix still has one row for each sample b drawn from data set m, it is compatible with the table of data set characteristics from the above illustration. Thus, the complete data matrix used for the following analysis consists of a $B \cdot M \times \frac{K \cdot (K-1)}{2}$ table for the paired comparisons and a $B \cdot M \times T$ table for the data set characteristics, that will be used to identify groups of data sets between which the performance comparisons of the algorithms differ.

When we now consider the paired comparisons, according to the Bradley-Terry model the three possible outcomes have the probabilities:

$$\begin{aligned} P\left(R(p_{mbk}, p_{mbk'}) = 1\right) &= \frac{\pi_k}{\pi_k + \pi_{k'} + \nu\sqrt{\pi_k \pi_{k'}}}, \\ P\left(R(p_{mbk}, p_{mbk'}) = 2\right) &= \frac{\pi_{k'}}{\pi_k + \pi_{k'} + \nu\sqrt{\pi_k \pi_{k'}}}, \\ P\left(R(p_{mbk}, p_{mbk'}) = 3\right) &= \frac{\nu\sqrt{\pi_k \pi_{j'}}}{\pi_k + \pi_{k'} + \nu\sqrt{\pi_k \pi_{k'}}}, \end{aligned}$$

where $\pi_k \geq 0$, $k = 1, \ldots, K$, are the parameters indicating the strength of each algorithm, and $\nu \geq 0$ is a discrimination constant governing the probability of ties. For parameter estimation via maximum likelihood, one restriction is necessary: usually, either one parameter is fixed to zero or the sum of all parameters constrained to 1, as in the following illustrations.

In order to assess whether there are groups of data sets with certain characteristics, for which the performance rankings of the candidate algorithms – and thus the parameters indicating the strength of each algorithm in the Bradley-Terry model – differ systematically, the model-based partitioning approach of Strobl et al. (2010) is used. The algorithm for model-based recursive partitioning is an extension of the popular CART algorithm for classification and regression trees (Breiman et al., 1984). However, while in the CART algorithm the aim is to detect groups of observations with different values of a response variable by means of recursively splitting the feature space, the aim in model-based recursive partitioning is to detect groups of observations that vary in the parameters of a certain model of interest. This model of interest could be, e.g., a linear regression model, where the intercept and slope parameters vary between different groups, whereas in our case, the model of interest is the Bradley-Terry model, that describes the performance of the candidate algorithms on different groups of data sets. The idea behind this is that one joint Bradley-Terry



(a)

(b)

Figure 5.3.: (a) Parameter instability in the strength of a candidate algorithm (simplified illustration). (b) Partitioned paired comparison model.

model would be suited to describe the benchmark study only if the ranking of the algorithms was equal for all data sets. If, however, different algorithms perform best on groups of data sets with different characteristics, this can be detected by the model-based recursive partitioning approach. The algorithm for model-based recursive partitioning of Bradley-Terry models consists of the following consecutive steps:

1. Fit a Bradley-Terry model for the paired comparisons of the algorithms based on all data sets in the current node (starting with the root node including all data sets).

2. Assess the stability of the Bradley-Terry model parameters with respect to each characteristic of the data sets.

3. If there is significant instability in the model parameters, split the data sets in two nodes along the characteristic with the strongest instability, and use the cutpoint with the highest improvement of the model fit.

4. Repeat steps 1–3 recursively in the resulting nodes until there are no more significant instabilities (or the number of data sets left in a node falls below a given stopping value).

The statistical framework employed here for testing the significance of instabilities in the model parameters is based on structural change tests adopted from econometrics, that can be used to detect the instability in the parameters of the Bradley-Terry models due to certain characteristics of the data sets, and is described in detail in Strobl et al. (2010) and Zeileis et al. (2008). The rationale of the underlying tests is that the individual deviations from a joint model are considered over the range of each potential splitting variable: As illustrated in Figure 5.3a, the strength parameter for algorithm a_k may show a systematic change when considered over the range of the characteristic c_j – such as the first canonical correlation, that indicates linear separability – while over the range of other characteristics the parameter may vary only randomly. A sound statistical framework is available to detect such systematic parameter instabilities and select the splitting variable or characteristic inducing the strongest instability (Strobl et al., 2010; Zeileis et al., 2008).

When the model-based partitioning approach of Strobl et al. (2010) is employed to detect groups of data sets with certain characteristics for which the performance-rankings of the candidate algorithms differ, the resulting partition can be displayed as a tree, as illustrated for the artificial toy example in Figure 5.3b: From all available characteristics in Table 5.1, the first canonical correlation i2r.fcc – that indicates whether the data set is linearly separable – is correctly identified as the characteristic that induces a significant change in the performance-ranking of the algorithms. For the 100 samples from data set `ds1`, that is not linearly separable, the values of the characteristic i2r.fcc are low and `lda` performs poorly (left node in Figure 5.3b), while for the 100 samples from data set `ds2`, that is linearly separable, the values of the characteristic i2r.fcc are high and `lda` performs well (whereas `svm` performs slightly worse when compared to the other algorithms, as displayed in the right node in Figure 5.3b). Note that in this example the performance measure quantifies errors, i.e., the smaller the better; and this is reflected in the tree nodes where small values indicate good performances as well.

5.4. Application example

This section demonstrates the framework by means of the UCI domain already used in Chapter 4. Computational details of the application example are described in Appendix A.

The application example consists of $M = 13$ data sets, that are all binary classification problems but cover a wide area of data set characteristics. Figure 5.4 names all data sets and shows the relative variation of their character-

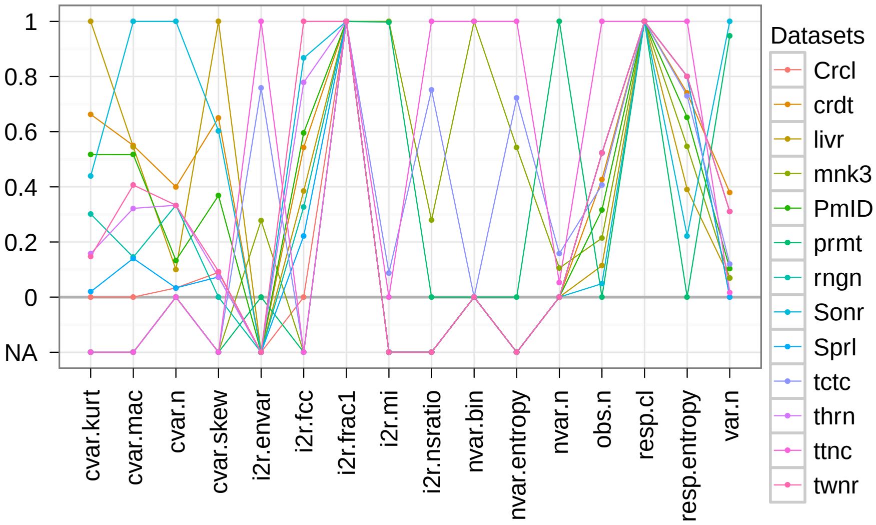


Figure 5.4.: Relative characterization of the UCI machine learning repository data sets.

istics (again the characteristics listed in Table 5.1 are used, i.e. $T = 16$). The $K = 6$ algorithms of interest are linear discriminant analysis (`lda`), k-nearest neighbor classifier (`knn`), classification trees (`rpart`), support vector machines (`svm`), neural networks (`nnet`) and random forests (`rf`) (see, e.g. Hastie et al., 2009). The performance measure p is the misclassification error and we draw $B = 100$ samples using 2/3-subsampling without replacement. The results of Strobl et al. (2007) and our preliminary experiments have shown that subsampling – rather than bootstrap sampling – is the resampling method of choice for this kind of benchmark experiments. The reason for this is that bootstrap sampling with replacement can induce artifacts in measures of the association between attributes, such as the entropy or the χ^2-statistic (Strobl et al., 2007).

On each data set the benchmark experiment is computed and the result is a 1300×15 table with paired comparisons of the algorithms and the corresponding 1300×16 table with data set characteristics. To fit the recursively partitioned Bradley-Terry model, categorical and ordinal characteristics are employed directly, while continuous characteristics are discretized based on their quantiles. (This discretization discards the order information, but allows to treat missing values as an extra category in a straightforward way – and if the order is important the model will find it nevertheless). We require a minimum of 200 observations per node; here the idea is to create nodes that contain in average the samples of at least two data set, so that the resulting partition is well interpretable. Based on this setup, Figure 5.5 shows the resulting tree.

Focusing first on the rightmost node (Node 11) in Figure 5.5, it becomes clear that the performance-rankings of the 6 algorithms is conditional on the domain specified by the data sets: Node 11 consists of big (obs.n > 667) and linearly well separable (i2r.fcc > 0.520) data sets. On these data sets, all algorithms perform well except classification trees. This is plausible from the method of operating of classification trees, where a linear separation can only be roughly approximated by means of stepwise binary partitioning. Another example where one algorithm is clearly inferior is displayed in Node 9, where the data sets are hardly linearly separable (i2r.fcc ≤ 0.520) and the dimensionality is low (var.n ≤ 2). In this case, linear discriminant analysis performs poor for obvious reasons, while the remaining algorithms perform well. With increasing dimensionality (var.n > 2, Node 10) it appears that support vector machines perform best, followed by random forests and neural networks; k-nearest neighbor classifiers perform equal to classification trees and again linear discriminant analysis comes last.

The data sets that are grouped in the remaining three nodes on the left hand side of the tree (Nodes 4, 6 and 6) based on their smaller samples sizes (obs.n $\leq$ 667) also show clear differences in the performance-rankings, but here the tree provides no reasonable explanation: the only characteristic used for splitting in this part of the tree is the number of observations obs.n. However, this characteristic may only serve as a proxy for other differences between the data sets, that are not yet covered by our list of characteristics.

This application example nicely illustrates the main challenge of the proposed method: the selection of the “right” data set characteristics. However, the benefit of the proposed method is that – at least from the set of characteristics provided – the relevant ones are selected for splitting automatically. In further consequence this allows to compute a huge set of characteristics, (e.g. join all characteristics available in the three papers cited in the introduction); the relevant ones are chosen automatically. Moreover, interactions between the characteristics are detected such that it becomes clear, e.g., that the performance of linear discriminant analysis depends both on the linear separability and on the dimensionality of the data set, as illustrated in our application example.

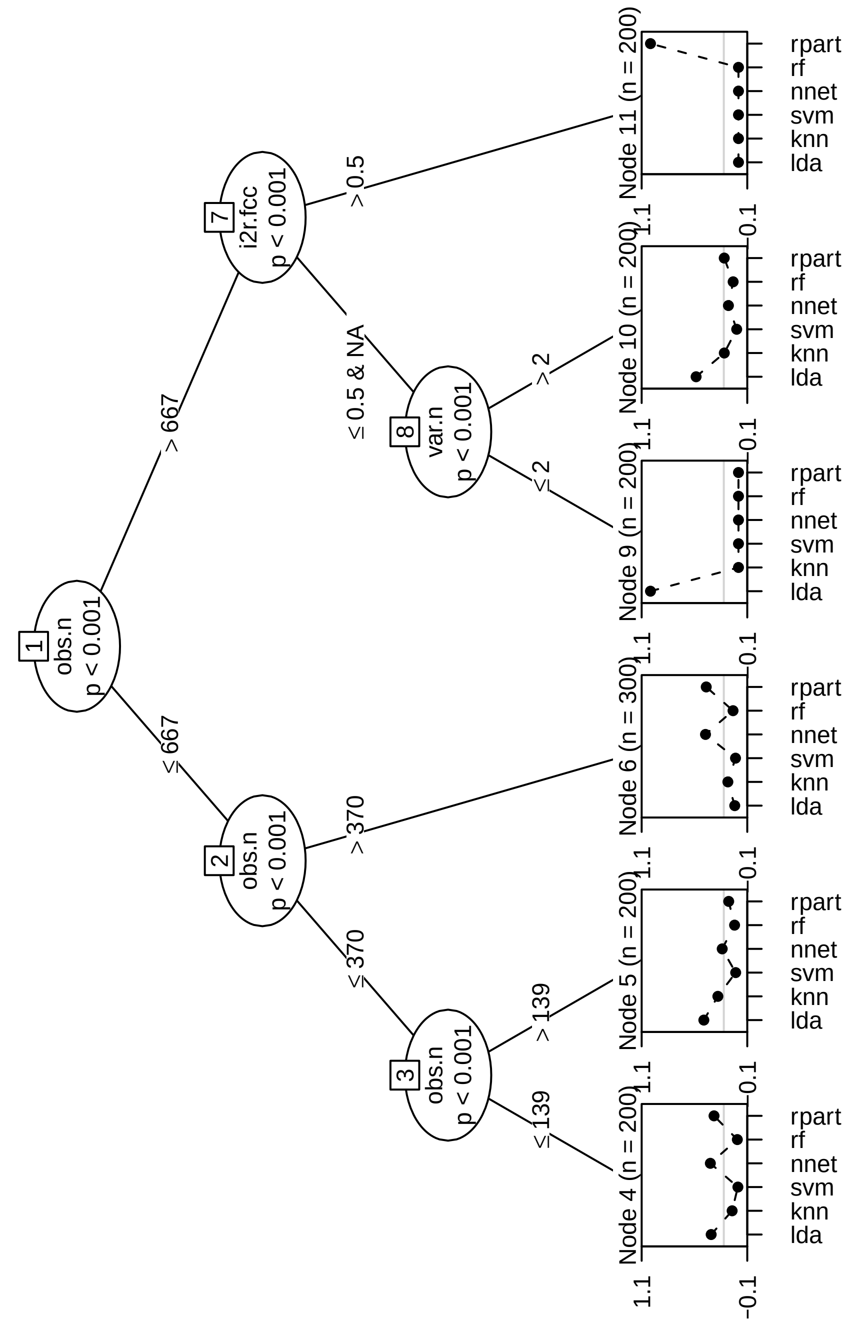


Figure 5.5.: Partitioned paired comparison model for the thirteen UCI machine learning repository data sets and the six candidate algorithms.

5.5. Summary

This chapter proposes a formal framework to determine the influence of data set characteristics on the performance of learning algorithms. The framework is a combination of the three methods, benchmark experiments, data set characterization and recursively partitioning Bradley-Terry models. Benchmark experiments are used to generate performance information for algorithms and relevant data set characteristics on various data sets. The recursively partitioning Bradley-Terry model then employs splits in characteristics which induce a significant change in the performance-ranking of the algorithms. Advantages of the resulting trees are that (1) they are easy to interpret by means of visualization and (2) from a potentially large number of data set characteristics those that correspond to a significant change in the performances of the algorithms are automatically detected.

The approach can be used both for exploring the impact of characteristics of a given sample of data sets, like the ones from UCI Machine Learning Repository (Asuncion and Newman, 2007) used in our example, and for analyzing the results of simulation experiments, where certain characteristics of data sets are systematically varied and the aim is to test their effect on the performance of candidate algorithms. In either case, it is important to note that – due to the fact that the number of bootstrap- or subsamples drawn form given data sets, and the number of samples drawn from a data generating process in a simulation study is arbitrary – one can detect very small performance differences with very high power when the number of learning samples B is large (see also Hothorn et al., 2005).

Future work will also include investigations on the stopping criteria for the recursive partitioning algorithm. For example, the number of minimum observations per node can be chosen greater than B as in our example, but it could also be chosen smaller than B, which would result in more than one node for a data set and could uncover different performance-rankings on sub-samples from one data set with different characteristics.

Chapter 6.

Sequential/adaptive benchmark experiments

Benchmark experiments draw B independent and identically distributed learning samples from a data generating process to estimate the (empirical) performance distribution of candidate algorithms. Formal inference procedures are used to compare these performance distributions and to investigate hypotheses of interests (cf. Chapter 1 and Chapter 2). In most benchmark experiments B is a "freely chosen" number, often specified depending on the algorithms' running time to setup experiments which finish in reasonable time. In this chapter we provide first thoughts on how to control B and remove its "arbitrary" aftertaste.

General sequential designs enable, amongst other things, to control the sample size, i.e., B (see, for example, Vandemeulebroecke, 2008). A benchmark experiment can be seen as a sequential experiment as each run, i.e., drawing a learning sample and estimating the candidates' performances is done one by one. Currently, no benefit is taken from this sequential procedure: The experiment is considered as a fixed-sample experiment with B observations and the hypothesis of interest is tested using a test T at the end of all B runs. We propose to take the sequential nature of benchmark experiments into account and to execute a test T successively on the accumulating data. In a first step, this enables to monitor the benchmark experiment – to observe p-value, test statistic and power of the test during the execution of the benchmark experiment. In a second step, this information can be used to make a decision – to stop or to go on with the benchmark experiment.

This chapter reviews the sequential benchmark experiment framework in Section 6.1. Section 6.2 discusses monitoring and its interpretation as indicator for the differences' relevance. In Section 6.3 the challenges of using the sequential information for decision making are discussed, i.e., possibilities for the analysis of accumulating data are presented. Recursive combination tests by Brannath et al. (2002) are then introduced as one first procedure towards a sound and flexible sequential benchmark experiment framework. Section 6.4 demonstrates the principle benefit in real-world applications using the UCI domain already used in the previous chapters. Section 6.5 concludes the chapter with a summary and an outline for future work.

6.1. Sequential framework

Section 1.1 defines a benchmark experiment as follows: Given is a data generating process DGP and we draw $b = 1, \dots, B$ independent and identically distributed learning samples of size n:

$$\mathfrak{L}^b = \{z_1^b, \dots, z_n^b\} \sim DGP$$

There are $K > 1$ candidate algorithms a_k $(k = 1, \dots, K)$ available to solve the underlying learning problem. For each algorithm a_k the function $a_k(\,\cdot \mid \mathfrak{L}^b)$ is the fitted model on the learning sample $\mathfrak{L}^b$. The performance of the algorithm a_k when provided with the learning sample $\mathfrak{L}^b$ is measured by a scalar function $p(\cdot)$:

$$p_{bk} = p(a_k, \mathfrak{L}^b) \sim \mathcal{P}_k(DGP)$$

The p_{bk} are samples drawn from the distribution $\mathcal{P}_k(DGP)$ of the performance measure of the algorithm a_k on the data generating process DGP. For practical issues computing $p(\cdot)$ in different learning tasks we refer to Chapter 2 for supervised learning problems and Dolnicar and Leisch (2010) for unsupervised clustering problems.

Now, given the K different random samples $\{p_{1k}, \dots, p_{Bk}\}$ with B iid samples drawn from the distributions $\mathcal{P}_k(DGP)$ the null hypothesis of interest for most problems is:

$$H_0 \ : \ \mathcal{P}_1 = \dots = \mathcal{P}_K$$

In order to specify an appropriate test procedure T we need to define an alternative to test against. The alternative depends on the optimality criterion of interest, which we assess using a scalar functional ϕ: algorithm a_k is better than an algorithm $a_{k'}$ with respect to a performance measure p and a functional ϕ iff $\phi(P_k) < \phi(P_{k'})$. The test procedure then is:

$$T \begin{cases} H_0: & \phi(\mathcal{P}_1) = \cdots = \phi(\mathcal{P}_K) \\ H_1: & \exists k, k' : \phi(\mathcal{P}_k) \neq \phi(\mathcal{P}_{k'}) \end{cases}$$

Benchmark experiments with this design are considered as fixed-sample experiments, i.e., B is defined, the experiment is executed, and at the end the hypothesis of interest is tested using the test T:

```
For $b = 1, \dots, B$:
    1. Draw learning sample $\mathfrak{L}^b$.
    2. Measure performance $p_{bk}$ of the $k = 1, \dots, K$ candidate algorithms.
Execute test procedure $T$ on the $K$ performance estimations $\{p_{1k}, \dots, p_{Bk}\}$ and make a decision for a given $\alpha$.
```

In most benchmark experiments B is just a freely chosen number without any statistical foundation; in fact the algorithms' running time often is the determining factor. However, the listing shows that the nature of benchmark experiments is sequential. We can take this into account and change the framework to execute test T in each replication (with a few initial replications as burn-in replications):

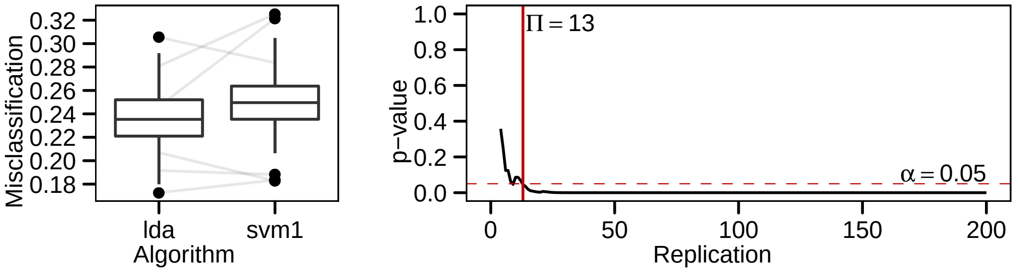
```
Do
    1. Draw learning sample $\mathfrak{L}^b$.
    2. Measure performance $p_{bk}$ of the $k = 1, \dots, K$ candidate algorithms.
    3. Execute test procedure $T$ on the $K$ performance estimations $\{p_{1k}, \dots, p_{bk}\}$.
While no decision for a given $\alpha$ (and $b \leq B$).
```

Now, in a first step this enables to monitor the benchmark experiment: we can observe the p-value on the accumulating data, and see, for example, how the decision would be after b replications, and after how many replications it is final. Anyhow, in this setup the experiment has B replications and the Bth test is the one used for the final decision. So, in a second step (group-)sequential and adaptive test methodology can be used to draw decisions during the experiment. Based on the particular method this enables to stop the experiment early, to change the null hypothesis or even to take off one candidate algorithm earlier.

6.2. Monitoring

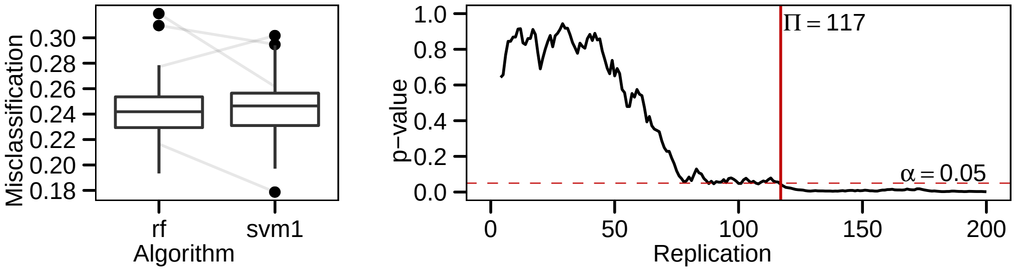
Monitoring benchmark experiments means that in every replication the hypothesis of interest is tested on the accumulating performance measures p_{bk} using the test procedure T. The test results, mainly the p-values, are observed and interpreted. One interesting observation is the number of replications from then on the p-value is continuously smaller than a given α; we define this as the point of consecutively significance Π. If there is such a point available Π ranges between 1 and B; otherwise, we define $\Pi = \infty$ or more precise $\Pi = \infty_B$. We propose to use the point of consecutively significance as a measure on how big the difference between the algorithms is: the bigger Π the bigger is the algorithms' difference based on the performance measure p. If one beliefs in the heuristic "relevant differences are often big differences" Π is an indicator for relevance as well.

In order to present this monitoring aspect of sequential benchmarking we use simple exemplar benchmark experiments which represent three typical benchmark situations. They follow this setup: DGP is the Pima Indians Diabetes (`PmID`) data set $\mathfrak{L}$ with bootstrapping as resampling scheme to draw $B = 200$ learning samples $\mathfrak{L}^b$. The candidate algorithms are the linear discriminant analysis (`lda`), a support vector machine with the cost parameter $C = 1.00$ (`svm1`), a support vector machine with the cost parameter $C = 1.01$ (`svm2`) and a random forest (`rf`). The performance p is the misclassification error measured on validation samples defined as out-of-bag samples $\mathfrak{L} \setminus \mathfrak{L}^b$. The individual three exemplar benchmark experiments compare two algorithms at a time, i.e., test if algorithm a_1 is better than algorithm a_2. We use the non-parametric (one-sided) Wilcoxon Signed Rank test (with three burn-in replications) as test procedure T and define $\alpha = 0.05$. This reflects the situation where an existing algorithm a_1 solves the given problem and a newly developed algorithm a_2 comes up in literature and is now.



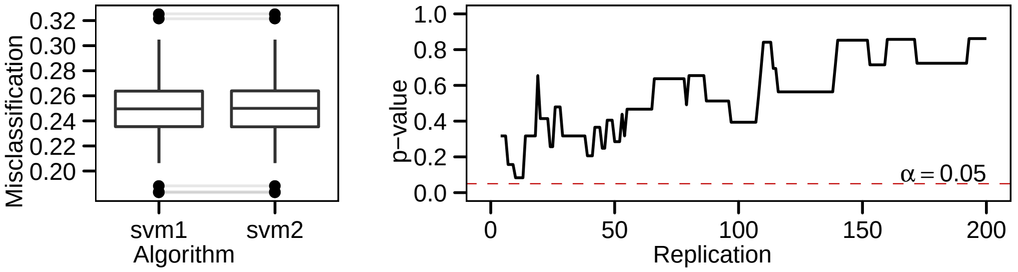
(a) (b)

Figure 6.1.: Scenario 1, different algorithm performances, `lda` versus `svm1`: (a) Misclassification error; (b) Sequential p-value.



(a) (b)

Figure 6.2.: Scenario 2, similar algorithm performances, `rf` versus `svm1`: (a) Misclassification error; (b) Sequential p-value.



(a) (b)

Figure 6.3.: Scenario 3, equal algorithm performances, `svm1` versus `svm2`: (a) Misclassification error; (b) Sequential p-value.

Scenario 1 – Different algorithm performances. The first scenario illustrates a clear situation, i.e., one algorithm (`lda`) is clearly better than the other (`svm1`). Figure 6.1a shows box plots of the misclassification errors (the sample dependency is only shown for samples which are hard to solve for at least one algorithm). After $B = 200$ replications the Friedman test procedure T rejects the null hypothesis of no algorithms' difference with a p-value of $1.1102 \cdot 10^{-16}$, i.e., `lda` $\prec$ `svm1` (see Section 2.4 for the definition of the preference relation $\prec$). The sequentially calculated p-value in Figure 6.1b displays that the point of consecutively significance Π is after 13 replications. So the performance difference between these two algorithms is big, only a few replications are needed to have a significant one. Using this information a decision-making framework could save a number of replications in this benchmark experiment.

Scenario 2 – Similar algorithm performances. The second scenario illustrates a not so clear situation, i.e., both algorithms (`rf` versus `svm1`) solve the learning problem nearly equally (Figure 6.2a). However, after $B = 200$ replications the Friedman test rejects the null hypothesis with a p-value of 0.0032, i.e., `rf` $\prec$ `svm1`. The sequential p-value plot in Figure 6.2b shows that there is a "wild burn-in phase" at the beginning (0 to 70), a phase where the p-value scatters around 0.05 (70 to 110), and then after 117 replications the p-value is consecutively under $\alpha = 0.05$. In comparison to Scenario 1 the difference between the two algorithms is small, a larger number of replications is needed to make it visible. This is the well known effect that one can make arbitrary small differences significant when drawing enough samples. Here, Π indicates that even the difference is significant it may be a small (not relevant) one.

A confirmation of monitoring its usefulness becomes explicit if supposed that we defined $B = 100$ for this scenario. In the sequential p-value plot we see that after 100 replications the p-value is below α, but a few replications before (and after) it is above the line:

replication:	95	96	97	98	99	100
p-value:	0.0532	0.0758	0.0792	0.0721	0.0620	0.0483

The final test after 100 replications reports a significant difference, but $\Pi = 100$ warns the researcher about the marginal result. From a decision-making framework additional replications are expected to confirm the decision.

Scenario 3 – Equal algorithm performances. The third scenario illustrates an undecidable situation, i.e, both algorithms (`svm1` versus `svm2`) solve the learning problem equally (Figure 6.3a). In the sequential p-value plot (Figure 6.3b) we see that the p-value increases continuously after a short burn-in phase (0 to 40). The test can not reject the null hypothesis (regardless of the concrete B), $\Pi = \infty$. In this scenario we expect from a decision-making framework to detect the non-difference, "accept" the null hypothesis, and save a number of replications.

6.3. Decision making

The typical benchmark scenarios in the previous section already hint at the possibilities of a benchmark experiment framework with sequential decision making: execute a benchmark experiment as long as needed, i.e., until H_0 is rejected or "accepted" (i.e., failed to reject).

Sequential analysis was pioneered by Wald (1945). Since then a lot of research was done, mainly in the field of pharmaceutical statistics with focus on clinical trials (for comprehensive reviews, see for example Wassmer, 2000; Hellmich and Hommel, 2004; Vandemeulebroecke, 2008). The analysis of accumulating data distinguish in three main threads (Vandemeulebroecke, 2008): strictly (or purely) sequential, group sequential and adaptive (or flexible) designs. In strictly sequential designs, observations are obtained one by one, and the experiment can be stopped at any time. Group sequential designs sample groups of observations, and the experiment can usually be stopped at any interim analysis, that is, after any of these groups. Any design that allows more flexibility, e.g., to change hypothesis, group sample size, etc., is called adaptive design. All these procedures deal with repeated significance testing (first addressed by Armitage et al., 1969): Testing not once but multiple times on accumulating data causes the inflation of the probability of the error of the first kind, i.e., the probability of rejecting the global null hypothesis when in fact this hypothesis is true; also known as alpha inflation.

For the sequential benchmark experiment framework we require that (1) decisions can be made for any arbitrary hypothesis of interest (2) using any appropriate test procedure T. These requirements ensure that analyses which can be done in the fixed-sample framework can be done in the sequential framework as well. Considering this, adaptive designs based on p-value combination functions and corresponding decision boundaries meet the needed requirements. The following section discusses a variant of p-value combination functions applicable for benchmark experiments.

6.3.1. Recursive combination tests

Recursive combination tests are defined by Brannath et al. (2002) as an adaptive test procedure based on the recursive application of two-stage combination tests. In the following we introduce the recursive combination tests principle using Fisher's product test and review the theory as required. For the complete theoretical validation of recursive combination tests we refer to the original publication by Brannath et al. (2002).

A one-sided null hypothesis H_0 is tested at level α in a two-stage combination test as follows: In the planning phase of Stage 1, the decision boundaries α_{01} and α_{11} with $0 \leq \alpha_{11} \leq \alpha \leq \alpha_{01} \leq 1$, and a p-value combination function $C(\cdot, \cdot)$ with its corresponding critical value c (dependent on α, α_{01}, and α_{11}) are defined. The p-value p_1 of Stage 1 is computed in the performing phase and then analyzed in the interim phase of Stage 1, i.e., H_0 is rejected if $p_1 \leq \alpha_{11}$, H_0 is accepted if $p_1 > \alpha_{01}$ and Stage 2 is performed if $\alpha_{11} < p_1 \leq \alpha_{01}$. In the latter case the Stage 2 p-value p_2 is computed on newly drawn learning samples and H_0 is rejected iff $C(p_1, p_2) \leq c$, otherwise accepted. A prominent choice for a two-stage combination test is Fisher's product test,

$$\begin{aligned} C(p_1, p_2) &= p_1 \cdot p_2 \text{ with} \\ c &= \frac{\alpha - \alpha_{11}}{\ln \alpha_{01} - \ln \alpha_{11}}. \end{aligned}$$

If stage-wise order of the sample space is assumed, a global p-value p can be defined as a function $q(p_1, p_2)$; in case of Fisher's product test this function is defined as

$$q(p_1, p_2) = \begin{cases} p_1, & p_1 \leq \alpha_{11} \text{ or } p_1 > \alpha_{01} \\ \alpha_{11} + p_1 \cdot p_2 \cdot (\ln \alpha_{01} - \ln \alpha_{11}), & p_1 \in (\alpha_{11}, \alpha_{01}] \text{ and } p_1 \cdot p_2 \leq \alpha_{11} \\ p_1 \cdot p_2 + p_1 \cdot p_2 \cdot (\ln \alpha_{01} - \ln p_1 \cdot p_2), & p_1 \in (\alpha_{11}, \alpha_{01}] \text{ and } p_1 \cdot p_2 \geq \alpha_{11}. \end{cases}$$

With this ordering, rejections at the first stage are considered to provide more evidence against H_0 than on the second stage, whereas acceptances at Stage 1 are assumed to support less evidence. A further design requirement is that the p-values p_1 and p_2 are p-clud (Brannath et al., 2002), meaning that under

H_0 the distributions of p_1 and p_2 conditional on p_1 are larger or equal to the uniform distribution on $[0, 1]$. This is fulfilled if independent samples are drawn at each stage and the test T is a level α test for any pre-chosen significance level α.

The generalization to an undefined number of stages is obvious – perform another two-stage combination test as second test, as third test, etc, until a final decision is reached. This requires for each stage t the definition of a conditional critical value c_t and conditional decision boundaries α_t, α_{0t} and α_{1t}. For Fisher's product test the conditional critical value c_t is defined as

$$c_t = \frac{\alpha_t - \alpha_{1t}}{\ln \alpha_{0t} - \ln \alpha_{1t}},$$

with α_t the conditional significance level computed by $\alpha_t^* = \frac{c_{t-1}}{p_{t-1}}$ $(\alpha_1 = \alpha)$. The conditional decision boundaries α_{1t} and α_{0t} are determined according to the constraint

$$\alpha_{1t} < \alpha_t^* \leq \alpha_{0t}. \tag{6.1}$$

This ensures that $p_t \leq \alpha_{1t}$ implies $p \leq \alpha$ and $p_t > \alpha_{0t}$ implies $p > \alpha$. The global p-value p is then computed by backward recursion, i.e.,

$$p = q(p_1, q(p_2, \ldots, q(p_{t-1}, p_t))). \tag{6.2}$$

In summary, the flowchart in Figure 6.4 represents the complete recursive combination test procedure. As one advantage of adaptive tests is that all information available after performing one stage may be used for the next stage, the flowchart also shows the possible design adaptions in each planning phase of a stage (e.g., changing the test procedure T_t or the number of learning samples B_t).

Two-sided null hypotheses. The generalization to two-sided null hypotheses is based on the fact that any two-sided null hypothesis H_0 at level α can be conducted with (two) one-sided adaptive test procedures at level $\alpha/2$; Wassmer (2000) and Hellmich and Hommel (2004) provide details. For the present, we focus on the one-sided null hypotheses, however, this generalization is a major part of our future research (see discussion in Section 6.5).

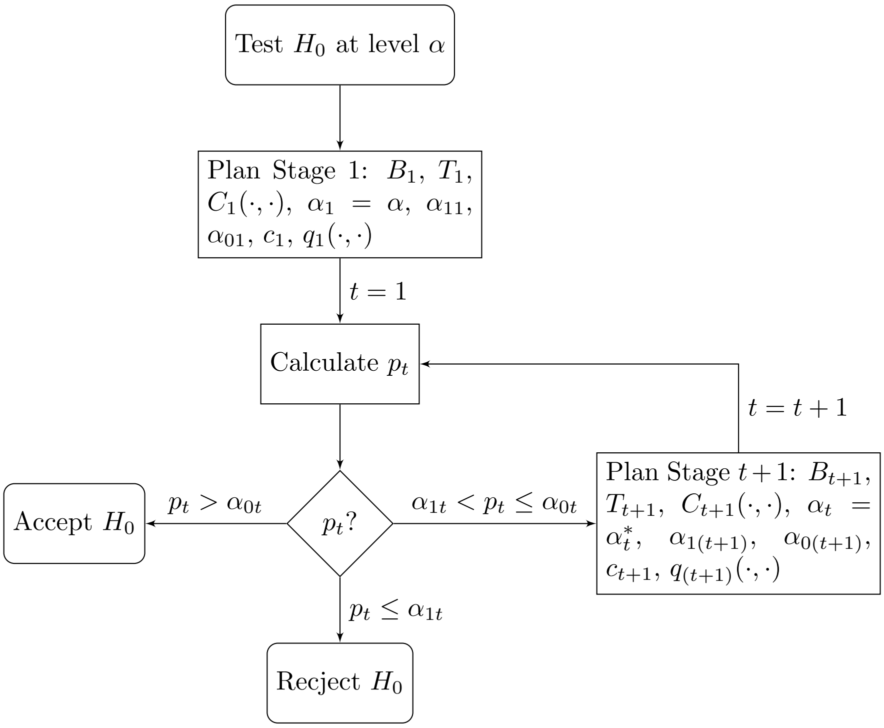

Test H_0 at level α
Plan Stage 1: B_1, T_1, $C_1(\cdot,\cdot)$, $\alpha_1 = \alpha$, α_{11}, α_{01}, c_1, $q_1(\cdot,\cdot)$
$t = 1$
Calculate p_t
$t = t + 1$
Plan Stage $t+1$: B_{t+1}, T_{t+1}, $C_{t+1}(\cdot,\cdot)$, $\alpha_t = \alpha_t^*$, $\alpha_{1(t+1)}$, $\alpha_{0(t+1)}$, c_{t+1}, $q_{(t+1)}(\cdot,\cdot)$
Accept H_0
$p_t > \alpha_{0t}$
p_t?
$\alpha_{1t} < p_t \leq \alpha_{0t}$
$p_t \leq \alpha_{1t}$
Recject H_0

Figure 6.4.: Flowchart representing the recursive combination test procedure. Allowing undecidable endings extends the decision block with another stop case, for example the case when $\sum_t B_t$ is smaller than a predefined maximum number of replications B.

6.3.2. Recursive combination tests in benchmark experiments

Applying recursive combination tests to benchmark experiments is (almost) along the lines of the framework represented in Figure 6.4. In fact, the node "Calculate p_t" simply is the execution of a fixed-design benchmark experiment with the components defined in its planning phase "Plan Stage p_{t-1}". However, there are differences between clinical trials – where sequential analysis is usually applied – and benchmark experiments that affect the application of sequential analysis fundamentally:

1. In contrast to clinical trials, with benchmark experiments it is easy and (comparatively) cheap to make additional replications until a final decision is reached; i.e., to reject or accept H_0. So, theoretically, in any situation a decision can be made. (In practice, however, sometimes it makes sense to stop after a large number of replications.)

2. Benchmark experiments are computer experiments often executed using remote servers, clusters, or other High-Performance Computing technologies. Therefore, decisions that are to be made in the interim and planning phases (e.g., to stop or to go on, to change the number of replications, or to change some decisions boundaries) need to be automatized soundly. ("Interactive" interim and planning phases would be an alternative option.)

3. Benchmark experiments usually compare more than two candidate algorithms simultaneously (i.e., multiple comparisons problems) as to a set of performance measures (i.e., multiobjective optimizations). In this case the procedure has to perform replications until a decision is made for each of the performance measures.

4. Benchmark experiments (often) use resampling schemes to randomly draw learning samples and corresponding test samples from a fixed data set (e.g., bootstrapping with out-of-bag observations). This leads to non-independent p-values. Independent p-values is important for Fisher's product test. However, the p-values' correlation vanishes with increasing data set size, we can rely on the asymptotic.

Aim of this chapter is to investigate the principle feasibility of sequential benchmark experiments, therefore we focus on the following basic scenario: First, we are interested on controlling the number of replications B and not on adapting the design. Therefore, the tests T_t and the combination tests C_t are the same for all stages; and for simplicity we define equal numbers of learning samples B_t for each stage. So, the only adaption in each planning phase is the (automatic) determination of the decision boundaries α_{0t} and α_{1t} according to Constraint (6.1). Second, we investigate one-sided null hypotheses of two candidate algorithms according to one performance measure; Point 3 of the enumeration above is the major part of our future research (along with the generalization to two-sided null hypotheses; see discussion in Section 6.5).

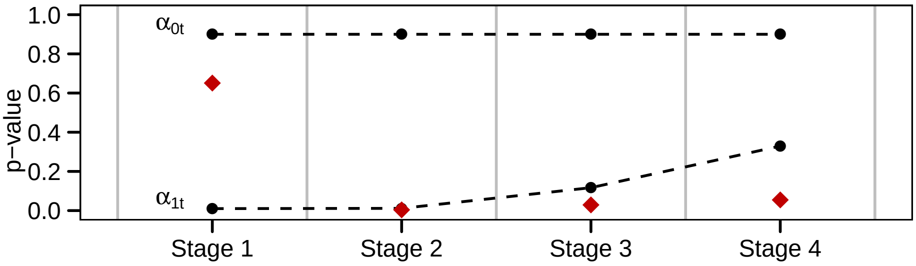


Figure 6.5.: Recursive combination test procedure applied on Scenario 2, similar algorithm performances, `rf` versus `svm1`.

Scenario 2 (Section 6.2) of the exemplar benchmark experiments is used to present and discuss recursive combination tests in benchmark experiments. For this scenario, monitoring shows that $\Pi = 117$, i.e., after 117 replications the p-value computed on the accumulating data is consecutively under $\alpha = 0.05$ (cf. Figure 6.2). Therefore, we expect the rejection of the null hypothesis before all $B = 200$ replications are used. We split the scenario into four stages, i.e., $B_t = \frac{B}{4} = 50$, and define $\alpha = \alpha_1^* = 0.05$, $\alpha_{11} = 0.01$, $\alpha_{01} = 0.9$. The rule for the conditional decision boundaries is defined as $\alpha_{0t} = \frac{\alpha_t^*}{1.2}$ and $\alpha_{1t} = \alpha_{1(t-1)}$; in each stage a small fraction of the conditional significance level is spent for the rejection boundary, whereas the acceptance boundary is unchanged. Figure 6.5 visualizes the result; the concrete values of the individual stages are:

	c_t	α_t^*	α_{1t}	p-value	α_{0t}
Stage 1:	0.0089	0.0500	0.0100	0.6509	0.9000
Stage 2:	0.0005	0.0137	0.0114	0.0037	0.9000
Stage 3:	0.0115	0.1404	0.1170	0.0290	0.9000
Stage 4:	0.0655	0.3952	0.3293	0.0545	0.9000

The conditional significance level α_t^* and therefore the lower decision boundary α_{0t} increases with each stage which restricts the “undecidable area” more and more. The p-value of the first stage does not allow a decision but then every p-value is lower than the rejection boundary. Therefore, H_0 can be rejected after the second stage and we save 100 replications. The recursively computed global p-value is 0.0209 using Formula (6.2).

6.4. Application example

We use the UCI domain to investigate the benefit of sequential/adaptive benchmarking in a real-world application (the domain is already introduced in Chapter 4; we use all data sets except `mnk3` due to numerical problems). In Section 4.5.1 we set $B = 250$ for the UCI experiment; we now take a look at the decisions of the recursive combination tests for the two leader algorithms of each data set. In detail: Figure 4.2b and Table 4.1 show the results of the fixed-sample experiments. For each data set $m = 1, \ldots, 20$ we take the two leader algorithms $a_{(1)}$ and $a_{(2)}$ such that $a_{(1)}$ has the smaller mean performance than $a_{(2)}$ (ties are broken randomly). For each pair the null hypothesis of an equal misclassification error is tested against $a_{(1)}$ has a smaller misclassification error than $a_{(2)}$. The non-parametric (one-sided) Wilcoxon Signed Rank test is used as test T_t, and Fisher's Product test as combination test C_t. We split each experiment into five stages, i.e., $t = 1, \ldots, 5$, $B_t = \frac{B}{5} = 50$, and define $\alpha = \alpha_1^* = 0.05$, $\alpha_{11} = 0.01$, $\alpha_{01} = 0.9$. The rules for the conditional decision boundaries are defined as $\alpha_{0t} = \frac{\alpha_t^*}{1.2}$ and $\alpha_{1t} = \alpha_{1(t-1)} - \frac{\|\alpha_{1(t-1)} - \alpha_t^*\|}{10}$; in each stage a small fraction of the conditional significance level is spent for the rejection boundary and the the acceptance boundary.

Figure 6.6 shows the results of the recursive combination tests for each data set (the red dots are the stage p-values, the lower dashed lines are the rejection boundaries, the upper dashed lines are the acceptance boundaries). For almost all data sets with a significant difference between the two leader algorithms we see that after two stages the test decisions are clear (`Sonr`, `Crcl`, `musk`, `rngn`, `chss`, `twnr`, `thrn`, `livr`, `Crds`, `HV84`, `PmID`, `Hrt1`). Only the plot for `hptt` shows a different image – all stage p-values are inside the two boundaries, and according to the sequential test procedure no significant difference can be stated after 250 replications. In case of the data sets with non-significantly different leaders we see that at least one stage p-value is always above the acceptance boundary (except for `Sprl`). But the decisions are not as clear as for the sig-

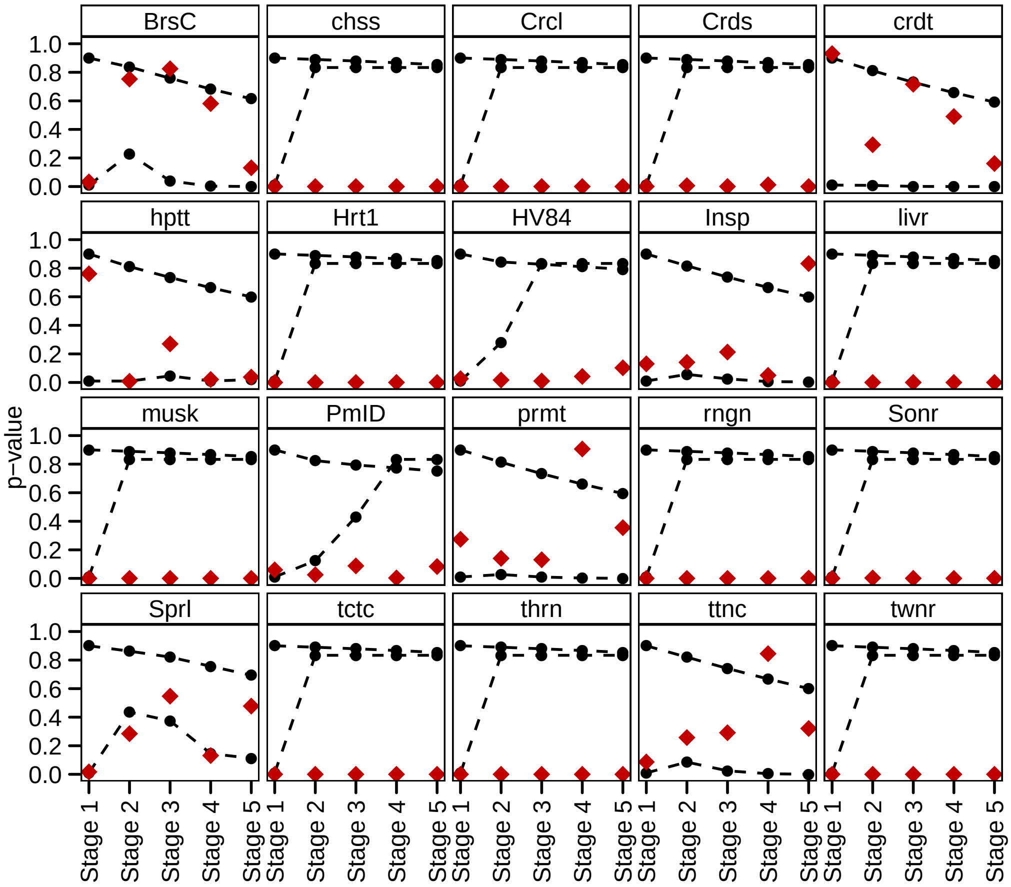


Figure 6.6.: Recursive combination test procedure applied on the two leader algorithms for each data set of the UCI domain (cf. Table 4.1). The red dots are the p-values for each stage, the lower dashed line is the rejection boundary, and the upper dashed line is the acceptance boundary.

nificant differences. On the one hand this can imply that more replications are needed for a clear decision, or on the other hand that the rule for computing the acceptance decision boundaries is not optimal. As already noted in Section 6.3.2, the automatization of the interim and planning phases is a crucial point. Our current believe is, that this is a problem of the automatization, and more research need to be done to have a sound automatization.

6.5. Summary

This chapter provides first thoughts on sequential/adaptive benchmarking. We show that taking the sequential nature of a benchmark experiment into account, enables to monitor and to make decisions during the execution of the experiment. We present an appropriate decision framework based on recursive combination tests with Fisher's product test. For the UCI application example the framework allows early stopping in case of significant decisions, however, in case of decisions towards non-significance the framework does not provide clear statements.

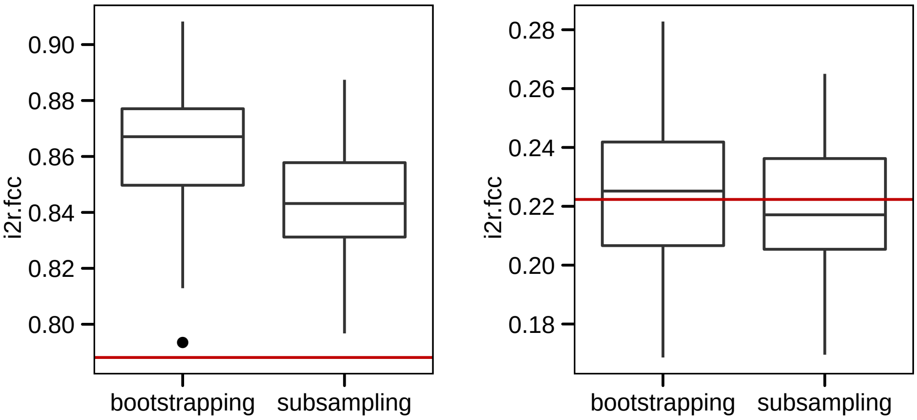
One major question is if the flexibility of adaptive procedures, like the recursive combination tests, is really needed for benchmark experiments. One can imagine that in many experiments, nothing is changed in the planning and interim phases – then less flexibility to the benefit of more efficiency would be appropriate. The most efficient approach is a strictly sequential approach, where a test is executed after each replication. Another approach, little less flexible but more efficient then the adaptive one, is the group sequential approach CRP by Müller and Schäfer (2001). This approach also includes the possibility of data-dependent modifications of the decision boundaries which probably leads to a better automatism. See Schäfer et al. (2006) for a comparison with other sequential/adaptive designs. So, one major part of future work contains intensive evaluation of different frameworks. Another, more general, part is the evaluation of the stability of such sequential/adaptive benchmarking frameworks. Stability can be investigated by calculating "all" possible test decisions under rearrangements of the individual replications (permutations). "Going wild" this idea could also lead to a general approach for the benchmarking of different sequential, group sequential, and adaptive test frameworks.

Chapter 7.

Discussion

Benchmark experiments nowadays are the method of choice to evaluate (new) learning algorithms in most research fields with applications related to statistical learning. Benchmark experiments are an empirical tool to analyze statistical learning algorithms on one or more data sets – to compare a set of algorithms, to find the best hyperparameters for an algorithm, or to make a sensitivity analysis of an algorithm. In essence, the experiments draw observations from theoretically intractable performance distributions of the candidate algorithms. Hothorn et al. (2005) define the theoretical framework, this dissertation uses their formalism and framework as fundament and extends it to a comprehensive toolbox for benchmark experiments.

This dissertation focus on the analysis of benchmark experiments used for algorithm comparisons. We present a systematic approach from exploratory analyses with specialized visualizations via formal investigations and their interpretation as preference relations through to a consensus order of the candidate algorithms – based on a set of performance measures and one or more data sets (Chapter 2 for single data sets, Chapter 4 for domains). Interactive visualizations can contribute a lot to gain further insights into the data created by benchmark experiments. Therefore, an interactive version of the exploratory tool presented in Chapter 2 is presented and discussed in Chapter 3. It is common knowledge that certain characteristics of data sets determine the performance of learning algorithms. A formal framework is presented to investigate such correlations based on recursive partitioning Bradley-Terry model (Chapter 5). In each former chapter we treat the benchmark experiments as fixed-sample experiments, but the nature of benchmark experiments is sequential. In Chapter 6 we provide first thoughts on a sequential framework, discuss its advantages – namely, controlling the number of replications by monitoring the experiments and decision-making during the experiments. Each chapter is concluded with a summary and an outlook for further improvements concerning the chapter's topic. Here, we want to discuss three major tasks of our future work.



(a)

(b)

Figure 7.1.: Box plot of the first canconical correlation characteristic of 100 samples drawn from the (a) `Sonr` and the (b) `Sprl` with bootstrapping and subsampling. The red line is the characteristic computed on the complete data sets.

Effect of resampling methods. Figure 7.1 shows box plots of the first canonical correlation characteristic (see Section 5.2) of 100 samples drawn from the (a) `Sonr` and the (b) `Sprl` data sets with bootstrapping and subsampling. The red line is the characteristic computed on the complete data sets. As we can see on the `Sonar` data set, using resampling can heavily affect the learning sample characteristics and therefore the result of the application where resampling is used (which is well-known). The characterization framework defined in Chapter 5 now provides an environment to systematically investigate such effects.

Sequential/adaptive benchmarking. Sequential/adaptive benchmarking is of great interest for our future work. Investigating different other sequential frameworks with less flexibility but more efficiency is one major task. Also important in our point of view is the systematical investigation of the stage p-values. Even the p-values are asymptotically independent, in practice there is a (small) dependence and its impact on the test decisions should be investigated systematically.

Grammar of benchmarking. A long-term goal is the development of a grammar of benchmarking. By this we mean on the one hand a formal language to specify and to describe arbitrary benchmark experiments and simulation studies (implementation independent); and on the other hand a guide to design experiments in an appropriate manner. For example, the benchmark experiments in this dissertation are described by natural language and by the concrete R code available in the computational details. The first one is too informal and allows a lot of misinterpretation; the second one is in a way too formal – in terms of implementation details – and one readers who do not know the R language and its tricks and treats have problems to understand the experiment. The motivation for the design guide primarily are sensitivity analyses. Currently the researcher is totally free to design the experiment – no rules state how to design "the grid" (as simplification) of different experiment settings through the experiment space. Now, the sequential/adaptive benchmarking approach suggests the idea to embed benchmark experiments into the theory of experimental design: the design elements of benchmark experiments are the candidate algorithms (treatments), the data sets (experimental units), the learning samples (and corresponding validation samples) drawn with a resampling scheme from each data set (blocking factor within treatments), and the performance measures of interest (observational units). Based on such a formal design framework, meaningful automatic recommendations of experiment designs could be possible. And our current opinion is that such a formal language and design framework would lead to better experiments in scientific fields dealing with benchmarking, simulation, and sensitivity analysis.

Part II.

Archetypal analysis

Chapter 8.

General framework

The Merriam-Webster Online Dictionary (2008) defines an archetype as "*the original pattern or model of which all things of the same type are representations or copies*". The aim of archetypal analysis is to find "pure types", the archetypes, within a set defined in a specific context. The concept of archetypes is used in many different areas, the set can be defined in terms of literature, philosophy, psychology and also statistics. Here, the concrete problem is to find a few, not necessarily observed, points (archetypes) in a set of multivariate observations such that all the data can be well represented as convex combinations of the archetypes. An illustrative example in literature is: the Spider-Man personality belongs to the generic Hero archetype, and archetypal analysis tries to find this coherence.

In statistics archetypal analysis was first introduced by Cutler and Breiman (1994). In their paper they laid out the theoretical foundations, defined the concrete problem as a nonlinear least squares problem and presented an alternating minimizing algorithm to solve it. It has found applications in different areas, e.g., in economics (Li et al., 2003; Porzio et al., 2008), astrophysics (Chan et al., 2003) and pattern recognition (Bauckhage and Thurau, 2009). In this part of the dissertation we introduce a general framework for archetypal analysis. Chapter 8 defines a methodological (and computational) framework which allows to generalize the archetypal problem: arbitrary loss functions, arbitrary conditions on the coefficients, or arbitrary matrix norms can be defined. A selection of these computational parts then determines the concrete optimization problem to solve; such a selection is called a family (as commonly called in the R community).

In Chapter 9 we take advantage of the framework and define two new problems – the weighted and the robust archetypal problems – and provide the corresponding family elements. Data points which vary from the majority have great influence on the solution; in fact one outlier can break down the solution. The original algorithm is adapted to be a robust M-estimator and an iteratively reweighted least squares fitting algorithm is presented. As required first step, the weighted archetypal problem is formulated and solved. The algorithm is demonstrated using an artificial example, the Air-Pollution example from Cutler and Breiman (1994), and a detailed benchmark experiment for sensitivity analysis.

Finally, as long-term outlook of the generalization of archetypes we discuss in Chapter 10 that the archetypal algorithm actually is similar to the least squares formulation of centroid clustering algorithms; and one can think of a general framework for the class of k-prototypes-like algorithms (k-means, k-median, Fuzzy k-means, etc.; see, e.g., Steinly, 2006; Leisch, 2006).

The R package archetypes implements the general framework defined in this dissertation's part. In Appendix A.2 we explain the design and the concept of the package; and show how to reproduce the application examples shown in the dissertation.

In the remaining of this chapter we review the theoretical background of classical archetypes in Section 8.1. Section 8.2 presents the structure of the general framework and family elements. And in Section 8.3 we present a real world example – the archetypes of human skeletal diameter measurements – to descriptively introduce archetypal analysis. Note that the mathematical nomenclature in this part of the dissertation is independent of the previous benchmark experiment part (due to compatibility with the publications); the next section defines the notation.

8.1. Archetypal problem

Consider an $n \times m$ matrix X representing a multivariate data set with n observations and m attributes. For given k the archetypal problem is to find the matrix Z of k m-dimensional archetypes. More precisely, to find the two $n \times k$ coefficient matrices α and β which minimize the residual sum of squares

$$\mathrm{RSS} = \|X - \alpha Z^\top\|_2 \text{ with } Z = X^\top \beta \tag{8.1}$$

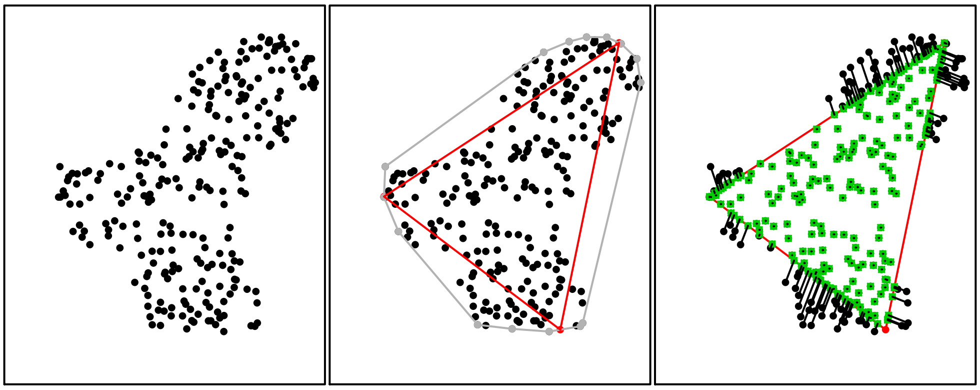


Figure 8.1.: (a) Artificial toy data set. (b) Approximation of the convex hull (outer polygon) by three archetypes (inner triangle). (c) Approximation of the data through the three archetypes and the corresponding α values.

subject to the constraints

$$\sum_{j=1}^{k} \alpha_{ij} = 1 \text{ with } \alpha_{ij} \geq 0 \text{ and } i = 1, \dots, n,$$

$$\sum_{i=1}^{n} \beta_{ij} = 1 \text{ with } \beta_{ij} \geq 0 \text{ and } j = 1, \dots, k.$$

The constraints imply that (1) the approximated data are convex combinations of the archetypes, i.e., $X = \alpha Z^\top$, and (2) the archetypes are convex combinations of the data points, i.e., $Z = X^\top \beta$. $\| \cdot \|_2$ denotes the Euclidean matrix norm.

Cutler and Breiman (1994) present an alternating constrained least squares algorithm to solve the problem: it alternates between finding the best α for given archetypes Z and finding the best archetypes Z for given α; at each step several convex least squares problems are solved, the overall RSS is reduced successively. The following Section 8.2 provides concrete details on the algorithm.

Figure 8.1a shows an artificial two-dimensional toy data set. We use this simple problem to introduce the methodology throughout this part of the dissertation; the advantage of such a simple problem is that we can visualize the result. Section 8.3 then shows a more realistic example. The toy data set consists of two attributes x and y, and 250 observations. It is generated in a way such that $k = 3$ archetypes are the optimal solution. Figure 8.1b shows the archetypes, their approximation of the convex hull (the inner triangle) and the convex hull of the data (outer polygon). Figure 8.1c shows the approximation of the data through the archetypes and the corresponding α values; as we can see, all data points outside the approximated convex hull are mapped on its boundary, all data points inside are mapped exactly.

8.2. Archetype algorithm

The description of the classical archetypal analysis in Formula (8.1) defines the basic principle of the estimation algorithm: it alternates between finding the best α for given archetypes Z and finding the best archetypes Z for given α; at each step several convex least squares problems are solved, the overall RSS is reduced successively.

The steps of the general framework for archetypal analysis are the following:

Given the number of archetypes k:

1. Data preparation and initialization: scale data, add a dummy row (see below) and initialize β in a way that the the constraints are fulfilled to calculate the starting archetypes Z.

2. Loop until RSS reduction is sufficiently small or the number of maximum iterations is reached:

   2.1. Find best α for the given set of archetypes Z: solve n convex least squares problems $(i = 1, \ldots, n)$

   $$\min_{\alpha_i} \frac{1}{2} \|X_i - Z\alpha_i\|_2 \text{ subject to } \alpha_i \geq 0 \text{ and } \sum_{j=1}^{k} \alpha_{ij} = 1.$$

   2.2. Recalculate archetypes $\tilde{Z}$: solve system of linear equations $X = \alpha \tilde{Z}^\top$.

2.3. Find best β for the given set of archetypes $\tilde{Z}$: solve k convex least squares problems ($j = 1, \ldots, k$)

$$\min_{\beta_j} \frac{1}{2} \|\tilde{Z}_j - X\beta_j\|_2 \text{ subject to } \beta_j \geq 0 \text{ and } \sum_{i=1}^{n} \beta_{ij} = 1.$$

2.4. Recalculate archetypes Z: $Z = X\beta$.

2.5. Calculate residual sum of squares RSS.

3. Post-processing: remove dummy row and rescale archetypes.

The framework defines a family element for each computational entity; see Appendix A.2 for the relation between the theoretical steps and the computational family elements in case of the archetypes package. The algorithm has to deal with several numerical problems, i.e. systems of linear equations and convex least squares problems. In the following we explain each step in detail.

Solving the convex least squares problems. In Step 2.1 and Step 2.3 several convex least squares problems have to be solved. Cutler and Breiman (1994) use a penalized version of the non-negative least squares algorithm by Lawson and Hanson (1974) (as general reference see,e.g., Luenberger, 1984). In detail, the problems to solve are of the form $\|u - Tw\|_2$ with u, w vectors and T a matrix, all of appropriate dimensions, and the non-negativity and equality constraints. The penalized version adds an extra element M to u and to each observation of T; then

$$\|u - Tw\|_2 + M^2 \|1 - w\|_2$$

is minimized under non-negativity restrictions. For large M, the second term dominates and forces the equality constraint to be approximately satisfied while maintaining the non-negativity constraint. The hugeness of the value M varies from problem to problem and thus can be seen as a hyperparameter of the algorithm. Default value in the archetypes package is 200.

Solving the system of linear equations. In Step 2.2 the system of linear equations

$$\tilde{Z} = \alpha^{-1}X$$

has to be solved. A lot of methods exist, one approach is the Moore-Penrose pseudoinverse which provides an approximated unique solution by a least squares approach: given the pseudoinverse α^+ of α,

$$\tilde{Z} = \alpha^+ X,$$

is solved. Another approach is the usage of QR decomposition: $\alpha = QR$, where Q is an orthogonal and R an upper triangular matrix, then

$$\tilde{Z} = Q^\top X R^{-1},$$

is solved. Default approach in the archetypes package is the QR decomposition using the `solve()` function.

Calculating the residual sum of squares. In Step 2.5 the RSS is calculated. It uses the spectral norm (see, e.g., Golub and Loan, 1996). The spectral norm of a matrix X is the largest singular value of X or the square root of the largest eigenvalue of X^*X,

$$\|X\|_2 = \sqrt{\lambda_{max}(X^*X)},$$

where X^* is the conjugate transpose of X.

Avoiding local minima. Cutler and Breiman (1994) show that the algorithm converges in all cases, but not necessarily to a global minimum. Hence, the algorithm should be started several times with different initial archetypes. It is important that these are not too close together, this can cause slow convergence or convergence to a local minimum.

Choosing the correct number of archetypes. As in many cases there is no rule for the correct number of archetypes k. A simple method the determine the value of k is to run the algorithm for different numbers of k and use the "elbow criterion" on the RSS where a "flattening" of the curve indicates the correct value of k.

Approximation of the convex hull. Through the definition of the problem, archetypes lie on the boundary of the convex hull of the data. Let N be the number of data points which define the boundary of the convex hull, then Cutler and Breiman (1994) showed: if $1 < k < N$, there are k archetypes on the boundary which minimize RSS; if $k = N$, exactly the data points which define the convex hull are the archetypes with RSS $= 0$; and if $k = 1$, the sample mean minimizes the RSS. In practice, these theoretical results can not always be achieved, as we show in the software presentation in Eugster and Leisch (2009).

8.3. Illustrative example

In this section we apply archetypal analysis on an interesting real world example: In Heinz et al. (2003) the authors took body girth measurements and skeletal diameter measurements, as well as age, weight, height and gender on 247 men and 260 women in their twenties and early thirties, with a scattering of older man and woman, and all physically active. We are only interested in a subset `skel2`, the skeletal measurements and the height (all measured in centimeters).

The skeletal measurements consist of nine diameter measurements: biacromial (Biac), shoulder diameter; biiliac (Biil), pelvis diameter; bitrochanteric (Bitro) hip diameter; chest depth between spine and sternum at nipple level, mid-expiration (ChestDp); chest diameter at nipple level, mid-expiration (ChestDiam); elbow diameter, sum of two elbows (ElbowDiam); wrist diameter, sum of two wrists (WristDiam); knee diameter, sum of two knees (KneeDiam); ankle diameter, sum of two ankles (AnkleDiam). See the original publication for a full anatomical explanation of the skeletal measurements and the process of measuring. We use basic elements of Human Modeling and Animation to model the skeleton and create a schematic representation of an individual; Figure 8.2 shows a generic individual with explanations of the measurements. For visualizing the full data set, parallel coordinates with axes arranged according to the "natural order" are used (see Figure 8.3).

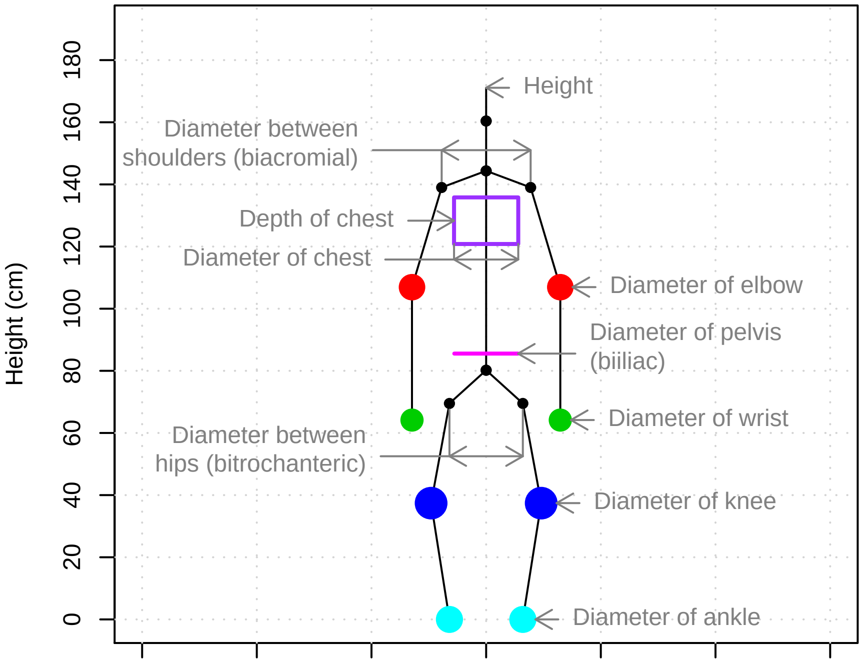


Figure 8.2.: Generic skeleton with explanations of the measurements.

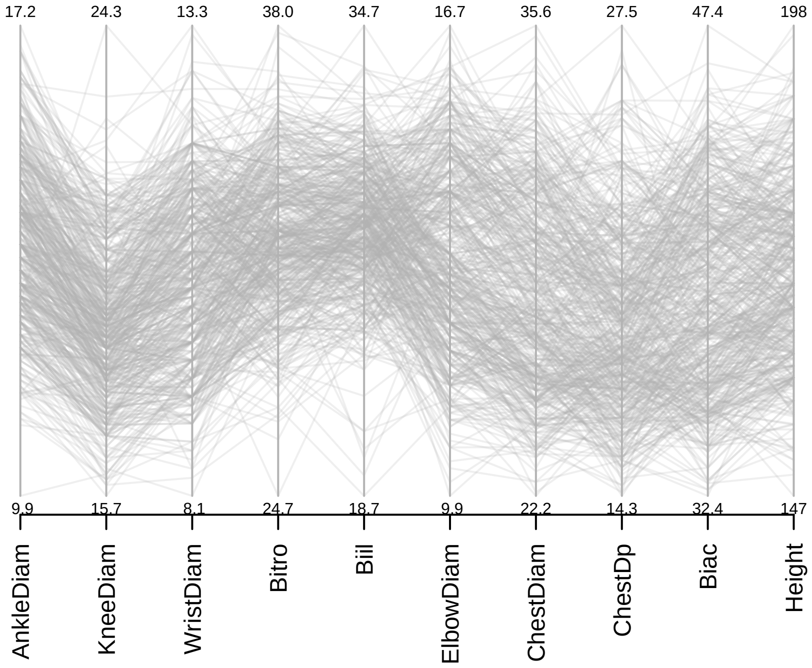


Figure 8.3.: Parallel coordinates of the `skel2` data set.

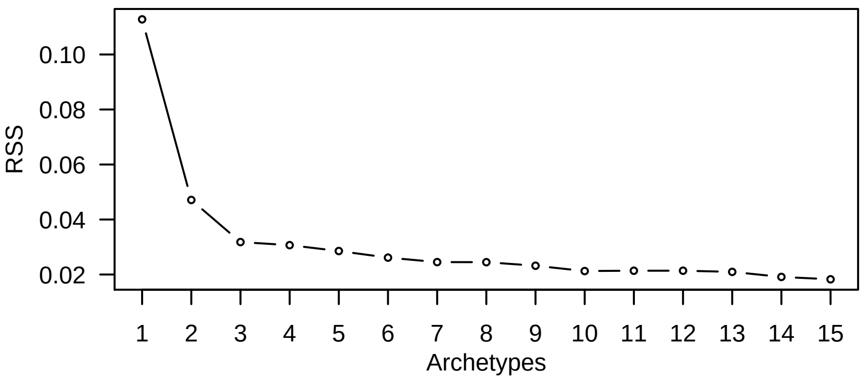


Figure 8.4.: Scree plot of the residual sum of squares.

At first view no patterns in Figure 8.3 are visible and it is difficult to guess a meaningful number of archetypes. Therefore, we calculate the archetypes for $k = 1, \dots, 15$ with three repetitions each time; the corresponding curve of the best model in each case is available in Figure 8.4. And according to the "elbow criterion" $k = 3$ or maybe $k = 7$ is the best number of archetypes. Corresponding to Occam's razor we proceed with three archetypes; Table 8.1 shows the raw measurements and Figure 8.5 as a bar plot in relation to the original data.

Archetype 2 (gray) represents individuals which are "huge" in all measurements; on the other hand, Archetype 3 (lightgray) represents individuals which are "small". Archetype 1 (darkgray) represents individuals with average measures except the bitrochanteric and biiliac – the meaning of this is best visible when looking at the data with gender information (men are blue, women are green colored, with alpha transparency) and the archetypes (red) (see Figure 8.7). Here we see that Archetype 2 reflects the primary difference between men and women in body structure – the comparatively wider hip and pelvis of women. A verification of this interpretation can be done by looking at the coefficients α and see how much each archetype contributes to the approximation of each individual observation. For three archetypes, a ternary plot (see Figure 8.6) is a usable graphical representation. Clearly, males cluster close to Archetype 2 and women mixes mainly the first and the third archetype. Therefore, males are primarily approximated by Archetype 2, women by linear combination of archetypes 1 and 3. For more than three archetypes parallel coordinates with an axis for each archetype projecting the corresponding coefficients (in range $[0, 1]$) can be used to investigate the coefficients α.

Measurements	Archetype 1	Archetype 2	Archetype 3
AnkleDiam	13.1850	15.9613	12.0225
KneeDiam	18.5906	21.0452	16.4354
WristDiam	9.7641	12.2378	9.2843
Bitro	34.4512	34.4547	27.2647
Biil	31.2224	29.3412	22.5222
ElbowDiam	12.2306	15.7404	11.2917
ChestDiam	26.1508	32.7796	24.3042
ChestDp	18.3922	22.9052	15.9521
Biac	36.3124	43.8883	34.5829
Height	167.0075	186.6854	157.8017

Table 8.1.: Raw measurements of the skeletal archetypes.

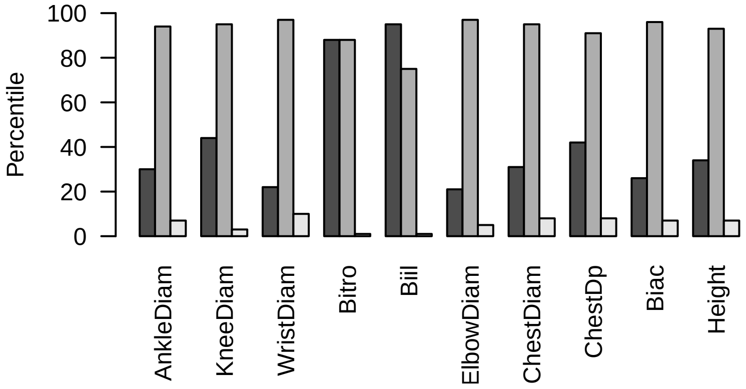


Figure 8.5.: Bar plot visualizing the three archetypes.

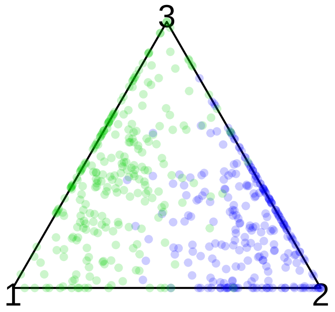


Figure 8.6.: Ternary plot visualizing the coefficients α, colored according to the gender.

Finally, we visualize the three skeleton archetypes itself, Figure 8.8. The left skeleton visualizes Archetype 2 with the wider hip and pelvis; the middle skeleton visualizes Archetype 1 which is “huge”, the right skeleton visualizes Archetype 3 which is “small” in all measurements. Assume that we want to automatically fabricate blue worker overalls from given templates (e.g., paper patterns). Combinations of archetypes 1 and 3 gives overalls in different sizes which each have similar width at shoulder and waist. If we add Archetype 2, we get overalls with a wider waist (compared to the shoulder).

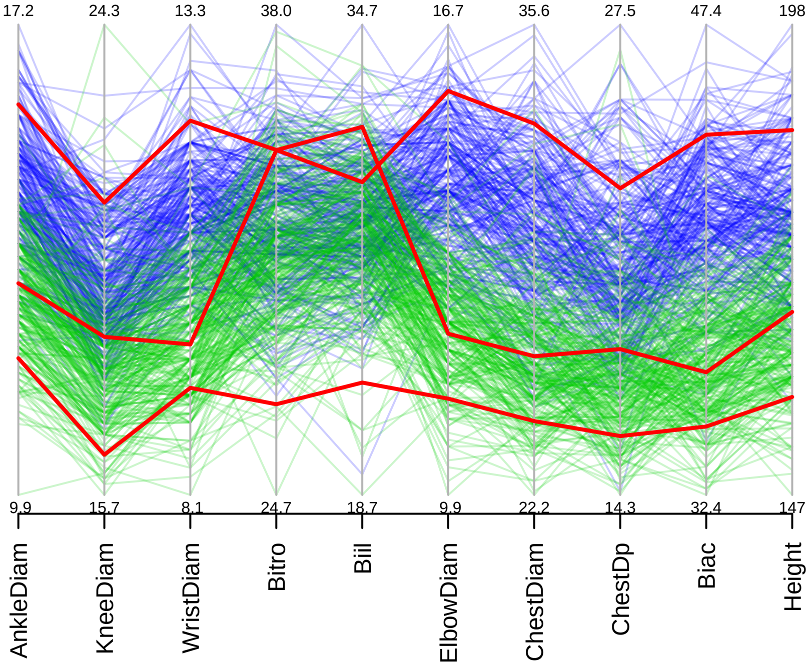


Figure 8.7.: Parallel coordinates with colors related to the gender and the three archetypes.

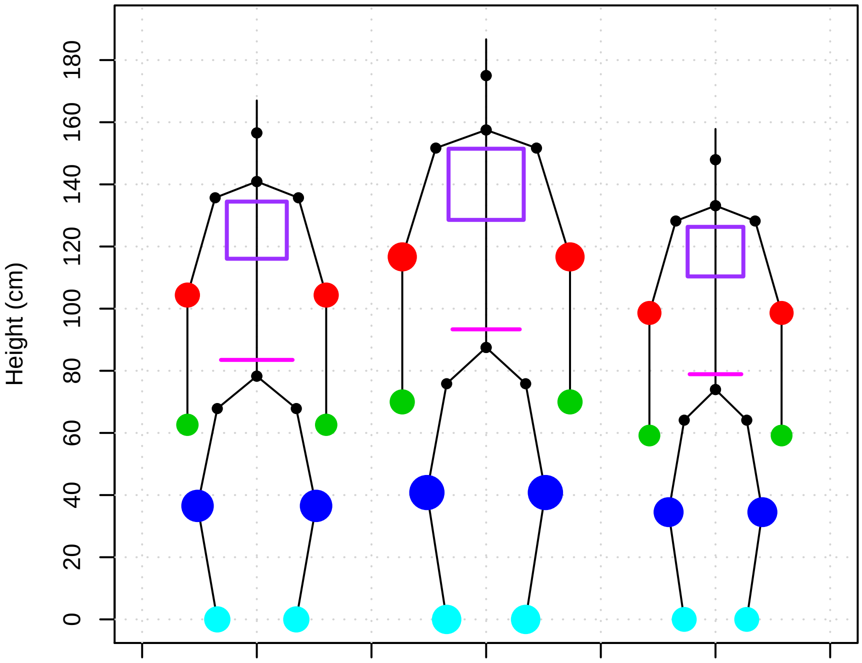


Figure 8.8.: Skeleton plot visualizing the three archetypes.

Chapter 9.

Weighted and robust archetypes

Archetypal analysis approximates the convex hull of the data set – this suggests itself that data points which “behave differently from the large majority of the other points” (Morgenthaler, 2007) have a great influence on the solution. In fact, the farther a data point is from the center of the data set the more influence it has on the solution. Although archetypal analysis is about the data set boundary, practice has shown that in many cases one primarily is interested in the archetypes of the large majority than of the totality. For example, Li et al. (2003) look at extreme consumers in segmenting markets – it is obvious that the extreme consumers should not be total outliers but related to the majority of the consumers. This chapter adapts the original archetypes estimator to be a robust M-estimator (Huber and Ronchetti, 2009) and presents an iteratively reweighted least squares (IRLS) fitting algorithm. This enables a robust analysis in terms of Rousseeuw and Leroy (2003, defined for robust regression): “*A robust analysis first wants to fit* an archetypal analysis *to the majority of the data and then to discover the outliers as those points which possess large residuals from that robust solution.*”

Robust archetypal analysis formulated in this way is based on weighting the residuals and observations respectively. On this account, the chapter formulates and solves the weighted archetypal problem in a first step. Weighted archetypal analysis enables to represent additional information available from the data set, like the importance of observations or the correlation between observations.

The chapter is organized as follows. In Section 9.1, the breakdown point of the original archetypes algorithm defined in Chapter 8 is discussed. In Section 9.2 the weighted archetypal problem is solved. Based on that, Section 9.3 introduces the robust M-estimator, the corresponding iteratively reweighted

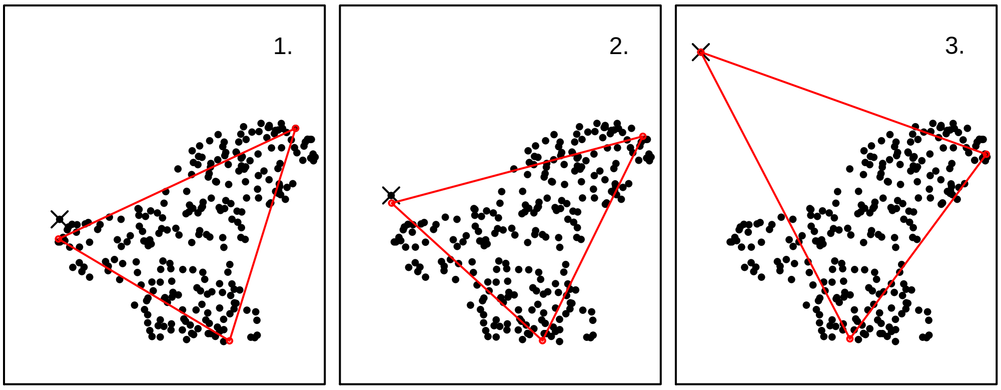


Figure 9.1.: Behavior of the archetypes (triangle) when one data point (cross) moves away.

least squares problem and the fitting algorithm. Each step is illustrated using the two-dimensional artificial toy example already presented in the previous chapter (cf. Figure 8.1). In Section 9.4 the robust algorithm is applied on the Air-Pollution data set (slightly modified to contain outliers) which is used in the original archetypal analysis paper by Cutler and Breiman (1994). Section 9.5 presents a structured simulation study to analyze the algorithm's robustness and convergence with respect to data dimension, number of outliers and distance of outliers to the majority of data. Finally, in Section 9.6 the conclusions are given.

9.1. Breakdown point of classical archetypes

In this section we discuss the breakdown point of the original archetypes algorithm introduced in Chapter 8. The breakdown point is the smallest amount of contamination that may cause an estimator to take arbitrary large values. We follow the sample version defined by Donoho and Huber (1983): Given the data set X with n observations, and T, an estimator based on X, we let $\epsilon^*_n(T, X)$ denote the smallest fraction of contaminated observations needed to break down the estimator T.

Here, we discuss the breakdown of the archetypes matrix Z, i.e., estimator $T = Z$. For a given k archetypal analysis reaches the worst possible value, $\epsilon^*_n(Z, X) = 1/n$ for every X; which converges to 0 as $n \to \infty$. To check this fact, suppose that one data point of the toy data set moves away; Figure 9.1

illustrates this scenario. Note how one of the archetypes has gone on to “catch” this outlier observation when the cross data point moves away from the majority of the data. In terms of the minimization problem this means that at one point (related to the distance of the outlier) the RSS is more reduced if the outlier is approximated well, then the remaining data points. Now, take the outlier to infinity to break down the archetype solution with one single outlier.

9.2. Weighted archetypes

In the original archetypal problem, Equation (8.1), each observation and therefore each residual contributes to the solution with equal weight. Remember that X is an $n \times m$ matrix and let W be a corresponding $n \times n$ square matrix of weights. The weighted archetypal problem is then the minimization of

$$\text{RSS} = \|W(X - \alpha Z^\top)\|_2 \text{ with } Z = X^\top \beta. \tag{9.1}$$

Weighting the residuals is equivalent to weighting the data set:

$$\begin{aligned} W(X - \alpha Z^\top) &= W(X - \alpha(X^\top \beta)^\top) = W(X - \alpha\beta^\top X) = \\ &= WX - W(\alpha\beta^\top X) = WX - (W\alpha)(\beta^\top W^{-1})(WX) = \\ &= \tilde{X} - \tilde{\alpha}\tilde{\beta}\tilde{X} = \tilde{X} - \tilde{\alpha}\tilde{Z}^\top \end{aligned}$$

Therefore the problem can be reformulated as minimizing

$$\text{RSS} = \|\tilde{X} - \tilde{\alpha}\tilde{Z}\|_2 \text{ with } \tilde{Z} = \tilde{\beta}\tilde{X} \text{ and } \tilde{X} = WX. \tag{9.2}$$

This reformulation allows the usage of the original algorithm with the additional pre-processing step to calculate $\tilde{X}$ and the additional post-processing step to recalculate α and β for the data set X given the archetypes $\tilde{Z}$.

The weight matrix W can express different aspects. In case of W a diagonal matrix the weights represents some kind of importance of the observations. The weight values are rescaled to the range $[0, 1]$ – values greater than one disperse the data points, and therefore the data set boundary, which is not meaningful in case of archetypal analysis. Furthermore, W can be an arbitrary square matrix, for example, a matrix to decorrelate the observations.

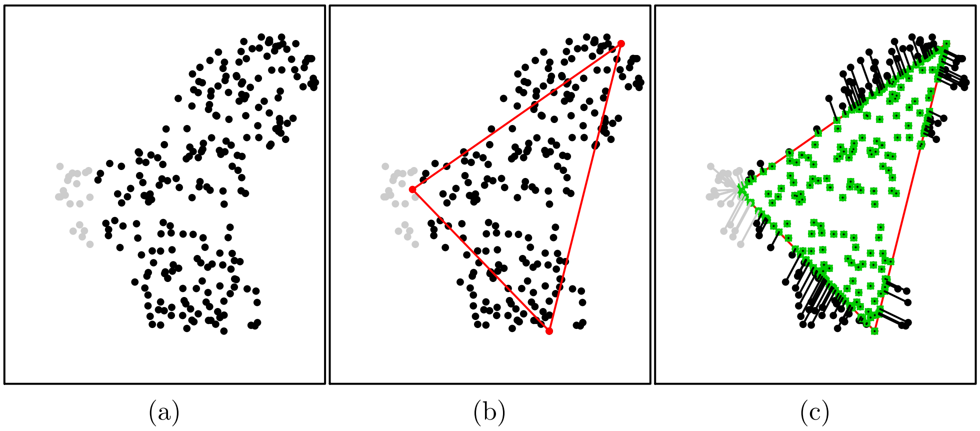


Figure 9.2.: Weighted archetypal analysis where gray data points weight 0.8 and black data points 1.

Figure 9.2 illustrates the weighted archetypal analysis of the toy data set for $k = 3$. (a) Gray data points weight 0.8 and black data points 1). (b) As expected, on the side of the lower weighted data points the corresponding archetype is inside the data set boundary. (c) These data points are mapped on the approximated convex hull boundary, their residuals contribute to the overall weighted RSS.

9.3. Robust archetypes

A popular robust technique is using M-estimators instead of least squares estimators. Let $R = (X - \alpha Z^\top)$ be the matrix of residuals. The standard archetypal analysis tries to minimize the Euclidean (matrix) norm of the residuals, i.e., $\min \|R\|_2$. Here, large residuals have large effects, which privileges outliers. M-estimators try to reduce the effect of outliers by replacing the squared residuals by another function $\rho(\cdot)$ less increasing than the square; this yields the optimization problem $\min \rho(R)$. Such a problem can be reformulated as an iterated reweighted least squares one, i.e., in the tth iteration $\min \|w(R^{(t-1)})R\|_2$ is solved with $w(\cdot)$ a weight function depending on the residuals of the $(t-1)$th iteration. (For general details on transforming the object function into the influence and weight functions we refer to, for example, Huber and Ronchetti, 2009.)

There is a large set of suitable objective functions $\rho(\cdot)$ and corresponding weight functions $w(\cdot)$ available – used, for example, in robust regression (Rousseeuw and Leroy, 2003) and locally weighted regression and scatter plot smoothing (Cleveland, 1979). Note that here the residual R_i of observation i ($i = 1, \ldots, n$) is of dimension m, therefore the one-dimensional distance calculations in the original functions are replaced by the corresponding norm functions. For an example, the (generalized) Bisquare objective $\rho(\cdot)$ and weight $w(\cdot)$ functions are defined as $\rho(R) = \sum_{i=1}^{n} \tilde{\rho}(R_i)$ and $w(R) = \mathrm{diag}(\tilde{w}(R_i))$, $i = 1, \ldots, n$ with R_i the m dimensional residual of the ith observation and

$$\tilde{\rho}(R_i) = \begin{cases} \frac{c^2}{6}(1 - (1 - \|\frac{R_i}{c}\|_1^2)^3), & \text{for } \|R_i\|_1 < c \\ \frac{c^2}{6}, & \text{for } \|R_i\|_1 \geq c \end{cases},$$

$$\tilde{w}(R_i) = \begin{cases} (1 - \|\frac{R_i}{c}\|_1^2)^2, & \text{for } \|R_i\|_1 < c \\ 0, & \text{for } \|R_i\|_1 \geq c \end{cases}.$$

The value c is a tuning parameter; practical application showed that $c = 6s$ with s the median of the residuals unequal to zero works well (following Cleveland, 1979). $\| \cdot \|_1$ denotes the 1-norm; on the lines of the absolute value in the original one-dimensional function definitions.

The iterative reweighted least squares algorithm at step t involves solving the weighted archetypal minimization problem

$$\begin{aligned} R^{(t)} &= \underset{R}{\mathrm{argmin}} \|w(R^{(t-1)})\ R\|_2 \\ &\quad \text{with } R = (X - \alpha Z^\top) \text{ and } Z = X^\top \beta, \end{aligned} \tag{9.3}$$

or according to equation (9.2),

$$\begin{aligned} R^{(t)} &= \underset{R}{\mathrm{argmin}} \|R\|_2 \\ &\quad \text{with } R = (X^t - \alpha Z^{t\top}) \\ &\quad \text{and } Z^t = X^{t\top}\beta,\ X^t = w(R^{(t-1)})X. \end{aligned} \tag{9.4}$$

The original algorithm proposed by Cutler and Breiman (1994) is an iterative alternating constrained least squares algorithm: it alternates between finding the best α for given archetypes Z and finding the best archetypes Z for given α. The algorithm has to deal with several numerical issues, e.g., each step requires the solution of several convex least squares problems. Chapter 8 describes the algorithm in detail. Here, we focus on the additional steps needed to enable weighted and robust archetypal analysis (marked with * in the following listing). Given the number of archetypes k and a weight matrix W (weighted archetypes) and a weight function $w(\cdot)$ (robust archetypes) the algorithm consists of the following steps:

*1. Data preparation: standardize and weight data, $X^0 = WX$.

2. Initialization: define α and β in a way that the constraints are fulfilled to calculate the starting archetypes Z.

3. Loop until RSS reduction is sufficiently small or the number of maximum iterations is reached:

    *3.1. Reweight data: $X^t = w(R^{(t-1)})X$.

    3.... Calculate Z, i.e., α and β given the data X^t.

    3.6. Calculate residuals R^t and residual sum of squares RSS.

*4. Recalculate α and β for the given set of archetypes Z and X.

5. Post-processing: rescale archetypes.

Standardization. Step 1 standardizes the data set to mean 0 and standard deviation 1. The mean is not robust and if outliers are in the data set available, a normalization toward the median is more suitable. Scale and median normalization (e.g., Quackenbush, 2002) is one simple approach we use: Transform the m attributes such that their distributions or their medians are equivalent.

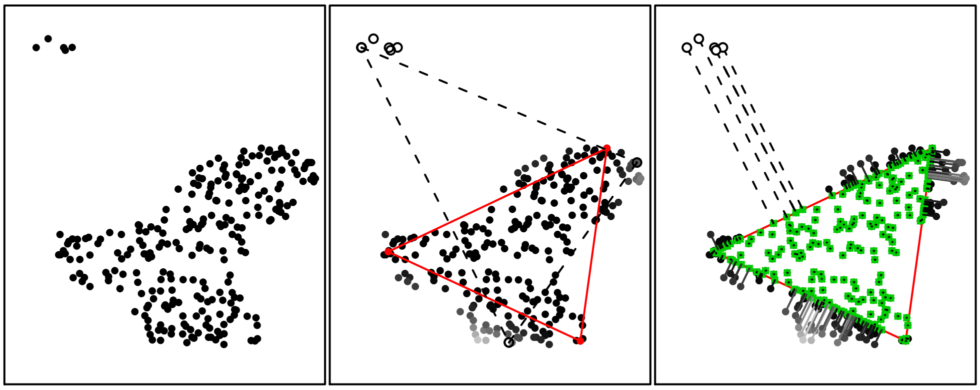
Initialization. Step 2 initializes the archetypes; a good initialization is important as a bad selection can cause slow convergence, convergence to a local minimum or even a non-robust solution. A common approach is to randomly draw the initial archetypes from the complete data set. This can lead to the selection of an outlier as initial archetype. Approaches to select initial archetypes from the majority of the data are, for example, to draw them from the subset of data points which are inside some quantiles in each attribute or which are in the neighborhood of the median. Note that these initialization methods do not ensure a good initialization (i.e., no outliers as initial archetypes) and several starts with different initializations are recommended.

Recalculation. Step 3 (the loop of the algorithm) computes the coefficient matrices α and β and the archetypes Z on the weighted data set X^t. For the final result the coefficient matrices have to be recalculated for the unweighted data set X. Step 4 again (as in each iteration of the loop) solves n and k convex least squares problems to find the best α and the best β given the set of archetypes Z and now the unweighted data set X (for details on the convex least squares problems see Chapter 8).

Computational complexity. The complexity of the algorithm is determined by the complexity of the underlying non-negative convex least squares method ($n + k$ problems per iteration and for the robust algorithm additional $n + k$ problems in Step 4) and the number of iterations until its convergence. In the concrete implementation we use the iterative NNLS algorithm defined by Lawson and Hanson (1974). The authors prove the convergence of the algorithm, but the number of iterations is dependent on the concrete problem. This behavior propagates to the archetypal analysis algorithm for which Cutler and Breiman (1994) also prove its convergence, but again the number of iterations is not fixed. Currently, we are not able to determine the theoretical computational complexity (i.e., Big O notation) for the algorithm. Therefore, Eugster and Leisch (2009) provide a simulation study to show how the (original) algorithm scales with numbers of observations, attributes and archetypes; and Section 9.5 provides a simulation study to compare the convergence of the original and robust algorithms. Note that the implementation is flexible and allows to replace NNLS with other algorithms.

Figure 9.3a shows the robust archetypal analysis of the toy data set extended with five outliers. (b) The dotted line indicates the solution of the original $k = 3$ archetype algorithm; one archetype has gone on to "catch" the outliers. The $k = 3$ Bisquare archetypes solution is similar to the solution on the data set without outliers (Figure 8.1). (c) The gray scale of the data points indicate their final weights. The outliers have weight 0 (therefore filled with white color), their residuals do not contributed to the overall RSS (indicated with the dotted lines).

Figure 9.4 illustrates individual algorithm iterations of the archetypal analysis which leads to the solution presented in Figure 9.3. The algorithm converges



(a) (b) (c)

Figure 9.3.: Robust archetypal analysis; the gray scale of the data points indicate their final weights. Note that the outliers have weight 0 (unfilled).

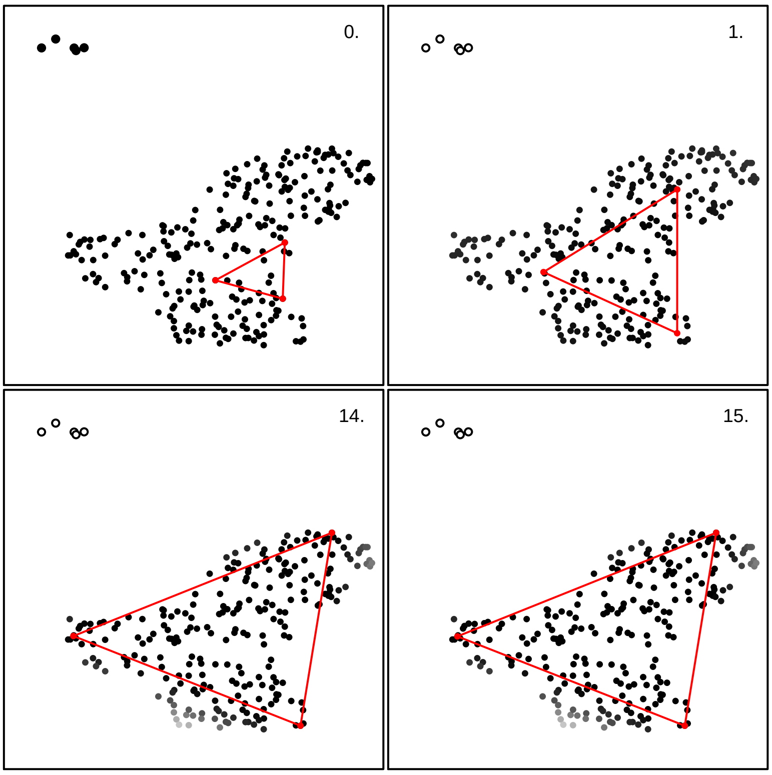


Figure 9.4.: Individual iterations of the robust archetypal analysis which led to the solution presented in Figure 9.3.

in fifteen iterations, the individual plots show the initial setup (randomly initialized archetypes), the first, fourteenth and final iteration. The gray scale of the data points indicate their current weights. Note that in the initial setup all data points have weight 1. Already in the first iteration the weights of the outliers are very low (closely to 0) and decrease to 0 at the final iteration.

9.4. Application example

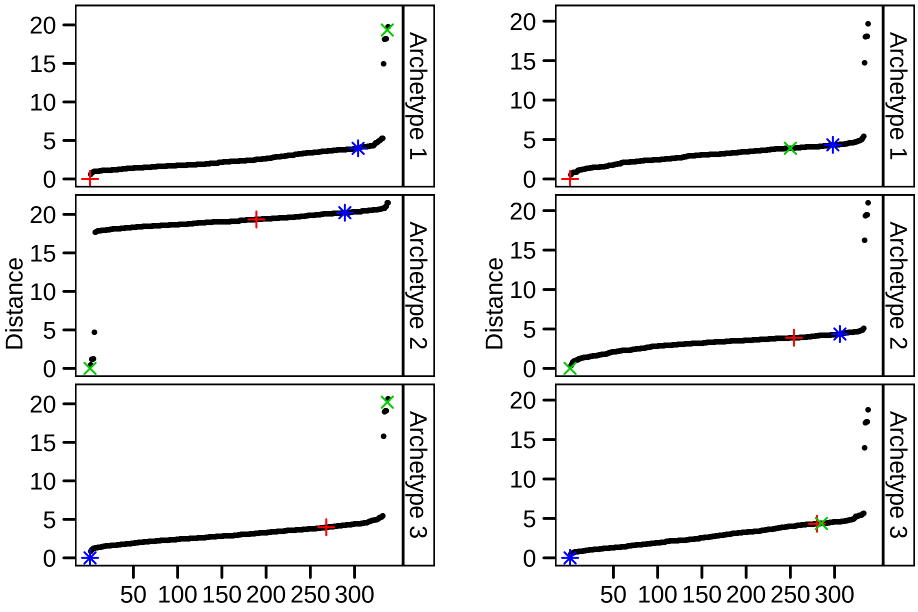
In this section we apply robust archetypal analysis on the Air-Pollution data set used by Cutler and Breiman in the original archetypal analysis paper (where they declare this problem as the "initial spark" for their study on archetypal analysis). Using a data set which already has been extensively studied allows us to compare the robust solutions with a well known solution.

The data consist of measurements of data relevant to air pollution in Los Angeles Basin in 1979. There are 330 cases consisting of daily measurements on the attributes ozone (OZONE), 500 millibar height (500MH), wind speed (WDSP), humidity (HMDTY), surface temperature (STMP), inversion base height (INVHT), pressure gradient (PRGRT), inversion base temperature (INVTMP), and visibility (VZBLTY). These data were standardized to have mean 0 and variance 1. Cutler and Breiman (1994) focus on three archetypes; left part of Table 9.1 lists the percentile value of each variable in an archetype as compared to the data (Figure 4 in Cutler and Breiman, 1994). The percentile value indicates in which percentile of the data set an archetype in a specific variable is. For example, the OZONE value in archetype 1 is 91, i.e., the archetype 1 OZONE value is in the 91th percentile of the 330 OZONE cases in the data. Cutler and Breiman interpret the archetypes as follows: "Archetype 1 is high in OZONE, 500MH, HMDTY, STMP, and INVTMP and low in INVHT and VZBLTY. This indicates a typical hot summer day. The nature of the other two archetypes is less clear; Archetype 3 seems to represent cooler days toward winter."

We contaminate the data set with a group of 5 outliers. The attributes of the outliers are calculated by $x *$MAX+IQR with MAX the maximum and IQR the interquartile range of the attribute and x randomly drawn from $[1.5, 2]$. Three archetypes are computed with the original and the robust algorithm (right part of Table 9.1). Figure 9.5 shows the panorama plot, a simple diagnostic tool to inspect arbitrary high-dimensional archetypes solutions. For each archetype

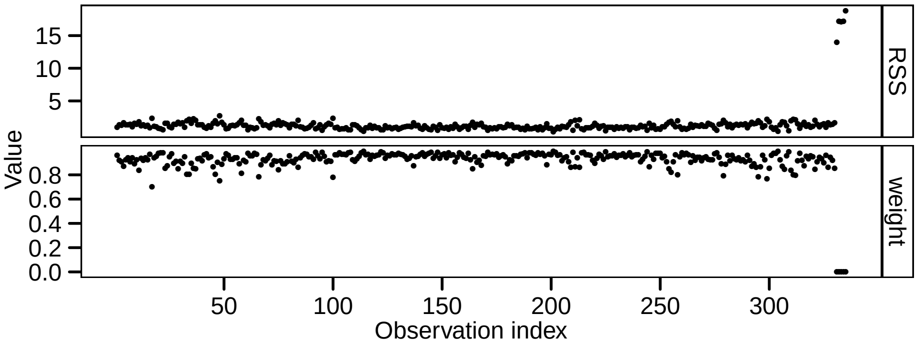
Data set	Air-Pollution						Air-Pollution+Outliers					
Algorithm	Original			Robust			Original			Robust		
Archetypes	A1	A2	A3	A1	A2	A3	A1	A2	A3	A1	A2	A3
OZONE	91	12	3	91	12	12	76	100	3	89	12	12
500MH	96	45	5	92	45	6	64	100	6	91	64	7
WDSP	43	8	91	43	8	91	27	100	63	43	8	89
HMDTY	78	11	74	78	12	81	50	100	19	77	11	63
STMP	95	15	6	95	16	11	73	100	5	91	21	11
INVHT	7	67	100	11	66	71	1	100	99	10	63	70
PRGRT	55	2	95	57	5	93	38	100	36	56	2	91
INVTMP	95	30	3	93	30	5	79	100	5	92	40	5
VZBLTY	15	88	77	15	88	77	9	100	87	15	76	76

Table 9.1.: Percentile profiles of the archetypes computed by the original algorithm and the robust algorithm on the original data set and the outlier data set.

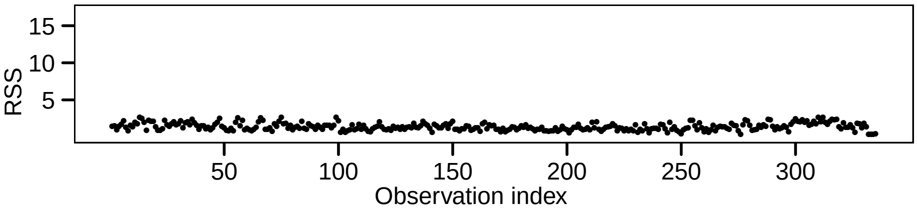


(a) (b)

Figure 9.5.: Panorama plots: The distance between each archetype and each data point in case of the (a) original algorithm and (b) robust algorithm.



(a)



(b)

Figure 9.6.: The residual length and weight of each individual data point. (a) In case of the robust algorithm the majority of the data have low residuals and high weights. The outliers have high residuals and low weights. (b) In case of the original algorithm all data points have low residuals (and weights 1).

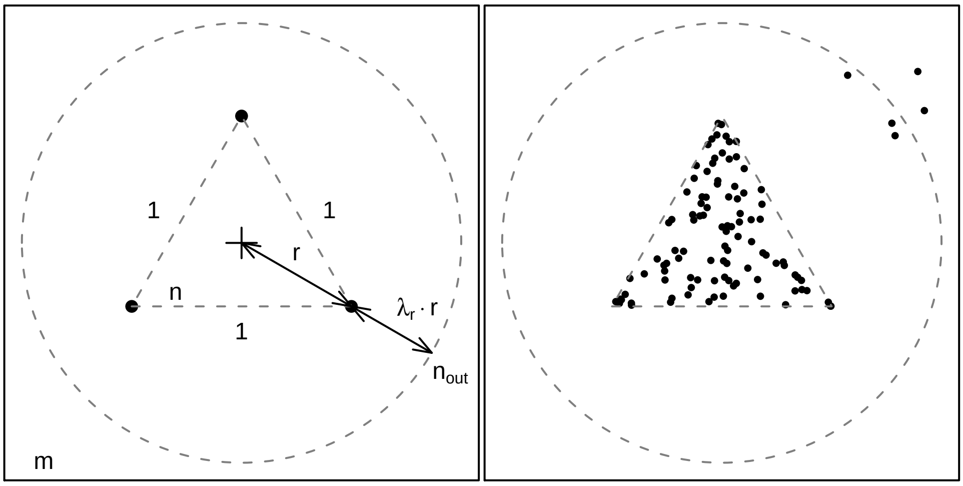
(individual panels) the Euclidean distance (y-axis) between the archetype and each data point is shown in ascending order (x-axis); other archetypes are shown as cross symbols. The underlying idea is to look at the data from the viewpoint of an archetype ("to watch its panorama"). This uncovers archetypes having only a few near data points – which then can be considered as candidates for outliers. In case of the original algorithm, Figure 9.5a, the second archetype is the archetype gone on to "catch" the outliers – it is the archetype near to the outliers. In contrast, the robust algorithm, Figure 9.5b, focuses on the majority of the data points – no archetype is in the neighborhood of the outlying observations.

In the sense of the adapted citation of Rousseeuw and Leroy (2003) on robust analysis in the introduction, Figure 9.6a (top panel) shows for each individual data point its residual length. The majority of the data has low residuals (note that variations from zero can occur due to numerical issues), whereas the five outliers stand out with high residuals. The calculated weights of the data points, Figure 9.6a (bottom panel), fit accordingly: the majority of the data has high weights, the outliers low weights. By comparison, all data points have low residuals (and weights 1) in case of the original algorithm, Figure 9.6b.

Finally, taking a look at the concrete robust archetypes values, right part of Table 9.1, shows that they are very similar to the archetypes calculated on the original data set (without the outliers).

9.5. Simulation study

So far, two exemplar data sets were used to present robust archetypal analysis and to demonstrate its proper functioning. This section now analyzes the algorithm's robustness and convergence with respect to data dimension, number of outliers and distance of outliers to the majority of data in a structured way. The simulation setup follows Hothorn et al. (2005, cf. the simulation problem), the analysis follows the framework defined in Part I of this dissertation. The two crucial points of this simulation study are (1) to define a basis setup which enables the measurement of robustness without the effects of other algorithm characteristics like structural stability or convergence, and (2) to define a performance measure which reflects robustness in numbers – as the residual sum of squares (RSS) is not meaningful for this case. Note that this section presents



(a)

(b)

Figure 9.7.: Schematic representation of the simulation setup: (a) Basis is an m-simplex as true data generating process, enclosed by an m-sphere as mean for the outlier generating process. (b) An exemplar data set with $m = 2$, $n = 100$, $n_{out} = 5$, $\lambda_r = 2$, $\sigma = 0.05$.

selected results of the simulation study, the complete results are available in the supplemental material (see the Appendix A on computational details).

We define an uniformly distributed regular m-simplex (an m-dimensional polytope of $m + 1$ vertices and equally long edges; note that the common name n-simplex conflicts with our notation) as true data generating process. The outlier generating process is defined as the multivariate normal distribution with the covariance matrix $\Sigma = \sigma I_m$ and the mean μ on an uniformly distributed m-sphere (the generalization of the surface of an ordinary sphere to dimension m) of radius $r + \lambda_r \cdot r$, with r the distance between the center and an m-simplex vertex and λ_r an expansion factor. n observations and n_{out} outliers are drawn from the data and outlier generating process, respectively. Figure 9.7a illustrates the setup, Figure 9.7b shows an exemplar data set with $m = 2$, $n = 100$, $n_{out} = 5$, $\lambda_r = 2$, $\sigma = 0.05$. We define the performance measure as the total Euclidean distance between the computed archetypes and the nearest true archetype in each case (Δ_1), and the total Euclidean distance between the true archetypes and the nearest computed archetypes in each case (Δ_2). Note that this definition must not yield in an one-to-one assignment; Figure 9.8 illustrates a potential case. Then, a good solution has small and

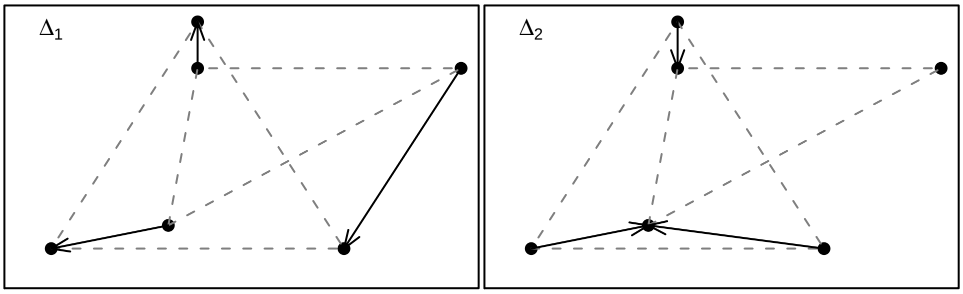


Figure 9.8.: The performance measure is the total distance between the computed archetypes and the nearest true archetypes (Δ_1) and the total distance between the true archetypes and the nearest computed archetypes (Δ_2).

nearly similar Δ_1 and Δ_2 (necessary condition). A perfect solution is an one-to-one assignment with Δ_1 and Δ_2 of value zero.

The first simulation investigates the original and robust algorithm when the correct number of archetypes k (i.e., $k = m + 1$) is known: A data set is generated for each combination of $m = 2, 3, 10, 15$ attributes, $n = 2000$ observations, $n_{out} = 5, 20, 100$ outliers, $\lambda_r = 2, 3, 5, 15$ radius expansion factors, $\sigma = 0.05$ covariance and 100 replications in each case. Each configuration is fitted with randomly chosen initial archetypes, stop criteria are 100 iterations or an improvement less than the square root of the machine epsilon ($\sqrt{(}2.22 \cdot 10^{-16})$). For each of the $m \; \times n \times \; n_{out} \; \times \; \lambda_r \; \times \; \Sigma \; \times \; (1, \dots, 100)$ original and robust fits the two distance measures, the number of iterations, the computation time and the RSS are reported (all in all 48000 measured values). The constant number of observations n allows the discussion of the algorithms in view of the curse of dimensionality.

Figure 9.9 shows the distances Δ_1 (y-axis, scale from 0 to 30) versus Δ_2 (x-axis, scale from 0 to 5) for each fit of the original and the robust algorithm in case of the configuration $m = 10$, $n_{out} = 100$, $\lambda_r = 2, 3, 5, 15$. In case of the original algorithm, Δ_1 increases with increasing outlier distance λ_r; one computed archetype always goes for the outlier group. One the other side, Δ_2 stays small, so the remaining computed archetypes fit well to the true archetypes. In case of the robust algorithm, both distances remain small with increasing outlier distance; the robust algorithm stays robust and finds good solutions for the majority of the data (with a few exceptions). Noticeable is that the robust algorithm's distances are more variable and that Δ_2 is slightly higher. The first fact indicates less stable solutions (part of our ongoing research, see discussion in Section 9.6). The second fact is, amongst other things, explored in the fol-

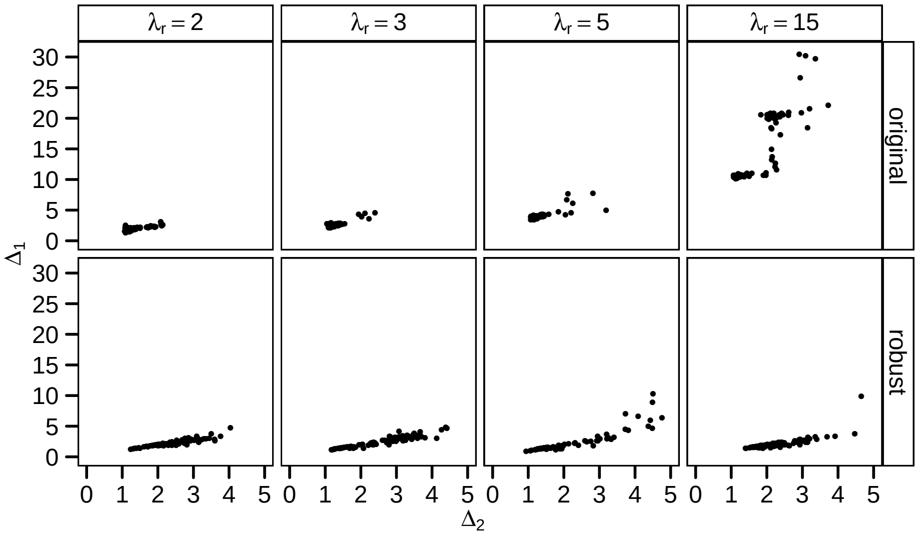


Figure 9.9.: Distances Δ_1 versus Δ_2 for sample fits of both algorithms in case of the configuration $m = 10$, $n_{out} = 100$, $\lambda_r = 2, 3, 5, 15$.

lowing paragraph. The described patterns appear for all other configurations as well.

Figure 9.10 shows the distances for each dimension $m = 2, 3, 10, 15$ and $n_{out} = 100$, $\lambda_r = 15$. Δ_1 also remains small over the number of dimensions for the robust algorithm and increases for the original algorithm (as expected). Interesting is the fact that Δ_2 increases over the number of dimensions for the robust algorithm. This indicates that the computed solution is smaller than the true m-simplex; observations near to the m-simplex boundary are weighted down even though they are from the true data generating process. This is an effect of the curse of dimensionality, in higher dimensions the robust algorithm finds the majority of the data's true shape but not in the exact size.

As already stated, the archetypal analysis algorithm is computer-intensive and a fast convergence is desired. Figure 9.11 compares the median number of iterations between the original (dashed line) and the robust (solid line) algorithm for each configuration. Ignoring $m = 2$, the robust algorithm needs less or nearly equal number of iterations in most of the cases. In the case $m = 2$ both algorithms generally converge much slower, and the robust algorithm needs twice as much iterations than the original algorithm. One supposable reason could be the large number of observations ($n = 2000$) in relation to the space of the m-simplex with side length 1 and the consequential closeness

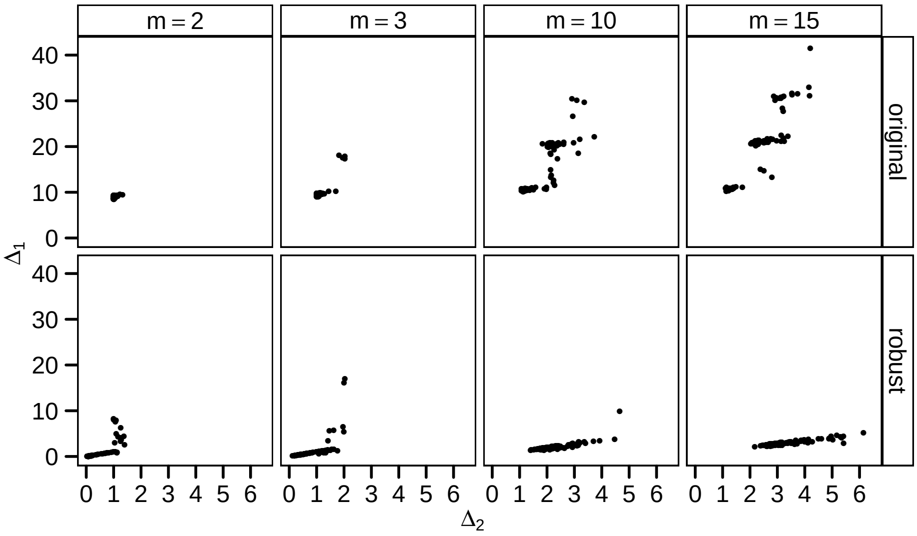


Figure 9.10.: Distances for each dimension $m = 2, 3, 10, 15$ and $n_{out} = 100$, $\lambda_r = 15$. Note that the y-axis scale is different per distance.

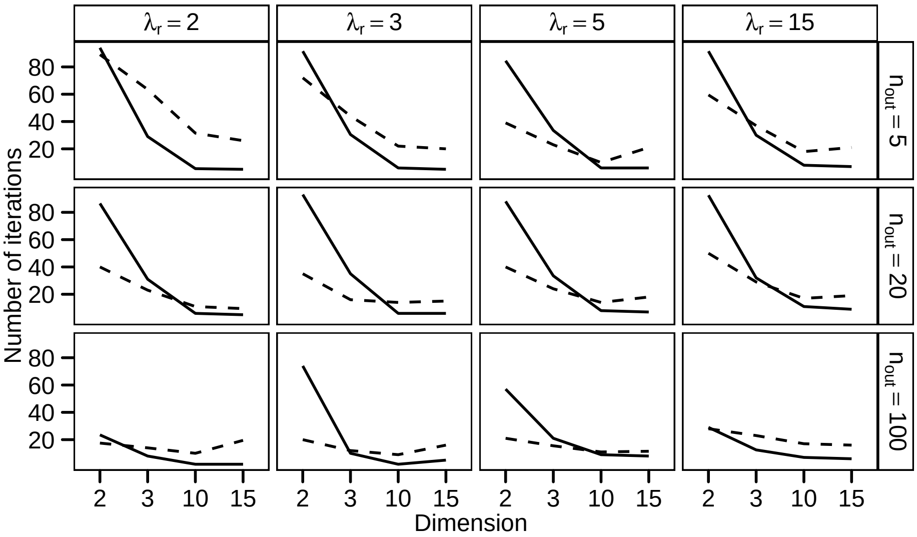


Figure 9.11.: Median number of iterations for the original (dashed line) and the robust algorithm (solid line).

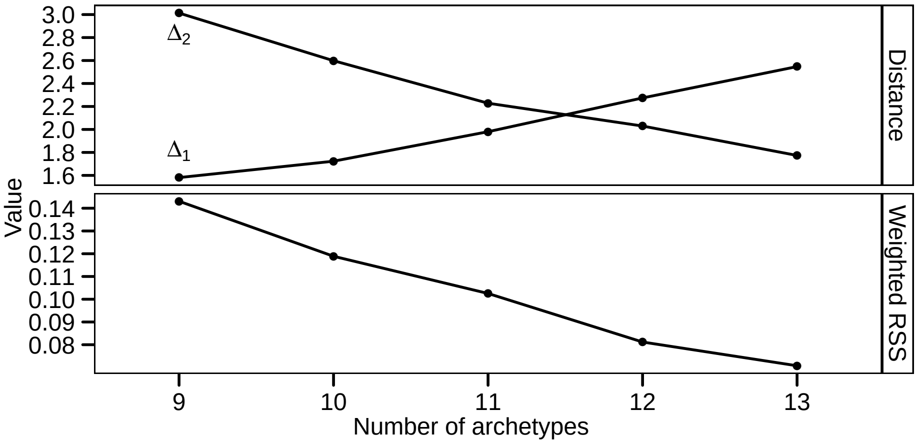


Figure 9.12.: Median distances Δ_1 and Δ_2 and median weighted RSS for the robust algorithm in case of the configuration $m = 10$, $n_{out} = 100$, $\lambda_r = 2, 3, 5, 15$, $k = 9, \ldots, 13$.

of the observations; but detailed simulations need to be done to analyze this phenomenon.

The second simulation investigates the robustness of the algorithm when the number of correct archetypes k (i.e., $k = m + 1$) is unknown: The simulation setup is equivalent to the first one, but an additional fit is computed for each $k = m - 1, m, m + 1, m + 2, m + 3$. Figure 9.12 (top panel) shows the median distances Δ_1 and Δ_2 of the 100 replications for the robust algorithm and $m = 10$, $n_{out} = 100$, $\lambda_r = 15$ and $k = 9, \ldots, 13$. The distances are inside a reasonable range, the algorithm is robust in case of wrong k. Δ_1 increases and Δ_2 decreases with increasing number of archetypes; the true solution is near to their intersection point. This reflects the different assignment scenarios: (1) less computed archetypes than true archetypes – one-to-one assignment with remaining true archetypes for Δ_1 and one-to-many assignment for Δ_2; (2) as many computed archetypes as true archetypes – one-to-one assignment; (3) more computed archetypes than true archetypes – one-to-many assignment for Δ_1 and one-to-one assignment with remaining computed archetypes for Δ_2. Obviously, this performance measure is only usable in simulation studies where the data generating process and the true archetypes are known. In real world applications the weighted RSS is a possible performance measure; Figure 9.12 (bottom panel) shows the median weighted RSS. However, as this scree plot indicates, this performance measure often allows no well-defined decision. This is a problem of great complexity – algorithm characteristics, like the structural stability or the convergence, play a major role. Currently we have no sound solution and this is part of our ongoing research (see discussion in Section 9.6).

9.6. Summary

This chapter adapts the archetypal analysis estimator by Cutler and Breiman (1994) to allow weighted and robust archetypal analysis. Weighted archetypes enables to represent additional information like importance of and correlation between observations. Robust archetypes focus on the majority of the data set; data points which behave differently from the large majority achieve less weight in the fitting process. The proposed estimator is an M-estimator whose minimization problem is solved by an iteratively reweighted least squares fitting algorithm. The artificial toy example and the real world application example shows that in presence of outliers the robust algorithm gives reliable archetypes which are greatly similar to the archetypes calculated on the same data set without outliers. The simulation study (benchmark experiment for sensitivity analysis) analyzes the algorithm with respect to data dimension, number of outliers, distance of outliers to the majority of data and number of archetypes in a structured way. The study shows that the algorithm is highly robust and often converges faster than the original algorithm.

Chapter 10.

Outlook

Prototype-based methods are widely used approaches in classification and clustering. Prototype methods represent the data set by a set of points; where the prototypes are defined by optimization of some fixed criteria (Hastie et al., 2009). Now, prototypes can be seen as a general concept and archetypes as one possible instance of prototypes. In fact, the archetypal analysis formulation (cf. Section 8.1) is similar to the least squares formulation of centroid clustering algorithms (cf. Steinly, 2006). As final outlook we present theoretical thoughts on the extension of our general framework for the class of k-prototypes-like algorithms (k-means, k-median, Fuzzy k-means, etc.; see, e.g., Steinly, 2006; Leisch, 2006).

The prototypes problem. Given is an $n \times m$ matrix X representing a multivariate data set with n observations and m attributes. For a given k, the matrix Z of k m-dimensional prototypes is calculated according to:

$$\begin{aligned}
&w\|X - \alpha Z^T\| \to \min \\
&\text{with } \sum_{j=1}^{k} \alpha_{ij} = 1 \text{ and } f_\lambda(\alpha_{ij}) > 0 \\
&Z = X^T \beta \\
&\text{with } \sum_{i=1}^{n} \beta_{ij} = 1 \text{ and } g_\lambda(\beta_{ij}) > 0
\end{aligned}$$

whereas $i = 1, \dots, n$, $j = 1, \dots, k$; α, the coefficients of the prototypes, is an $n \times k$ matrix; β, the coefficients of the observations, is an $n \times k$ matrix; w, the weighting vector, is an n vector; $\| \cdot \|$ is an appropriate matrix norm; f and g

are evaluation functions of the coefficients for a specific constraint controlled by the parameter λ.

The evaluation functions f_λ and g_λ define the nature of the constraints and therefore the nature of the problem and the resulting prototypes:

Archetypal analysis problem: Imagine a λ_1 so that

$$f_{\lambda_1}(\alpha_{ij}) = \begin{cases} 1, & 0 \leq \alpha_{ij} \leq 1 \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad g_{\lambda_1}(\beta_{ij}) = \begin{cases} 1, & 0 \leq \beta_{ij} \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

The functions allow values arbitrary values between $[0, 1]$ (see Figure 10.1, top row); this describes the constraints $\alpha_{ij} \geq 0$ and $\beta_{ij} \geq 0$. With $\| \cdot \|$ the spectral norm, the obtained problem is equivalent to the archetypal analysis problem defined in Section 8.1; the prototypes are the archetypes.

***k*-centroids cluster problem:** Imagine a λ_2 so that

$$f_{\lambda_2}(\alpha_{ij}) = \begin{cases} 1, & \alpha_{ij} = 0 \text{ or } \alpha_{ij} = 1 \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad g_{\lambda_2}(\beta_{ij}) = \begin{cases} 1, & \beta_{ij} = \frac{1}{n_j} \\ 0, & \text{otherwise} \end{cases}$$

The functions allow pointwise values (see Figure 10.1, bottom row); this describes the constraints $\alpha_{ij} \in \{0, 1\}$ and

$$\beta_{ij} = \begin{cases} \frac{1}{n_j}, & \text{observation } i \text{ is element of cluster } j \\ 0, & \text{otherwise} \end{cases}$$

with n_j is the size of cluster j. The obtained problem is equivalent to the k-centroids cluster problem; the prototypes are the canonical centroids. Based on the norm $\| \cdot \|$ different canonical centroids are obtained, see Leisch (2006) for details.

This formulation reveals the relation of the two problems as instances of a more general problem class. Future work contains the analysis of this relationship. One interesting question is, if the two problems and their solutions are transferable into each other (i.e., for a known k-centroids clustering of a data set, the corresponding k archetypes can be estimated; and vice versa). Furthermore, the above problem formulation allows a mixture of the two problems by defining appropriate evaluation functions. As the two prototype solutions can be seen as the two extreme solutions for k prototypes – the maximum and the mean prototypes – an appropriate mixture of the problems could lead to quantile prototypes.

Structural stability. Seeing the archetypal problem as an instance of the more general prototypal problem allows to use similar methods to evaluate the algorithm. Structural stability, especially in the case of unknown and wrong number of archetypes, is an important characteristic. Therefore, a major goal of our future work is the development of a framework along the lines of the framework for "structure and reproducibility of cluster solutions" by Dolnicar and Leisch (2010). This includes, among other things, the definition of a criterion for stability of archetypal solutions.

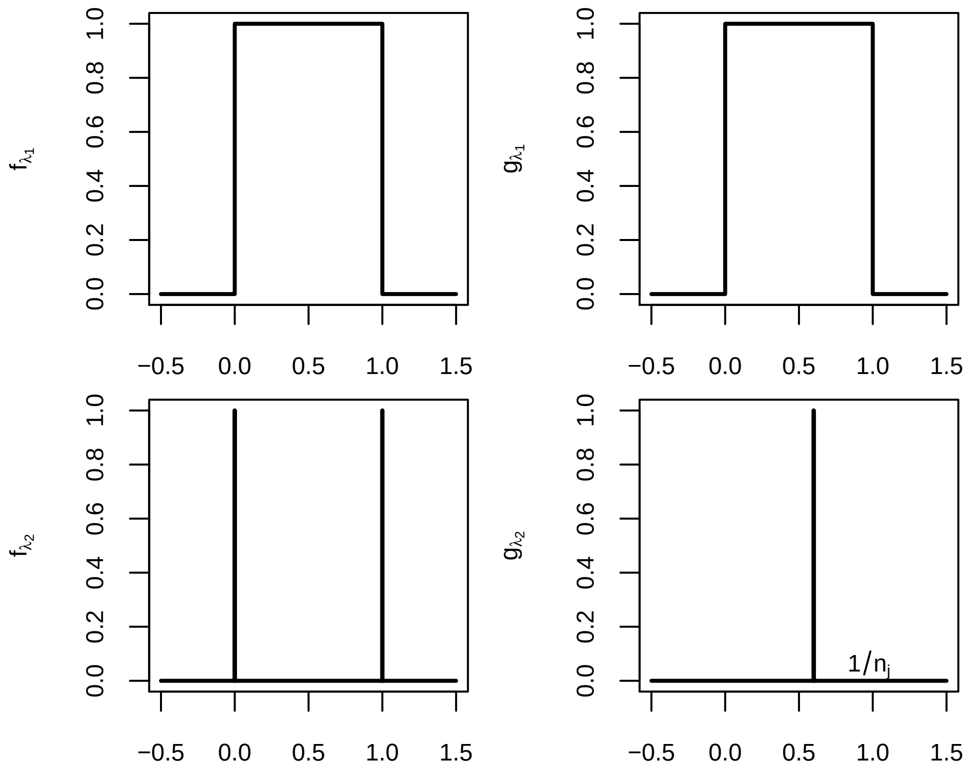


Figure 10.1.: Evaluation functions for the archetypal analysis problem (top row) and the k-centroids cluster problem (bottom row).

Appendix

Appendix A.

Computational details

This appendix provides detailed information on the computational parts of the dissertation. All computations and graphics have been done using R (R Development Core Team, 2010) and add-on packages. R itself and all packages used are freely available under the terms of the General Public License from the Comprehensive R Archive Network at http://CRAN.R-project.org.

Section A.1 provides information on the package and the reproducibility for Part I – benchmark experiments – of this dissertation. Section A.2 provides information on the package and the reproducibility for Part II – archetypal analysis – of this dissertation. In both cases are demos and vignettes available to reproduce examples, applications, and simulations of the dissertation.The packages are documented using Roxygen (Danenberg and Eugster, 2010). Roxygen is a documentation system for R; allowing, among other things, in-source specification of Rd files, collation, and namespace directives. This reduces the error-proneness and therefore increases the quality of the documentation.

A.1. Benchmark experiments

A.1.1. Package benchmark

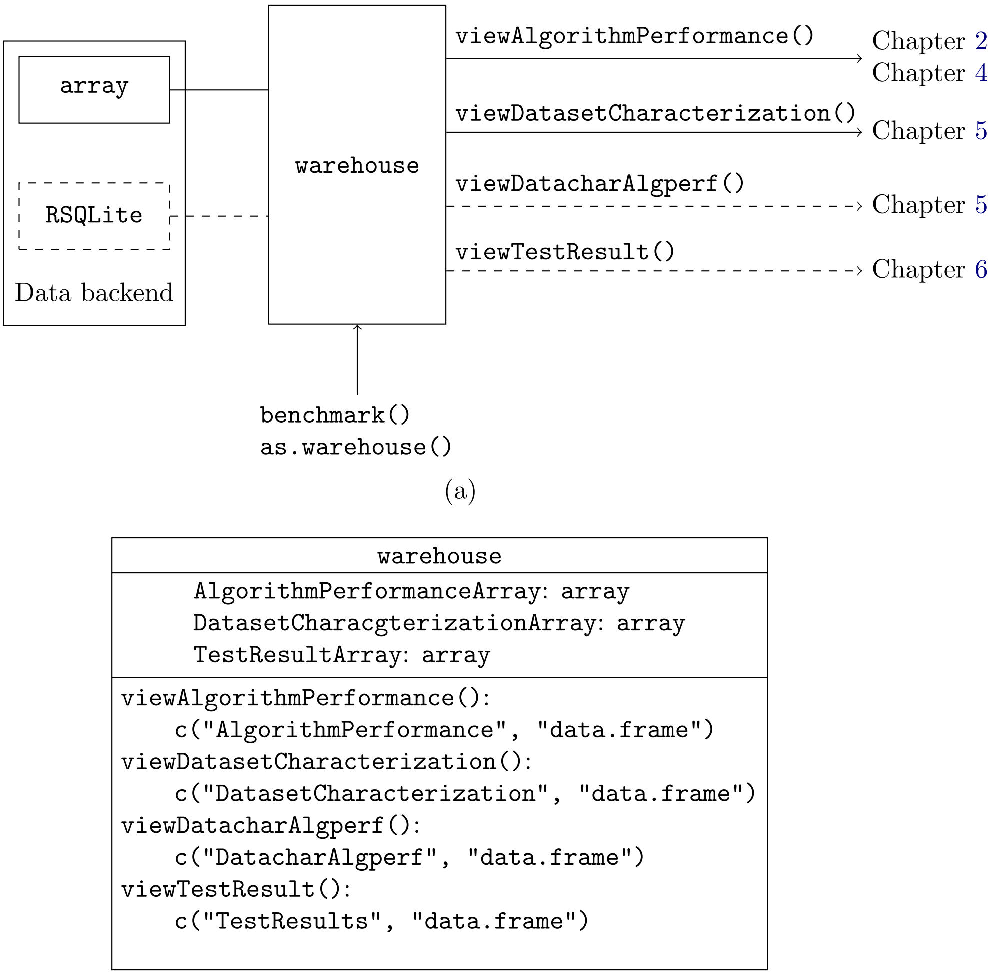
The Benchmark Experiments Toolbox (R package benchmark version 0.3-2, Eugster, 2011) provides a toolbox for setup, execution, and analysis of benchmark experiments. Main focus is the analysis of data accumulating during the execution – and one major objective is the statistical correct computation of an order of the candidate algorithms. The benchmark package uses functionality of other packages: coin (Hothorn et al., 2006) for permutation tests; lme4 (Bates and Maechler, 2010) for mixed effects models; multcomp (Hothorn et al., 2008) for the calculation of simultaneous confidence intervals; relations (Hornik and Meyer, 2010) to handle relations and consensus rankings; psychotree (Strobl et al., 2010) for recursively partitioning Bradley-Terry models; and ggplot2 (Wickham, 2009) to produce graphics.

The idea of the package is to provide an extensible warehouse data structure (`warehouse`) to collect all interesting results arising during the execution of benchmark experiments. Currently implemented are data structures to collect algorithm performances (`AlgorithmPerformanceArray`), data set characterizations (`DatasetCharacterizationArray`), and sequential test results (`TestResultArray`). The backend data structure of the warehouse is a collection of `array`s, but this is exchangeable; for example with an `RSQLite` data base (James, 2010, not implemented in this benchmark version).

The function `benchmark()` executes a benchmark experiment and fits as reference implementation of how to fill up such a warehouse. Different implementations of the generic `as.warehouse()` function coerce benchmark results from other data structures into a `warehouse` object (for example, from the data structure used by the `mlr` package, Bischl et al., 2010).

Different accessor functions then process the raw data (e.g., clean and join different arrays) and return `data.frame` objects for the analysis. Implemented accessor functions are `viewAlgorithmPerformance()`, `viewDatasetCharacterization()`, and `viewTestResult()`. Each accessor function returns a `data.frame` with an additional class attribute set. The analysis methods introduced in this dissertation are implemented for these `data.frame`s. For example, the methods introduced in Chapter 2 and Chapter 4 are based on the result from the `viewAlgorithmPerformance()` accessor which uses the data available in the `AlgorithmPerformanceArray`. And the methods introduce in Chapter 5 are based on the result from the `viewDatacharAlgperf()` accessor which returns a `data.frame` from `AlgorithmPerformanceArray` and `DatasetCharacterizationArray`.

Figure A.1 shows a schematic representation of the package concept (dashed lines are not implemented in version 0.3-2). The following sketches an exemplar session from the UCI benchmark experiment (Chapter 2; note that this just illustrates the principal usage):



(a)

```
                      warehouse
      AlgorithmPerformanceArray: array
      DatasetCharacgterizationArray: array
      TestResultArray: array

viewAlgorithmPerformance():
    c("AlgorithmPerformance", "data.frame")
viewDatasetCharacterization():
    c("DatasetCharacterization", "data.frame")
viewDatacharAlgperf():
    c("DatacharAlgperf", "data.frame")
viewTestResult():
    c("TestResults", "data.frame")
```

(b)

Figure A.1.: Schematic representation of the benchmark package concept. (a) The general concept. (b) Details of the current `warehouse` implementation; see package documentation for the description of the resulting `data.frames`.

```
R> library("benchmark")

R> ### Data set:
R> # Load and process mnk3 data using as.dataset()

R> ### Setup and execution:
R> w <- benchmark(mnk3,
+                sampling = bs.sampling(250),
+                algorithms = c(lda, knn, nnet, rf, rpart, svm),
+                performances = c(miscl, fittime, predicttime))

R> ### Analysis:
R> apm <- w$viewAlgorithmPerformance(performances =
                                       "Misclassification")
R> ## Exploratory:
R> boxplot(apm)                      # Figure 2.2b
R> beplot0(apm)                      # Figure 2.3
R> beplot0(apm, lines.show = TRUE)  # Figure 2.4

R> ## Inferential:
R> p2 <- paircomp(apm,               # Lmer model used in 2.3.2
+                family = LmerTestPaircomp,
+                type = "<",
+                significance = 0.05,
+                relevance = 0.01)
```

The package documentation provides detailed information for each function. However, this package is ongoing research and not all methods are already released in the CRAN version of the benchmark package. Especially because of the grammar of benchmark and the design of experiment ideas (Chapter 7) the package is in a structural redesign phase at the time of writing.

A.1.2. Reproducibility

UCI domain. The benchmark experiment of the UCI domain is used throughout the dissertation. For the candidate algorithms the following functions and packages have been used: Functions lda() and qda() from package MASS for linear and quadratic discriminant analysis. Function knn() from package class for the k-nearest neighbor classifier. The hyperparameter k (the number of neighbors) has been determined between 1 and $\sqrt{n}$, n the number of observations, using 10-fold cross-validation (using the function `tune.knn()` from package e1071, Dimitriadou et al., 2010). Function `nnet()` from package nnet for fitting neural networks. The number of hidden units has been determined between 1 and $\log(n)$ using 10-fold cross-validation (using function `tune.nnet()` from package e1071), each fit has been repeated 5 times. All four algorithms `lda()`, `qda()`, `knn()`, and `nnet()` are described by Venables and Ripley (2002). Function `rpart()` from package rpart (Therneau and Atkinson, 2010) for fitting classification trees. The 1-SE rule has been used to prune the trees. Functions `naiveBayes()` and `svm()` from package e1071 for fitting naive Bayes models and C-classification support vector machines. The two C-classification support vector machines hyperparameters γ (the cost of constraints violation) and C (the kernel parameter) have been determined using a grid search over the two-dimensional parameter space (γ, C) with γ from 2^{-5} to 2^{12} and C from 2^{-10} to 2^{5} (using function `tune.svm()` from package e1071). And function `randomForest()` from package randomForest (Liaw and Wiener, 2002) for fitting random forests.

The data sets are from the UCI Machine Learning Repository (Asuncion and Newman, 2007) and preprocessed by Meyer et al. (2003). The data sets basic characteristics are (* indicates artificial data sets):

Problem	Abbr.	#Attributes		#Samples		Class distribution
		nom	con	comp	incomp	(%)
promotergene	prmt	57		106		50.00/50.00
hepatitis	hptt	13	6	80	75	20.65/79.35
Sonar	Sonr		60	208		53.37/46.63
Heart1	Hrt1	8	5	296	7	54.46/45.54
liver	livr		6	345		42.03/57.97
Ionosphere	Insp	1	32	351		35.90/64.10
HouseVotes84	HV84	16		232	203	61.38/38.62
musk	musk		166	476		56.51/43.49
monks3	mnk3	6		554		48.01/51.99
Cards	Crds	9	6	653	37	44.49/55.51
BreastCancer	BrsC	9		683	16	65.52/34.48
PimaIndiansDiabetes	PmID		8	768		65.10/34.90
tictactoe	tctc	9		958		34.66/65.34
credit	crdt		24	1000		70.00/30.00
Circle (*)	Crcl		2	1200		50.67/49.33
ringnorm (*)	rngn		20	1200		50.00/50.00
Spirals (*)	Sprl		2	1200		50.00/50.00
threenorm (*)	thrn		20	1200		50.00/50.00
twonorm (*)	twnr		20	1200		50.00/50.00
titanic	ttnc	3		2201		67.70/32.30
chess	chss	36		3196		47.78/52.22

The raw performances of the executed benchmark experiment are available in the package (`data("uci621raw")`). Source codes for replicating the analyses of the benchmark experiment are available as package demos. For example, Chapter 2 is reproduced via

```
R> demo("benchplot", package = "benchmark")
```

Chapter 4 via the demo `lsbenchplot-cs621`; and the archetypal analysis of the UCI domain via the demo `lsbenchplot-cs621-atypes`.

A.2. Archetypal analysis

A.2.1. Package archetypes

The R package archetypes (version 2.0-2, Eugster, 2010) provides a framework for archetypal analysis supporting arbitrary problem solving mechanisms for the different conceptual parts of the algorithm. The archetypes package uses functionality of the package nnls (Mullen and van Stokkum, 2010) to solve non-negative least squares problems.

The main function `archetypes()` implements the algorithm framework with the steps defined in Section 9.3 (based on Section 8.2):

```
function (data, k, weights = NULL, maxIterations = 100,
          minImprovement = sqrt(.Machine$double.eps),
          maxKappa = 1000, verbose = FALSE, saveHistory = TRUE,
          family = archetypesFamily("original"), ...)
```

The most important argument of this function is the `family` argument specifying the problem to solve, i.e., the classical, weighted, or robust archetypal problem. The structure of a `family` object is:

```
List of 14
 $ normfn      :function (m, ...)            # Step 3.6
 $ scalefn     :function (x, ...)            #      1
 $ rescalefn   :function (x, zs, ...)        #      4 and 5
 $ dummyfn     :function (x, ...)            #      2
 $ undummyfn   :function (x, zs, ...)        #      5
 $ initfn      :function (x, p, ...)         #      2
 $ alphasfn    :function (coefs, C, d, ...)  #      3.2
 $ betasfn     :function (coefs, C, d, ...)  #      3.4
 $ zalphasfn   :function (alphas, x, ...)    #      3.3
 $ globweightfn:function (x, weights)        #      1
 $ weightfn    :function (x, weights)        #      3.1
 $ reweightsfn :function (x, weights)        #      3.1
 $ class       : NULL
 $ which       : chr ""
```

The package provides three `archetypesFamily("...")` definitions: `original` for the classical archetypes, `weighted` for the weighted archetypes, and `robust` for the robust archetypes. However, own families can be simply defined by replacing the desired functions with user-defined ones (see the package documentation for a description of the corresponding arguments).

As exemplar session sketches the following code the computation of the classical archetypes for the toy example used throughout the dissertation's part:

```
R> library("archetypes")

R> ### Data set:
R> data("toy", package = "archetypes")

R> ### Classical archetypes:
R> a <- archetypes(toy, 3)

R> ### Visualizations:
R> xyplot(a, toy, chull = chull(toy))  # Figure 8.1b
R> xyplot(a, toy, adata.show = TRUE)   # Figure 8.1c
```

The package documentation provides detailed information for each function.

A.2.2. Reproducibility

Skeletal archetypes. The source code of the illustrative example in Section 8.3 is available in the package vignette "From Spider-Man to Hero – Archetypal Analysis in R":

```
R> v <- vignette("archetypes", package = "archetypes")
R> v        # View PDF file
R> edit(v)  # View source code
```

Weighted and robust archetypes. The source codes of the artificial two-dimensional toy example (used throughout this dissertation's part), the Air-Pollution example (Section 9.4), and the robustness simulation (Section 9.5) are available as package demos. The demos are executed via:

```
R> demo("robust-***", package = "archetypes")
```

The source code file for a demo is accessible via:

```
R> edit(file = system.file("demo", "robust-***.R",
+                              package = "archetypes"))
```

Replace `***` with `toy`, `ozone`, and `simulation` respectively.

Mathematical nomenclature

In trying to find a notation which is internally consistent, we have defined a few general bounded variables with corresponding index variables. Note that in some cases we have to use the same symbol with different meanings in the benchmark experiment part and the archetypal analysis part.

General symbols used in Part I:

$b = 1, \dots, B$ Number of replications.
$k = 1, \dots, K$ Number of candidate algorithms.
$j = 1, \dots, J$ Number of performance measures.
$m = 1, \dots, M$ Number of data sets.
$t = 1, \dots, T$ Number of data set characteristics.

DGP Data generating process; either a known data generating process (represented by a function) or an unknown data generating process (represented by a finite data set $\mathfrak{L}$ with some resampling scheme).
$\mathfrak{L} = \{z_1, \dots, z_N\}$ Data set with N observations.
$z_i = (y_i, x_i)$ Observation with x_i a vector of input variables and y_i the response variable; $i = 1, \dots, N$.
$\mathfrak{D} = \{\mathfrak{L}_m\}$ Domain of M data sets.

$\mathfrak{L}^b$ Learning samples with $n \leq N$ observations drawn from the data generating process DGP.
$\mathfrak{T}^b$ Validation samples with the number of observations dependent on the corresponding learning sample $\mathfrak{L}^b$.

a, a_k Candidate algorithms.

$p(\cdot)$, $p_j(\cdot)$ Performance measures.
$p_{mbkj} \sim \mathcal{P}_{mbkj}$ Theoretical performance p_j of algorithm a_k on learning sample $\mathfrak{L}^{\mathfrak{b}}_{\mathfrak{m}}$ of data set $\mathfrak{L}_{\mathfrak{m}}$ drawn from the performance distribution $\mathcal{P}_{mbkj}$.

General symbols used in Part II:

X	Data matrix of dimension $n \times m$.
$i = 1, \ldots, n$	Number of observations.
$j = 1, \ldots, m$	Number of attributes.
k	Number of archetypes.
Z	Archetypes matrix of dimension $k \times m$.
α_{ij}	Archetypes coefficient matrix of dimension $n \times k$.
β_{ij}	Observations coefficient matrix of dimension $n \times k$.
RSS	Residual sum of squares.

Bibliography

Jacob Abernethy and Percy Liang. MLcomp. Website, 2010. http://mlcomp.org/; visited on April 13, 2011.

P. Armitage, C. K. McPherson, and B. C. Rowe. Repeated significane test on accumulating data. *Journal of the Royal Statistical Society*, 132(2), 1969.

Kenneth J. Arrow. *Social Choice and Individual Values*. Yale University Press, second edition, 1963. ISBN 0300013647.

A. Asuncion and D.J. Newman. UCI machine learning repository, 2007. URL http://www.ics.uci.edu/~mlearn/MLRepository.html.

Douglas Bates and Martin Maechler. *lme4: Linear Mixed-Effects Models using S4 Classes*, 2010. URL http://CRAN.R-project.org/package=lme4. R package version 0.999375-33.

Christian Bauckhage and Christian Thurau. Making archetypal analysis practical. In *Proceedings of the 31st DAGM Symposium on Pattern Recognition*, pages 272–281, 2009. doi: 10.1007/978-3-642-03798-6_28.

Richard A. Becker, William S. Cleveland, and Ming-Jen Shyu. The visual design and control of Trellis display. *Journal of Computational and Graphical Statistics*, 5(2):123–155, 1996. doi: 10.2307/1390777.

Bernd Bischl, Max Wornowizki, and Katharina Borg. *mlr: Machine Learning in R*, 2010. URL http://R-Forge.R-project.org/projects/mlr/. R package version 0.3.1206.

Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995. ISBN 0198538642.

Ralph A. Bradley and Milton E. Terry. Rank analysis of incomplete block designs. I. the method of paired comparisons. *Biometrica*, 39(3/4):324–345, 1952.

Werner Brannath, Martin Posch, and Peter Bauer. Recursive combination tests. *Journal of the American Statistical Association*, 97(457):236–244, 2002. doi: 10.1198/016214502753479374.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman and Hall, 1984. ISBN 0412048418.

Robert C. Camp. A bible for benchmarking, by Xerox. *Financial Executive*, 9(23), 1993.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 161–168, 2006. ISBN 1595933832. doi: 10.1145/1143844.1143865.

Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pages 96–103, 2008. doi: 10.1145/1390156.1390169.

Ben H. P. Chan, Daniel A. Mitchell, and Lawrence E. Cram. Archetypal analysis of galaxy spectra. *Monthly Notice of the Royal Astronomical Society*, 338:790–795, 2003. doi: 10.1046/j.1365-8711.2003.06099.x.

William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979.

Dianne Cook and Deborah F. Swayne. *Interactive and Dynamic Graphics for Data Analysis: With R and GGobi*. Springer, 2007. ISBN 9780387717616.

Douglas E. Critchlow and Michael A. Fligner. Paired comparison, triple comparison, and ranking experiments as generalized linear models, and their implementation on GLIM. *Psychometrika*, 56(3):517–533, 1991. doi: 10.1007/BF02294488.

Adele Cutler and Leo Breiman. Archetypal analysis. *Technometrics*, 36(4):338–347, 1994.

Peter Danenberg and Manuel J. A. Eugster. *Roxygen: Literate Programming in R*, 2010. URL http://CRAN.R-project.org/package=roxygen. R package version 0.1-2.

Andrew Davey and Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, second edition, 2002. ISBN 0521367662.

Marcilio C. P. de Souto, Ricardo B. C. Prudencio, Rodrigo G. F. Soares, Daniel A. S. Araujo, Ivan G. Costa, Teresa B. Ludermir, and Alexander Schliep. Ranking and selecting clustering algorithms using a meta-learning approach. In *International Joint Conference on Neural Networks*, pages 3729–3735, 2008. doi: 10.1109/IJCNN.2008.4634333.

Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923, 1998. doi: 10.1162/089976698300017197.

Evgenia Dimitriadou, Kurt Hornik, Friedrich Leisch, David Meyer, , and Andreas Weingessel. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2010. URL http://CRAN.R-project.org/package=e1071. R package version 1.5-24.

Sara Dolnicar and Friedrich Leisch. Evaluation of structure and reproducibility of cluster solutions using the bootstrap. *Marketing Letters*, 21:83–101, 2010. doi: 10.1007/s11002-009-9083-4.

David L. Donoho and Peter J. Huber. The notion of breakdown point. In *A Festschrift for Erich Lehmann*, pages 157–184, 1983.

Bradley Efron. Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, 7(1):1–26, 1979. doi: 10.1214/aos/1176344552.

Bradley Efron. Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983.

Bradley Efron. How biased is the apparent error rate of a prediction rule? *Journal of the American Statistical Association*, 81(394):461–470, 1986.

Bradley Efron and Robert Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1993. ISBN 0412042312.

Matthias Ehrgott. *Multicriteria Optimization*. Springer, second edition, 2005. ISBN 3540213988.

Manuel J. A. Eugster. *archetypes: Archetypal Analysis*, 2010. URL http://cran.r-project.org/package=archetypes. R package version 2.0-2.

Manuel J. A. Eugster. *benchmark: Benchmark Experiments Toolbox*, 2011. URL http://cran.r-project.org/package=benchmark. R package version 0.3-2.

Manuel J. A. Eugster and Friedrich Leisch. Bench plot and mixed effects models: First steps toward a comprehensive benchmark analysis toolbox. In Paula Brito, editor, *Compstat 2008—Proceedings in Computational Statistics*, pages 299–306. Physica Verlag, Heidelberg, Germany, 2008. ISBN 978-3-7908-2083-6. Preprint available from http://epub.ub.uni-muenchen.de/3206/.

Manuel J. A. Eugster and Friedrich Leisch. From Spider-man to Hero – archetypal analysis in R. *Journal of Statistical Software*, 30(8):1–23, 2009. URL http://www.jstatsoft.org/v30/i08.

Manuel J. A. Eugster and Friedrich Leisch. Exploratory analysis of benchmark experiments – an interactive approach. *Computational Statistics*, 2010. doi: 10.1007/s00180-010-0227-z. Accepted for publication on 2010-06-08, preprint available from http://epub.ub.uni-muenchen.de/10604/.

Manuel J. A. Eugster and Friedrich Leisch. Weighted and robust archetypal analysis. *Computational Statistics and Data Analysis*, 55(3):1215–1225, 2011. doi: 10.1016/j.csda.2010.10.017. Preprint available from http://epub.ub.uni-muenchen.de/11498/.

Manuel J. A. Eugster, Torsten Hothorn, and Friedrich Leisch. Exploratory and inferential analysis of benchmark experiments. Under review, preprint available from http://epub.ub.uni-muenchen.de/4134/, 2010a.

Manuel J. A. Eugster, Torsten Hothorn, and Friedrich Leisch. Domain-based benchmark experiments: Exploratory and inferential analysis. Under review, preprint available from http://epub.ub.uni-muenchen.de/4134/, 2010b.

Manuel J. A. Eugster, Friedrich Leisch, and Carolin Strobl. (Psycho-)analysis of benchmark experiments – a formal framework for investigating the relationship between data sets and learning algorithms. Under review, preprint available from http://epub.ub.uni-muenchen.de/11425/, 2010c.

Robin Farquharson. *Theory of Voting*. Oxford Blackwell, 1969. ISBN 0300011210.

DLR-DFD Federal Environment Agency. CORINE Land Cover (CLC2006). Deutsches Zentrum für Luft- und Raumfahrt e.V., 2004. URL http://www.

corine.dfd.dlr.de/.

Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software—Practice and Experience*, 30(11):1203–1233, 2000. doi: 10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N.

Salvador Garcia and Francisco Herrera. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996. ISBN 0801854148.

Alexander Gouberman and Simon Urbanek. *icp: Interactive Custom Plots – Customizable Interactive Graphics for R*, 2008. URL http://www.rosuda.org/iPlots/. R package version 1.1-0.

Georg Hager and Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, 2010. ISBN 9781439811924.

David J. Hand. Recent advances in error rate estimation. *Pattern Recognition Letters*, 4(5):335–346, 1986. doi: 10.1016/0167-8655(86)90054-1.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, second edition, 2009. ISBN 9780387848570.

William D. Heavlin. Archetypal analysis of computer performance. Conference talk at the the "38th Symposium on the Interface of Computing Science, Statistics and Applications" (Interface 2006), May 23–26, Pasadena, CA, USA, 2006. URL http://home.comcast.net/~bill.heavlin/heavlin_2006Interface.pdf.

Grete Heinz, Louis J. Peterson, Roger W. Johnson, and Carter J. Kerk. Exploring relationships in body dimensions. *Journal of Statistics Education*, 11(2), 2003.

Martin Hellmich and Gerhard Hommel. Multiple testing in adaptive designs: A review. *Institute of Mathematical Statistics Lecture Notes-Monograph Series: Recent Developments in Multiple Comparison Procedures*, 47:33–47, 2004. doi: 10.1214/lnms/1196285624.

Sebastian Henschel, Cheng Soon Ong, Mikio L. Braun, Soeren Sonnenburg, and Patrik O. Hoyer. MLdata: Machine learning benchmark repository. Website,

2010. http://mldata.org/; visited on April 13, 2011.

Robert J. Hijmans, Susan E. Cameron, Juan L. Parra, Peter G. Jones, and Andy Jarvis. Very high resolution interpolated climate surfaces for global land areas. *International Journal of Climatology*, 25(15):1965–1978, 2005. doi: 10.1002/joc.1276. URL http://worldclim.org.

Myles Hollander and Douglas A. Wolfe. *Nonparametric Statistical Methods*. John Wiley & Sons, second edition, 1999. ISBN 0471190454.

Kurt Hornik and David Meyer. Deriving consensus rankings from benchmarking experiments. In R. Decker and H.-J. Lenz, editors, *Advances in Data Analysis (Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation e.V., Freie Universität Berlin, March 8–10, 2006*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 163–170. Springer-Verlag, 2007. doi: 10.1007/978-3-540-70981-7_19.

Kurt Hornik and David Meyer. *relations: Data Structures and Algorithms for Relations*, 2010. URL http://CRAN.R-project.org/package=relations. R package version 0.5-8.

Torsten Hothorn, Friedrich Leisch, Achim Zeileis, and Kurt Hornik. The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, 14(3):675–699, 2005. doi: 10.1198/106186005X59630.

Torsten Hothorn, Kurt Hornik, Mark A. van de Wiel, and Achim Zeileis. A Lego system for conditional inference. *The American Statistician*, 60(3), 2006. doi: 10.1198/000313006X118430.

Torsten Hothorn, Frank Bretz, and Peter Westfall. Simultaneous inference in general parametric models. *Biometrical Journal*, 50(3), 2008. doi: 10.1002/bimj.200810425.

Lung-Cheng Huang, Sen-Yen Hsu, and Eugene Lin. A comparison of classification methods for predicting Chronic Fatigue Syndrome based on genetic data. *Journal of Translational Medicine*, 7(1):81–89, 2009. doi: 10.1186/1479-5876-7-81.

Peter J. Huber and Elvezio M. Ronchetti. *Robust Statistics*. John Wiley & Sons, Inc., 2 edition, 2009. ISBN 0471418056.

Raj K. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1991. ISBN 0471503361.

David A. James. *RSQLite: SQLite interface for R*, 2010. URL http://CRAN.R-project.org/package=RSQLite. R package version 0.9-3.

Alexandros Kalousis and Melanie Hilario. Model selection via meta-learning: A comparative study. In *Proceedings of the 12th International IEEE Conference on Tools with Artificial Intelligence*, pages 406–413, 2000. doi: 10.1109/TAI.2000.889901.

Alexandros Kalousis, Joao Gama, and Melanie Hilario. On data and algorithms: Understanding inductive performance. *Machine Learning*, 54:275–312, 2004. doi: 10.1023/B:MACH.0000015882.38031.85.

John G. Kemeny and James L. Snell. *Mathematical Models in the Social Sciences*. MIT Press, 1972. ISBN 0262610302.

Ross D. King, C. Feng, and A. Sutherland. STATLOG: Comparison of classification algorithms on large real-world problems. *Applied Artifcial Intelligence*, 9:289–333, 1995. doi: 10.1080/08839519508945477.

Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, 1974. ISBN 0898713560.

Friedrich Leisch. A toolbox for k-centroids cluster analysis. *Computational Statistics and Data Analysis*, 51(2):526–544, 2006. doi: 10.1016/j.csda.2005.10.006.

Shan Li, Paul Wang, Jordan Louviere, and Richard Carson. Archetypal analysis: A new way to segment markets based on extreme individuals. In *A Celebration of Ehrenberg and Bass: Marketing Knowledge, Discoveries and Contribution. Proceedings of the ANZMAC 2003 Conference, December 1-3, 2003*, pages 1674–1679, 2003.

Andy Liaw and Matthew Wiener. Classification and regression by randomForest. *R News*, 2(3):18–22, 2002. URL http://CRAN.R-project.org/doc/Rnews/.

Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228, 2000. doi: 10.1023/A:1007608224229.

Robert D. Luce and Howard Raiffa. *Games and Desicions*. John Wiley & Sons, 1957. ISBN 0486659437.

David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, second edition, 1984. ISBN 0201157942.

David Martens, Bart Baesens, Tony Van Gestel, and Jan Vanthienen. Comprehensible credit scoring models using rule extraction from support vector machines. *European Journal of Operational Research*, 183(3):1466–1476, 2007. doi: 10.1016/j.ejor.2006.04.051.

Kevin McGarigal, Sam A. Cushman, Maile C. Neel, and Eduard Ene. *FRAGSTATS: Spatial Pattern Analysis Program for Categorical Maps*, 2002. Computer software program produced by the authors at the University of Massachusetts, Amherst.

Geoffrey J. McLachlan. Error rate estimation in discriminant analysis: Recent advances. In Arjun K. Gupta, editor, *Advances in Multivariate Statistical Analysis*, pages 233–252. 1987.

Donald R. McNeil. *Interactive Data Analysis*. Wiley, New York, 1977. ISBN 978-0471026310.

Merriam-Webster Online Dictionary. Archetype. In *Merriam-Webster Online Dictionary*. 2008. Retrieved October 6, 2008, from http://www.merriam-webster.com/dictionary/archetype.

David Meyer, Friedrich Leisch, and Kurt Hornik. The support vector machine under test. *Neurocomputing*, 55:169–186, September 2003. doi: 10.1016/S0925-2312(03)00431-4.

Stephan Morgenthaler. A survey of robust statistics. *Statistical Methods and Applications*, 15(3):271–293, 2007. doi: 10.1007/s10260-007-0057-5.

Katharine M. Mullen and Ivo H. M. van Stokkum. *nnls: The Lawson-Hanson algorithm for non-negative least squares (NNLS)*, 2010. URL http://CRAN.R-project.org/package=nnls. R package version 1.3.

Hans-Helge Müller and Helmut Schäfer. Adaptive group sequential designs for clinical trials: Combining the advantages of adaptive and of classical group sequential approaches. *Biometrics*, 57(3):886–891, 2001. doi: 10.1111/j.0006-341X.2001.00886.x.

Claude Nadeau and Yoshua Bengio. Inference for the generalization error. *Machine Learning*, 52(3):239–281, 2003. doi: 10.1023/A:1024068626366.

Hannu Nurmi. Discrepancies in the outcomes resulting from different voting schemes. *Theory and Decision*, 25(2):193–208, 1988. doi: 10.1007/BF00134159.

Bernhard Pfahringer and Hilan Bensusan. Meta-learning by landmarking various learning algorithms. In Pat Langley, editor, *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 743–750, 2000.

José C. Pinheiro and Douglas M. Bates. *Mixed-Effects Models in S and S-PLUS*. Springer-Verlag, 2000. ISBN 0387989579.

Dimitris N. Politis and Joseph P. Romano. Large sample confidence regions based on subsamples under minimal assumptions. *Annals of Statistics*, 22(4):2031–2050, 1994. doi: 10.1214/aos/1176325770.

Giovanni C. Porzio, Giancarlo Ragozini, and Domenico Vistocco. On the use of archetypes as benchmarks. *Applied Stochastic Models in Business and Industry*, 24(5):419–437, 2008. doi: 10.1002/asmb.v24:5.

John Quackenbush. Microarray data normalization and transformation. *Nature Genetics*, 32:496–501, 2002. doi: 10.1038/ng1032.

John R. Quinlan. *C4.5: Programms for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993. ISBN 1558602380.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. URL http://www.R-project.org. ISBN 3-900051-07-0.

Peter J. Rousseeuw and Annick M. Leroy. *Robust Regression and Outlier Detection*. John Wiley & Sons, Inc., 2003. ISBN 0471852333.

Helmut Schäfer, Nina Timmesfeld, and Hans-Helge Müller. An overview of statistical approaches for adaptive designs and design modifications. *Biometrical Journal*, 48(4):507–520, 2006. doi: 10.1002/bimj.200510234.

Theresa Scharl and Friedrich Leisch. gcExplorer: Interactive exploration of gene clusters. *Bioinformatics*, 25(8):1089–1090, 2009. doi: 10.1093/bioinformatics/btp099.

Rosa A. Schiavo and David J. Hand. Ten more years of error rate research. *International Statistical Review*, 68(3):295–310, 2000. doi: 10.1111/j.1751-5823.2000.tb00332.x.

Helmut Schlumprecht and Georg Waeber. *Heuschrecken in Bayern*. Ulmer, 2003. ISBN 3800138832.

Douglas Steinly. k-means clustering: A half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, pages 1–34, 2006. doi: 10.1348/000711005X48266.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society, Series B*, 36(2):111–147, 1974.

Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(25), 2007. doi: 10.1186/1471-2105-8-25.

Carolin Strobl, Florian Wickelmaier, and Achim Zeileis. Accounting for individual differeneces in Bradley-Terry models by means of recursive partitioning. *Journal of Educational and Behavioral Statistics*, 2010. In press.

Terry M. Therneau and Beth Atkinson. ***rpart**: Recursive Partitioning*, 2010. URL http://CRAN.R-project.org/package=rpart. R package version 3.1-46; R port by Brian Ripley.

Antony Unwin, Chris Volinsky, and Sylvia Winkler. Parallel coordinates for exploratory modelling analysis. *Computational Statistics & Data Analysis*, 43:553–564, 2003. doi: 10.1016/S0167-9473(02)00292-X.

Simon Urbanek. *Acinonyx: iPlots Extreme*, 2009. URL http://www.rforge.net/Acinonyx/. R package version 3.0-0.

Simon Urbanek and Martin Theus. iPlots: High interaction graphics for R. In Kurt Hornik, Friedrich Leisch, and Achim Zeileis, editors, *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, 2003.

Simon Urbanek and Tobias Wichtrey. *iplots: iPlots - Interactive Graphics for R*, 2008. URL http://www.iPlots.org/. R package version 1.1-3.

Marc Vandemeulebroecke. Group sequential and adaptive designs – a review of basic concepts and points of discussion. *Biometrical Journal*, 50(3), 2008. doi: 10.1002/bimj.200710436.

William Venables and Brian Ripley. *Modern Applied Statistics with S*. Springer-Verlag, fourth edition, 2002. ISBN 0387954570.

Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002. doi: 10.1023/A:1019956318069.

Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2), 1945. doi: 10.1214/aoms/1177731118.

Gernot Wassmer. Basic concepts of group sequential and adaptive group sequential test procedures. *Statistical Papers*, 41, 2000. doi: 10.1007/BF02925923.

Stefan Wellek. *Testing Statistical Hypotheses of Equivalence.* Chapman & Hall, 2003. ISBN 1584881607.

Hadley Wickham. Meifly: Models explored interactively. Website ASA Sections on Statistical Computing and Graphics (Student Paper Award Winner 2007), 2007. Available online at `http://stat-computing.org/awards/student/winners.html`.

Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis.* Springer, 2009. ISBN 978-0-387-98140-6.

Achim Zeileis, Torsten Hothorn, and Kurt Hornik. Model-based recursive partitioning. *Journal of Computational and Graphical Statistics*, 17(2):492–514, 2008. doi: 10.1198/106186008X319331.