AUSTRIAN JOURNAL OF STATISTICS
Volume 41 (2012), Number 1, 5–26

Domain-Based Benchmark Experiments: Exploratory and Inferential Analysis

Manuel J. A. Eugster[1], Torsten Hothorn[1], and Friedrich Leisch[2]

[1]Institut für Statistik, LMU München, Germany
[2]Institut für Angewandte Statistik und EDV, BOKU Wien, Austria

Abstract: Benchmark experiments are the method of choice to compare learning algorithms empirically. For collections of data sets, the empirical performance distributions of a set of learning algorithms are estimated, compared, and ordered. Usually this is done for each data set separately. The present manuscript extends this single data set-based approach to a joint analysis for the complete collection, the so called problem domain. This enables to decide which algorithms to deploy in a specific application or to compare newly developed algorithms with well-known algorithms on established problem domains.

Specialized visualization methods allow for easy exploration of huge amounts of benchmark data. Furthermore, we take the benchmark experiment design into account and use mixed-effects models to provide a formal statistical analysis. Two domain-based benchmark experiments demonstrate our methods: the UCI domain as a well-known domain when one is developing a new algorithm; and the Grasshopper domain as a domain where we want to find the best learning algorithm for a prediction component in an enterprise application software system.

Zusammenfassung: Benchmark-Experimente sind die Methodik der Wahl um Lernalgorithmen empirisch zu vergleichen. Für eine gegebene Menge von Datensätzen wird die empirische Güte-Verteilung verschiedener Lernalgorithmen geschätzt, verglichen und geordnet. Normalerweise geschieht dies für jeden Datensatz einzeln. Der vorliegende Artikel erweitert diesen Datensatz-basierten Ansatz zu einem Domänen-basierten Ansatz, in welchem nun eine gemeinsame Analyse für die Menge der Datensätze durchgeführt wird. Dies ermöglicht unter anderem die Auswahl eines Algorithmus in einer konkreten Anwendung, und der Vergleich neu entwickelter Algorithmen mit bestehenden Algorithmen auf wohlbekannten Problemdomänen.

Speziell entwickelte Visualisierungen ermöglichen eine einfache Untersuchung der immensen Menge an erzeugten Benchmark-Daten. Des Weiteren verwenden wir Gemischte Modelle um eine formale statistische Analyse durchzuführen. Anhand zweier Benchmark-Experimenten illustrieren wir unsere Methoden: die UCI-Domäne als Vertreter einer wohlbekannten Problemdomäne für die Entwicklung neuer Lernalgorithmen; und die Grasshopper-Domäne, für welche wir den besten Lernalgorithmus als Vorhersagekomponente innerhalb einer konkreten Anwendung finden wollen.

Keywords: Benchmark Experiment, Learning Algorithm, Visualisation, Inference, Mixed-Effects Model, Ranking.

1 Introduction

In statistical learning, benchmark experiments are empirical investigations with the aim of comparing and ranking learning algorithms with respect to a certain performance measure. In particular, on a data set of interest the empirical performance distributions of a set of learning algorithms are estimated. 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 by Hothorn et al. (2005); they introduce a theoretical framework for inference problems in benchmark experiments and show that standard statistical test procedures can be used to test for differences in the performances. The practical toolbox is provided by Eugster (2011, 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 paper is 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.

From our point of view the benchmarking process consists of three hierarchical levels: (1) In the *Setup* level the design of the benchmark experiment is defined, i.e., data sets, candidate algorithms, performance measures and a suitable resampling strategy 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 exploratory and inferential methods. This paper covers the Setup and Analysis level and is organized as follows: Section 2 reviews the theoretical framework for benchmark experiments defined by Hothorn et al. (2005) and extends it for sets of data sets. In Section 3 we first define 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 3.1 introduces visualization methods to present the results in their entirety. In Section 3.2 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. In Section 4 we demonstrate the methods on two problem domains: The *UCI domain* (Section 4.1) as a well-known domain; useful, for example, when one is developing a new algorithm. The *Grasshopper domain* (Section 4.2) 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 5 concludes the article with a summary and future work. All computational details are provided in the section "Computational details" on page 23. All methods proposed in this paper have been fully automated and will be made available as open source software upon publication of this manuscript.

2 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 trial the algorithms are fitted on a learning sample and validated on the corresponding validation sample according to the specified performance measures.

Formally, following Hothorn et al. (2005), a benchmark experiment (for one data set and one performance measure) is defined as follows: Given is a data set $\mathfrak{L} = \{z_1, \dots, z_N\}$. We draw $b = 1, \dots, B$ learning samples of size n using a resampling scheme, such as sampling with replacement (bootstrapping, usually of size $n = N$) or subsampling without replacement ($n < N$):

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

We assume that there are $K > 1$ candidate algorithms a_k, $k = 1, \dots, K$, available for the solution of the underlying learning problem. For each algorithm a_k the function $a_k(\,\cdot \mid \mathfrak{L}^b)$ is the fitted model based on a learning sample $\mathfrak{L}^b$ ($b = 1, \dots, B$). This function itself has a distribution $\mathcal{A}_k$ as it is a random variable depending on $\mathfrak{L}$:

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

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

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

The p_{bk} are samples drawn from the distribution $\mathcal{P}_k(\mathfrak{L})$ of the performance measure of the algorithm a_k on the data set $\mathfrak{L}$.

This paper focuses on the important case of supervised learning tasks, i.e., each observation $z \in \mathfrak{L}$ 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 $i = 1, \dots, N$ for x, y, and z). The aim of a supervised learning task is to construct a learner $\hat{y} = a_k(x \mid \mathfrak{L}^b)$ 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 $z \in \mathfrak{L}$ is measured by a scalar loss function $l(y, \hat{y})$. The performance measure p is then defined by some functional μ of the loss function's distribution over all observations:

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

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

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

and the deviance for predicted class probabilities $\hat{y}_g$ ($g = 1, \dots, G$ 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). With a view to practicability in real-world applications, further interesting performance measures are the algorithm's execution time and the memory requirements (for fitting and prediction, respectively).

The focus of benchmark experiments is on the general performance of the the candidate algorithms. Therefore, using the performance on the learning data set $\mathfrak{L}^b$ as basis for further analyses is not a good idea (as commonly known). Thus – as in most cases we are not able to compute μ analytically – we use the empirical functional $\hat{\mu}_{\mathfrak{T}}$ based on a test sample $\mathfrak{T}$:

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

This means we compute the performance of the model (fitted on the learning sample $\mathfrak{L}^b$) for each observation in the test sample $\mathfrak{T}$ and apply the empirical functional $\hat{\mu}$ to summarize over all observations. Due to resampling effects it is not given that $\hat{\mathcal{P}}_k$ approximates $\mathcal{P}_k$; for example cross-validation overestimates the true performance distribution. The most common example for the empirical functional $\hat{\mu}$ is the mean, that is the empirical analogue of the expectation. Most further analysis methods require independent observations of the performance measure, therefore we define the validation sample $\mathfrak{T}$ in terms of out-of-bag observations: $\mathfrak{T} = \mathfrak{L} \setminus \mathfrak{L}^b$.

Now, in real-world applications we are often interested in more than one performance measure (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 $\mathfrak{D} = \{\mathfrak{L}_1, \dots, \mathfrak{L}_M\}$:

$$p_{mbkj} = p_j(a_k, \mathfrak{L}_m^b) \sim \mathcal{P}_{mk}^j = \mathcal{P}_k^j(\mathfrak{L}_m) .$$

The p_{mbkj} are samples drawn from the jth performance distribution $\mathcal{P}_k^j(\mathfrak{L}_m)$ 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 $\hat{\mathcal{P}}_k^j(\mathfrak{L}_m)$ is estimated.

3 Analysis of Experiments

The execution of a benchmark experiment as defined in Section 2 results in $M \times B \times K \times J$ raw performance measures, i.e., $M \times K \times J$ empirical performance distributions $\hat{\mathcal{P}}_{mk}^j$. 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 paper we assume that the following systematic stepwise approach 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 $\hat{\mathcal{P}}_{mk}^j$ 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 (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 $\mathcal{R} = \{\bar{R}_1, \dots, \bar{R}_M\}$ 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 $\mathfrak{D}$.

3.1 Visualizing the Domain

A benchmark experiment results in $M \times K \times J$ estimated performance distributions $\hat{\mathcal{P}}^j_{mk}$. 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 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 4.1 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 $\hat{\mathcal{P}}^j_{mk}$ 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 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 2b is explained further down.)

Now, in addition to the empirical performance distributions $\hat{\mathcal{P}}^j_{mk}$, 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 introduce an appropriate 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 (we refer to the original publication for the definition and explanation of the axioms). The *symmetric difference distance* d_Δ satisfies these axioms; it 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

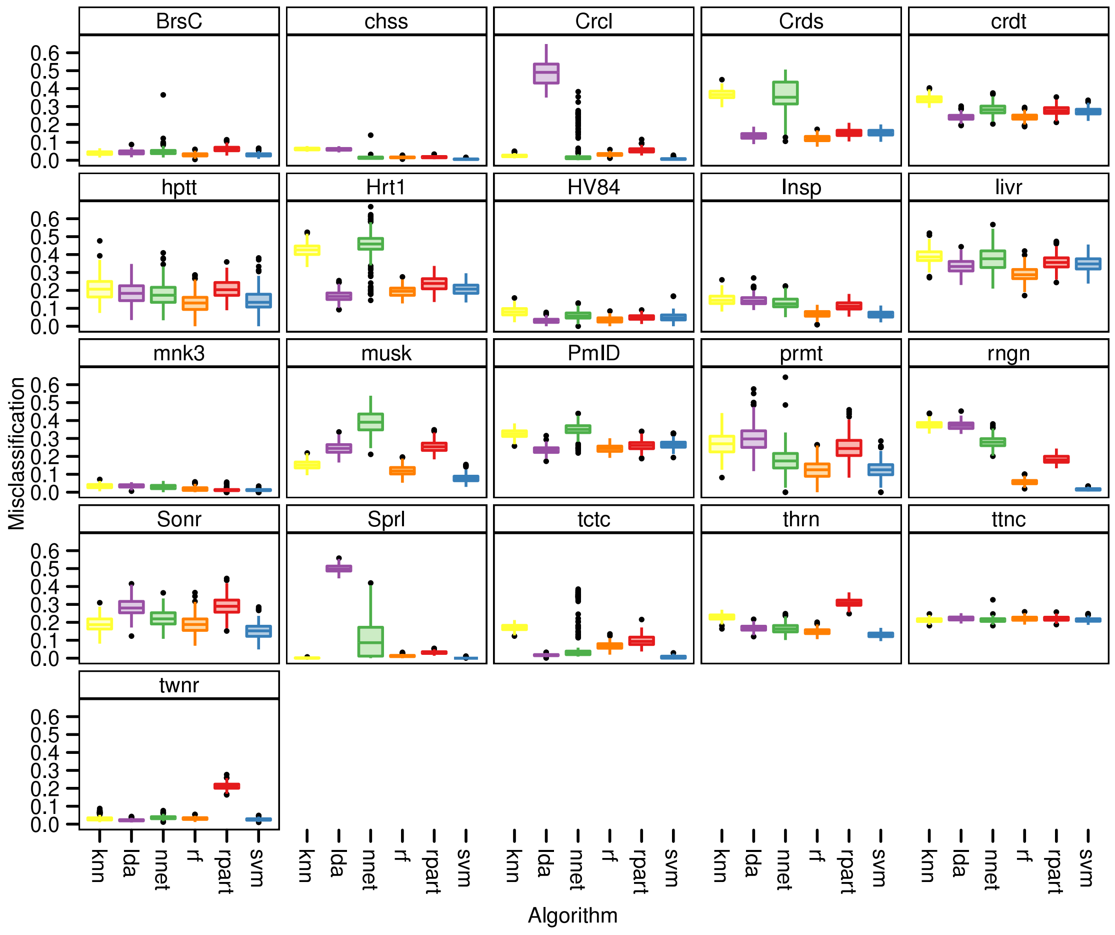


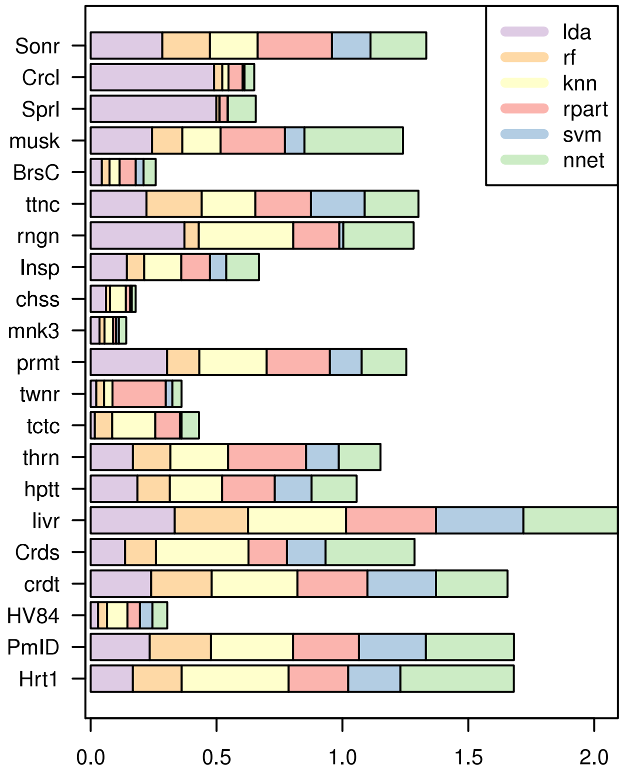
Figure 1: Trellis graphic with box plot panels. The plot shows the misclassification error of the UCI domain benchmark experiment described in Section 4.1.

XOR operator):

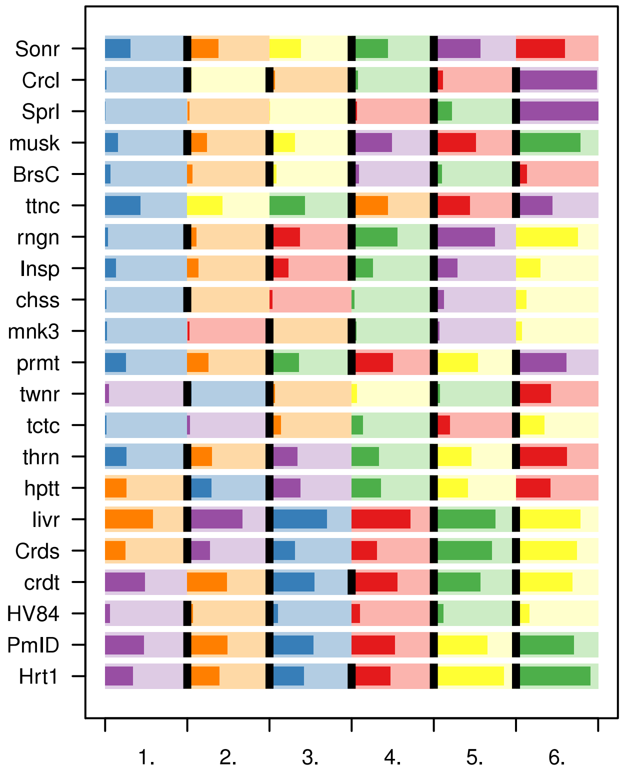
$$d_{\triangle}(R_1, R_2) = \#\{(a_k, a_{k'}) | (a_k, a_{k'}) \in R_1 \oplus (a_k, a_{k'}) \in R_2, k, k' = 1, \dots, K\} .$$

Computing all pairwise distances for the relations R_m $(m = 1, \dots, M)$ 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 to use agglomerative hierarchical clustering with complete linkage (see, e.g., Hastie et al., 2009). Figure 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 2a), even if the data sets are ordered according to the data sets' linear order determined by the hierarchical clustering.

The *benchmark summary plot* (bsplot) overcomes these difficulties by adapting the



(a)



(b)

Figure 2: Visualizations of the candidate algorithms' misclassification errors on the UCI domain: (a) stacked bar plot; (b) benchmark summary plot (legend omitted).

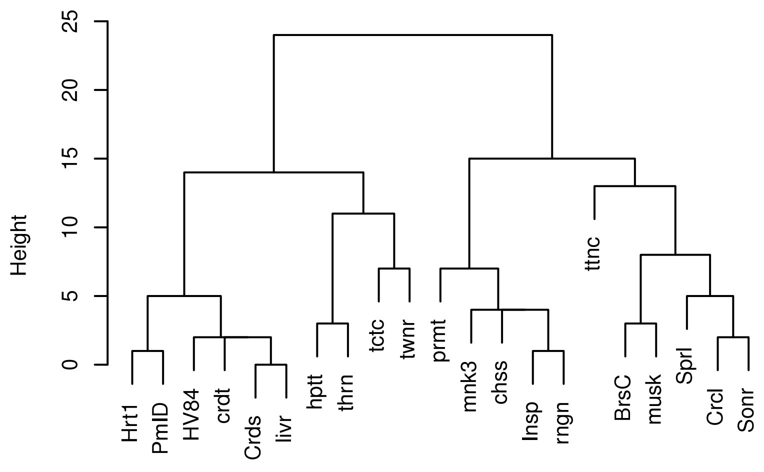


Figure 3: Dendrogram showing the clustering of the UCI domain's data sets based on their candidate algorithms' performance relations.

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 $\hat{\mathcal{P}}^j_{mk}$ 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 2b shows the UCI domain's bsplot. In comparison with the stacked bar plot (Figure 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.

In the UCI problem domain the resulting relations are all transitive, this is not generally true for relations based on statistical tests (see Eugster, 2011), 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

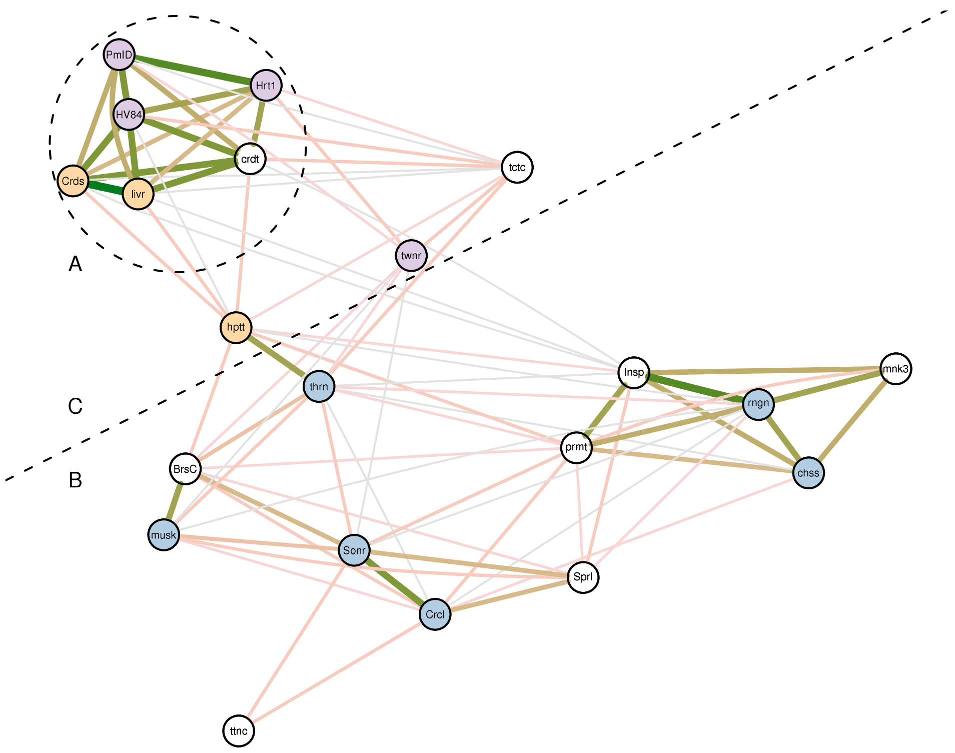


Figure 4: Benchmark summary graph visualizing the relation of the UCI domain's data sets based on the distance matrix. The color of the nodes indicate the unique winner algorithm; otherwise unfilled. The dashed circle highlights a subset A with similar algorithm performances. The dashed line indicates two sub-domains; B – where `svm` performs best, and C – where `rf` and `lda` perform best.

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 edges' widths and colors 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 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 subdomain algorithms are always winner as well together with other algorithms (compare with Figure 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. See Eugster and Leisch (2010) for the interactive analysis of benchmark experiments based on a single data set. 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? We investigate this question in Eugster et al. (2010), 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.

3.2 Modeling 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 $\hat{\mathcal{P}}^j_{mk}$ 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

$$p_{mbk} = \kappa_k + \beta_m + \beta_{mk} + \beta_{mb} + \epsilon_{mbk} \quad (1)$$

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.

Linear mixed-effects models are the appropriate tool to estimate the parameters described in Formula 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 Eugster, 2011).

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

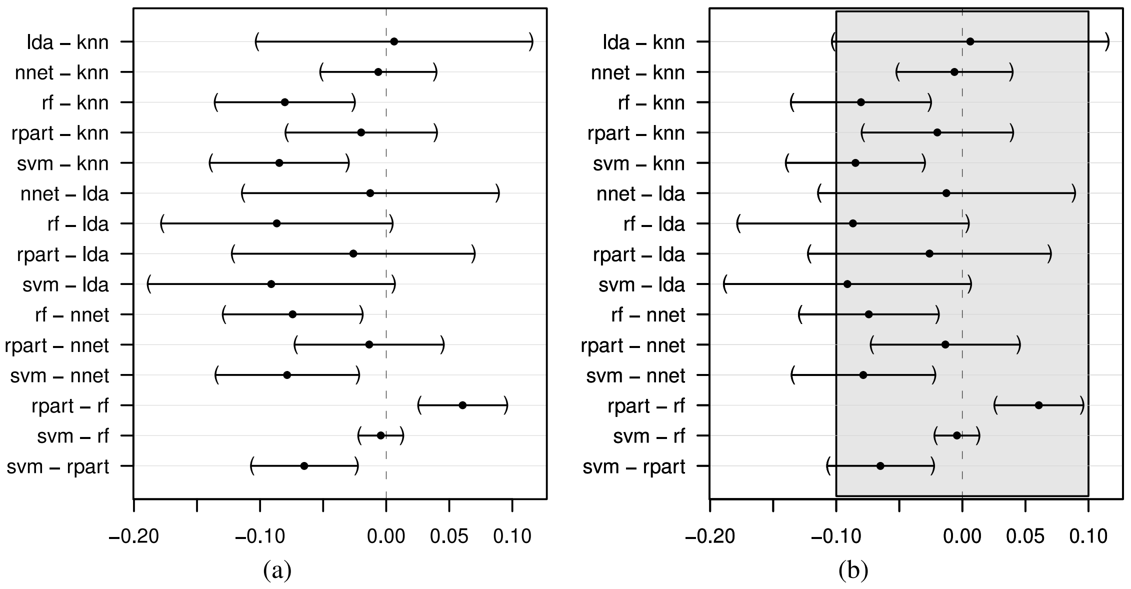


Figure 5: The UCI domain's simultaneous 95% confidence intervals for multiple (a) significant and (b) relevant comparisons for a fitted linear mixed-effects model on the algorithms' misclassification errors.

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 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 2b; Section 4.1 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 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 data set-based benchmark experiments. Following Eugster (2011, Chapter 2), we use 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 ($j = 1, \dots, J$). Consensus methods aggregate the J preference relations to single domain-based order relation of the candidate algorithms. This can be seen as a multi-criteria or multi-objective 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 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.

4.1 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-

Table 1: UCI domain's chains of preference relations $\mathcal{R} = \{R_1, \dots, R_{21}\}$.

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

3) given in the beginning of Section 3 (and defined in Eugster, 2011). 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 the section on computational details on page 23).

The Trellis plot in Figure 1 shows the box plots of the estimated performance distributions. Table 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 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

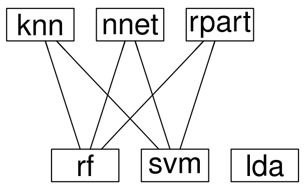
`lda` has the worst, `svm` the best mean performance. Data set `mnk3` has the lowest and data set `livr` the highest estimated performance effect (i.e., complexity) among the domain's data sets:

	$\hat{\beta}_m$	$\hat{\beta}_{mk}$					
		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 1 and 2.

	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 6: Interpretation of the pairwise significant differences, i.e., Figure 5a, as preference relation: (a) incidence matrix, (b) the corresponding Hasse diagram (nodes are ordered bottom-up).

Figure 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 6a. 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 6b 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.

4.2 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

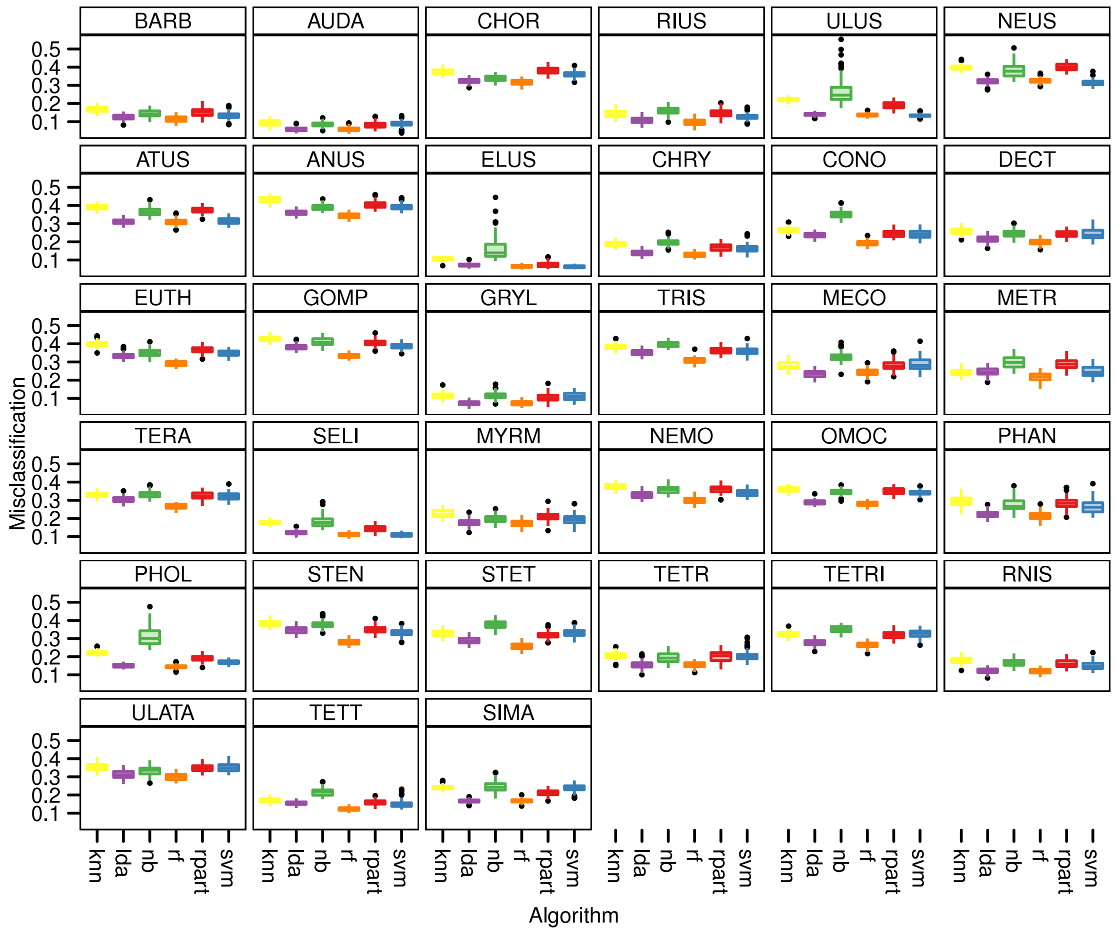


Figure 7: Trellis graphic with box plot of the candidate algorithms' misclassification error on the Grasshopper domain. Each data set is one grasshopper species.

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`, green), 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 the section on computational details on page 23).

Figure 7 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 $\mathcal{R} = \{R_1, \dots, R_{33}\}$ (using mixed-effects models; see Eugster, 2011) 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 6b) 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 8a 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 7).

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 8b shows the corresponding simultaneous 95% confidence intervals and the resulting order is:

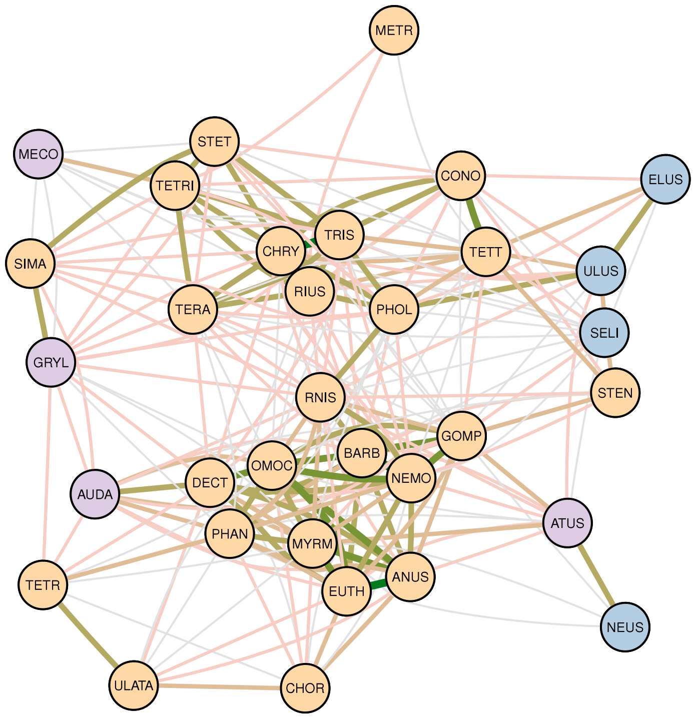
$$\texttt{rf} \prec \texttt{lda} \prec \texttt{svm} \prec \texttt{rpart} \prec \texttt{nb} \sim \texttt{knn}\,.$$

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.

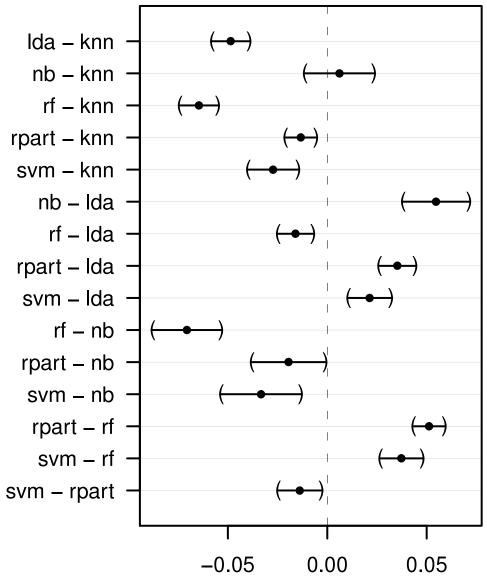
5 Summary

The great many of published benchmark experiments show that this method is the primary choice to evaluate learning algorithms. Hothorn et al. (2005) define the theoretical framework for inference problems in benchmark experiments. Eugster (2011) introduce the practical toolbox with a systematic four step approach from exploratory analysis via formal investigations through to a preference relation of the algorithms. The present publication 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 paper 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



(a)



(b)

Figure 8: (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.

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. The two domain-based benchmark experiments show that this domain-based relation conforms with the exploratory analysis and the approach of aggregating the individual local relations using consensus methods.

Computational Details

All computations and graphics have been done using R 2.11.1 (R Development Core Team, 2010) and additional add-on packages.

Setup and Execution: For the candidate algorithms of the two benchmark experiments the following functions and packages have been used: Function `lda` from package `MASS` for linear 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., 2009). 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 three algorithms are described by Venables and Ripley (2002). Function `rpart` from package `rpart` (Therneau and Atkinson, 2009) 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.

Analysis: The presented toolbox of exploratory and inferential methods is implemented in the package `benchmark` (Eugster, 2010). 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; `ggplot2` (Wickham, 2009) to produce the graphics.

Code for replicating our analysis is available in the `benchmark` package via `demo(package = "benchmark")`.

References

Abernethy, J., and Liang, P. (2010). *MLcomp.* Website. (`http://mlcomp.org/`; visited on December 20, 2011)

Asuncion, A., and Newman, D. (2007). *UCI machine learning repository.* Website. Available from `http://www.ics.uci.edu/~mlearn/MLRepository.html`

Bates, D., and Maechler, M. (2010). `lme4`: Linear mixed-effects models using S4 classes [Computer software manual]. Available from `http://lme4.r-forge.r-project.org/` (R package version 0.999375-35)

Becker, R. A., Cleveland, W. S., and Shyu, M.-J. (1996). The visual design and control of Trellis display. *Journal of Computational and Graphical Statistics*, *5*(2), 123–155.

Dimitriadou, E., Hornik, K., Leisch, F., Meyer, D., , and Weingessel, A. (2009). `e1071`: Misc functions of the department of statistics (e1071), tu wien [Computer software manual]. Available from `http://CRAN.R-project.org/package=e1071` (R package version 1.5-19)

Eugster, M. J. A. (2010). `benchmark`: Benchmark experiments toolbox [Computer software manual]. Available from `http://CRAN.R-project.org/package=benchmark` (R package version 0.3)

Eugster, M. J. A. (2011). *Benchmark Experiments – A Tool for Analyzing Statistical Learning Algorithms.* Dr. Hut-Verlag. Available from `http://edoc.ub.uni-muenchen.de/12990/` (PhD thesis, Department of Statistics, Ludwig-Maximilians-Universität München, Munich, Germany)

Eugster, M. J. A., and Leisch, F. (2010). Exploratory analysis of benchmark experiments – an interactive approach. *Computational Statistics.* (Accepted for publication on 2010-06-08)

Eugster, M. J. A., Leisch, F., and Strobl, C. (2010). *(Psycho-)analysis of benchmark experiments – a formal framework for investigating the relationship between data sets and learning algorithms* (Technical Report No. 78). Institut für Statistik, Ludwig-Maximilians-Universität München, Germany. Available from `http://epub.ub.uni-muenchen.de/11425/`

Federal Environment Agency, D.-D. (2004). *CORINE Land Cover (CLC2006).* Available from http://www.corine.dfd.dlr.de/ (Deutsches Zentrum für Luft- und Raumfahrt e.V.)

Gansner, E. R., and North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software — Practice and Experience*, *30*(11), 1203–1233.

Hager, G., and Wellein, G. (2010). *Introduction to High Performance Computing for Scientists and Engineers.* CRC Press.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (second ed.). Springer-Verlag.

Henschel, S., Ong, C. S., Braun, M. L., Sonnenburg, S., and Hoyer, P. O. (2010). *MLdata: Machine learning benchmark repository.* Website. (http://mldata.org/; visited on December 20, 2011)

Hijmans, R. J., Cameron, S. E., Parra, J. L., Jones, P. G., and Jarvis, A. (2005). Very high resolution interpolated climate surfaces for global land areas. *International Journal of Climatology*, *25*(15), 1965–1978. Available from http://worldclim.org

Hornik, K., and Meyer, D. (2007). Deriving consensus rankings from benchmarking experiments. In R. Decker and H.-J. Lenz (Eds.), *Advances in Data Analysis (Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation e.V., Freie Universität Berlin, March 8–10, 2006* (pp. 163–170). Springer-Verlag.

Hornik, K., and Meyer, D. (2010). `relations`: Data structures and algorithms for relations [Computer software manual]. Available from http://CRAN.R-project.org/package=relations (R package version 0.5-8)

Hothorn, T., Bretz, F., and Westfall, P. (2008). Simultaneous inference in general parametric models. *Biometrical Journal*, *50*(3), 346–363. Available from http://cran.r-project.org/package=multcomp

Hothorn, T., Hornik, K., Wiel, M. A. van de, and Zeileis, A. (2006). A Lego system for conditional inference. *The American Statistician*, *60*(3). Available from http://CRAN.R-project.org/package=coin

Hothorn, T., Leisch, F., Zeileis, A., and Hornik, K. (2005). The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, *14*(3), 675–699.

Kemeny, J. G., and Snell, J. L. (1972). *Mathematical Models in the Social Sciences.* MIT Press.

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

McGarigal, K., Cushman, S. A., Neel, M. C., and Ene, E. (2002). Fragstats: Spatial pattern analysis program for categorical maps [Computer software manual]. (Computer software program produced by the authors at the University of Massachusetts, Amherst)

Pfahringer, B., and Bensusan, H. (2000). Meta-learning by landmarking various learning algorithms. In *In Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 743–750). Morgan Kaufmann.

Pinheiro, J. C., and Bates, D. M. (2000). *Mixed-Effects Models in S and S-PLUS*. Springer.

R Development Core Team. (2010). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Available from `http://www.R-project.org` (ISBN 3-900051-07-0)

Scharl, T., and Leisch, F. (2009). gcExplorer: Interactive exploration of gene clusters. *Bioinformatics*, *25*(8), 1089–1090.

Schlumprecht, H., and Waeber, G. (2003). *Heuschrecken in Bayern*. Ulmer.

Therneau, T. M., and Atkinson, B. (2009). `rpart`: Recursive partitioning [Computer software manual]. Available from `http://CRAN.R-project.org/package=rpart` (R package version 3.1-43. R port by Brian Ripley)

Venables, W. N., and Ripley, B. D. (2002). *Modern Applied Statistics with S* (Fourth ed.). New York: Springer. Available from `http://www.stats.ox.ac.uk/pub/MASS4` (ISBN 0-387-95457-0)

Vilalta, R., and Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial Intelligence Review*, *18*(2), 77–95.

Wellek, S. (2003). *Testing Statistical Hypotheses of Equivalence*. Chapman & Hall.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer New York. Available from `http://had.co.nz/ggplot2/book`

Authors' addresses:

Manuel Eugster and Torsten Hothorn
Department of Statistics
LMU Munich
Ludwigstr. 33
80539 Munich
Germany

Friedrich Leisch
Institute of Applied Statistics and Computing
University of Natural Resources and Life Sciences
Gregor Mendel-Str. 33
1180 Wien
Austria