Wrapper Algorithms and their Performance Assessment on High-dimensional Molecular Data

Christoph Bernau

München 2014

Wrapper Algorithms and their Performance Assessment on High-dimensional Molecular Data

Christoph Bernau

Dissertation
an der Fakultät Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Christoph Bernau
aus Erding

München, den 13. März 2014

Erstgutachter: Prof. Dr. Anne-Laure Boulesteix
Zweitgutachter: Prof. Dr. Harald Binder
Tag der mündlichen Prüfung: 8. August 2014

Contents

Abstract **xv**

Zusammenfassung **xvii**

Introduction **1**

1 Estimating the error rate of wrapper algorithms **5**
1.1 Introduction 5
1.2 Internal CV and the unconditional error rate of wrapper algorithms 7
1.2.1 Settings and notations 7
1.2.2 Conditional and unconditional error rate 8
1.2.3 The unconditional error rate of wrapper algorithms 9
1.2.4 Revisiting internal cross-validation 10
1.3 A smooth analytical alternative to ICV 11
1.3.1 Basic Idea 11
1.3.2 A weighted mean approach 12
1.3.3 Algorithmic description of WMC 13
1.3.4 Shrinkage approach (WMCS) 15
1.3.5 Algorithmic description of WMCS 16
1.3.6 More details on estimating $\mathrm{P}(k^*(S) = k)$ 16
1.3.7 Implementation 17
1.4 Empirical results and comparison of the three estimators 17
1.4.1 Study design 17
1.4.2 Results on correlated gaussian data 20
1.4.3 Summary of results and comparison 22
1.5 Further analysis and theoretic considerations 25
1.5.1 Normality assumption 25
1.5.2 Runtimes 26
1.5.3 Decision Theoretic Interpretation 28
1.5.4 Asymptotic consideration – justification of $Err = \varepsilon^{n_L}(\phi)$ as target of Err_{ICV} 39
1.6 Discussion and concluding remarks 41

2 Ranking High-Dimensional Wrapper Algorithms 45
2.1 Introduction . . . 45
2.1.1 Notations and settings . . . 46
2.1.2 Algorithms considered . . . 47
2.1.3 Ranking algorithms by cross-study validation: the CSV matrix . . . 47
2.1.4 Summarization of the CSV matrix . . . 47
2.1.5 True global ranking . . . 48
2.1.6 Description of datasets . . . 48
2.1.7 Simulation design . . . 50
2.1.8 Further inspection of the implemented simulation scenario . . . 52
2.1.9 Evaluation criteria in simulations . . . 56
2.2 Results . . . 57
2.2.1 Simulated Data . . . 57
2.2.2 Application to breast cancer prognostic modelling . . . 59
2.2.3 $\overline{CV}$ performance is not necessarily indicative of cross-study performance . . . 61
2.2.4 Specialist and generalist algorithms . . . 63
2.3 Discussion . . . 63

3 Computational Aspects & Software 67
3.1 Introduction . . . 67
3.2 *survHD*: An application programming interface . . . 68
3.2.1 General concept . . . 68
3.2.2 Custom Survival Models . . . 71
3.3 Short introduction to parallel programming in **R** . . . 76
3.3.1 The package *Rmpi* . . . 76
3.3.2 The packages *snow* and *doSNOW* . . . 78
3.3.3 pbdMPI: Overcoming the master-worker concept . . . 81
3.4 Comparison of Cluster and Cloud Computing . . . 84
3.4.1 Simple Trend Benchmarks . . . 84
3.4.2 Test Project: Optimization Bias . . . 87
3.4.3 Cost Aspects: An example . . . 88
3.4.4 A Hybrid Cloud Solution: Combining Computer Cluster & Cloud Resources . . . 90
3.4.5 Hybridcloud: Case study . . . 91
3.4.6 Parallelization for *survHD* . . . 93
3.5 Working with large datasets in **R** . . . 96
3.5.1 Motivating Example: Normalization of microarrays . . . 96
3.5.2 Implementation . . . 98
3.5.3 Parallel file access in **R** . . . 98
3.6 Discussion . . . 102

Summary & Outlook 107

A Estimating the error rate of wrapper algorithms 111
A.1 Additional results on real datasets of Chapter 1 . . . 112
A.1.1 Alon data . . . 112
A.1.2 Singh data . . . 115
A.1.3 Golub data . . . 118
A.1.4 Chiaretti data . . . 121
A.2 Additional results of the simulations in Chapter 1 . . . 124
A.2.1 Correlated non-informative data . . . 128
A.2.2 Uncorrelated Gaussian data . . . 130
A.2.3 Differences in covariances between groups . . . 133
A.3 Normality assumption . . . 136
A.4 Further information on runtimes . . . 137

B Ranking High-Dimensional Wrapper Algorithms 139
B.1 Correlation of the rankings defined by CV and CSV in the simulation example 140
B.2 Simulation example with removal of outlier studies . . . 141
B.3 CV as an estimator of independent within-study discrimination performance 144
B.4 Rankings on experimental microarray data (outlier studies included) . . . 145
B.5 Rankings on experimental microarray data (outlier studies removed) . . . 146

C Computational Aspects & Software 147
C.1 Package Vignette: SurvHD - Survival Analysis for High-Dimensional Data 148
C.2 Bigmemory Benchmark at the IBE-Cluster . . . 173

D Papers containing parts of the work presented in this thesis 175
D.1 Correcting the Optimal Resampling-Based Error Rate by Estimating the Error Rate of Wrapper Algorithms (Bernau et al., 2013) . . . 175
D.2 Cross-study validation for assessment of prediction models and algorithms (conditionally accepted at Bioinformatics) . . . 186
D.3 Application of Microarray Analysis on Computer Cluster and Cloud Platforms (Bernau et al., 2013) . . . 196

Acknowledgments 212

List of Figures

1.1 Schematic illustration contrasting the weighted mean correction and internal cross-validation. . . . 14
1.2 Comparison of ICV, WMC and WMCS for the setup with correlated Gaussian data. . . . 21
1.3 Distribution of the shrinkage factor ξ for different numbers of observations for the setup with correlated Gaussian data. . . . 22
1.4 Comparison of ICV, WMC and WMCS for the selection setup on the prostate cancer dataset by Singh. . . . 24
1.5 Normal quantile plots of $e(k\|S)$ for the case $n = 40$ in simulation setup A.2. 26
1.6 Normal quantile plots of $e(k\|S)$ for the case $n = 40$ in simulation setup A.2.1. 27
1.7 Global decision problem. . . . 33
1.8 Local decision problem. . . . 35
1.9 Resampling scheme for local decision problem. . . . 38

2.1 Cross-study validation matrices $\mathbf{Z}^k$ in simulated and experimental data for Ridge Regression. . . . 51
2.2 Validity checks for the CoxBoost model fitted on the data set ST1. . . . 53
2.3 Simulated concordance indices on the $I = 8$ studies for the 'true' coefficients fitted on the data set ST1. . . . 54
2.4 Average concordance indices of the $\mathbf{Z}^k$-matrices for the first 50 iterations of the simulation. . . . 56
2.5 Comparison of cross-study validation and cross-validation on simulated data. 58
2.6 Distribution of Kendall's correlation between true algorithm rankings and estimated algorithm rankings . . . 60
2.7 Comparison of $\overline{CSV}$ and $\overline{CV}$ experimental data. . . . 62

3.1 Most important functions and objects in *survHD*. . . . 70
3.2 'Leave-One-In-Validation matrix' for the algorithm 'Plusminus' on 10 breast cancer studies. . . . 72
3.3 Speedup for pairwise logistic models on different computing clusters. . . . 80
3.4 Comparison of parallelization in *snow* and *pbdMPI*. . . . 83
3.5 Time needed for the broadcast of a 2GB Affybatch. . . . 86
3.6 Speedup for Project 1 on cluster and cloud instances. . . . 87

3.7 Hybrid-Cloud based on **R** and the doRedis-package. 92
3.8 Distribution of the 1000 subtasks over the involved clusters in the hybrid cloud at AWS EC2, IBE and LRZ. 94
3.9 Gantt chart of *doRedis / multicore*-implementation of a resampling-based hyper-parameter optimization on SuperMUC. 95
3.10 Cluster analysis of separately preprocessed microarrays in artificially created batches. 99
3.11 Cluster analysis of addon-preprocessed microarrays. 99
3.12 Gantt chart for a *bigmemory*-experiment with 500 processes at LRZ SuperMig. 101
3.13 Gantt chart for MPI-IO with 160 processes at LRZ SuperMig. 101
3.14 Parallel normalization of microarrays using a separate memory attached file. 103

A.1 Distribution of the shrinkage factor ξ for all simulation setups on the Alon dataset. 114
A.2 Distribution of the shrinkage factor ξ for all simulation setups on the Singh dataset. 117
A.3 Distribution of the shrinkage factor ξ for all simulation setups on the Golub dataset. 120
A.4 Distribution of the shrinkage factor ξ for all simulation setups on the Chiaretti dataset. 123
A.5 Comparison of ICV, WMC and WMCS for the setup with correlated Gaussian data and weaker signal. 125
A.6 Distribution of the shrinkage factor ξ for the setup with correlated Gaussian data and weaker signal. 127
A.7 Comparison of ICV, WMC and WMCS for the setup with correlated non-informative Gaussian data. 128
A.8 Distribution of the shrinkage factor ξ for the setup with correlated non-informative Gaussian data. 130
A.9 Comparison of ICV, WMC and WMCS for the setup with uncorrelated Gaussian data. 131
A.10 Distribution of the shrinkage factor ξ for the setup with uncorrelated Gaussian data. 133
A.11 Comparison of ICV, WMC and WMCS for the setup with differences in covariances between response groups. 134
A.12 Distribution of the shrinkage factor ξ for for the setup with differences in covariances between response groups. 136
A.13 Normal quantile plots of $e(k\|S)$ for the case $n = 80$ in simulation setup 1.4.2. 137

B.1 Correlation of the rankings defined by CV and CSV in the simulation example. 140
B.2 Comparison of cross-study validation and cross-validation on simulated data after removal of outlier studies. 141
B.3 Distribution of Kendall's correlation between true algorithm rankings and estimated algorithm rankings after removal of outlier studies. 142

B.4 Correlation of the rankings defined by CV and CSV after removal of outlier studies. 143
B.5 Comparison of CV and independent within-study validation. 144

C.1 Ganttchart for bigmemory-experiment with 50 processes at the IBE-cluster. 173
C.2 Ganttchart for bigmemory-experiment with 50 processes at the IBE-cluster reading 500x5000 matrices from a memory attached file. 173

List of Tables

1.1 Mean, average absolute difference to ICV and standard deviation of the different correction methods. . . . 20
1.2 Approximation of $\varepsilon^{n_L}(\phi)$ based on 1000 independent training sets and a large validation set containing 20000 observations. . . . 21
1.3 Average corrected errors for informative pls and selection. . . . 23
1.4 Average corrected errors for non-informative pls and selection. . . . 25
1.5 Average runtimes WMC, WMCS and Internal Cross-Validation for the kNN-setups. . . . 27
1.6 Average runtimes WMC, WMCS and Internal Cross-Validation for the selection-setups. . . . 27
1.7 Loss matrix $\mathbf{W}$ for the classical testing problem. . . . 29
1.8 Loss matrix for the classification problem. . . . 31

2.1 Public microarray datasets of breast cancer patients as curated and summarized by Haibe-Kains et al. (2012). . . . 49
2.2 True global ranking and median rank estimate of CV and CSV on simulated data. . . . 59

3.1 Results for the basic benchmarks on the various instance included in the study. . . . 85
3.2 Estimated costs for the complete Project 1 on the EC2 instance *c1.medium*. 90
3.3 Estimated costs for the complete optimization bias project on the EC2 instance *m1.small*. . . . 90

A.1 Average corrected errors, mean absolute difference to ICV, and standard deviations for all setups on the Alon dataset. . . . 113
A.2 Average corrected errors, mean absolute difference to ICV, and standard deviations for all setups on the Singh dataset. . . . 116
A.3 Average corrected errors, mean absolute difference to ICV, and standard deviations for all setups on the Golub dataset. . . . 119
A.4 Average corrected errors, mean absolute difference to ICV, and standard deviations for all setups on the Chiaretti dataset. . . . 122

A.5 Mean, average absolute difference to ICV and standard deviation of the different correction methods in the setup with correlated Gaussian data and weaker signal. . . . 126
A.6 Estimation of $\varepsilon^{n_L}(\phi)$ based on 1000 independent training sets and a large validation set. . . . 126
A.7 Mean, average absolute difference to ICV and standard deviation of the different correction methods in the setup with correlated non-informative Gaussian data. . . . 129
A.8 Estimation of $\varepsilon^{n_L}(\phi)$ based on 1000 independent training sets and a large validation set. . . . 129
A.9 Mean, average absolute difference to ICV and standard deviation of the different correction methods in the setup with uncorrelated Gaussian data. 132
A.10 Approximated value of $\varepsilon^{n_L}(\phi)$ computed from 1000 independent training sets and a large validation set. . . . 132
A.11 Mean, average absolute difference to ICV and standard deviation of the different correction methods in the setup with differences in covariances between response groups. . . . 135
A.12 Estimation of $\varepsilon^{n_L}(\phi)$ based on 1000 independent training sets and a large validation set. . . . 135
A.13 Average runtimes WMC, WMCS and Internal Cross-Validation for the $PLSLDA$-setups. . . . 138

B.1 Rankings as estimated by CV and CSV on the experimental microarray data with outlier studies CAL and MSK included. . . . 145
B.2 Rankings as estimated by CV and CSV on the experimental microarray data after the removal of outlier studies. . . . 146

Abstract

Prediction problems on high-dimensional molecular data, e.g. the classification of microarray samples into normal and cancer tissues, are complex and ill-posed since the number of variables usually exceeds the number of observations by orders of magnitude. Recent research in the area has propagated a variety of new statistical models in order to handle these new biological datasets. In practice, however, these models are always applied in combination with preprocessing and variable selection methods as well as model selection which is mostly performed by cross-validation. Varma and Simon (2006) have used the term 'wrapper-algorithm' for this integration of preprocessing and model selection into the construction of statistical models. Additionally, they have proposed the method of nested cross-validation (NCV) as a way of estimating their prediction error which has evolved to the gold-standard by now.

In the first part, this thesis provides further theoretical and empirical justification for the usage of NCV in the context of wrapper-algorithms. Moreover, a computationally less intensive alternative to NCV is proposed which can be motivated in a decision theoretic framework. The new method can be interpreted as a smoothed variant of NCV and, in contrast to NCV, guarantees intuitive bounds for the estimation of the prediction error. The second part focuses on the ranking of wrapper algorithms. Cross-study-validation is proposed as an alternative concept to the repetition of separated within-study-validations if several similar prediction problems are available. The concept is demonstrated using six different wrapper algorithms for survival prediction on censored data on a selection of eight breast cancer datasets. Additionally, a parametric bootstrap approach for simulating realistic data from such related prediction problems is described and subsequently applied to illustrate the concept of cross-study-validation for the ranking of wrapper algorithms.

Eventually, the last part approaches computational aspects of the analyses and simulations performed in the thesis. The preprocessing before the analysis as well as the evaluation of the prediction models requires the usage of large computing resources. Parallel computing approaches are illustrated on cluster, cloud and high performance computing resources using the R programming language. Usage of heterogeneous hardware and processing of large datasets are covered as well as the implementation of the **R**-package *survHD* for the analysis and evaluation of high-dimensional wrapper algorithms for survival prediction from censored data.

Zusammenfassung

Prädiktionsprobleme für hochdimensionale genetische Daten, z.B. die Klassifikation von Proben in normales und Krebsgewebe, sind komplex und unterbestimmt, da die Anzahl der Variablen die Anzahl der Beobachtungen um ein Vielfaches übersteigt. Die Forschung hat auf diesem Gebiet in den letzten Jahren eine Vielzahl an neuen statistischen Methoden hervorgebracht. In der Praxis werden diese Algorithmen jedoch stets in Kombination mit Vorbearbeitung und Variablenselektion sowie Modellwahlverfahren angewandt, wobei letztere vorwiegend mit Hilfe von Kreuzvalidierung durchgeführt werden. Varma und Simon (2006) haben den Begriff 'Wrapper-Algorithmus' für eine derartige Einbettung von Vorbearbeitung und Modellwahl in die Konstruktion einer statistischen Methode verwendet. Zudem haben sie die genestete Kreuzvalidierung (NCV) als eine Methode zur Schätzung ihrer Fehlerrate eingeführt, welche sich mittlerweile zum Goldstandard entwickelt hat. Im ersten Teil dieser Doktorarbeit, wird eine tiefergreifende theoretische Grundlage sowie eine empirische Rechtfertigung für die Anwendung von NCV bei solchen 'Wrapper-Algorithmen' vorgestellt. Außerdem wird eine alternative, weniger computerintensive Methode vorgeschlagen, welche im Rahmen der Entscheidungstheorie motiviert wird. Diese neue Methode kann als eine geglättete Variante von NCV interpretiert werden und hält im Gegensatz zu NCV intuitive Grenzen bei der Fehlerratenschätzung ein. Der zweite Teil behandelt den Vergleich verschiedener 'Wrapper-Algorithmen' bzw. das Schätzen ihrer Reihenfolge gemäß eines bestimmten Gütekriteriums. Als eine Alternative zur wiederholten Durchführung von Kreuzvalidierung auf einzelnen Datensätzen wird das Konzept der studienübergreifenden Validierung vorgeschlagen. Das Konzept wird anhand von sechs verschiedenen 'Wrapper-Algorithmen' für die Vorhersage von Überlebenszeiten bei acht Brustkrebsstudien dargestellt. Zusätzlich wird ein Bootstrapverfahren beschrieben, mit dessen Hilfe man mehrere realistische Datensätze aus einer Menge von solchen verwandten Prädiktionsproblemen generieren kann. Der letzte Teil beleuchtet schließlich computationale Verfahren, die bei der Umsetzung der Analysen in dieser Dissertation eine tragende Rolle gespielt haben. Die Vorbearbeitungsschritte sowie die Evaluation der Prädiktionsmodelle erfordert die extensive Nutzung von Computerressourcen. Es werden Ansätze zum parallelen Rechnen auf Cluster-, Cloud- und Hochleistungsrechenressourcen unter der Verwendung der Programmiersprache **R** beschrieben. Die Benutzung von heterogenen Hardwarearchitekturen, die Verarbeitung von großen Datensätzen sowie die Entwicklung des **R**-Pakets *survHD* für die Analyse und Evaluierung von 'Wrapper-Algorithmen' zur Überlebenszeitenanalyse werden thematisiert.

Introduction

The upcoming of new biological high-dimensional data has introduced great challenges as far as their processing and analysis are concerned. Microarrays, which will be the main focus of this thesis, are just the first generation of modern and ever more detailed techniques for measuring transcriptomes or gene expressions. From a statistical point of view, microarray data analysis combines many problematic features. The original dataset consists of approximately 500000 measurements per patient which are usually combined into probe sets of sizes between 20 and 30. Thus, a common microarray dataset encompasses several tens of Gigabytes, i.e. it exceeds the main memory of most desktop PCs or workstations even nowadays. Likewise, the mere copying or data transfer via email, as common practice with many statistical analysis projects, is not possible in this case. Even normal intranets or network file systems, which are not dedicated to transfer of large amounts of data, cannot cope appropriately with raw microarray data. The third chapter of this thesis, will cover several aspects of software and hardware approaches in the context of microarray analysis.

During a preprocessing step, consisting of background correction, normalization and summarization, the number of variables per observation is strongly reduced to several thousands. This dimension reduction can distinctly alleviate many problems mentioned above and one can easily stick to normal computer resources for further analysis. As will be shown in Chapter 3, the high-level analysis, e.g. classification or survival analysis, can also be efficiently performed on cloud resources or meta clouds consisting of cloud and inhouse-resources (Bernau et al., 2013). The cited paper has been written in the context of a special issue for *Methods of Information in Medicine* in cooperation with Jochen Knaus (Institute of Medical Biometry and Medical Informatics, Albert-Ludwigs-University Freiburg) and Anne-Laure Boulesteix (Department for Medical Informatics, Biometry and Epidemiology [IBE]). A more detailed description of their contribution to the paper and the corresponding sections, which are presented in this thesis, can be found in Section D.3.

Even after the preprocessing step, however, microarray data are still high-dimensional from a statistical point of view. The large number of variables in combination with a small number of observations – many datasets include less than 100 observations – leads to a plethora of problems. If individual differentially expressed genes have to be identified via univariate tests, one is faced with a serious multiple testing problem. Various methods, controlling e.g. the false discovery rate (Benjamini and Hochberg, 1995), have been developed in this context in order to approach this problem adequately.

This thesis rather focuses on the development of gene profiles and risk scores using algorithms like penalized regression (Goeman, 2010) or boosting (Binder et al., 2009). Further, the attention is shifted rather to the design of good predictors for classes (e.g. cancerous or normal tissue) or survival times. In this context, the detection of the 'true' gene expressions causing or indicating a disease are less important as long as the prediction quality remains on an adequate level. This can also be achieved if gene expressions highly correlated to the actually relevant gene expressions are identified and weighted appropriately. Nonetheless, also in the context of this definition of the problem at hand, several serious problems have to be approached.

First and foremost, algorithms had to be established and optimized which can handle problems where the number of variables is distinctly outpacing the number of observations. The traditional linear model has no unique solution in this case which leads to approaches like penalized regression (Hastie et al., 2001). Other methods have been adopted from the machine learning community, e.g. the k-nearest-neighbors algorithm. Sometimes these methods were developed in a more intuitive fashion in the context of concrete problems like letter recognition. Hence, some of these methods lack a sound statistical foundation although they have proved useful and effective in practice.

A common aspect of all these new techniques is that they include tuning parameters (also denoted as hyper parameters) adjusting several features, e.g. complexity or flexibility, to the respective needs of the problem at hand. In most cases, no analytic solution for the adjustment of these tuning parameters exists. Thus, one has to resort to rules of thumb, trial and error or grid searches. The latter approach induces a potential for an optimistic 'tuning' or 'optimization bias' (see Boulesteix and Strobl, 2009) especially if no appropriately large dataset can be reserved for independent validation which is a common problem in many microarray studies (Varma and Simon, 2006). Often, the tuning parameters are chosen based on their performance in resampling approaches. If one uses this optimally chosen performance estimate as an estimate for the error rate of the underlying classifier, this estimate can be distinctly optimistically biased. In the first chapter of this thesis, an approach for correcting this bias will be introduced and analyzed. Although this problem has gained in importance in the context of the analysis of modern microarray data, it shares many parallels with traditional decision theory since one has to choose a parameter value in order to minimize a loss. In this context, another term, which has been introduced in Varma and Simon (2006), will play an important role: the wrapper algorithm. This term describes the combination of an algorithm with a tuning parameter and a procedure for adjusting this tuning parameter. It eventually epitomizes the main focus of this thesis. One can also define terms like *expected loss or risk* for such wrapper algorithms, which can be paralleled to strategies in decision theory. Furthermore, this concept is not restricted to the case where a tuning parameter has to be chosen, but can be extended to the case where one chooses among several different algorithms, i.e. to all types of model selection. Most parts of the work presented in this chapter are also included in the paper Bernau et al. (2013) which has been published in *Biometrics*. Section D.1 provides more details on the contribution of the coauthors to the paper and the corresponding sections in this thesis.

The second chapter is also related to wrapper algorithms. The work presented therein has evolved from a cooperation with Levi Waldron (Harvard Medical School and Dana-Faber Cancer Institute) and Lorenzo Trippa (Harvard Medical School and Dana-Faber Cancer Institute) and will be published in a paper which has been conditionally accepted by *Bioinformatics*. The coauthors' contributions to this paper and the corresponding sections presented in this thesis are described in more detail in Section D.2. Furthermore, the **R**-package *survHD* has been developed in cooperation with Levi Waldron and Markus Riester (Dana-Faber Cancer Institute) for this project (see Section C.1 for a more exhaustive description of their contribution). The chapter mainly covers a recent method for evaluating and ranking of algorithms (mostly wrapper algorithms), namely cross-study validation (Waldron et al., 2013). This validation scheme aims to integrate all problems which occur if a prediction rule is transferred from a training or pilot study to a validation study. In many cases, published promising results from microarray studies could not be validated or reproduced in follow-up studies. Apart from the tuning, such a mismatch between training study and validation study performance can also be caused e.g. by different biological conditions in the microarray laboratories. The reduction of such effects epitomizes the goal of a lot of tools for preprocessing of microarrays (e.g. Kostka and Spang, 2008; McCall et al., 2009). Nonetheless, for the time being, these problems cannot be avoided completely. Thus, this type of validation tries to account for such problems, by turning the focus primarily to the performance on independent validation studies. Since merging the studies is a delicate task, each of a compendium of microarray datasets is once used as training and tuning dataset, and subsequently validated on all other datasets. This procedure can help to avoid any bias that could be induced by incomplete cross-validation, since tuning and performance estimation are strictly separated in this design.

Nonetheless, it also increases the possible discrepancy between two important quantities of interest, which are by far more similar for traditional statistical approaches. For physicians, the performance of a concrete model is usually the primary interest. Cross-study validation, however, validates several models and resulting risk scores, which have been trained on different studies and can be substantially different from each other. Their commonality is that they have been trained by the same algorithm, which is usually a wrapper algorithm in the context of microarray studies. Thus, this validation scheme is probably more important for machine learners who want to gain insights into the performance of algorithms and are not primarily interested in concrete models. The conflict, which can best be described by the terms conditional (on the training data) and unconditional error rate or performance will also be an important aspect in the first and second chapter.

Chapter 1

Correcting the optimal resampling-based error rate by estimating the error rate of wrapper algorithms

The first chapter will start with the introduction of wrapper algorithms in the context of binary classification and the tuning bias that is induced choosing a best-performing classifier from a number of potential candidate classifiers. The term wrapper algorithm will be defined in this chapter and it will be shown that the well-known nested cross-validation (NCV) estimator can provide a good estimate for its unconditional error rate. Furthermore, a computationally less intensive alternative (WMC) is presented, which has the same estimation target and stays within certain reasonable bounds. Finally, the performance of both estimators is presented on experimental microarray data as well as simulated data and a decision theoretic interpretation of the WMC estimator and some of the assumptions it is based on is provided.

1.1 Introduction

Resampling-based procedures are routinely applied in order to assess the performance of statistical learning methods by estimating their prediction error. If the available dataset were large enough, it would be recommended to partition the data into learning and validation data, to fit a model using the learning data, and to estimate its error based on the validation data. In the common case of small sample high-dimensional data considered here, however, the available dataset is usually too small for such a partitioning. Resampling-based procedures are thus particularly useful in the context of "$n \ll p$" data analysis, i.e. when the number of predictors exceeds the number of observations.

In practice, most common classification methods for high-dimensional data involve a tuning parameter, e.g. the cost parameter in linear Support Vector Machines (SVM) or the

number of neighbors in k-nearest-neighbors (kNN). If the error of a classification method is estimated by a resampling method several times with different values of the tuning parameter successively, each parameter value possibly yields a different estimated error. The approach consisting in selecting the parameter value yielding the smallest resampling error estimate and only reporting this resampling estimate is biased (Dupuy and Simon, 2007). That is because the minimal resampling error can be seen as the result of an optimal selection. As such, it is a biased estimate of the generalization error rate, i.e. of the error that would be obtained with this parameter value on independent data. This bias, that is quantitatively assessed by Varma and Simon (2006) in the "$n \ll p$" setting, is denoted here as *tuning bias*. Note that the term "tuning" may be ambiguous since researchers from different fields might have different understandings of tuning. In this thesis, a parameter is considered a tuning parameter if it is not optimized by an analytical method (like the least squares criterion for the coefficients in linear regression), but rather by trying several values successively and using the value yielding the best prediction performance on test data. When choosing the parameter value based on the performance yielded by different candidate values, one indirectly uses the test data for learning the decision function, leading to an optimistic bias.

A similar bias, called method selection bias in the sequel, occurs if a researcher tries out several classification methods successively and reports only the results of the method yielding the minimal error rate. For instance, suppose the resampling error rate of Support Vector Machines (SVM), Random Forests (RF), k-nearest-neighbors (kNN), and L_1-penalized regression are computed for a particular dataset. Suppose further that kNN yields the smallest error rate in the resampling approach. This error rate is likely to be smaller than the error rate of kNN on independent data, because it was *optimally* selected across four error estimates that all show some variability. The resulting bias may be considerable, as illustrated by Boulesteix and Strobl (2009); Jelizarow et al. (2010).

To correct for the tuning bias outlined above in the context of microarray-based classification, Varma and Simon (2006) recommend to embed an *internal* cross-validation (ICV) into the *external* resampling-based error estimation procedure. If the external resampling procedure is also CV as considered by Varma and Simon (2006), it is usual to denote the whole procedure as "nested CV". In this thesis, however, the more general terminology "ICV" will be used here. No matter which resampling scheme is used externally for error estimation, the principle of ICV for parameter tuning is as follows. In each external resampling iteration, an internal CV is performed based on the current learning set for different tuning parameter values. The parameter value yielding the smallest error is selected and then evaluated by predicting the test observations. In this way, for each external iteration the choice of the parameter value is performed without using information from the test set. Note that, as Varma and Simon (2006) have already stressed in their paper, the ICV procedure estimates the error rate of a so-called wrapper algorithm including the tuning process and not the error rate of a specific tuning parameter value. A similar procedure might also be used to address the method selection bias induced by the optimal choice of the classification method. However, the ICV technique is computationally expensive, since it requires an additional CV loop on each learning set. The computational burden

might rapidly become intractable. Furthermore, ICV tends to yield highly variable results, sometimes leading to obviously inappropriate "corrected" errors outside the range of the original errors of the considered methods.

An alternative bias correction approach will be suggested here, which can be applied to address both the tuning and the method selection bias. The procedure, which can be interpreted as a smooth, analytical variant of ICV, guarantees intuitive bounds, increases stability compared to ICV, and reduces the computation time drastically since it does not rely on an internal CV loop.

Apart from ICV, the literature on the bias of the optimally selected error rate is scarce. Tibshirani and Tibshirani (2009) introduce an approach addressing several of the mentioned inconveniences, first and foremost the computational burden. In contrast to ICV, however, it does not target the unconditional error rate of wrapper algorithms, but the conditional error rate of the optimal method/tuning parameter. Therefore, this approach will not be treated as a direct competitor in the following study because the differing estimation targets impede a fair comparison.

1.2 Internal CV and the unconditional error rate of wrapper algorithms

1.2.1 Settings and notations

From a statistical point of view, binary supervised classification can be described in the following way. On the one hand there is a response variable taking values in $\mathcal{Y} = \{0, 1\}$. On the other hand there are predictors taking values in $\mathcal{X} \subset \mathbf{R}^p$ that will be used for constructing a classification rule. Predictors and response follow an unknown joint distribution on $\mathcal{X} \times \mathcal{Y}$ denoted by $P(\mathbf{x}, y)$. The observed i.i.d. sample of size n is denoted by $s_0 = \{(\mathbf{x}_1, y_1)...(\mathbf{x}_n, y_n)\}$. The classification task consists in building a decision function $\hat{f}^S : \mathcal{X} \mapsto \mathcal{Y}$, $\mathbf{x} \mapsto \hat{f}^S(\mathbf{x})$, which maps elements $\mathbf{x}$ of the predictor space $\mathcal{X}$ into the response space $\mathcal{Y}$. The superscript S indicates that the decision function is built using the sample S.

From now on, the considered combination of method and tuning parameter values will be denoted by method k (with $k \in 1, \ldots, K$) . As an example, method $k = 1$ may stand for SVM with cost= 1, method $k = 2$ for kNN with 10 nearest neighbors, and so on. As a special case, $1, \ldots, K$ might represent different parameter values of the same method. The decision function obtained by fitting the prediction method k to the sample s_0 is denoted as $\hat{f}_k^{s_0}$. Using this notation, each method k can be defined as a function $k : \mathcal{S} \mapsto \mathcal{F}_\mathcal{X}, \quad S \mapsto k(S) = \hat{f}_k^S$, which maps any possible sample S to the prediction function $\hat{f}_k^S$. Here, $\mathcal{S}$ denotes the space of possible samples S and $\mathcal{F}_\mathcal{X}$ denotes the space of decision functions on $\mathcal{X}$.

1.2.2 Conditional and unconditional error rate

For a decision function $\hat{f}_k^{s_0}$, the true prediction error $\varepsilon[\hat{f}_k^{s_0}]$ depends on the expectation $\mathbf{E}_P$ over the joint distribution P and an adequately chosen loss function $L(\cdot,\cdot)$, e.g. the indicator loss considered here. Abbreviating the true error $\varepsilon[\hat{f}_k^{s_0}]$ of method k constructed from sample s_0 by the simplified notation $\varepsilon(k \parallel s_0)$, one obtains

$$\varepsilon(k \parallel s_0) = \mathbf{E}_P\left[L\left(\hat{f}_k^{s_0}(\mathbf{x}), y\right)\right] = \int_{\mathcal{X}\times\mathcal{Y}} L\left(\hat{f}_k^{s_0}(\mathbf{x}), y\right)\, dP(\mathbf{x}, y), \tag{1.1}$$

$\varepsilon(k \parallel s_0)$ is commonly referred to as *conditional* error since it refers to the decision function constructed from the specific sample s_0. The corresponding conditional error rate $\varepsilon(k \parallel S)$ can be seen as a random variable, where S stands for a random sample that follows the distribution P^n.

The expectation $\varepsilon^n(k) = \mathbf{E}_{P^n}\left[\varepsilon(k \parallel S)\right]$ of this random variable $\varepsilon(k \parallel S)$ is usually referred to as the *unconditional* true error rate of method k. It depends only on the method k, on the size n of the sample S and on the joint distribution P, and can be seen as a fixed quantity for every method k. Since the joint distribution $P(\mathbf{x}, y)$ is unknown, the conditional errors $\varepsilon(1 \parallel s_0), \ldots, \varepsilon(K \parallel s_0)$ and the unconditional errors $\varepsilon^n(1), \ldots, \varepsilon^n(K)$ have to be estimated. Standard estimation approaches are based on CV or repeated subsampling. The repeated subsampling method is applied here, because the new correction method involves the estimation of the unconditional variance of the estimated error. The estimator proposed by Nadeau and Bengio (2003) – which is used here – is expected to work for repeated subsampling only, and no convincing alternative estimator applicable to CV could be found in current literature after extensive research.

In repeated subsampling the whole dataset is randomly split into learning and test sets several times. Each learning set L_b, $b = 1, \ldots, B$ of size n_L (with $n_L < n$) is used to estimate a decision function that is subsequently evaluated on the corresponding test set $S \setminus L_b$. For each iteration $b = 1, \ldots, B$ and each method k, $k = \{1, \ldots, K\}$, one obtains an estimated error $e(k \parallel L_b, S \setminus L_b)$, where the notation "$L_b, S \setminus L_b$" means that method k is fitted to the learning set L_b and evaluated on the test set $S \setminus L_b$. Note that the notation e is used for estimators and ε for true errors.

In contrast to the conditional true error $\varepsilon(k \parallel S)$, the estimated error $e(k \parallel L_b, S \setminus L_b)$ is conditional on the considered sample not only with regard to the estimation of the decision function, but in addition with regard to the estimation of the error. For each method k, the iteration-wise test errors are eventually combined into an error rate estimate by averaging over the iterations $b = 1, \ldots, B$, yielding

$$e(k \parallel S) = \frac{1}{B}\sum_{b=1}^{B} e(k \parallel L_b, S \setminus L_b), \tag{1.2}$$

which obviously may depend on the random choices of the partitions $\{L_b, T_b\}$, $b = 1, \ldots, B$, a fact that is however omitted in the notation.

1.2.3 The unconditional error rate of wrapper algorithms

Further, the method yielding the smallest estimated error rate based on S is denoted by $k^*(S)$, i.e. $k^*(S) = \arg\min_k e(k \parallel S)$. For a sample s_0, the error estimate $e(k^*(s_0) \parallel s_0)$ obtained by repeated resampling incorporates a source of a downward bias because $k^*(s_0)$ is chosen such that $e(k^*(s_0) \parallel s_0)$ is minimal. If one simply chooses the method yielding the minimal error rate $e(k^*(s_0) \parallel s_0)$, this minimal error rate underestimates the true conditional error rate $\varepsilon(k^*(s_0) \parallel s_0)$ of the chosen method. This problem is due to the fact that the same sample s_0 is used both for error estimation and for the choice of the optimal classification method $k^*(s_0)$. The corresponding prediction rule $\hat{f}^{s_0}_{k^*(s_0)}$ is expected to perform worse on an independent sample which was not used for choosing the method. This bias is related to the problem of multiple comparisons. The minimal error rate out of K methods decreases with increasing K.

In the context of ICV, the current gold standard for avoiding this bias, Varma and Simon (2006) reformulate the estimation task by defining wrapper algortihms.

A wrapper algorithm consists of two steps. The first step is a tuning process k^* on S which chooses a parameter or method. It can be described as a function:

$$k^* : \mathcal{S} \mapsto \mathcal{K}, \quad S \mapsto k^*(S) = \arg\min_k e(k \parallel S),$$

where $\mathcal{K}$ is the space of tuning parameters or candidate methods (here: $\mathcal{K} = \{1, \ldots, K\}$). Note that $k^*(S)$ is actually a random variable even for a fixed sample S because it depends on the randomly drawn learning sets L_b, $b = 1, \ldots, B$. For simplicity, however, this dependence on the specific learning sets is ignored in the notation.

The second step consists in learning a prediction rule by applying the chosen tuning parameter or candidate method $k^*(S)$ on S. For a sample S, this learning process can be described using the function $\psi : \mathcal{K} \times \mathcal{S} \mapsto \mathcal{F}_\mathcal{X}$, $k \mapsto \psi^S(k, S) = \hat{f}^S_k$. Using these functions, one can define the wrapper algorithm ϕ:

$$\phi : \mathcal{S} \mapsto \mathcal{F}_\mathcal{X}, \quad S \mapsto \phi(S) = \psi^S(k^*(S), S) = \hat{f}^S_{k^*(S)}.$$

Please note that this definition of ϕ is parallel to the definition of the methods k. However, ϕ incorporates an additonal source of randomness – the tuning process – whereas the methods k are deterministic for a fixed sample S.

Now, the main idea for bias correction in ICV and the new method consists in estimating the unconditional error rate of such wrapper algorithms ϕ:

$$Err = \mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi). \tag{1.3}$$

The error $\varepsilon(k^*(s_0) \parallel s_0)$ of the s_0-best method fitted on s_0 is a realization of the random variable $\varepsilon(k^*(S) \parallel S)$ whose mean over P^{n_L} is Err. It will be shown in the next section that the well-known ICV estimator actually targets at Err, which is also the reason why the simulated counterpart of Err is used for evaluation in the simulation studies later on.

Before turning the focus to ICV, another problem has to be addressed. Note that $e(k \parallel S)$ is a good estimator for $\varepsilon^{n_L}(k)$ but an upwardly biased estimator of $\varepsilon(k \parallel S)$

and $\varepsilon^n(k)$, because the decision functions are estimated based on n_L observations instead of n, with $n_L < n$. This bias affects any resampling based approach for estimating the generalization error. Since the mixture of this bias and the tuning bias would seriously distort the evaluation of the correction methods, Err (using n_L) from Eq. (1.3) as the estimation target will be defined as estimation target here.

1.2.4 Revisiting internal cross-validation

ICV is often performed within a nested CV procedure, i.e. with test sets $T_b := S \setminus L_b$ forming a partition of the sample S. In the following, however, ICV is formulated in a general way without specifying how the learning and test sets (L_b, T_b) are chosen. Here, they are chosen by repeated subsampling. In the current notation, the ICV error estimate can be written as

$$\widehat{Err}_{ICV} = \frac{1}{B} \sum_{b=1}^{B} e(k^*(L_b) \parallel L_b, S \setminus L_b). \tag{1.4}$$

This formula, which is substantially different from formula (1.2) due to referring to $k^*(L_b)$ instead of a fixed method k, can be interpreted as follows. For each iteration b ($b = 1, \ldots, B$), the following procedure is repeated. Firstly, the "L_b-best method" $k^*(L_b)$ is determined by ICV within L_b. Secondly, the classification rule fitted on L_b using the best method $k^*(L_b)$ is evaluated on $S \setminus L_b$, yielding $e(k^*(L_b) \parallel L_b, S \setminus L_b)$. When introducing ICV Varma and Simon (2006) already explain that their method actually estimates the error rate of wrapper algorithms. In addition to this explanation, an asymptotic consideration is presented, which further clarifies that the ICV-estimator is interpreted best as a natural estimator of Err (Eq. 1.3).

The difference between Eq. (1.4) and Eq. (1.2) is that ICV builds the average error of the best methods $k^*(L_b)$ (as assessed in ICV), instead of averaging the error rates of a specific method k. Note that these L_b-best methods again vary with the choice of the internal learning sets, which means that one may not obtain the same final results when repeating the same procedure twice – even if the outer learning sets L_b are fixed. Roughly speaking, in ICV the quantity Err (Eq. (1.3)) is estimated through averaging over B subsets of s_0. Each term $e(k^*(L_b) \parallel L_b, S \setminus L_b)$ can be seen as an estimator of $\varepsilon(k^*(L_b) \parallel L_b)$, which roughly plays the role of a realization of $\varepsilon(k^*(S) \parallel S)$. The determination of $k^*(L_b)$ within each iteration is computationally expensive, which makes ICV very difficult to apply in practice when the prediction methods are time consuming, especially when they involve a tuning step that itself has to be performed through ICV. As a consequence, researchers often run ICV with a small number of folds (e.g. 3-fold-CV), yielding even more variable results. In extreme cases, this high variability may lead to estimates $\widehat{Err}_{ICV}$ larger than $\max_k e(k \parallel s_0)$ or lower than $\min_k e(k \parallel s_0)$, which is very unintuitive. Motivated by these disadvantages, an alternative computationally effective estimator is presented in the following section.

1.3 A smooth analytical alternative to ICV

1.3.1 Basic Idea

The rationale behind ICV is that the construction of the decision function *and* the tuning/ method selection process, which are typically applied to the whole sample $S = s_0$, are mimicked on each external learning set L_b. In this way the tuning/method selection process is empirically incorporated into the estimation procedure. In practice, the best method $k^*(L_b)$ is typically not the same for all iterations $b = 1, \ldots, B$. ICV builds a so-to-say hard-weighted average of error estimates obtained with different methods or tuning parameters. The term hard-weighted is used here to emphasize that for each resampling iteration b only one of the $e(k \parallel L_b, S \setminus L_b)$ $(k = 1, \ldots, K)$ is chosen by ICV to be included in the average. The weight of $e(k^*(L_b) \parallel L_b, S \setminus L_b)$ is 1, while the weight of all other $e(k \parallel L_b, S \setminus L_b)$ (for $k \neq k^*(L_b)$) is 0. This way, results from different tuning parameters are eventually combined, which basically imitates the wrapper algorithm ϕ whose unconditional error rate is estimated.

The new method is also based on a combination of error estimates of different parameter values/methods, albeit in a completely different and more direct way. While ICV combines errors of different parameter values/methods $e(k \parallel L(b), S \setminus L_b)$ computed for different test sets, the new procedure combines the global error estimates $e(k \parallel s_0)$ of the different parameter values/methods k. Furthermore, the way these average errors are combined does not depend on an empirical experiment as performed in ICV. The main idea is to decompose the unconditional error rate $\mathbf{E}_{P^{n_L}}\left[\varepsilon(k^*(S) \parallel S)\right]$ with regard to the random variable $k^*(S)$, i.e. the index of the best method, as follows:

$$\mathbf{E}_{P^{n_L}}\left[\varepsilon(k^*(S) \parallel S\right] = \sum_{k=1}^{K} \mathrm{P}\left(k^*(S) = k\right) \cdot \mathbf{E}_{P^{n_L}}\left[\varepsilon(k \parallel S) | k^*(S) = k\right]. \tag{1.5}$$

As argued below, in most cases, it is reasonable to assume that, for a fixed method k,

$$\varepsilon(k \parallel S) \perp k^*(S), \tag{1.6}$$

i.e. the conditional error rate of method k constructed on S is independent from $k^*(S)$. It follows from Eq. (1.6) that the conditional expectations in Eq. (1.5), which cannot be estimated easily, can be replaced by the respective unconditional expectations,

$$\mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S) \approx \sum_{k=1}^{K} \mathrm{P}\left(k^*(S) = k\right) \cdot \mathbf{E}_{P^{n_L}}\left[\varepsilon(k \parallel S)\right], \tag{1.7}$$

and thus, to estimate $\mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)$, the quantities in Eq. (1.7) have to be estimated, as discussed in Sections 1.3.2 and 1.3.3.

Before that, assumption (1.6) has to be revisited. It means that the true error rate $\varepsilon(k \parallel s_0)$ of method k fitted on s_0 does not depend on which method performed best in repeated subsampling based on s_0 – the unconditional error rates $\varepsilon^{n_L}(1), \ldots, \varepsilon^{n_L}(K)$

being fixed. Note that this assumption, of course, should not be misinterpreted in the sense that parameter tuning with CV is useless. Even if assumption (1.6) holds, tuning is useful to identify which method may have the smallest unconditional error rate $\varepsilon^{n_L}(k)$. A counter-example for which assumption (1.6) does not completely hold is support vector machines – denoted as $k = 1$ here – in the case of a sample with a mislabeled observation. The error rate $\varepsilon(1 \parallel s_0)$ of SVM is likely to be large, because SVM classifiers are strongly affected by mislabeled observations that often take the role of support vectors. Hence, $k^*(s_0) = 1$ is not likely. Thus, in this case, one obviously does not have $\varepsilon(k \parallel S) \perp k^*(S)$. However, especially in the presence of variable selection, assumption (1.6) holds in most cases, as illustrated by Hanczar et al. (2007) based on an extensive empirical study. Note that, if assumption (1.6) does not hold, it is more likely that $\mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S)|k^*(S) = k] < \mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S)]$, i.e. that a classifier performs better if it is chosen by the model selection procedure. Consequently, this assumption could be a potential source for a pessimistic bias in the approach. This aspect will be revisited after analyzing the correction method on simulated data.

1.3.2 A weighted mean approach

The terms $\mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S)]$ (for $k = 1, \ldots, K$) in Eq. (1.7) can be naively estimated by $e(k \parallel s_0)$ (for $k = 1, \ldots, K$), suggesting to estimate the quantity of interest $\mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)$ by a weighted mean of the average errors $e(k \parallel s_0)$. Thus, the following estimator is proposed, denoted from now on as "Weighted Mean Correction" (WMC):

$$\widehat{Err}_{WMC} = \sum_{k=1}^{K} \hat{\mathrm{P}}\left(k^*(S) = k\right) \cdot e(k \parallel s_0), \tag{1.8}$$

where $\hat{\mathrm{P}}(k^*(S) = k)$ stands for an adequate estimator of $\mathrm{P}(k^*(S) = k)$. Two such estimation procedures are presented in Section 1.3.3, where a variant of $\widehat{Err}_{WMC}$ is introduced based on improved estimates of $\mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S)]$, which is called WMCS.

An illustrative way to explain the WMC estimator is to parallel it to the raw mean and to $\widehat{Err}_{ICV}$. The raw mean is obtained by giving equal weights to all parameters/methods, i.e. by replacing $\hat{\mathrm{P}}(k^*(S) = k)$ by $1/K$ in Eq. (1.8). This equal weight approach can be considered as a sensible upper bound for the corrected error because it corresponds to a random choice of the parameter/method. By definition, a random choice cannot lead to a tuning or method selection bias. That is why any corrected error is not expected to be higher than

$$\widehat{Err}_{RawMean} = \sum_{k=1}^{K} \frac{1}{K} \cdot e(k \parallel s_0). \tag{1.9}$$

Regardless of the choice of the weights, there is a connection between the new approach and ICV. The WMC estimator can be paralleled to the ICV estimator through a reformulation as

$$\widehat{Err}_{WMC} = \frac{1}{B}\sum_{k=1}^{K}\sum_{b=1}^{B}\hat{\mathrm{P}}\left(k^*(S)=k\right)\cdot e(k \parallel L_b, S\setminus L_b). \tag{1.10}$$

Similarly, the estimator $\widehat{Err}_{ICV}$ can also be reformulated as

$$\widehat{Err}_{ICV} = \frac{1}{B}\sum_{k=1}^{K}\sum_{b=1}^{B} I(k^*(L_b)=k)\cdot e(k \parallel L_b, S\setminus L_b). \tag{1.11}$$

Bringing these two estimators to a similar form highlights their crucial difference. $\widehat{Err}_{WMC}$ smoothly weights the errors $e(k \parallel L_b, S\setminus L_b)$ with the probabilities $\hat{\mathrm{P}}(k^*(S)=k)$ estimated from an analytical parametric model (whose parameters are estimated from the quantities $e(k \parallel L_b, S\setminus L_b)$ only). On the contrary, in $\widehat{Err}_{ICV}$ the weights are empirical, discrete and depend on the results of a computationally intensive internal CV. Figure 1 illustrates the similarities and differences of WMC and ICV graphically.

1.3.3 Algorithmic description of WMC

The new WMC estimator is obtained through the following steps:

1 *Inputs of the WMC algorithm:* estimated fold errors $\{e(k \parallel L_b, s_0\setminus L_b)\}_{b=1,\ldots,B}^{k=1,\ldots,K}$

2 *Estimating the weights* $\mathrm{P}(k^*(S)=k)$ based on the assumption of a multivariate normal distribution $MVN(\mu,\boldsymbol{\Sigma})$ of the vector
$(e(1\|S),\ldots,e(K \parallel S))^\top$:

a *Estimating the parameters μ and $\boldsymbol{\Sigma}$* of the multivariate normal distribution

- *The vector of means* is estimated by $\hat{\mu}=(e(1\|s_0),\ldots,e(K \parallel s_0))^\top$.
- *The covariance matrix* is estimated by the nearest positive definite $\hat{\boldsymbol{\Sigma}}$ of $\widehat{\mathbf{V}}^{\frac{1}{2}}\tilde{\mathbf{C}}\widehat{\mathbf{V}}^{\frac{1}{2}}$ as computed by the algorithm in Higham (1988), where $\widehat{\mathbf{V}}$ is the diagonal matrix with diagonal elements $\hat{v}_1,\ldots,\hat{v}_K$ computed using the procedure by Nadeau and Bengio (2003) as $\hat{v}_k=\widehat{var}(e(k \parallel S))=\frac{1}{B-1}\sum_{b=1}^{B}\left(e(k \parallel L_b, s_0\setminus L_b)-e(k \parallel s_0)\right)^2$ and the element $\hat{\rho}_{k_1,k_2}$ (for $k_1\neq k_2$) of the correlation matrix $\tilde{\mathbf{C}}=(\hat{\rho}_{k_1,k_2})_{k_1=1,\ldots,K}^{k_2=1,\ldots,K}$ is obtained as the empirical correlation between the corresponding fold errors: $\hat{\rho}_{k_1,k_2}=\widehat{cor}(e(k_1 \parallel L_b, S\setminus L_b), e(k_2 \parallel L_b, S\setminus L_b))$.

b *Estimating the weights* $\mathrm{P}(k^*(S)=k)$ (for $k=1,\ldots,K$) based on the assumption of a multivariate normal distribution of $(e(1 \parallel S),\ldots,e(K \parallel S))^\top$ by plugging $\hat{\mu}$ and $\hat{\boldsymbol{\Sigma}}$ into Eq. (1.13) given in Section 1.3.6.

3 *Computing the weighted average* of the original average resampling errors $e(k \parallel s_0)$, yielding the WMC estimator $\widehat{Err}_{WMC}=\sum_{k=1}^{K}\hat{\mathrm{P}}\left(k^*(S)=k\right)\cdot e(k \parallel s_0)$.

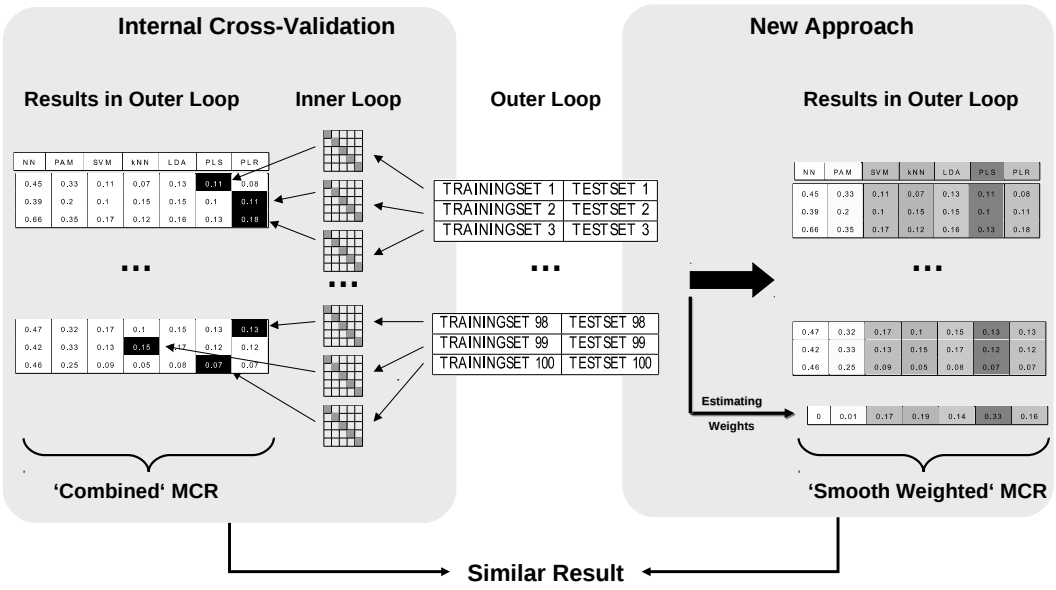


Figure 1.1: Schematic illustration contrasting the weighted mean correction (WMC) and internal cross-validation (ICV). The misclassification rate (MCR) obtained by WMC (compare to Eq. (1.10)) can be interpreted as a smoothed variant of the one obtained by internal cross-validation (compare to Eq. (1.11)).

1.3.4 Shrinkage approach (WMCS)

As supported by the results in Section 1.4, the WMC procedure yields an optimistic estimate. $\widehat{Err}_{WMC}$ uses $e(k \parallel s_0)$ to estimate both $\mathbf{E}_{P^{n_L}}[\varepsilon(k||S)]$ in Eq. (1.5) and the mean of $e(k||S)$ for the estimation of the weights $\hat{\mathrm{P}}(k^*(S)=k)$. Thus, the optimistic bias, which shall actually be corrected, is – to some extent – incorporated into the correction procedure. This may lead to under-correction in some cases, as supported by some of the results in the non-informative settings (see Section 4.2). Thus, a shrinkage procedure is suggested to alleviate this problem.

The idea is to shrink $e(k||s_0)$ (for $k=1,\ldots,K$) towards the average of the K methods by considering a shrunken estimator $e^{\xi}(k \parallel s_0) = (1-\xi)\cdot e(k \parallel s_0) + \xi \cdot \frac{\sum_{h=1}^{K} e(h \parallel s_0)}{K}$ instead of $e(k \parallel s_0)$, where ξ is a shrinkage parameter that has to be adequately chosen. The term $e^{\xi}(k^*(s_0) \parallel s_0) - e(k^*(s_0) \parallel s_0)$ can be viewed as the bias in the sense considered by Tibshirani and Tibshirani (2009). At this stage, the parametric model is used which has been described in Section 3.3 (step (2) of the original WMC algorithm) for the distribution of $\boldsymbol{e} = (e(1 \parallel S), \ldots, e(K \parallel S))^{\top}$. Under this model, the bias ζ for method $k^*(s_0)$ can be estimated as $e(k^*(s_0) \parallel s_0)$ (the $k^*(s_0)$-th component of the mean vector) minus the conditional mean of the $k^*(s_0)$-th component of random $\boldsymbol{e}$ vector given that the $k^*(s_0)$-th component is the smallest component, i.e. conditional on $k^*(s_0) = k^*(S)$. The latter conditional mean can be derived by Monte-Carlo simulation. A sensible shrinkage factor ξ can then be obtained by equating the bias $e^{\xi}(k^*(s_0) \parallel s_0) - e(k^*(s_0) \parallel s_0)$ with the value ζ of the bias derived from the parametric model. Provided that $\zeta > 0$ (otherwise ξ is set to 0), the shrinkage factor is obtained as

$$\xi = \begin{cases} \zeta \left[\widehat{Err}_{RawMean} - e(k^*(s_0)|s_0)\right]^{-1} & if \quad \zeta < \widehat{Err}_{RawMean} - e(k^*(s_0)|s_0) \\ 1 & if \quad \zeta \geq \widehat{Err}_{RawMean} - e(k^*(s_0)|s_0). \end{cases}$$

If the expected bias ζ is larger than the difference between the minimal error rate and the raw mean, the shrinkage factor ξ is thus set to 1, leading to $e^{\xi}(k \parallel s_0) = \widehat{Err}_{RawMean}$. In contrast, if $e(k^*(s_0) \parallel s_0)$ is considerably different from the raw mean, ξ is small and $e^{\xi}(k \parallel s_0)$ is similar to $e(k \parallel s_0)$. Roughly speaking, the shrinkage factor ξ measures the plausibility that the classifier $k^*(s_0)$ does not perform better than the average classifier, while taking into account the variability of the resampling approach and the correlations between the candidate classifiers. Additional details on the empirical properties of this shrinkage factor are available in Section 1.4.

Finally, the second variant $\widehat{Err}_{WMCS}$ of the estimator is computed by replacing $\boldsymbol{e}$ by $\boldsymbol{e}^{\xi} = (e^{\xi}(1 \parallel s_0), \ldots, e^{\xi}(K \parallel s_0))^{\top}$ in both $\hat{\boldsymbol{\mu}}$ and Eq. (1.5):

$$\widehat{Err}_{WMCS} = \sum_{k=1}^{K} \hat{\mathrm{P}}^{\xi}(k^*(S)=k) \cdot e^{\xi}(k \parallel s_0),$$

where $\hat{\mathrm{P}}^{\xi}(k^*(S)=k)$ denotes the estimated probability obtained by replacing $\hat{\boldsymbol{\mu}}$ through $\hat{\boldsymbol{\mu}}^{\xi} = \boldsymbol{e}^{\xi}$ in the parametric model.

1.3.5 Algorithmic description of WMCS

Parallel to WMC, the WMCS algorihm can be formulated in the following way:

1 *Inputs of the algorithm:* estimated fold errors $\{e(k \parallel L_b, s_0 \setminus L_b)\}_{b=1,\ldots,B}^{k=1,\ldots,K}$

2 *Estimating the shrinkage factor* ξ based on a preliminary multivariate normal distribution $MVN_{pre}(\mu, \mathbf{\Sigma})$ for $(e(1\|S), \ldots, e(K \parallel S))^\top$:

a *Estimating the parameters* μ *and* $\mathbf{\Sigma}$ *of* MVN_{pre} in the same way as in step (2a) of the original WMC algorithm.

b *Computing the expected bias* $\hat{\zeta}$ as $\hat{\zeta} = e(k^*(s_0) \parallel s_0) - \mathbf{E}_{MVN_{pre}(\hat{\mu}, \hat{\mathbf{\Sigma}})}(e(k^*(s_0) \parallel S) | k^*(S) = k^*(s_0))$, whereby the latter term is determined by Monte-Carlo simulation.

c *Determining the shrinkage factor* ξ as

$$\hat{\xi} = \begin{cases} \hat{\zeta} \left[\widehat{Err}_{RawMean} - e(k^*(s_0)|s_0) \right]^{-1} & \text{if } \hat{\zeta} < \widehat{Err}_{RawMean} - e(k^*(s_0)|s_0) \\ 1 & \text{otherwise.} \end{cases}$$

3 *Shrinking the average resampling errors* $e(k \parallel s_0)$ *with shrinkage factor* ξ*:*

$$e^{\xi}(k \parallel s_0) = (1 - \xi) \cdot e(k \parallel s_0) + \hat{\xi} \cdot \widehat{Err}_{RawMean} \qquad \text{for } k \text{ in } 1, \ldots, K.$$

4 *Estimating the weights* by $\hat{\mathrm{P}}^{\xi}(k^*(S) = k)$ similarly to the WMC procedure (step (2b)), except that μ is now estimated by $(e^{\xi}(1 \parallel S), \ldots, e^{\xi}(K \parallel S))^\top$ instead of $(e(1 \parallel S), \ldots, e(K \parallel S))^\top$.

5 *Computing the weighted average* yields the WMCS estimator

$$\widehat{Err}_{WMCS} = \sum_{k=1}^{K} \hat{\mathrm{P}}^{\xi}(k^*(S) = k) \cdot e^{\xi}(k \parallel s_0). \tag{1.12}$$

1.3.6 More details on estimating $\mathbf{P}(k^*(S) = k)$

Under the assumption that the vector $(e(1 \parallel S), \ldots, e(K \parallel S))^\top$ follows a multivariate normal distribution with mean vector $\hat{\mu}$ and covariance matrix $\hat{\mathbf{\Sigma}}$, the probability $\hat{\mathrm{P}}(k^*(S) = k)$ is obtained as $\widehat{\mathrm{P}}(k^*(S) = k) = \mathrm{P}_{MVN(\hat{\mu}, \hat{\mathbf{\Sigma}})}(e(k \parallel S) \leq e(k' \parallel S), \forall k' : k' \neq k)$. This quantity can be derived analytically by considering the $K - 1$ differences $\delta_{k'} = e(k \parallel S) - e(k' \parallel S)$ for $k' \neq k$, which are simple linear combinations of the original random vector $(e(1 \parallel S), \ldots, e(K \parallel S))^\top$. The probability $\mathrm{P}(k^*(S) = k) = \mathrm{P}(e(k \parallel S) - e(k' \parallel S) \leq 0, \forall k' : k' \neq k)$ is then obtained from the density of the multivariate normal distribution of the random vector δ as

$$\int_{-\infty}^{0} \cdots \int_{-\infty}^{0} \frac{1}{(2\pi)^{\frac{K-1}{2}} \sqrt{\mathbf{T}\hat{\mathbf{\Sigma}}\mathbf{T}^\top}} \exp\left((\delta - \mathbf{T}\hat{\mu})^\top \left(\mathbf{T}\hat{\mathbf{\Sigma}}\mathbf{T}^\top \right)^{-1} (\delta - \mathbf{T}\hat{\mu}) \right) \Pi_{k'} d\delta_{k'}, \tag{1.13}$$

where the $(K-1)\times K$ matrix $\mathbf{T}$ contains the linear combinations yielding the corresponding differences, i.e. such that $\delta = \mathbf{Te}$. These integrals can be approximated very precisely by common statistical software like the function *pmvnorm* from the **R**-package *mvtnorm* (Genz et al., 2011). Computation times of this function are marginally small in comparison with computation times of other steps of the analysis. Of course, the normality assumption cannot hold exactly since the considered errors are averages of binary variables. In order to assess the deviation from the normal distribution section 1.5.1 as well as in Appendix A provide a selection of representative normal quantile plots for the distribution of the average errors in simulation settings. In many cases the assumption seems to hold whereas in some cases the distributions tend to more extreme values than expected under normality assumptions.

1.3.7 Implementation

The weighted mean correction method is implemented in the **R** function *weighted.mcr* included in a new version of the **R** / Bioconductor package *CMA* (Slawski et al., 2009) that can be downloaded from the companion website (`http://www.ibe.med.uni-muenchen.de/organisation/mitarbeiter/090_ehemalige/20130630_bernau/index.html/cvbias/index.html`). The codes implementing the analyses are also provided there. For larger datasets and computationally intensive methods like SVMs, ICV runtimes can be drastically higher in comparison to the two WMC variants. Detailed information on the runtimes of the different methods are provided in section 1.5.2.

1.4 Empirical results and comparison of the three estimators

1.4.1 Study design

The new estimator $\widehat{Err}_{WMC}$ (Eq. (1.10)) and its shrunken version $\widehat{Err}_{WMCS}$ (Eq. (1.12)) are evaluated on simulated data (see Sections 1.4.2 and A.2) as well as several experimental microarray datasets, and compared to the widely established estimator $\widehat{Err}_{ICV}$ (Eq. (1.4)) and to the naive raw mean estimator $\widehat{Err}_{RawMean}$ (Eq. (1.9)).

Experimental data study The real data study includes four microarray datasets: a colon cancer dataset (Alon et al., 1999) included in Bioconductor package *colonCA* with $n = 62$ diseased or healthy tissues and $p = 1991$ variables, a prostate cancer dataset (Singh et al., 2002) with $n = 102$ diseased or healthy patients and $p = 12625$ variables, a leukemia dataset (Golub et al., 1999) included in Bioconductor package *CMA* with $n = 38$ patients with two different leukemia subtypes and $p = 3051$ variables, and an ALL-leukemia dataset included in Bioconductor package *ALL* (Chiaretti et al., 2004) with $n = 100$ patients with and without relapse and $p = 12625$ variables. Additionally, modified versions of

these four datasets are considered which are obtained by replacing the response Y by a randomly generated Bernoulli distributed variable $Y' \sim \mathcal{B}(1, 0.5)$. These modified datasets are denoted as "non-informative" setup, in contrast to the original version of the datasets including "informative" predictors.

In the whole study, error rate estimation is performed through repeated subsampling into learning and test sets with $B = 100$ subsampling iterations. The proportion of observations included in the learning sets is set to 80% and 63.2% successively. In contrast to the two WMC variants, $\widehat{Err}_{ICV}$ involves another parameter, the number of folds in internal CV. There are no commonly accepted guidelines to choose the number of folds in internal CV, which can be seen as a further inconvenience of ICV. In this study, this number is chosen such that each internal test set contains approximately 5 observations. In each setup, the whole procedure is repeated $T = 50$ times in order to analyze the variability of the results. The term "repeated" indicates that $T = 50$ different *sets* of partitions $(L_b, T_b)_{b=1,\ldots,B}$ are considered successively for the original datasets, and that $T = 50$ different randomly generated responses Y' are considered successively for the modified datasets. The whole analysis thus considers a total of $50 \times 100 = 5000$ splittings into learning data L_b and test data T_b.

As outlined in the introduction and in Section 1.2, the methodology can both be applied to the correction of the tuning bias or to the correction of the method selection bias. To illustrate these two features, two setups are considered successively. In the first setup (illustrating the correction of the tuning bias and denoted as "tuning setup" (labels in tables and figures according to the methods used: "pls" and "knn")), methods $1, \ldots, K$ stand for different parameter values of a unique classification method. Two classifiers are considered successively. The first classifier is k-nearest-neighbors (kNN), where methods $1, \ldots, K$ correspond to different values $(1, \ldots, 15)$ of the parameter "number of neighbors". The second classifier is Partial Least Squares dimension reduction followed by Linear Discriminant Analysis (PLS-LDA) where methods $1, \ldots, K$ correspond to different numbers $(1, \ldots, 10)$ of PLS components. In both cases, a preliminary variable selection is performed by selecting the variables yielding the lowest p-values with the two-sample t-test (50 variables for kNN, 250 variables for PLS-LDA). Note that, in all resampling iterations, variable selection is performed using the learning set only. For ICV, this holds for the outer as well as for the inner loop.

In the second setup (illustrating the correction of method selection bias and denoted as "selection setup" (label in tables and figures: "sel")), methods $1, \ldots, K$ correspond to different combinations of classification methods and parameter values. The parameters are fixed, because tuning them with internal CV would imply three embedded CVs for $\widehat{Err}_{ICV}$, which is computationally intractable. The following classification methods are considered: nearest shrunken centroids with $\Delta = 0.5$, linear SVM with $cost = 50$, kNN with 1 neighbor based on the 20 top-variables, kNN with 18 neighbors based on the 50 top-variables, Diagonal Linear Discriminant Analysis (DLDA) based on the 20 top-variables, PLS-LDA with 3 PLS components based on the 100 top-variables and L_2-penalized logistic regression with penalty $\lambda = 0.01$. The simulation study is performed for the selection setup only, since it is generally more challenging due to the larger bias.

Note that these three different classifier pools in combination with the resampling approach for tuning/selection define wrapper algorithms ϕ, whose unconditional error rates are the actual estimation target of ICV and WMC. A summary of the results on these experimental datasets can be found in Section 1.4.3 whereas more detailed results are presented in Appendix A.1.

Simulated data The simulation study treats five different data generating processes (DGPs), including DGPs with correlated and uncorrelated predictors as well as different signal strengths and sample sizes. $T = 200$ datasets are randomly drawn from each setting. For each dataset the true error rate of the wrapper algorithm is approximated on a large validation set. Thus, the methods (WMC and WMCS) can be compared to the existing methods (ICV and raw mean) based on data, where the true value of the parameter of interest $Err = \mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi)$ –the unconditional error rate of the wrapper algorithm ϕ– is known. For simulating the predictors, a block design is used because drawing data from a 2000-dimensional multivariate normal distribution is computationally intractable. That is why blocks of 100 or 200 correlated predictors are simulated and subsequently combined. The correlations between variables from different blocks are thus zero. Since the analyses on real data have shown that the selection setups produce larger optimization biases and are generally more challenging the simulations focus on this setup. See Section 1.4.1 for details on the considered methods. The proportion of observations in the training data is set to $q = 0.8$. Note that the true parameter to be estimated primarily depends on $n_L = q \cdot n$ (not n itself), i.e. changing the proportion q also changes the true parameter. In order to analyze the behavior of the different estimators for different sample sizes, n is set successively to $n = 40$, $n = 60$ or $n = 80$ observations.

Approximating the true value of $Err = \varepsilon^{n_L}(\phi)$ The estimates produced by the investigated methods (WMC, WMCS, ICV, raw mean) are compared to the true value of the parameter of interest $Err = \mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi)$. In the simulation, this true value is approximated based on 1000 independent randomly generated datasets S of size $n_L = 0.8n$ (corresponding to the number of training observations in the outer subsampling loop). Each dataset S is used both for determining $k^*(S)$ (based on cross-validation with folds of approximate size 5) and for constructing the prediction rule using method $k^*(S)$, i.e. the wrapper algorithm is applied on each of these datasets. The 1000 constructed prediction rules are subsequently evaluated using a large independent test dataset of size 20000 generated from the same data generating process, and $\varepsilon^{n_L}(\phi)$ is estimated as the average error of the 1000 prediction rules on this test dataset.

One of the five simulation setups will be described here in depth. The remaining setups can be found in Appendix A.2.

	$n = 40$			$n = 60$			$n = 80$		
	mean	absdicv	sd	mean	absdicv	sd	mean	absdicv	sd
ICV	0.088	0.000	0.039	0.058	0.000	0.026	0.047	0.000	0.020
WMCS	0.094	0.013	0.043	0.061	0.007	0.029	0.050	0.005	0.021
WMC	0.078	0.012	0.035	0.054	0.006	0.024	0.045	0.004	0.018
Raw	0.159	0.034	0.039	0.134	0.054	0.026	0.117	0.056	0.024
Min	0.064	0.024	0.034	0.044	0.014	0.022	0.038	0.009	0.017
Max	0.255	0.167	0.058	0.224	0.167	0.048	0.206	0.159	0.041

Table 1.1: Mean, average absolute difference to ICV (absdicv) and standard deviation of the different correction methods for datasets of size $n = 40$, $n = 60$ and $n = 80$ in the setup with correlated Gaussian data.

1.4.2 Results on correlated gaussian data

This setup analyzes the performance of the different estimators in the case of correlated predictors. It tries to mimic the structure of microarray data in the following way. The variables of the Singh dataset (Singh et al., 2002) are ordered according to the absolute value of their t-statistic (t-test between groups). 200 informative predictors are generated which have the same differences $\boldsymbol{\delta}$ in group means as observed for the top 200 predictors from the Singh dataset. Moreover, the covariance matrix $\hat{\boldsymbol{\Sigma}}_{info}$ of these predictors is estimated and used for simulating the informative predictors according to the multivariate normal distribution $MNV(\boldsymbol{\delta}, \hat{\boldsymbol{\Sigma}}_{info})$ (group 1) or $MNV(\mathbf{0}_{200}, \hat{\boldsymbol{\Sigma}}_{info})$ (group 2), respectively. The non-informative predictors are simulated from the multivariate normal distribution with mean zero (regardless of the response group) and a block-diagonal covariance matrix. The covariance of each block of size 200 is chosen as the empirical covariance matrix of the corresponding predictors from the Singh dataset, i.e. the covariance of the first block of non-informative predictors equals the empirical covariance of predictors 201 to 400 from the Singh dataset (ordered according to their t-statistic), for the second the covariance of predictors 401 to 600 are used, and so on.

The boxplots in Figure 1.2 display the estimated errors obtained for the 200 simulated datasets with the five considered methods as well as the minimal and maximal error over the K investigated methods (top: $n = 40$, middle: $n = 60$, bottom: $n = 80$). The (approximated) target value $\varepsilon^{n_L}(\phi)$ is represented as an horizontal gray line. The exact values can be found in Table 1.2.The histograms in Figure 1.3 display the values of the shrinkage factors computed within the WMCS procedure for sample sizes $n = 40$, $n = 60$ and $n = 80$.

In the case of $n = 40$, WMC is closer to ICV than WMCS as far as the absolute difference is concerned (see Table 1.1). However, it constantly underestimates the bias and has a slightly bigger difference to the ICV mean. Due to the strong signal the bias in the setups with $n = 60$ and $n = 80$ is quite small and there is not much difference in the performance of the different methods. The distribution of the shrinkage factor ξ, which is

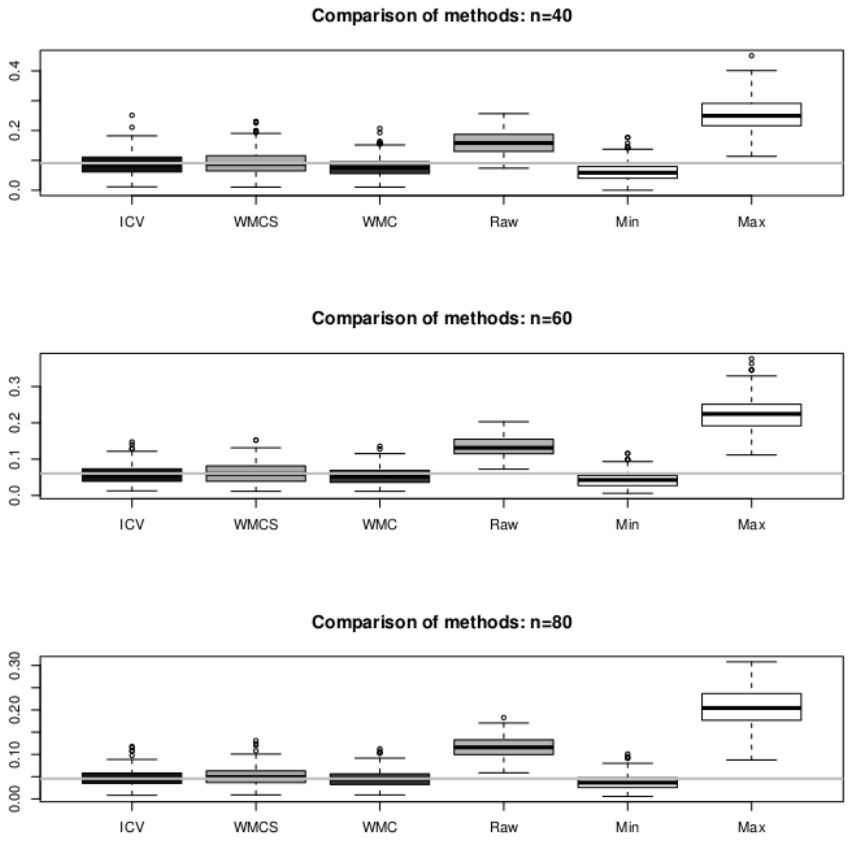


Figure 1.2: Comparison of ICV (Internal Cross-Validation), WMC (Weighted Mean Correction) and WMCS (Weighted Mean Correction with Shrinkage) for the setup with correlated Gaussian data. The gray line represents the approximation of the true value of $\varepsilon^{n_L}(\phi)$ as described in A.2.

	n=40	n=60	n=80
$\varepsilon^{n_L}(\phi)$	0.091	0.061	0.046

Table 1.2: Approximation of $\varepsilon^{n_L}(\phi)$ based on 1000 independent training sets and a large validation set containing 20000 observations (see A.2) in the setup with correlated Gaussian data.

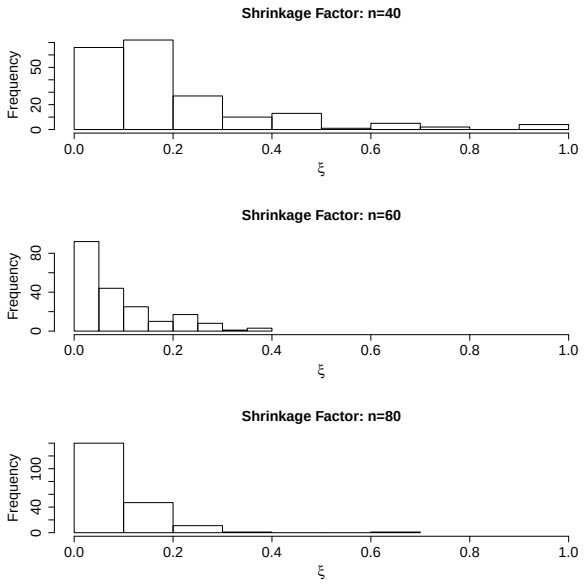


Figure 1.3: Distribution of the shrinkage factor ξ for different numbers of observations for the setup with correlated Gaussian data. The distribution gets closer to 0 as the size of the dataset increases.

illustrated in Figure 1.3, reflects this small bias. Most values are close to 0, i.e. the optimal error rate is just slightly shrunk towards the raw mean. Furthermore, the distribution is shifted even closer to 0 if the number of observations is increased. This behaviour of the WMCS estimator is appropriate in this case. Table 1.2 shows well that the tuning bias must be analyzed separately from the pessimistic bias that is induced by using less observations on the CV training sets than for the construction of the classifier on the full dataset. One can see that the true parameter $\varepsilon^{n_L}(\phi)$ distinctly decreases between 40 to 80 observations. In Section A.2 in the appendix the signal is being decreased.

1.4.3 Summary of results and comparison

An overview of the results on real datasets is given in Table 1.3 (informative setup) and Table 1.4 (non-informative setup) displaying the averages over the $T = 50$ replications of the corrected estimates $\widehat{Err}_{WMC}$, $\widehat{Err}_{WMCs}$, $\widehat{Err}_{ICV}$ and of the raw mean $\widehat{Err}_{RawMean}$ as well as the minimal and the maximal error rates $\min_k e(k \parallel s_0)$ and $\max e(k \parallel s_0)$. Figure 1.4 displays the boxplots of these error estimates for a representative real dataset in the informative setting (top) and non-informative setting (bottom), while Figure 1.2 in the previous section has shown the boxplots obtained in a simulation setting with strong signal. The most important aspects of both experimental and simulated data study are described in the following paragraphs. More details can be found in Appendix A.

In most simulation and real data settings the ICV and WMCS estimates are similar and

Table 1.3: Average corrected errors (over 50 replications) for informative pls and selection (sel) setups with 80% of the observations in the training datasets.

Setup	ICV	WMCS	WMC	Raw	Min	Max
pls-alon	0.176	0.183	0.168	0.186	0.149	0.210
pls-singh	0.087	0.092	0.080	0.091	0.075	0.180
pls-golub	0.048	0.034	0.030	0.035	0.024	0.041
pls-chiaretti	0.431	0.434	0.417	0.441	0.398	0.459
sel-alon	0.164	0.179	0.163	0.190	0.142	0.257
sel-singh	0.097	0.104	0.092	0.133	0.083	0.319
sel-golub	0.026	0.021	0.018	0.061	0.004	0.226
sel-chiaretti	0.398	0.403	0.383	0.420	0.365	0.452

range between the WMC errors and the raw mean errors. In the tuning setups, WMCS often yields results close to the raw mean indicating that its shrinkage mechanism considers differences among classifiers only marginally relevant. As pointed out before, the raw mean error is a sensible upper bound for the corrected error. The new WMCS method yields the raw mean if ξ equals 1. As mentioned in Section 1.3, the raw mean corresponds to a random choice of the parameter/method, which obviously cannot lead to a tuning or method selection bias. A good correction method is not expected to produce estimates higher than the raw mean approach. Corrected errors estimated by ICV, however, fall beyond this upper bound in some of the investigated setups, which makes poor sense in most situations and may be considered as an important disadvantage. Such a failure may also exceptionally occur with the WMC and WMCS methods, but to a much lesser extent.

Most WMC estimates are slightly more optimistic than ICV in the informative setups with the real datasets. In simulations, this tendency to under-correction is even more pronounced.

In contrast, WMCS slightly over-corrects, i.e. tends to over-estimate the error, but this tendency decreases with increasing sample size in simulations. The analysis of the shrinkage parameter ξ suggests that this slight over-correction especially occurs in intermediate cases where ξ takes values around 0.5 or 0.6. In general, however, the shrinkage parameter behaves as expected: larger values are selected more often for non-informative setups ($\xi \to 1$) than for informative setups ($\xi \to 0$). See Appendix A for more details on the shrinkage parameter.

On the Golub dataset, which is characterized by very small errors, one rarely observes ICV-corrected error rates higher than the average maximal error rate, $\max_k e(k \parallel S)$. Corrected error rates exceeding the error rate of the worst classifier can be considered as obvious failures of the correction method. In contrast to ICV, both variants of the new estimator are upper-bounded by $\max_k e(k \parallel S)$.

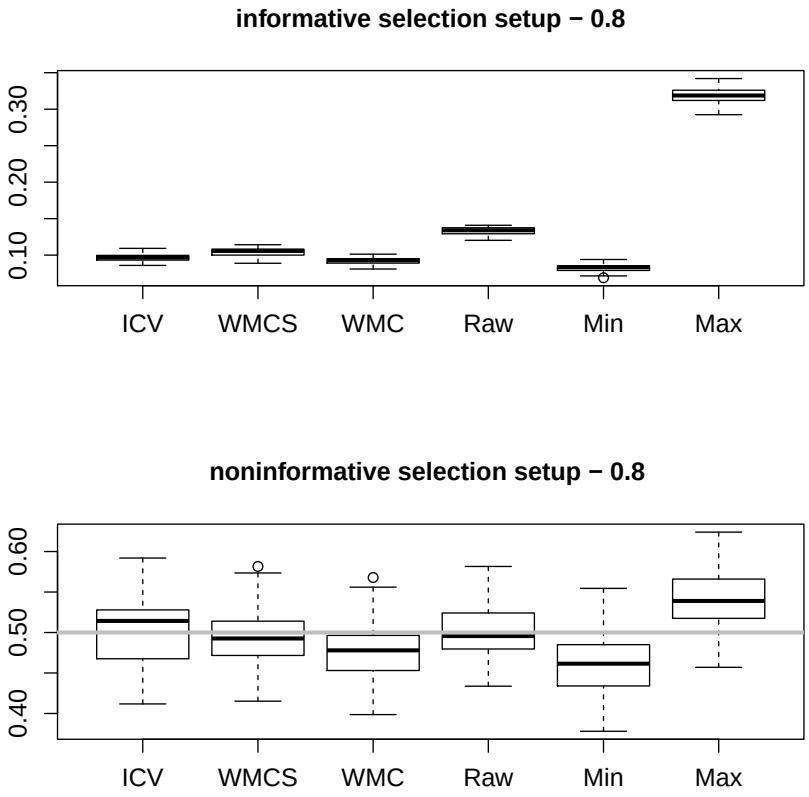


Figure 1.4: Comparison of ICV (Internal Cross-Validation), WMC (Weighted Mean Correction) and WMCS (Weighted Mean Correction with Shrinkage) for the selection setup on the prostate cancer dataset by Singh with 80% of the observations in the training datasets (top: informative setting, bottom: non-informative setting with an horizontal line at 50%).

Table 1.4: Average corrected errors (over 50 replications) for non-informative pls and selection (sel) setups with 80% of the observations in the training datasets.

Setup	ICV	WMCS	WMC	Raw	Min	Max
pls-alon	0.502	0.497	0.483	0.504	0.465	0.538
pls-singh	0.494	0.490	0.482	0.492	0.468	0.524
pls-golub	0.500	0.489	0.479	0.495	0.463	0.533
pls-chiaretti	0.495	0.492	0.482	0.498	0.469	0.523
sel-alon	0.492	0.482	0.462	0.488	0.440	0.531
sel-singh	0.504	0.496	0.479	0.503	0.463	0.541
sel-golub	0.498	0.487	0.466	0.493	0.443	0.536
sel-chiaretti	0.505	0.498	0.484	0.501	0.468	0.532

To conclude, WMCS yields more convincing results than WMC in most settings, in the sense that it leads to estimates i) near 0.5 in non-informative settings, ii) near the true error rate of the wrapper algorithms Err in simulated informative settings, and iii) near the ICV estimates in real data informative settings. The strongest deviations from ICV occur on the 0.63-setups. In the tuning setups, WMCS mostly yields results close to the raw mean. Further analyses are needed to show whether WMCS is substantially superior to the raw mean in this case. It is worth mentioning, however, that the shrinkage factor ξ always provides important evidence for the applicability of the raw mean, and WMCS is thus more informative.

1.5 Further analysis and theoretic considerations

1.5.1 Normality assumption

WMC and WMCS assume that $e(k\|S)$ follows a normal distribution. The basic idea is that this term is the average of the fold errors $e(k \parallel L_b, S \setminus L_b)$, which actually suggests a line of argument closely related to the Central Limit Theorem. However, since the fold errors are complex correlated terms, the Central Limit Theorem (CLT) is not applicable directly and no variant of the CLT adapted to this particular situation could be found after extensive literature research. Thus, this assumption is substantiated by providing normal quantile plots for $e(k\|S)$ in the context of the simulation study. Since in the simulation study the average errors $e(k\|S)$ are computed from independently drawn samples, these plots are more meaningful than their counterparts from the analysis on real datasets. A small selection of representative qq-plots is presented here and the reader is referred to `http://www.ibe.med.uni-muenchen.de/organisation/mitarbeiter/090_ehemalige/20130630_bernau/cvbias/index.html` for more exhaustive results.

The normal quantile plots displayed below were obtained in the case $n = 40$ of the

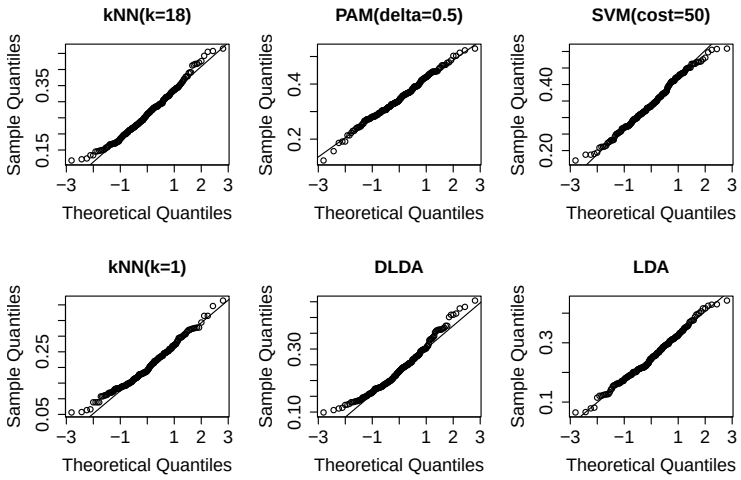


Figure 1.5: Normal quantile plots of $e(k\|S)$ for the case $n = 40$ in simulation setup A.2. Although the number of observations is quite small the deviations from the normal distribution are marginal.

simulation setup described in A.2. Although n is small here, one cannot see any evidence for large deviations from normality for any of the classifiers. Especially in the middle region, empirical and theoretical quantiles are almost the same. On the tails there are noticeable deviations. However, even though these plots are based on 200 observations, the respective points at both ends of it are computed using a very small fraction of the data. Thus, these deviations are to be expected even in the case of perfect normality as can be checked in simulations with perfectly normally distributed data (data not shown).

Figure 1.6 illustrates the *largest* deviations from normality ever observed in the simulation study, corresponding to the case $n = 40$ of the simulation setup described in A.2.1. Four of these plots show deviations at the tails (not in the middle region) that are slightly larger that one might expect under perfect normality. Nonetheless, they do not provide any evidence for substantial deviations from normality. Moreover one has to keep in mind that $n = 40$ is an extremely small sample size considering the high dimension of the data.

1.5.2 Runtimes

Especially for larger datasets and computationally more intensive classifiers like SVMs (see Table 16) runtimes for ICV can be very high. Computation times are at least 7 times smaller with the new method – in particular cases even up to 15 to 18 times smaller. Since both variants of WMC need only the outer resampling approach their runtimes are almost the same. WMC (≈ 0.044 seconds runtime for $B = 100$ fold errors of $K = 7$ classifiers) is slightly faster than WMCS (≈ 0.54 seconds runtime for the same setup.) because it does not involve the shrinkage step. However, considering the large (unavoidable) runtimes of the external resampling approach, even the 0.5 seconds of WMCS are comparatively

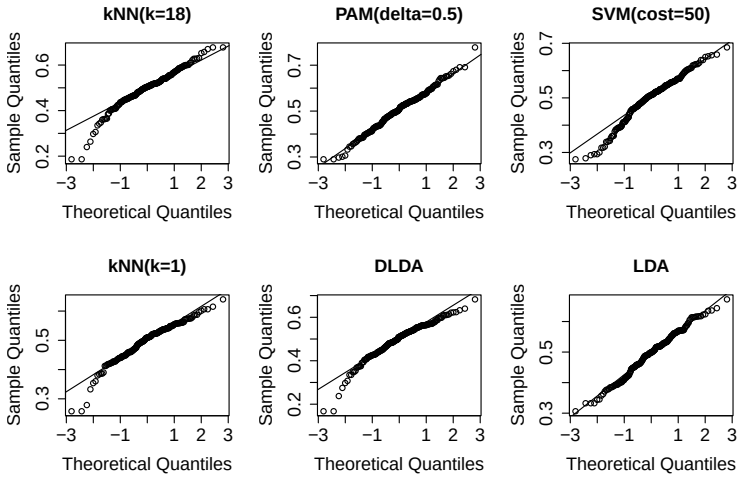


Figure 1.6: Normal quantile plots of $e(k\|S)$ for the case $n = 40$ in simulation setup A.2.1 illustrating the largest deviations from normality observable in the simulation study.

Runtimes	WMC	WMCS	ICV
Alon	14.2	14.9	167.3
Singh	29.1	29.2	547.7
Golub	13.3	13.8	99.0
Chiaretti	33.7	32.1	579.6

Table 1.5: Average runtimes (in seconds) WMC (Weighted Mean Correction), WMCS (Weighted Mean Correction with Shrinkage) and Internal Cross-Validation for the kNN-setups on the different datasets. The difference between WMC and WMCS is negligible in comparison to the absolute runtimes.

Runtimes	WMC	WMCS	ICV
Alon	67.7	68.8	906.9
Singh	610.3	611.3	11158.4
Golub	88.3	89.5	716.1
Chiaretti	664.6	646.7	11705.7

Table 1.6: Average runtimes (in seconds) WMC (Weighted Mean Correction), WMCS (Weighted Mean Correction with Shrinkage) and Internal Cross-Validation for the selection-setups on the different datasets. The difference WMC and WMCS is negligible in comparison to the absolute runtimes.

negligible. Note that in the selection setups the implementation of SVM from the **R**-package *e1071* (Meyer et al., 2012, based on *libsvm*) is used, which is probably not the most efficient one for larger datasets. Nonetheless, it seems to be **R**'s standard routine for SVMs and is commonly used. The following runtimes have been obtained using a Linux multiprocessor computer with an Intel Xeon 5150 CPU at 2.66GHz and 6GB RAM (the computation has been performed by a single core).

1.5.3 Decision Theoretic Interpretation

The problem discussed above evidently includes several elements which can be found in the standard decision theoretic problem. Many aspects of the WMC estimator have also been conceptualized within this decision theoretic framework in mind. As will be shown in a later part of this section, the crucial assumptions and quantities all have a decision theoretic interpretation. Also the important term wrapper algorithm can be paralleled to strategies which are a central element of decision theory. This section describes the parallels to decision theory, thereby providing further justification and explanation for the approach of the WMC estimator by embedding it into this classical framework. At first, this subsection presents a short treatise of the most fundamental terminology of decision theory for readers unfamiliar with this discipline. This small introduction is adapted from the classical work by Berger (1980). The book may also serve the interested reader as an exhaustive description of the subject.

Short introduction to decision theory

Decision theory is primarily concerned with the problem of finding reasonable decisions under certain aspects of uncertainty. In the standard problem one can choose among several actions $a \in \mathscr{A}$ which produce certain losses W if they are implemented in different possible states of nature $\theta \in \Theta$. Actions and states of nature are at first abstract terms. Their crucial role consists in the fact that the Cartesian product of the set of possible actions $\mathscr{A}$ and the set of possible states of nature Θ builds the domain of the loss function W:

$$\begin{aligned} W : \Theta \times \mathscr{A} &\mapsto \mathbb{R}^- \\ (\theta, a) &\mapsto W(\theta, a). \end{aligned} \tag{1.14}$$

The restriction to $\mathbb{R}^-$ is a convenience without loss of generality in the case of symmetric, linear losses [1]. One could also formulate this function with regard to gains which are just negative losses. One can assume, it is the differences between losses which matter rather than their absolute level.

The introduced terms shall now be illustrated in an example which is related to common hypothesis testing. Suppose one has to judge whether a certain hypothesis H_0 is true or

[1] In psychology and economic sciences, losses are often considered to have stronger effects than gains. Here, however, such an imbalance is not expected.

	$\mathscr{A}$	
Θ	reject H_0	accept H_0
H_0 is true	W_α	0
H_0 is not true	0	W_β

Table 1.7: Loss matrix **W** for the classical testing problem. In concordance to common notation, W_α and W_β refer to the losses in the case of type 1 error and type 2 error respectively.

false. As actions one has a_1 (accept H_0) and a_2 (reject H_0). If no zone of indifference is allowed, these two actions build the complete domain of possible actions $\mathscr{A}$. The set of states of nature Θ is equally small consisting of θ_1 (H_0 is true) and θ_2 (H_1 is not true). Consequently, Table 1.7 already depicts the complete 'loss matrix', **W**, where the possible losses are specified for all combinations of a and θ.

It is intuitive that the difference between the values W_α and W_β, the loss of the type 1 error and the type 2 error, primarily impacts the way one will design the testing procedure. In classical testing theory, W_α is considered dominating which leads to the approach of accepting H_0 unless there is substantial evidence against it, in order to avoid W_α. This approach is mandatory in testing theory. Contrastingly, one could handle these quantities in a more flexible fashion in decision theory. Related to this issue, a very interesting discussion can be found in Armitage (1989) where the author treats the question whether decision theory could be more appropriate than classical testing theory in certain studies in epidemiology. For example it is questionable how one has to proceed after a study where a new drug is better than the gold standard, but the difference was not significant. In testing theory one has to stick to the gold standard H_0 in order to avoid any harm caused by a drug for which superiority has not been shown on a sufficiently high level. In decision theory, one would have to carefully consider the loss associated to withholding a 'seemingly' better drug as well.

Another important aspect, which has to be considered in this discussion, is the distribution $\pi(\theta)$ of the different states of nature. These probabilities specify how probable the occurrence of a certain loss is. Since this distribution is unknown in practice, $\pi(\theta)$ is usually introduced as a Bayesian subjective a-priori distribution $\pi_0(\theta)$. Here, however, there is no need to enter the discussion about a-priori distributions and it is simply assumed that a distribution of the states of nature $\pi(\theta)$ exists. This distribution leads directly to another crucial quantity, the expected loss of an action:

$$\mathcal{W}_a = \mathbb{E}_\pi W(\theta, a) = \sum_\Theta W(\theta, a)\pi(\theta). \tag{1.15}$$

If further information on the actual state of nature is missing, a good decision consists in choosing the action with the smallest expected loss. The crucial task consists in collecting information about the actual state of nature. Note that the actual state can usually not be determined directly. In the drug example, one can usually not determine the superiority of

a drug even though one might have a sound chemical explanation for its effectiveness. This gathering of information is called *experiment* in decision theory and in this example one would probably conduct a randomized trial. In an experiment, a sample $\mathbf{G} = (G_1, \ldots, G_n)$ is drawn from a random variable G. The distribution of this random variable P_G is usually clearly affected by the actual state of nature, so that one might draw a conclusion from the sample $\mathbf{G}$ to the actual state of nature. The corresponding sample space is denoted by $\mathscr{G}$. If one has to compare two drugs, this random variable could be the difference in blood pressure under the new drug and the gold standard, which one could measure for n patients. Subsequently, one can deduct a decision rule or strategy $\delta(\mathbf{G})$ from this experiment, which can be defined as the following measurable function:

$$\begin{aligned} \delta : \mathscr{G} &\mapsto \mathscr{A} \\ \mathbf{G} &\mapsto \delta(\mathbf{G}). \end{aligned} \tag{1.16}$$

The strategy maps each possible outcome of the experiment to a certain action. The common strategy for the introduction of a new blood pressure drug would be to test $\mathbf{G}$, the difference in blood pressure described above, for a significant difference from zero in favor of the new drug (using maybe a one-sided paired t-test) and grant the admission accordingly. The quality of such a decision rule can subsequently be assessed by its risk $\mathcal{R}$:

$$\mathcal{R}(\theta, \delta) = \mathbb{E}_\theta^{\mathbf{G}}[W(\theta, \delta(\mathbf{G}))] = \int_{\mathscr{G}} W(\theta, \delta(\mathbf{g}))\, dP_{\mathbf{G}|\theta}, \tag{1.17}$$

where $\mathbf{g}$ denotes a realization of $\mathbf{G}$. Although the risk is an expectation, it is rather to be considered the natural counterpart of the term loss in the context of actions. On the contrary, the expected loss of an action can be paralleled to the expected risk which is defined as the following expectation:

$$r(\pi, \delta) = \mathbb{E}^\pi[\mathcal{R}(\theta, \delta)] = \sum_{\Theta} \mathcal{R}(\theta, \delta)\pi(\theta), \tag{1.18}$$

i.e. here the expectation over the different states of nature is computed just as in the case of the expected loss of an action. In order to minimize $r(\pi, \delta)$, a strategy must be able to find the actions with minimal loss with a high probability in all states of nature just as cross-validation (when used for model selection) has to find the appropriate classifier with a high probability. Before the WMC estimator is treated, the introduced terms are illustrated in another example that has already been described in this thesis.

Class prediction from a decision theoretic point of view Another example for a decision problem as well as the corresponding loss function have already been introduced in Section 1.2.2, which is also the reason why the classical notation L for the loss could not be used here. The classifier in this section had two possible actions because it could choose from two classes. Here one has the special case of $\Theta = \mathscr{A}$ because the states of nature are also the two possible classes. The loss was defined symmetrically here because the misclassification rate was used. Correct classification produced no loss and misclassification

	$\mathscr{A}$	
Θ	predict class 1	predict class 2
class 1 is true	1	0
class 2 is true	0	1

Table 1.8: Loss matrix for the classification problem. The loss is symmetric because the loss is the same regardless of the predicted class.

produced the same loss regardless of the true class (see Table 1.8). Consequently, the expected loss of action a_1 can be computed by:

$$\mathcal{W}_{a_1} = \mathbb{E}_\pi W(\theta, a_1) = \sum_\Theta W(\theta, a_1)\pi(\theta) = 0 \times \pi(\theta_1) + 1 \times \pi(\theta_2) = \pi(\theta_2),$$

i.e. simply the probability for class 2. The next question is how one might construct a reasonable experiment. Here one has to further specify the problem. Suppose that the only measurements one can get from the new patient, which has to be classified, are the gene expressions $\mathbf{x}^\sharp$. His state of nature is denoted by $\theta^\sharp \in \Theta$. Imagine that one already has a classifier $\hat{f}^{\mathbf{s}_0}$, constructed on a previous sample $\mathbf{s}_0$, which can detect (with an acceptably high probability) cancer from the gene expressions at a time where no other symptoms can be observed. Consequently, the experiment does not consist in determining the actual state of nature by diagnostic techniques like a histological analysis. However, a possible approach consists in measuring the gene expressions $\mathbf{x}^\sharp$ of the current patient, i.e. the random vector $\mathbf{x}^\sharp$ is the experiment sample $\mathbf{G} = G$ which this time comprises only a single vector of gene expressions. Its sample space is denoted by $\mathcal{X}$ (see Section 1.2.2) and its distribution is denoted by $P_{\mathbf{x}^\sharp}$. Now, the strategy is to predict the patient's class as suggested by the classifier $\hat{f}^{\mathbf{s}_0}$ using his gene expressions $\mathbf{x}^\sharp$. The corresponding risk is:

$$\mathcal{R}(\theta, \delta(G)) = \mathbb{E}_\theta^{\mathbf{x}^\sharp}[W(\theta, \hat{f}^{\mathbf{s}_0}(\mathbf{x}^\sharp))] = \int_\mathcal{X} W(\theta, \hat{f}^{\mathbf{s}_0}(\mathbf{x}^\sharp))\, dP_{\mathbf{x}^\sharp|\theta}.$$

The corresponding expected risk is $r(\pi, \delta) = \sum_\Theta \mathcal{R}(\theta, \delta(G))\pi(\theta)$ and can be reformulated as the conditional error rate from Section 1.2.2 in the following way:

$$\begin{aligned} r(\pi, \delta) &= \sum_\Theta \int_\mathcal{X} W(\theta, \hat{f}^{\mathbf{s}_0}(\mathbf{x}^\sharp))\, dP_{\mathbf{x}|\theta}\pi(\theta) \overset{1}{\rightsquigarrow} \int_{\mathcal{X}\times\mathcal{Y}} W(y, \hat{f}^{\mathbf{s}_0}(\mathbf{x}^\sharp))\, dP(\mathbf{x}^\sharp, y) \\ &\overset{1}{\rightsquigarrow} \int_{\mathcal{X}\times\mathcal{Y}} L(\hat{f}^{\mathbf{s}_0}(\mathbf{x}), y)\, dP(\mathbf{x}, y) = \varepsilon(\hat{f}^{\mathbf{s}_0} \| \mathbf{s}_0) = \varepsilon(k \| \mathbf{s}_0). \end{aligned}$$

[1]Here one needs to consider the unavoidable differences to the notation in Section 1.2.2. The classes y are identical to the states of nature θ and hence $\mathcal{Y} = \Theta$. Consequently, $P(\mathbf{x}^\sharp, \theta) = P(\mathbf{x}, y)$ is the joint distribution of $\mathbf{x}$ and y as already defined in Section1.2.2. L is the Loss function defined in Section 1.2.2 and corresponds to W in the current notation. In Section 1.2.2, $\varepsilon(\hat{f} \| \mathbf{s}_0)$ has been introduced where k was shorthand for a classifier algorithm $\hat{f}_k$. This classifier is then used to construct a prediction rule on $\mathbf{s}_0$ which is indicated by $(k \| \mathbf{s}_0)$.

Thus, one can say that the class prediction from Section 1.2.2 can also be formulated within decision theory. Now, all the relevant terms have been introduced which are needed to embed the tuning/model selection task, which is addressed with the WMC estimator, into a decision theoretic framework.

The WMC estimator in a decision theoretic framework

At first, the more obvious elements are described. One has to choose adequately from a selection of possible actions $\mathscr{A}$ (the different algorithms k) which lead to different losses or errors W if they are applied in various states of nature. The states of nature are the distributions of $\mathbf{X}$ and $\mathbf{y}$ which define different classification problems. These distributions are denoted by P_θ here to clarify that they represent the states of nature which are commonly denoted by θ. Further, one conducts an experiment (cross-validation or resampling in general) in order to obtain information on the actual state of nature and the respective suitability of the available actions.

Global interpretation

A first difference consists in the fact that cross-validation is an experiment with internal variation, i.e. for a fixed dataset drawn from the distribution P_θ, it can still lead to different results depending on the partitioning of the data into the different folds. Thus, pseudo-deterministic experiments are assumed in which the result of the experiment (here the resampling) is deterministic on such a fixed dataset because a random seed is set before the partitioning is sampled by a computer program. Under this assumption, one can formulate the problem as a classical decision theoretic problem as shown in Figure 1.7.

As actions one has the different classifier algorithms k, $k = 1, \ldots, K$, and as states of nature one has a set of classification problems which can be represented by the distribution of regressors and response $P_\theta(\mathbf{X}, \mathbf{y})$, $\theta = \theta_1, \ldots, \theta_\Xi$. The loss $W^u(k_1, P_\theta)$ is the unconditional error rate (the index u is used to avoid confusion with the conditional loss W^c which will be introduced below) of the respective classifier on the distribution $P_{\theta_\cdot}$.

Now, a crucial point is addressed where this interpretation theoretically deviates from the common procedure for resampling based model selection. Classical decision theory assumes that one performs a single experiment each time one faces a new state of nature. Consequently, one has to postulate that there is only a single global or unconditional decision. This means that once the CV has been performed on a single dataset from the distribution P_θ the winning algorithm is declared to be used for the current state of nature, P_θ. The experiment, the resampling, is not reperformed on new datasets from this same distribution (which are then only used for retraining the algorithm) because one is still facing the same current state of nature. Evidently, one is not facing a new distribution P_θ just because one has drawn a new sample from it. This viewpoint directly leads to the central formula of the WMC estimator:

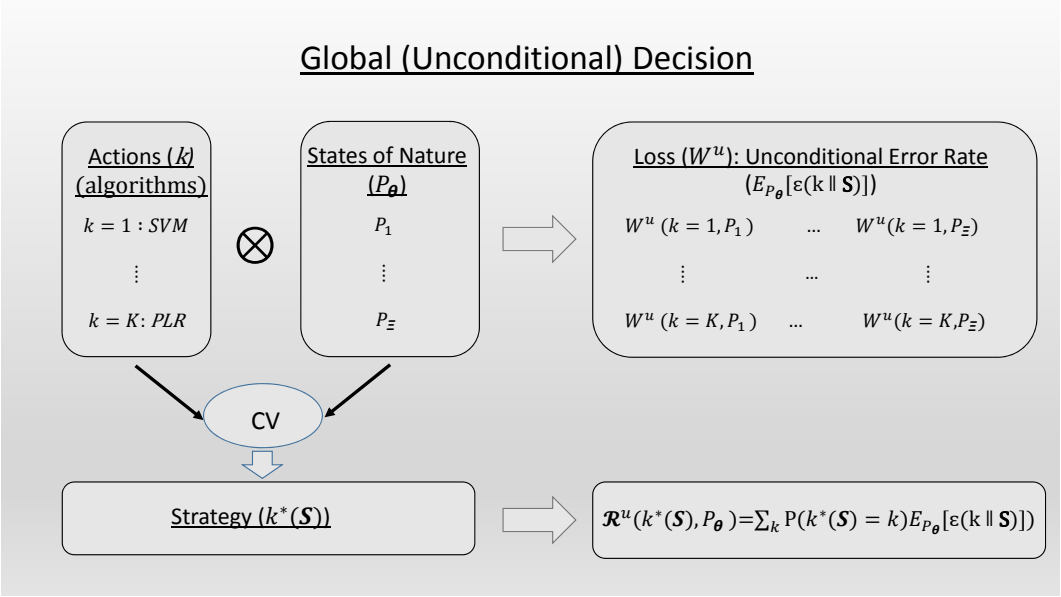


Figure 1.7: Global (unconditional) decision problem. The algorithms k, $k = 1, \ldots, K$, can be interpreted as actions and the distributions P_θ as states of nature. The corresponding losses are the unconditional error rates.

$$\sum_{k=1}^{K} \mathrm{P}\left(k^*(CV) = k\right) \cdot \mathbf{E}_{P_\theta^{n_L}}\left[\varepsilon(k \parallel S) | k^*(CV) = k\right],$$

where S denotes a sample of size n_L (the number of observations in a training set of the outer cross-validation loop) drawn from P_θ. By postulating a single decision, the condition $k^*(S) = k$ can be erased because one only decides once and the chosen algorithm is used for all possible training datasets to come from the current state of nature, P_θ. Hence, one can define the loss matrix $\mathbf{W}$ depicted in Figure 1.7 using $\mathbf{E}_{P_\theta^{n_L}}\left[\varepsilon(k \parallel S)\right]$, $k = 1 \ldots, K$, $\theta = \theta_1, \ldots, \theta_\Xi$. Parallel to common decision theory, one only needs to obtain the probabilities $\mathrm{P}\left(k^*(S) = k\right)$, in order to compute the risk $\mathcal{R}$ of a strategy $\delta = k^*$ on a specific state of nature P_θ.

In this context, one has to discuss the valid objection that this interpretation does not match the practical procedure because the model selection step is repeated on each dataset in nested cross-validation. These datasets in the nested cross-validation correspond to the new datasets from the same current state of nature (P_θ) in the decision theoretic interpretation. Even during the NCV procedure itself, however, different classifiers might be chosen on different datasets of the outer loop.

Local interpretation

Now, a second decision theoretic interpretation of the problem is presented which is also useful to shed further light on this objection. One can formulate the problem as a whole set of local or conditional decision problems. Figure 1.8 illustrates an example for these decision problems.

Here, one has a training dataset $\mathbf{s}^{TR}$ on which the models or prediction rules of the different algorithms, $\hat{f}_k^{\mathbf{s}^{TR}}$, $k = 1, \ldots, K$, have already been trained. These models are fixed now and even though they have been fitted using different algorithms, e.g. support vector machines or penalized logistic regression, the origin of the models is irrelevant for the decision problem itself. The possible actions consist in the different fixed prediction rules $\hat{f}_k^{\mathbf{s}^{TR}}$, $k = 1, \ldots, K$, whereas the states of nature remain the same as in the previous interpretation, i.e. the distributions P_θ. The appropriate losses are the conditional error rates of the different models $\hat{f}_k^{\mathbf{s}^{TR}}$.

In this interpretation the shift of the actions from algorithms to fixed prediction rules or models is crucial. Although one could still formulate that one chooses among the different algorithms from which the different models originate, this would not be in line with decision theory. The loss W^c of an action in a specific state of nature has to be a fixed quantity which is true for the conditional error rate of a fixed model. For an algorithm, the conditional error rate would be different on another training set. One can say that a distribution P^{TR} for the training dataset and a specific selection of algorithms induce a set of similar (local) decision problems as depicted in Figure 1.8 by providing a different set of actions for each sample $\mathbf{S}^{TR}$ drawn from P^{TR}.

A straightforward way for summarizing these decision problems for the actions consists

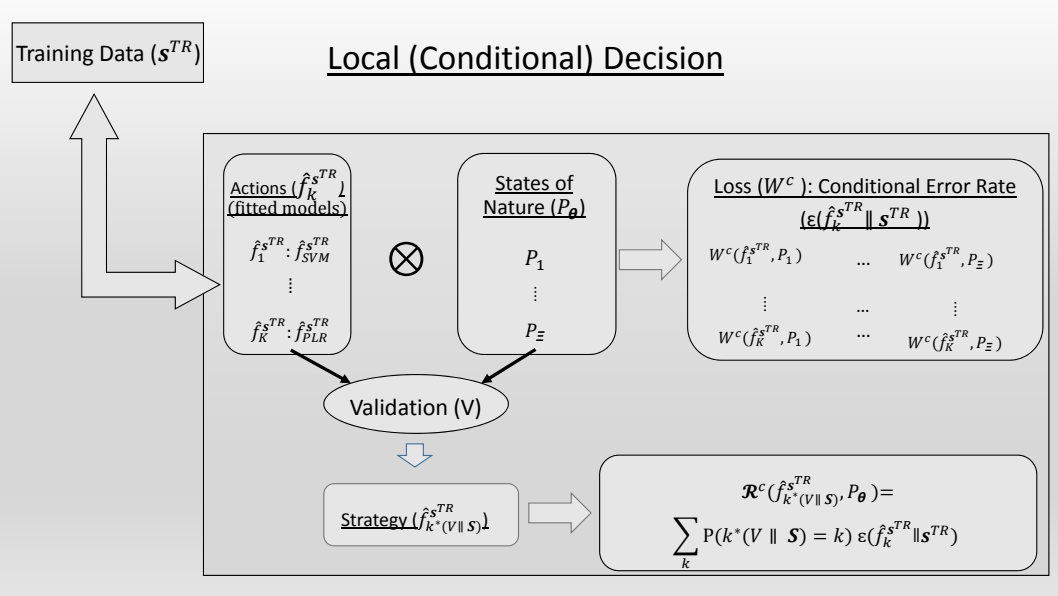


Figure 1.8: Local (conditional) decision problem. The fitted predictions rules, $\hat{f}_k^{\mathbf{s}^{TR}}$, $k = 1, \ldots, K$ can be interpreted as actions and the distributions P_θ as states of nature. The corresponding losses are the conditional error rates of the fitted prediction rules.

in integration over all possible samples of $\mathbf{S}^{TR}$ drawn from P^{TR} whereby always the risk of the model trained by the same algorithm k is taken[1]:

$$\int \varepsilon(\hat{f}_k^{\mathbf{S}^{TR}} \parallel \mathbf{S}^{TR})\, dP^{TR} = \int \varepsilon(k \parallel \mathbf{S}^{TR})\, dP^{TR}.$$

If one additionally assumes $P^{TR} = P_\theta$, one will arrive at the global decision problem stated above as far as the actions are concerned because one has the same actions k and the same losses ($\int \varepsilon(k \parallel \mathbf{S}^{TR})\, dP_\theta = \mathbf{E}_{P_\theta}\left[\varepsilon(k \parallel \mathbf{S}^{TR})\right]$). For strategies, however, this would not be the case because here one would be allowed to choose models from different algorithms on the respective training samples $\mathbf{S}^{TR}$ (also drawn from P_θ) depending on the experiment performed on $\mathbf{S}$ drawn from P_θ. This again reveals the basic problem of defining the algorithms as actions on the local level because their loss varies over the different training datasets (For $P^{TR} = P_\theta$, the basic problem consists in the fact that the state of nature redefines the actions via the drawn training dataset $\mathbf{S}^{TR}$. This training data set is in practice also used for the CV experiment, i.e. training and model selection are not performed on separate datasets).

The key assumption for bringing these two interpretations on a common basis is the crucial assumption used in the WMC estimation:

$$\mathbf{E}_{P_\theta}\left[\varepsilon(k \parallel \mathbf{S}) | k^*(\mathbf{S}) = k\right] \approx \mathbf{E}_{P_\theta}\left[\varepsilon(k \parallel \mathbf{S})\right].$$

In the notation used for the two decision theoretic interpretations, this can be written as [2]:

$$\begin{aligned} \mathbf{E}_{P_\theta}\left[\varepsilon(k \parallel \mathbf{S}^{TR}) | k^*(V \parallel \mathbf{S}^{TR}) = k\right] &= \int \varepsilon(\hat{f}_k^{\mathbf{S}^{TR}} \parallel \mathbf{S}^{TR})\, dP_\theta^{\mathbf{S}^{TR}|k^*(V\parallel\mathbf{S}^{TR})=k} \\ &\approx \int \varepsilon(\hat{f}_k^{\mathbf{S}^{TR}} \parallel \mathbf{S}^{TR})\, dP_\theta \mathbf{S}^{TR} = \mathbf{E}_{P_\theta}\left[\varepsilon(k \parallel \mathbf{S}^{TR})\right], \end{aligned} \tag{1.19}$$

where the notation $k^*(V \parallel \mathbf{S}^{TR}) = k$ means that the experiment CV performed on the sample $\mathbf{S}^{TR}$ by the strategy k^* has chosen the model $\hat{f}_k^{\mathbf{S}^{TR}}$ fitted by algorithm k. Likewise, the ponderous expression $P_\theta^{\mathbf{S}^{TR}|k^*(V\parallel\mathbf{S}^{TR})=k}$ simply denotes the distribution of $\mathbf{S}^{TR}$ under exactly this condition. If the assumption in Eq. (1.19) holds, one can reformulate the 'aggregated' (aggregated over the conditional decsision problems) risk for the strategy in the following way [3]:

$$\sum_{k=1}^{K} \int \varepsilon(\hat{f}_k^{\mathbf{S}^{TR}} \parallel \mathbf{S}^{TR})\, dP_\theta^{\mathbf{S}^{TR}|k^*(V\parallel\mathbf{S}^{TR})=k} \mathrm{P}(k^*(V \parallel \mathbf{S}^{TR}) = k)$$

[1] Note that the expectation ε is taken over P_θ whereas $\mathbf{S}^{TR}$ is sampled from P^{TR}.
[2] $\mathbf{S}^{TR}$ is drawn from P_θ and thus corresponds to $\mathbf{S}$ in the previous formula.
[3] $\mathrm{P}(k^*(V \parallel \mathbf{S}^{TR}) = k)$ is the marginal probability that the experiment V chooses k.

$$\overset{2}{\approx} \sum_{k=1}^{K} \mathrm{P}(k^*(V \parallel \mathbf{S}^{TR}) = k)\, \mathbf{E}_{P_\theta}\left[\varepsilon(k \parallel \mathbf{S}^{TR})\right],$$

which is equal to the risk defined in Figure 1.7 on the bottom line. The assumption in Eq. (1.19) states that the (conditional) loss W^c of the models $\hat{f}_k^{(\cdot)}$ of a certain algorithm k on those datasets, where this algorithm is chosen by the experiment (V), is on average not smaller than the unconditional error rate of this algorithm. Or put another way: the experiment cannot find locally good strategies which re-decide on each training what algorithm is to be used. This matches the common assumption that algorithms are able to fit into patterns or signals which are present in specific distributions and not just on specific datasets. In the context of experimental microarray data, Hanczar et al. (2007) have provided evidence for this assumption, which has already been discussed in Section 1.3.1.

Additionally, one can substantiate this assumption for the case of nested cross-validation by looking at the experiment which is applied ($V = CV$). If possible, one would certainly use a single large dataset drawn from P_θ in order to choose among the different prediction rules. This is especially true in cases where one does not know by which algorithm the decision rules or models $\hat{f}^{\mathbf{s}^{TR}}$ were constructed (i.e. CV cannot be performed because one cannot retrain the prediction rules with an unknown method) as well as in all cases where $P_\theta \neq P_{TR}$ (as in Chapter 2).

Even in the standard case $P_\theta = P_{TR}$, however, it is very questionable whether CV can really find locally best models (i.e. datasets for which the model constructed by a certain algorithm performs very well although the algorithm performs clearly worse when fitted on another dataset from the same distribution $P_\theta = P^{TR}$). For this, the CV estimate has to be very specific for different datasets. This actually requires that the model is quite stable, i.e. that in each resampling step the model is almost the same as the one fitted on the dataset. Formulated another way, the CV estimate will be very close to the conditional error rate of the model fitted on the complete dataset. This can be assumed for models that will not change strongly if several observations are removed or modified. In this case, however, this model will perform equally well on most datasets. This means that CV cannot find datasets, for which $\varepsilon(\hat{f}_k^{\mathbf{S}} \parallel \mathbf{S}) << \mathbf{E}_{P_\theta}[\varepsilon(k \parallel \mathbf{S})]$ holds, because they actually do not exist. Usually, this is the case for algorithms where the resubstitution rate could also be used as an estimation for the conditional error rate. Such algorithms are fairly uncommon in the context of high-dimensional small sample data. On the contrary, for unstable algorithms, the CV estimate will be closer to the unconditional error rate. Hence, the CV experiment will lack the necessary specificity for the good datasets.

An alternative resampling scheme

A resampling approach that would be more in line with the interpretation of a set of local decision problems would be the following (see Figure 1.9). One first splits the dataset

[1] Note that $\mathrm{P}(k^*(V \parallel \mathbf{S}^{TR}) = k)$ is just an extended version of the notation $\mathrm{P}(k^*(\mathbf{S}^{TR}) = k)$ explicitly mentioning the experiment V.

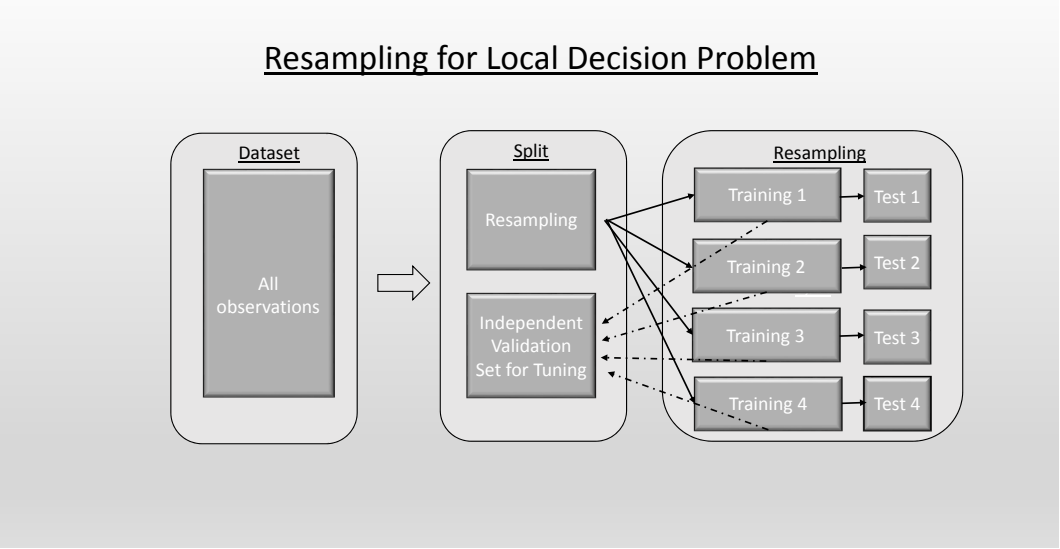


Figure 1.9: Resampling scheme for local (conditional) decision problem. An independent validation set is reserved for the tuning step that is performed on each training set in the resampling approach. On each training set in the resampling one is faced with a local decision problem whereby each of these problems includes newly defined actions, the different predictions rules fitted on the current training set.

into a 'resampling' set and a validation set reserved for the validation that is performed during the model selection experiment (V in Fig. 1.8). Subsequently, one performs a cross-validation on the 'resampling' set whereby on each fold the model performing best on the independent validation set is chosen, i.e. the model selection experiment V now uses data for its validation step which have not been used to define the actions, i.e. the constructed models. In this approach, it is really the local models which are evaluated on the independent evaluation set and even for highly unstable algorithms it is possible to obtain a specific local performance estimate, if the tuning validation set is large enough.

Concluding, one can say that almost all important assumptions and quantities in the WMC estimator can be embedded in the decision theoretic standard problem. Also the discussion about the usefulness of unconditional and conditional error rate, which mainly characterizes a crucial difference between the interests of statisticians and physicians, can be explained in this context as well as the term wrapper algorithm. It can be derived from the term strategy or decision rule in decision theory. From this point of view, wrapper algorithms have actually already been a subject of statistical research for a long time.

1.5.4 Asymptotic consideration – justification of $Err = \varepsilon^{n_L}(\phi)$ as target of Err_{ICV}

In order to further motivate the new method, the following asymptotic consideration of Err_{ICV} is presented which justifies the quantity $Err = \varepsilon^{n_L}(\phi)$, that has been introduced in Section 1.2.3, as the actual estimation target of Err_{ICV}. Additionally, a decomposition of this quantity has been described in Section 1.3.1 (Eq. 1.5) which constitutes the fundament of the estimators Err_{WMC} and Err_{WMCS}.

In the notation used in this thesis, the ICV estimator can be written as:

$$Err_{ICV} = \frac{1}{B}\sum_{b=1}^{B}\sum_{k=1}^{K} I(k^*(L_b) = k) \cdot e(k \parallel L_b, S \setminus L_b),$$

where $b = 1, \ldots, B$ corresponds to the subsampling iterations, $k = 1, \ldots, K$ denotes the different classifiers and $k^*(\cdot)$ indicates the classifier chosen by the internal cross-validation, i.e. $k^*(L_b) = k$ means that classifier k has been chosen by the internal cross validation procedure on the training set in the bth subsampling/resampling iteration, and $e(k \parallel L_b, S \setminus L_b)$ is the misclassification rate of classifier k on the bth test set.

Now, suppose that nested resampling is performed on D independent datasets of the same data generating process, whereby each dataset S_d has n observations and is drawn from P^n. Then the average is built:

$$\frac{1}{D}\sum_{d=1}^{D}\frac{1}{B}\sum_{b=1}^{B}\sum_{k=1}^{K} I(k^*(L_{db}) = k) e(k \parallel L_{db}, S_d \setminus L_{db}).$$

For $D \to \infty$ one obtains:

$$\lim_{D\to\infty} \frac{1}{D}\sum_{d=1}^{D}\frac{1}{B}\sum_{b=1}^{B}\sum_{k=1}^{K} I(k^*(L_{db})=k)e(k \parallel L_{db}, S_d \setminus L_{db})$$

$$\overset{1}{=} \lim_{D\to\infty} \frac{1}{B}\sum_{b=1}^{B}\frac{1}{D}\sum_{d=1}^{D}\sum_{k=1}^{K} I(k^*(L_{db})=k)e(k \parallel L_{db}, S_d \setminus L_{db})$$

$$(***)$$

Consider the expectation of the error $e^1_{new} = |y^{new} - \hat{f}^{S_{n_L}}_{k^*(S_{n_L})}|$ of a single (new) observation from P^1 before S_{n_L} has been drawn from P^{n_L}. Relying on $y^{new} \perp S_{n_L}$, and $y^{new} \perp k^*(S_{n_L})$ one obtains:

$$\begin{aligned}
\mathbf{E}(e^1_{new}) &= \int \sum_{k=1}^{K} \int |y^{new} - \hat{f}_k^{s_{n_L}}|\, dP^1 \mathrm{P}(k^*(s_{n_L})=k|S_{n_L}=s_{n_L})\, dP^{n_L} \\
&= \int \sum_{k=1}^{K} \varepsilon_k[\hat{f}_k^{s_{n_L}}] \mathrm{P}(k^*(s_{n_L})=k|S_{n_L}=s_{n_L})\, dP^{n_L} \\
&= \sum_{k=1}^{K} \int \varepsilon_k[\hat{f}_k^{s_{n_L}}]\, dP^{S_{n_L}|k^*(S_{n_L})=k} \mathrm{P}(k^*(S_{n_L})=k) \\
&= \sum_{k=1}^{K} \mathbf{E}\left[\varepsilon(k \parallel S_{n_L})|k^*(S_{n_L})=k\right] \mathrm{P}(k^*(S_{n_L})=k) \\
&\overset{2}{=} \mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi) = Err
\end{aligned}$$

Consequently, one obtains [3]:

$$\mathbf{E}\left[\sum_{k=1}^{K} I(k^*(L_{db})=k)e(k \parallel L_{db}, S_d \setminus L_{db})\right] = \frac{n_{ts} \times \mathbf{E}(e^1_{new})}{n_{ts}} = Err = \varepsilon^{n_L}(\phi),$$

where n_{ts} denotes the number of test observations. Applying the law of large numbers at $(***)$ leads to:

$$\lim_{D\to\infty} \frac{1}{B}\sum_{b=1}^{B}\frac{1}{D}\sum_{d=1}^{D}\sum_{k=1}^{K} I(k^*(L_{db})=k)e(k \parallel L_{db}, S_d \setminus L_{db}) \overset{\text{a.s.}}{\to} \frac{1}{B}\sum_{b=1}^{B}\varepsilon^{n_L}(\phi) = \varepsilon^{n_L}(\phi) = Err.$$

[1] This reordering is allowed because the series is absolutely convergent.

[2] compare to Eq.(1.6)

[3] Note that the observations in a single testfold are independent from each other and the training set.

The natural interpretation of this asymptotic consideration is that ICV estimates a combined error rate instead of focusing on a specific classifier or tuning parameter. This interpretation appropriately reflects the symmetry of the estimator Err_{ICV}. As explained in Sections 2.3 and 3.1, Err_{WMC} tries to approximate this quantity, and thus also ICV, based on a multivariate normal distribution and the assumption in Eq. 1.6

1.6 Discussion and concluding remarks

In the context of error estimation through repeated subsampling, this chapter has introduced two variants of a new weighting-based method for correcting the tuning bias and the method selection bias by estimating the unconditional error rate of the corresponding wrapper algorithms as with ICV. Both of the methods avoid the additional computational costs of ICV while producing comparable results. The shrinkage-based variant, WMCS, addresses the optimistic tendency of WMC. WMCS yields the most accurate estimates in simulations and the best approximation to ICV on real data despite a slight tendency to over-correction. The correction method cannot only be applied in the well-known context of parameter tuning but also to address the method selection bias. Correction of the latter bias has seemingly never been addressed explicitly in the literature, neither with ICV nor with any other approach. It can also be suggested to extend ICV to the method selection setup, based on the idea that a "method" can – in a broad sense – be considered as a categorically scaled tuning parameter. There are however some differences between the tuning setup and the method selection setup. Quite generally, the bias is larger in the method selection setup than in the tuning setup. That is probably because in the tuning setup the error is expected to depend smoothly on the parameter value. Large differences between errors obtained with similar parameter values are unlikely. In contrast, in the selection setup methods do not have a natural ordering and may yield more contrasted errors.

Besides the lower computational effort, an important advantage of the method over ICV is that the obtained corrected error remains within reasonable bounds defined by the minimal and maximal errors. As shown in Section 1.4, ICV may produce estimates outside this interval. Regardless of whether the ICV-estimates fall above the highest error or below the lowest error, such a "correction" makes poor sense. The WMC correction method is clearly superior in such cases. Another extreme situation where the new method yields more plausible results is when all tuning parameter values/methods lead to very similar resampling error rates ($e(1 \parallel s) \approx \cdots \approx e(K \parallel s)$). In this case WMC and WMCS do not perform any correction, which is intuitively reasonable. On the contrary, ICV may produce a different corrected error. Whereas the results of the new weighted mean correction are deterministic (once the outer learning sets are fixed), ICV depends on the specific choice of the internal learning sets when selecting the L_b-best method $k^*(L_b)$. This aspect of ICV is consistent with its main idea of mimicking the selection or tuning process on each learning set of the resampling approach. However, by this dependence, ICV suffers from another source of variability which is difficult to correct within a reasonable time.

In contrast to ICV, WMC and WMCS directly use the information on the correlation

between the errors of different parameter values/methods, which allows an assessment of the "effective cardinality" of the pool of parameter values/methods. Obviously, the potential for tuning or method selection bias increases with the number K of parameter values/methods. However, if they are all very similar the bias is not expected to increase dramatically. The new method automatically takes into account correlation between errors including such highly correlated "blocks" of similar parameter values/methods. Another practical advantage over ICV is that WMC and WMCS can be applied "a posteriori" as long as one has used the same training sets for all classifiers and saved all fold errors $(e(k||L_b, S \setminus L_b), \forall b, k)$. With ICV the whole procedure has to be performed again if the classifier pool or the tuning grid is changed or enlarged.

Next, the small optimistic bias of WMC and the even smaller pessimistic bias of WMCS has to be discussed. The WMC procedure is based on a number of assumptions and possibly biased estimation steps. On the one hand, if assumption (1.6) is violated the method can be expected to be conservative i.e. to over-correct the error, because a method chosen by internal CV based on a specific dataset is expected to perform better rather than worse when applied to this dataset, yielding $\mathbf{E}_{P_{n_L}}[\varepsilon(k \parallel S)|k^*(S) = k] \leq \mathbf{E}_{P_{n_L}}[\varepsilon(k \parallel S)]$. On the other hand there is the optimistic bias which is induced by the biased mean estimate $\hat{\mu}$. This bias is corrected by the shrinkage approach in WMCS. However, WMCS sometimes over-corrects the optimistic tendency of WMC especially in setups where ξ is close to 0.5 or 0.6. In these particular cases, WMC often performs better than WMCS. In the present context of bias correction, however, the optimistic bias of WMC epitomizes a clear disadvantage and it is not recommended to apply it in practice. The pessimistic bias of WMCS is definitely more acceptable, the more so as it is substantially smaller on average. In the tuning setup, the WMCS method often produces results very similar to the raw mean. In future work the methods have to be further assessed and refined in this context. Moreover, the method could also be extended to the case where the tuning parameter is the number of variables used by a specific method. This setup can be considered an intermediate case between selection setup and tuning setup.

Finally, one needs to discuss the shift of focus from the conditional error rate of a classifier to the unconditional error rate of wrapper algorithms, which have been introduced in Varma and Simon (2006) and used as a basis by the new approach. On the one hand, one can certainly argue that the ultimate quantity of interest for a physician is the conditional error rate of the eventually constructed prediction rule. In this context, ICV and the WMC variants can only provide a "surrogate estimate" based on the assumption that Err is usually close to this conditional error rate. Especially in the case of method selection setups, this assumption may be violated because the models constructed in the respective ICV iterations can be substantially different from the model constructed on the whole dataset. However, even in this case, Err can be an informative quantity if one is more interested in the general utility of the data at hand. In pilot studies meant as proof of concept, Err can provide a more realistic picture of the signal in the data because it considers the high variability of the model construction procedure of wrapper algorithms. Considering the conditional error rate of a single model only may lead to highly variable

conclusions.

On the other hand, from a statistician's point of view, the unconditional error rate of the wrapper algorithm, Err, can be an interesting quantity for its own sake. If several wrapper algorithms have to be compared, it is recommendable to compare them on the basis of Err and not of the conditional error rate. For this purpose, Err has clear advantages over the conditional error rate because it indeed reports the performance of the underlying wrapper algorithm whereas the conditional error rate is the performance of a single prediction rule only. At least on simulated data (where the corresponding quantities can be obtained via Monte Carlo simulations), one can also consider as a next step to include the variance of the conditional error rate of wrapper algorithms ($\mathbf{Var}_{P^{n_L}}\left[\varepsilon(\phi(S))\right] = \mathbf{Var}_{P^{n_L}}\left[\varepsilon(k^*(S) \parallel S)\right] = \mathbf{E}_{P^{n_L}}\left[\varepsilon(k^*(S) \parallel S) - Err\right]^2$) into the comparison in order to provide additional insight into the characteristics of the competing wrapper algorithms.

Chapter 2

Ranking High-Dimensional Wrapper Algorithms using multiple studies

The following chapter covers the ranking of wrapper algorithms as they have been introduced in Chapter 1. This time, however, the main focus is on a new, extended validation scheme, leave-one-in cross-study validation (CSV, Waldron et al. (2013)), which tries to account for several problems concerning the transfer of prediction rules from the training to a validation study. The performance estimates will be compared as well as the resulting rankings for classical cross-validation and CSV on experimental microarray data as well as on an extensive simulation study. Moreover, the section tries to find guidelines for the aggregation of the numerous performance estimates that are computed during the procedure of CSV for each wrapper algorithm. Finally, it is analyzed whether certain algorithms rank distinctly higher in CV than in CSV and may therefore be considered 'specialist algorithms' in contrast to 'generalist algorithms' which rank higher in CSV.

2.1 Introduction

Cross-validation and related resampling methods are *de facto* standard for ranking supervised learning algorithms. They allow estimation of prediction accuracy using subsets of data that have not been used to train the algorithms. This avoids over-optimistic accuracy estimates caused by "re-substitution," a property that has been carefully considered (Molinaro et al., 2005; Baek et al., 2009; Simon et al., 2011). It is common to evaluate algorithms and prediction models by estimating accuracy via cross-validation based on several datasets, with results summarized across datasets to rank algorithms (Demšar, 2006; Boulesteix, 2013). This approach recognizes possible variations in the performance of learning algorithms across studies. However, it is not fully consistent with the ultimate goal of providing accurate predictions for fully independent samples originating from different institutions and processed by different laboratories.

It has been observed that accuracy estimates of genomic prediction models based on independent validation data are often substantially inferior to cross-validation estimates

(Castaldi et al., 2011). In some cases this has been attributed to incorrect application of cross-validation; however even strictly performed cross-validation may not avoid over-optimism resulting from potentially unknown sources of heterogeneity across datasets. These include differences in design, acquisition, and ascertainment strategies (Simon et al., 2009), hidden biases, measured variables, technologies used for measurements, and populations studied. In addition, many genomics studies are affected by experimental batch effects (Baggerly et al., 2008; Leek et al., 2010). Quantifying these heterogeneities and predicting their impact on the performance of prediction algorithms is critical in the practical implementation of personalized medicine procedures that use genomic information.

There are potentially conflicting, but valid, perspectives on what constitutes a good learning algorithm. The first perspective is that a good learning algorithm should perform well when trained and applied to a single population and experimental setting, but it is not expected to perform well when the resulting model is applied to different populations and settings. Such an algorithm is called "*specialist*", in the sense that it can adapt and specialize to the population at hand. This is the mainstream perspective for assessing prediction algorithms and is consistent with validation procedures performed within studies (Molinaro et al., 2005; Baek et al., 2009; Simon et al., 2011). However, it does not reflect the reality that "samples of convenience" and uncontrolled specimen collection are the norm in genomic biomarker studies (Simon et al., 2009).

Another perspective shall be promoted here: a good learning algorithm should be "*generalist*", in the sense that it yields models that may not be optimal for the training population, that is likely not fully representative of the prediction problem at hand, but that perform reasonably well across populations or laboratories employing comparable but not identical methods. *Generalist* algorithms may be preferable in important settings, for instance when a researcher develops a model using samples from a highly controlled environment, but hopes the model to be applicable to other hospitals, labs, or more heterogeneous populations. Although concern has been expressed about the lack of independent validation of genomic prediction *models* (Subramanian and Simon, 2010; Micheel et al., 2012), computational scientists have not systematically adopted independent validation in the comparison of learning *algorithms*. Thus, the so-called "leave-one-dataset-in" cross-study validation will be promoted here in order to formalize the use of independent validation in the evaluation of learning algorithms. Through data-driven simulations and an example involving eight publicly available estrogen receptor-positive breast cancer microarray datasets, several established survival prediction algorithms are assessed using the proposed approach which will be compared to conventional cross-validation.

2.1.1 Notations and settings

Suppose one considers multiple datasets $i = 1, \ldots, I$ with sample sizes $N_1, \ldots, N_I$. Each observation s appears only in one dataset i (datasets do not overlap), and the corresponding record includes a primary outcome Y_i^s and a vector of predictor variables $\mathbf{X}_i^s$; $\mathbf{X}_i^s$ will be gene expression measurements. The goal is to compare the performance of different learning algorithms $k = 1, \ldots, K$ that generate prediction models for Y_i^s using $\mathbf{X}_i^s$. Here,

the primary outcome Y_i^s is a possibly censored survival time. One is interested in evaluating and ranking competing prediction methods $k = 1, \ldots, K$. Since the ranking may depend on the application field, the first step is to define the prediction task of interest. Here, the focus is on the prediction of survival time in breast cancer patients based on high-throughput gene expression measurements. The approach and the concept of cross-study validation, however, can be applied to any other type of response variable.

2.1.2 Algorithms considered

Six learning algorithms ($k = 1, \ldots, 6$) are assessed which are appropriate for high-dimensional continuous predictors and possibly censored survival time outcome: LASSO regression (Goeman, 2010), CoxBoost (Binder and Schumacher, 2008), SuperPC (Blair and Tibshirani, 2004), Unicox (Tibshirani, 2009), and PlusMinus (Zhao et al., 2013). The focus is not to provide a comprehensive array of algorithms, but simply to use a few popular, representative algorithms to study the properties of cross-study validation.

2.1.3 Ranking algorithms by cross-study validation: the CSV matrix

Here, k-fold cross-validation and related resampling methods are collectively denoted by cross-validation (CV).

The ranking procedure for learning algorithms is based on a squared matrix $\mathbf{Z}^k$ of scores ($k = 1, \ldots, K$), with the element in the i-th row and j-th column measuring how well the model produced by algorithm k trained on dataset i performs when validated on dataset j. Since K methods are considered here, one ends up with K method-specific squared matrices $\mathbf{Z}^1, \ldots, \mathbf{Z}^K$. The diagonal entries of the matrices are set equal to the performance estimates obtained with 4-fold CV in each dataset. $\mathbf{Z}^k$ is also called the cross-study validation matrix, or *CSV matrix*.

Possible definitions for the non-diagonal $\mathbf{Z}_{i,j}^k$ scores include the concordance index in survival analysis (Harrell et al., 1996; Gönen and Heller, 2005), which is used here, the area under the operating characteristic curve in binary classification problems, or the mean squared distance between predicted and observed values in regression problems. As an illustration, Figure 2.1a displays the *CSV* matrix of C-statistics obtained through validation of ridge regression models for the eight studies of Table 2.1.

2.1.4 Summarization of the CSV matrix

In order to rank learning algorithms $k = 1, \ldots, K$, each matrix $\mathbf{Z}^k$ must be summarized by a single score. Two candidate approaches are considered:

1) **Simple Average** $\frac{\sum_i \sum_{i \neq j} \mathbf{Z}_{i,j}^k}{I(I-1)}$ of all non-diagonal elements of the $\mathbf{Z}^k$-matrix.

2) **Median** or more generally $q-$**quantile** of the non-diagonal entries of $\mathbf{Z}^k$. Quantiles

offer robustness to outlier values, and the possibility to reduce the influence of uninformative studies where all algorithms perform poorly by selection of an appropriate quantile.

2.1.5 True global ranking

From a statistical perspective the score $\mathbf{Z}^k_{i,j}$ is a random variable. First, studies i and j can be seen as randomly drawn from a population of studies. Second, observations within each study can be considered as randomly drawn from the unknown and possibly different distributions F_i and F_j underlying studies i and j.

With this view of $\mathbf{Z}^k_{i,j}$ as random variable, one can consider the theoretical counterparts of the empirical aggregating scores (simple average and quantiles) described in Section 2.1.4 to summarize $\mathbf{Z}^k$. The theoretical counterparts are the expected value or quantiles of each $\mathbf{Z}^k_{i,j}$ score, $i \neq j$, obtained by integrating the two levels of randomness described above. The *true global* ranking of the learning algorithms $k = 1, \ldots, K$ is then defined by these expected values (or quantiles), one for each algorithm. The ranking is called *global* because it is not specific to the available studies.

The true global ranking can be considered as the estimation target of evaluation procedures such as CV or CSV. Section 2.1.7 presents the design of a data-driven simulation study in which the true ranking is obtained through Monte Carlo integration. Thus, one can evaluate and compare the ability of CV and CSV to recover the true global ranking.

2.1.6 Description of datasets

A compendium of breast cancer microarray studies is used which has been curated for the meta-analysis of Haibe-Kains et al. (2012) and is available as a supplement to that article. All datasets have been selected for which metastasis-free survival (DMFS), the most commonly available survival endpoint, as well as Estrogen Receptor (ER) status, were available, and which were generated with Affymetrix HGU GeneChips HG-U133A, HG-U133B and HG-U133PLUS2. Exclusively, ER-positive tumors have been considered. Of the remaining 8 datasets (Table 2.1), only one originated from a population-based cohort (Schmidt et al., 2008). Four studies considered only patients who did not receive hormone therapy or chemotherapy adjuvant treatment. Only four provided date ranges of patient recruitment Foekens et al. (2006); Desmedt et al. (2007); Schmidt et al. (2008); Chin et al. (2006). This variability in design strategies and reporting, and cohort differences in survival that are not easily explicable (Table 2.1, column 3Q survival) highlight the prevalence of "samples of convenience" in biomarker studies discussed by Simon et al. (2009).

Samples from dataset ST1 duplicated in dataset VDX were removed. Expression of each gene was summarized using the probeset with maximum mean (Miller et al., 2011). The 50% of genes with lowest variance were removed. Subsequently, gene expression values were scaled by linear scaling of the 2.5% and 97.5% quantiles as described by Haibe-Kains et al. (2012).

No.	Name	Adjuvant therapy	number of patients	number of ER+	3Q survival [*mo.*]	Median follow-up [*mo.*]	Original identifiers ‡	Reference
1	CAL	chemo, hormonal	118	75	42	82	CAL	Chin et al. (2006)
2	MNZ	none	200	162	120	94	MAINZ	Schmidt et al. (2008)
3	MSK	combination	99	57	76	82	MSK	Minn et al. (2005)
4	ST1	hormonal	512*	507*	114	106	MDA5, TAM, VDX3	Foekens et al. (2006)
5	ST2	hormonal	517	325	126	121	EXPO, TAM	Symmans et al. (2010)
6	TRB	none	198	134	143	171	TRANSBIG	Desmedt et al. (2007)
7	UNT	none	133	86	151	105	UNT	Sotiriou et al. (2006)
8	VDX	none	344	209	44	107	VDX	Minn et al. (2007)

Table 2.1: Public microarray datasets of breast cancer patients as curated and summarized by Haibe-Kains et al. (2012). Datasets are referred to using the following acronyms: CAL = University of California, San Francisco and the California Pacific Medical Center (United States), MNZ = Mainz hospital (Germany). MSK = Memorial Sloan-Kettering (United States), ST1, ST2 are meta-datasets as provided by Haibe-Kains et al. (2012), therein named SUPERTAM1 and SUPERTAM2. TRB = dataset collected by the TransBIG consortium (Europe), UNT = cohort of untreated patients from the Oxford Radcliffe (United Kingdom), VDX = Veridex (the Netherlands). **# ER+** refers to the number of patients with the Estrogen Receptor positive subtype. **3Q survival** provides the Kaplan-Meier estimate of 75% survival probability for each cohort. **Median follow-up** is calculated from the reverse Kaplan-Meier estimate. *Numbers shown are after removal of samples duplicated in the dataset VDX. ‡ Dataset identifier(s) as specified in Haibe-Kains et al. (2012).

2.1.7 Simulation design

Heterogeneous datasets from a joint probability model with survival outcomes are simulated. The probability model is defined by a resampling procedure that is applied to the eight breast cancer datasets in Table 2.1. The resampling scheme is a combination of parametric and nonparametric bootstrap (Efron and Tibshirani, 1993; Bender et al., 2005). The goal of the simulation study is to compare CV and CSV when used for evaluation and ranking of competing learning algorithms, in synthetic data that realistically simulate multiple independent datasets, where the true relationship between the independent and dependent variables is known. CV and CSV are then assessed with respect to their ability to recover the true global ranking, which is computed through Monte-Carlo integration. The ability to recover the ranking is assessed by Kendall correlation between the true global ranking and the CV or CSV estimates.

For $b = 1, \ldots, B = 1000$ iterations, a collection of $I = 8$ datasets is generated as follows. First, 8 studies are sampled with replacement from the list of breast cancer studies. In other words, a collection of studies is simulated in order to mimic the fact that studies are not considered as fixed but rather drawn from a population. This step only involves simulations from a multinomial Mult$(8, [1/8, \ldots, 1/8])$ distribution. Second, for each of the generated studies, $N = 150$ patients are sampled from the corresponding original dataset with replacement. Each of the 150 predictor vectors is directly generated from a study-specific empirical distribution (non-parametric bootstrap). Finally, the corresponding survival times are simulated from a proportional hazards model (parametric bootstrap) fitted to one of the available studies:

$$M^i_{true} : \lambda^i(t|x) = \lambda^i_0(t) \times exp(x^T \boldsymbol{\beta}_i), \tag{2.1}$$

$i = 1, \ldots, I$, where $\lambda^i(t|x)$ is the individual hazard function when the vector of predictors is equal to x and $\boldsymbol{\beta}_i$ denotes a vector of regression coefficients. The truncated inversion method in Bender et al. (2005) and the Nelson-Aalen estimator for cumulative hazard functions are combined to simulate survival times that reflect survival distributions and follow-up of the real studies. The vector $\boldsymbol{\beta}_i$ is set identical to the coefficients fitted in study $i = 1, \ldots, I$ using the *CoxBoost* method (Binder and Schumacher, 2008). Note that a different regression method could have been used at this stage.

The collections of simulated datasets are then used both (i) to compute by Monte Carlo method the true global ranking defined in Section 2.1.5, and (ii) to compute estimated ranks by CV and CSV.

Figure 2.1a displays, for each pair of studies (i, j) in Table 2.1, C-index obtained when training a model by ridge regression on dataset i (rows), and validating that model on dataset j (columns). Diagonal elements $(i = j)$ are obtained by 4-fold CV. Figure 2.1b displays mean C-indices for each (i, j) combination across simulations, when the training and validation studies are generated resampling the i-th and j-th study. The diagonal elements are computed by averaging C-indices with the training and validation datasets independently generated by resampling from the same study.

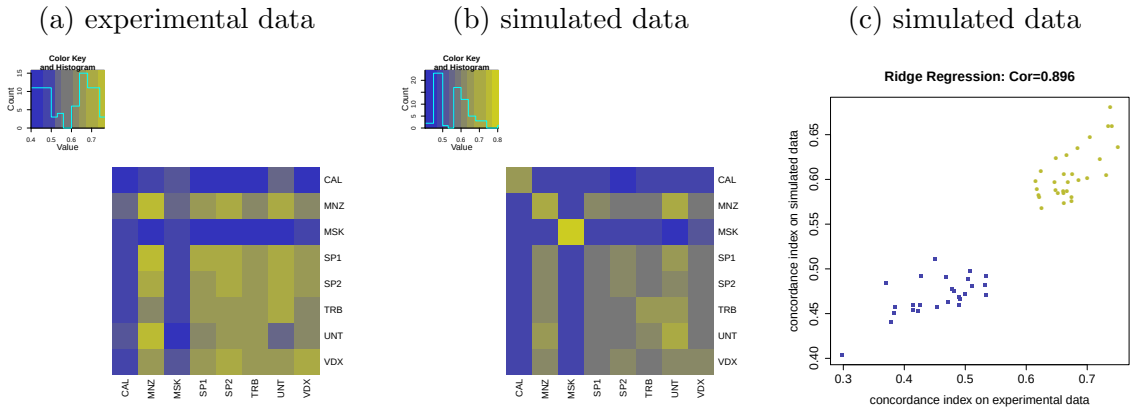


Figure 2.1: Cross-study validation matrices $\mathbf{Z}^k$ in simulated and experimental data for Ridge Regression. Panel (a) displays C-indices for training and validation on each pair of actual datasets in Table 2.1. The diagonal of this matrix shows estimates obtained through 4-fold CV. The heatmap in panel (b) displays, for each pair of studies (i, j), the average C-index obtained when ridge regression is fit on simulated datasets generated by resampling gene expression data from the i-th study in Table 2.1 by non-parametric bootstrap and simulating censored survival outcome by parametric bootstrap; the resulting model is validated on a simulated dataset generated by resampling study j. Two independent datasets from the same study are sampled for the diagonal elements. Both heatmaps strongly resemble each other, indicating a realistic simulation scenario. CAL and MSK are outlier studies: cross-study C-index is approximately 0.5 when they are used either for training or validation. All values of the Z-matrix corresponding to these two studies build the blue "bad performance" cluster in panel (c) which compares the C-indices obtained for study pairs (i, j), $i \neq j$, on simulated data (y-axis) and experimental data (x-axis). Pearson correlation is ≈ 0.9. The three plots illustrate high similarity between simulated and real data in the application.

The similarity between the two panels is striking, in particular with respect to the clear separation of the eight studies into two groups. The first group includes the studies MNZ, ST1, ST2, TRP, UNT and VDX, and seems to produce more accurate prediction models than the remaining studies. The datasets in this group seem also associated with higher values of the concordance index when used for validation. This difference between the two groups is also illustrated in Figure 2.1c. It displays the non-diagonal entries of the matrices represented in the left and middle panels, i.e. average C-indices from simulated datasets vs. C-indices from real data. This scatterplot shows a clear two-cluster structure: the yellow dots display the 30 training and validation combinations within the larger group of studies, i.e MNZ, ST1, ST2, TRP, UNT and VDX.

2.1.8 Further inspection of the implemented simulation scenario

In order to investigate the appropriateness of the simulated data, it is also necessary to perform several validity checks on the models that have been fitted on the real data sets and are used as blueprints for the simulated data. Figure 2.2 shows some of these checks for the CoxBoost model that has been fitted on the data set ST1. The upper row shows two plots representing the cumulative hazard of the real and estimated survival and censoring times respectively. The gray line corresponds to the Nelson-Aalen estimate of the cumulative base-line hazard on the real data set based on the true (CoxBoost) model that has been fitted on this data set. On the contrary, the black line corresponds to the Nelson-Aalen estimate that has been obtained on 10000 observations which have been simulated from the model. In the case of the survival times, the estimate is based on the 'true' linear predictor which is taken from the corresponding model fitted on the real data. The lines almost coincide which represents a good evidence that the simulated data behave as expected and that the truncation of censoring and survival times does not have a strong influence on the simulated observation in the relevant time frame.

In the middle, two plots illustrating the vector of coefficients and the distribution of the linear predictor are presented. It is important that the vector of coefficients represents a realistic gene profile consisting of several non-zero coefficients which do not have too strong an impact. Otherwise, the algorithms cannot be expected to correctly estimate these coefficients. The distribution of the linear predictor is almost centered around zero due to the transformation of the gene expression variables. Moreover, values range between -3 and 4 in this case. Extreme values of the linear predictor are numerically problematic since they are the argument of an exponential function during the simulation process.

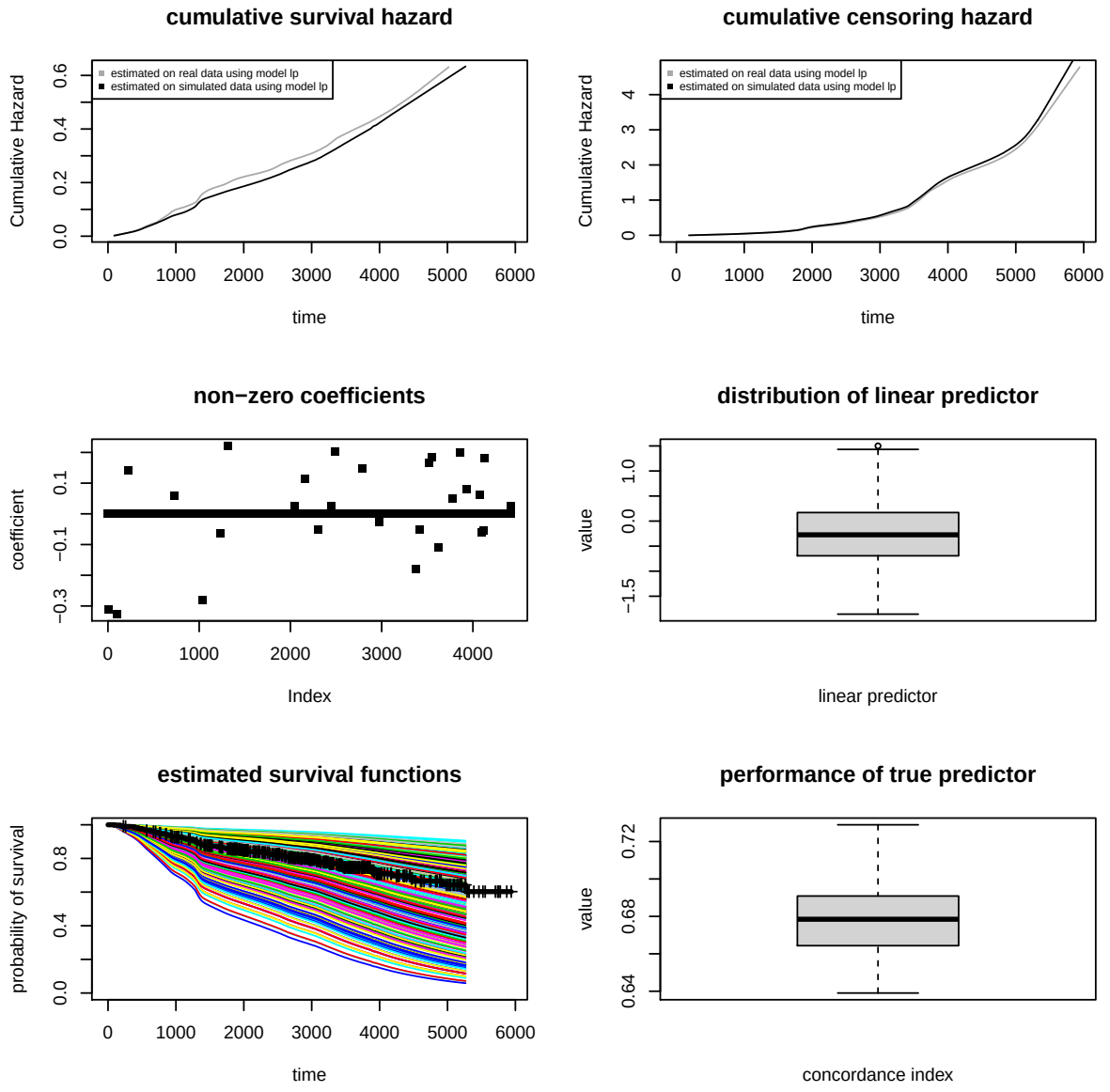


Figure 2.2: Validity checks for the CoxBoost model fitted on data set ST1. One can see that the cumulative hazard for survival times as well as for the censoring times can be recovered from the simulated data very accurately. This means that their Nelson-Aalen-estimate (under the usage of the true linear predictor) obtained on a large set of simulated data almost coincides with the 'true' cumulative hazards (see two top panels), i.e. the simulated data indeed follow the true model. The plots in the middle panels illustrate the coefficients as estimated by *CoxBoost* as well as the resulting distribution of the linear predictor on the original ST1 data set. The plot in the left bottom panel shows the survival curves estimated by the model and the empirical Kaplan-Meier estimate. Finally, the right panel in the bottom row describes the distribution of the C-indices the true coefficients achieve on data sets of size 1000 on the simulated data. In summary, these plots show that the simulated data are similar to the original ones and do not feature problematic, numerical artifacts, e.g. extremely large values for the linear predictor.

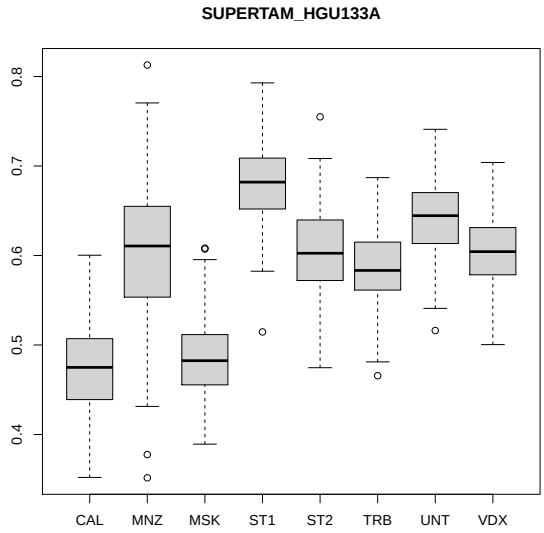


Figure 2.3: Simulated concordance indices on the $I = 8$ studies for the 'true' (CoxBoost) coefficients fitted on the data set ST1. These distributions represent the potential C-Indices an algorithm can achieve on data sets (n=150) of the respective validation studies if it exactly recovers the the 'true' coefficients vector of the true model on data set ST1 correctly.

Values as high as 20 were observed in the case of the Unicox algorithm which usually result in numerical artifacts in the simulation, even if they are theoretically correct in combination with an extremely small estimate of the cumulative base line hazard. If they exist, these numerical problems can be best observed in the plot on the left side in the last row which depicts the fitted survival curves, i.e.:

$$\mathcal{S}(t|\mathbf{x}_i^s) = \exp\left(-\exp(\mathbf{x}_i^s \beta_T^i) \int_0^t \lambda_0^i(t)\, dt\right), \tag{2.2}$$

where β_T^i is the coefficient estimated by the CoxBoost model. The plot shows that the model estimates relatively variable survival curves for the different patients which is necessary if one wants the evaluated algorithms to be able to achieve good concordance indices on the simulated data. As another validity check, the Kaplan-Meier estimate of the overall survival is depicted in black. This estimate is right in the middle of the fitted survival curves as it is expected for a realistic model. The last plot shows a boxplot of the concordance indices that can be obtained on an independent simulated data set of size 1000 if an algorithm is able to find the correct true coefficients. In this case, the median is approximately at 0.68. This value refers to a good but not unrealistic discrimination performance.

This approach can also be repeated for the case where the true coefficients of the model on ST1 are evaluated by their concordance index on large simulated data sets simulated

from the models fitted on the other studies. One can see that for several data sets even the true coefficients do not achieve a C-Index higher than 0.5 (see Figure 2.3), i.e. they are just as bad as a coefficient vector of zeros. For the MNZ, TRB, UNT and VDX dataset, however, they clearly provide evidence that cross-study-prediction is possible for the simulated data. The data sets for which cross-study prediction seems to be possible, coincide with the data sets for which cross-study prediction worked well on the real data sets (see Figures 2.5a and 2.5b).

Implementation of the algorithms

All the considered methods are used as wrapper algorithms and are tuned using cross-validation on the simulated taining sets as implemented in the package *survHD*. For most algorithms standard tuning grids are used, whereas the *Glmnet* implementation for Ridge and Lasso Cox-regression applies an inbuilt tuning algorithm. Apart from the removal of the low variance gene expressions, no further variable selection is applied, especially no supervised variable selection method. In this context, it is worth mentioning that the focus lies more on the illustration of the concept of ranking algorithms using cross-study-validation. Since the algorithms have to be fitted several thousands of times during the simulation, the different algorithms are tuned as well as possible in a reasonable amount of time. One certainly cannot guarantee that the performance of the algorithms reported here is the absolutely optimal performance the respective algorithm might achieve. The performance of each algorithm could probably be further enhanced by investing more time into a more detailed tuning or combining the algorithm with a supervised variable selection technique.

Variability over the iterations

The CSV results can be expected to be highly variable as can be seen from the heatmaps on the real data. An important aspect is the presence of outlier studies for which cross-study-validation does not seem to work properly. Consequently, even for good algorithms there will be several C-indices close to 0.5 when their prediction rules are evaluated on the other studies. Thus, the question of how the values of the **Z**-matrix should be aggregated gains in importance. In order to illustrate this aspect, one can have a look at the iteration-wise averaged C-Indices for the first 50 iterations of the six algorithms (see Figure 2.4).

One can see several spikes in the curves of the different algorithms, at which the performance of almost all algorithms is distinctly higher. This is mainly caused by the bootstrapping of the studies. Studies can occur several times in a simulated **Z**-matrix which means that in some **Z**-matrices there are more of the good studies whereas other **Z**-matrices are based on more of the outlier studies. This portion of good studies basically dominates the curves. Due to the high variability between good and outlier studies the average C-index is not expected to be the best criterion.

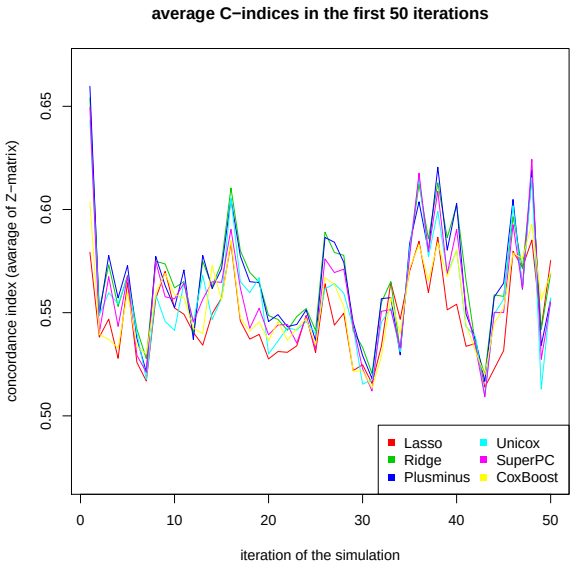


Figure 2.4: Average concordance indices of the $\mathbf{Z}^k$-matrices for the first 50 iterations of the simulation ($n = 150$). The lines correspond to the $K = 6$ considered algorithms. The spikes, where performances are distinctly higher for all algorithms, correspond to iterations with a higher portion of 'good' studies.

2.1.9 Evaluation criteria in simulations

The simulation study can assess and rank learning algorithms based on their ability to recover the true underlying models M^i_{true}. This section introduces a criterion that reflects the similarity between the true regression coefficients $\boldsymbol{\beta}_i$, that were used to simulate the i-th dataset and the coefficients $\hat{\boldsymbol{\beta}}^{(k)}_j$ fitted on dataset j. Pairs of studies with $i \neq j$ and pairs with $i = j$ are considered separately. The standard way to assess similarity between vectors is to compute the euclidean distance between them. However, since the focus is on prediction the alternative criterion $\widehat{\text{cor}}(\mathbf{X}_i\boldsymbol{\beta}_i, \mathbf{X}_i\hat{\boldsymbol{\beta}}^{(k)}_j)$ is considered in order to measure the similarity between the true $\boldsymbol{\beta}_i$ and fitted regression coefficients $\hat{\boldsymbol{\beta}}^{(k)}_j$. Here $\mathbf{X}_i$ is the matrix of predictors of dataset i and $\widehat{\text{cor}}$ denotes the empirical Pearson's correlation. The average

$$S^k_{self} = (1/I) \cdot \sum_i \widehat{\text{cor}} \left(\mathbf{X}_i\boldsymbol{\beta}_i, \mathbf{X}_i\widehat{\boldsymbol{\beta}}^{(k)}_i \right), \tag{2.3}$$

over the I studies, provides a measure of the ability of learning algorithm k to recover the model that has generated the training dataset, hence the index *self*.

Another criterion of interest is the ability of a learning algorithm to recover the vector of regression coefficients $\boldsymbol{\beta}_i$ when one relaxes the assumption that the unknown models

underlying training and validation datasets coincide. This can be quantified with

$$S^k_{across} = (1/(I(I-1))) \cdot \sum_i \sum_{j \neq i} \widehat{\text{cor}} \left(\mathbf{X}_i \boldsymbol{\beta}_i, \mathbf{X}_i \widehat{\boldsymbol{\beta}}^{(k)}_j \right), \tag{2.4}$$

where the index *across* emphasizes the focus on cross-study similarity, i.e. on the ability of algorithm k to recover the coefficients $\boldsymbol{\beta}_i$ when fitted on dataset j, with $j \neq i$. Instead of taking simple averages in Eqs. (2.3-2.4), one could also use different summaries, e.g. median or quantiles, as presented in Section 2.1.4.

Both S^k_{self} and S^k_{across} are criteria to assess and compare learning algorithms. The ranking obtained by ordering the algorithms according to their value of S_{self} (S_{across}) are denoted by R_{self} (R_{across}). Note, however, that S^k_{self} and S^k_{across} are by definition specific to the considered collection of studies and datasets: they involve the vectors $\boldsymbol{\beta}_i$ and the matrices $\boldsymbol{X}_i$ ($i = 1, \ldots, I$). The corresponding rankings are called *local* because they are specific to the collection of datasets at hand.

2.2 Results

2.2.1 Simulated Data

The focus in the simulation study is on differences between the rankings and performance estimates obtained by CV and CSV. Figure 2.5a shows the distributions of $\overline{CSV}$ and $\overline{CV}$, and Figure 2.5b shows the distribution of the rankings, across 1000 simulated 8-dataset compendia. Table 2.2 shows the median of these rank distributions, along with true global median ranks. The rank of method k is 1 if it yields the largest score of the K training algorithms. One can observe large differences in the distributions of $\overline{CSV}$ and $\overline{CV}$ across simulations (Figure 2.1a): the average of the $\overline{CV}$ scores is around 0.65, while $\overline{CSV}$ scores are centered around 0.55. The variability of $\overline{CV}$ and $\overline{CSV}$ across simulations, however, is comparable.

Performance differences across algorithms, whether estimated by CV or CSV, are relatively small compared to the overall difference between CV and CSV performance estimates. One can also observe differences between the rank distributions produced by CV and CSV. By both CV and CSV, *Glmnetlasso* is ranked as one of the worst performing algorithms, while *Glmnetridge* and *Plusminus* are ranked first or second. However the consistent advantage of *Glmnetridge* over *Plusminus* in $\overline{CV}$ vanishes under $\overline{CSV}$. The median rank of CoxBoost across simulations is two positions better as estimated by $\overline{CV}$ than by $\overline{CSV}$; in this case CSV is more consistent with the global true rankings (Table 2.2). Note that the exchange in top global true ranking between *Glmnetridge* over *Plusminus* when one considers average and median summaries (see section 2.1.4 and criterion in Table 2.2) occurs because the global true performance of these two algorithms is nearly indistinguishable.

The true *local rankings* of the $K = 6$ algorithms, defined by S^k_{across} or S^k_{self} in Section 2.1.9 vary over the 1000 simulated collections of studies. Furthermore, the median Kendall's correlation between R^k_{across} and R^k_{self} amounts to approximately 0.5, i.e. the

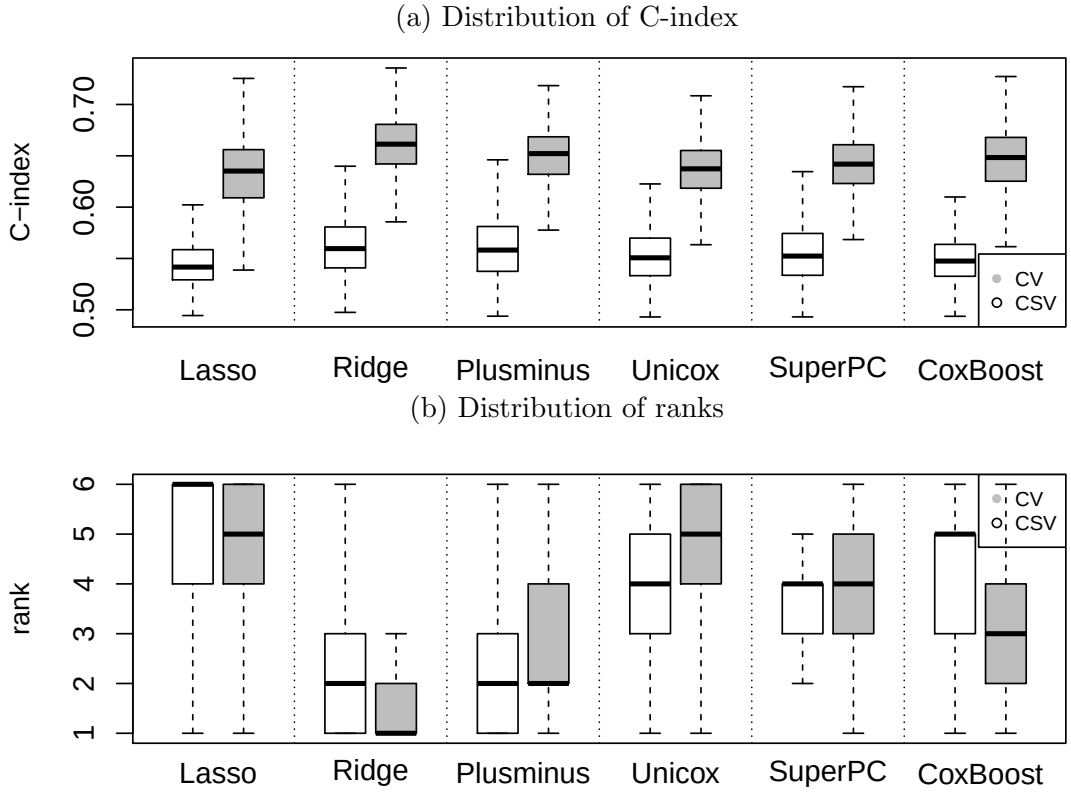


Figure 2.5: Comparison of cross-study validation ($\overline{CSV}_k$) and cross-validation ($\overline{CV}_k$) on simulated data. Each boxplot represents evaluations of $K = 6$ algorithms in 1000 simulations of a compendium of $I = 8$ datasets. For each simulation the diagonal and off-diagonal elements of the Z^k matrix of validation C-statistics are summarized by (a) simple mean and (b) rank of the simple mean across algorithms. CV estimates of the C-index are much higher (approximately increased by 0.1) than CSV estimates. Lasso is ranked worst by both CV and CSV and Ridge / Plusminus are ranked best, however some differences are apparent. CoxBoost ranks two positions worse in CSV, and the distinct advantage seen in CV for Ridge over Plusminus vanishes in CSV.

local performance measures S^k_{across} and S^k_{self} define distinctly different rankings (see Figure B.1 in the appendix). A natural question is how well $\overline{CV}_k$ and $\overline{CSV}_k$ can recover the unknown rankings R^k_{across} and R^k_{self}. The boxplots in Figure 2.6 display the Kendall's correlation between local rankings R_{across} (a) or R_{self} (b) and the ranking found by $\overline{CV}$ (gray boxes) or $\overline{CSV}$ (white boxes) across simulations. Figure 2.6(c) shows the same for true global ranking. Both local rankings (R_{across} and R_{self}) can be recovered by $\overline{CSV}$ with a Kendall correlation around 0.5. $\overline{CV}$ tends to be less correlated with R_{across}. This pattern is reversed for the local ranking R_{self}. Recall that, similarly to $\overline{CV}$, R_{self} is defined in terms of within-study validation. However, the difference between the medians in the second panel is less pronounced than in the first one. Finally, $\overline{CSV}$ features a con-

siderably higher correlation to the true global ranking. This suggests that $\overline{CSV}$ is more suitable for recovering the global ranking. If the two outlier studies (CAL and MSK) are removed, the advantage of $\overline{CSV}$ over $\overline{CV}$ in recovering true global ranking is further increased (median Kendall correlation: 0.8 vs. 0.6, see Figures B.2- B.4 in the appendix), and also surpasses $\overline{CV}$ for recovering true local self ranking R_{self}. Overall, as displayed by the Figure B.3 in the appendix, it appears that, after outlier studies are removed, CSV outperforms substantially CV when used for ranking algorithms.

Algorithm	global true ranking		CSV (med. ranks)		CV (med. ranks)	
Glmnetridge	1	2	2	2	1	2
Plusminus	2	1	2	2	2	2
Superpc	3	3	4	3	4	4
Unicox	4	4	4	4	5	4
CoxBoost	5	5	5	5	3	4
Glmnetlasso	6	6	6	6	5	6
criterion	Av.	Med.	Av.	Med.	Av.	Med.

Table 2.2: True global ranking and median rank estimate of CV and CSV on simulated data. Ranks shown for CV and CSV are the median across 1000 simulations; individual columns refer to summarization of Z_{ij}^k by Average or Median value. Third quartile ranks are the same as the median ranks, these are not shown. All methods rank GLMnetridge and Plusminus well, and GLMnetlasso poorly; however CSV ranks with Z_{ij}^k summarized using the median are the closest to the true global ranking. Variability of CV and CSV rank estimates across simulations is shown in Figure 2.5b.

2.2.2 Application to breast cancer prognostic modelling

This section applies CV and CSV to the $I = 8$ breast cancer studies described in Section 2. Generally, the results resemble those obtained on simulated data. The top panel in Figure 2.7 illustrates the distributions of $\overline{CSV}$ and $\overline{CV}$ for each of the $K = 6$ algorithms. Except for the distinctly larger interquartile ranges of the boxes, the same patterns are observed as in Figure 2.5. Note that in these boxplots an observation represents a single entry of the $\mathbf{Z}^k$-matrix, whereas in Figure 2.5 each box represents the distribution across iterations of the average of the $\mathbf{Z}^k$-matrix. This explains the higher variance observed in Figure 2.7. One can observe in Figure 2.7 that:

- $\overline{CV}$ estimates are approximately 0.06 higher than $\overline{CSV}$ estimates on the C-index scale. To illustrate the magnitude of this improvement on the C-index scale consider a population with two groups of patients, high and low risk patients, covering identical proportions 0.5 of the population. A *perfect discrimination model* that correctly recognizes the subpopulation of each individual, when the hazard ratio between high

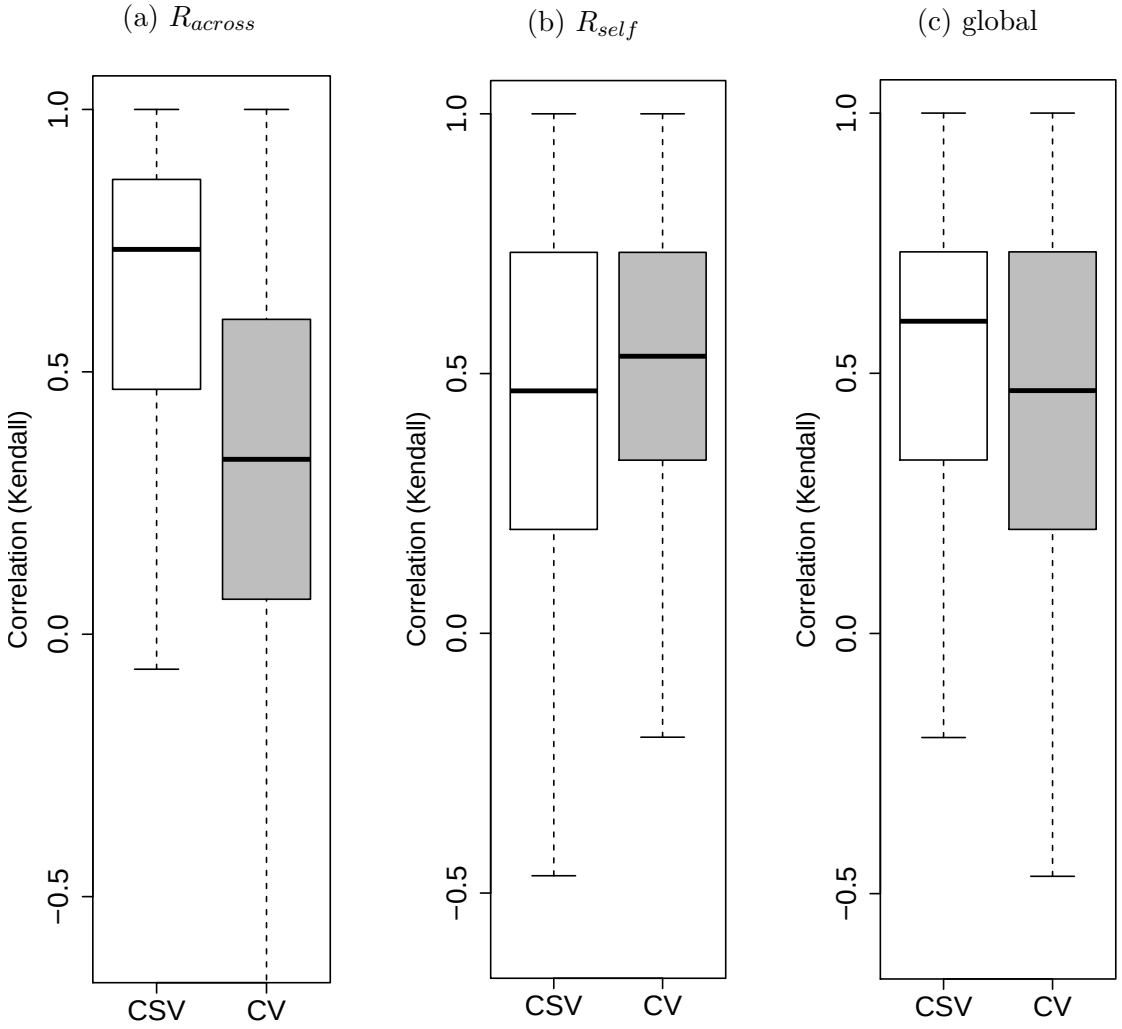


Figure 2.6: Distribution of Kendall's correlation between true algorithm rankings and rankings estimated by $\overline{CSV}$ (cross-study validation, white) and $\overline{CV}$ (cross-validation, gray) on simulated data. Panels (a) and (b) compare CV and CSV in terms of their correlation to the *local rankings* (true rankings for each simulation iteration); (c) compares these to the global ranking based on all iterations. R_{across} and R_{self} are the ranks calculated from S_{across} and S_{self}, respectively. Each box represents correlations computed in each of the 1000 iterations of the simulation study. $\overline{CSV}$ achieves higher correlations, and lower variance, to the global ranking and this improvement is more pronounced for R_{across}. For R_{self} (within-study validation), this pattern is reversed as expected, although the difference between medians is smaller. Thus in simulated independent datasets, $\overline{CSV}$ recovers true across-study rankings more accurately and with less variability than $\overline{CV}$.

versus low risk patients is 2.7, achieves on average a C-index of 0.62. For the average C-index of the perfect discrimination model to be increased to 0.68, a doubling of the true hazard ratio to 5.4 is necessary. Thus it is fair to characterize the validation

results seen here by CV as much more optimistic than as seen by CSV.

- The presence of outlier studies (CAL and MSK, see Section 2.3 for a brief discussion on their specific characteristics) has a strong effect on the ranking estimates when one uses the mean to summarize $\mathbf{Z}^k$ matrices. Using mean summarization, both $\overline{CSV}$ and $\overline{CV}$ rank *Superpc* first. This is mainly caused by high variability around the $C = 0.5$ of the $\mathbf{Z}^k_{i,j}$ validation scores obtained by models trained by outlier studies. In particular, *Superpc* and *Unicox* are the only algorithms that produce models with good discrimination when trained on the MSK study. With median summarization, ranking estimates are less influenced by the presence or absence of outlier studies. Therefore, the use of the median can be recommended in order to summarize $\mathbf{Z}^k$ matrices.

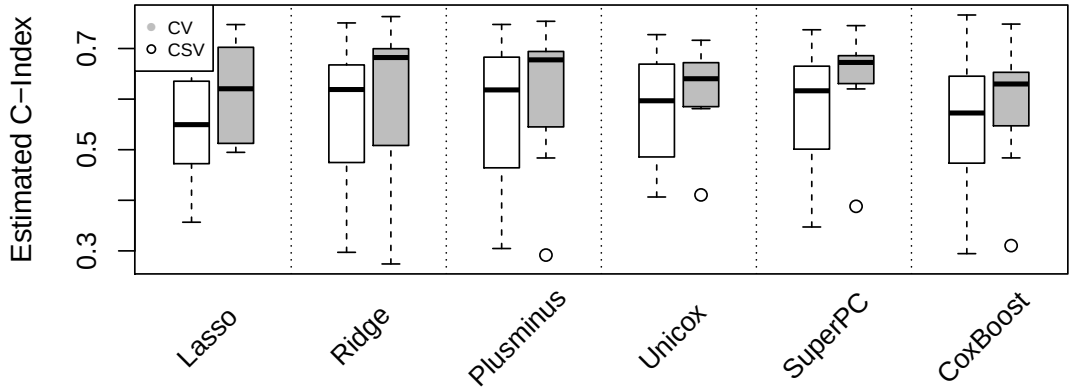
- Using median aggregation of the $\mathbf{Z}^k_{ij}$ scores, the ranking defined by $\overline{CSV}$ is identical to the ranking found in the simulation example (cf. Table B.1 in the appendix and Table 2.2), reflecting the realistic simulation scenario and the robustness of median aggregation. For both median and third quartile aggregation the rankings defined by $\overline{CV}$ and $\overline{CSV}$ differ substantially (Kendall's correlations 0.6 and 0.07). This is consistent with results of the simulation study, where median correlation of the rankings defined by $\overline{CSV}$ and $\overline{CV}$ was approximately 0.5.

- Figure 2.7b illustrates that CSV performance estimates do not necessarily reflect CV results. This panel shows $\overline{CV}$ and $\overline{CSV}$ estimates for *Glmnetridge* averaged over all datasets combinations, with a fixed study used for training (black) or validation (gray). Cross-validation statistics are only moderately correlated with CSV statistics for models trained from the same dataset ($\rho = 0.2$), and negatively correlated with CSV statistics when that dataset is used for validation ($\rho = -0.33$).

2.2.3 $\overline{CV}$ performance is not necessarily indicative of cross-study performance

A more detailed analysis of the correlation between $\overline{CSV}$ and $\overline{CV}$ shows that cross-study prediction and within-study prediction are indeed less related than one might expect. In Figure 2.7b, study specific values for $\overline{CSV}$ are plotted against $\overline{CV}$ for the learning algorithm *Glmnetridge* as an example (outlier studies removed). For each study one has a single value for CV but two values for CSV depending whether one averages the $\mathbf{Z}$-matrix column-wise (identical validation study) or row-wise (identical training study). The correlations vary between algorithms but their values are usually around 0.5 and in the latter case (identical validation study) mostly negative. This phenomenon could be caused by 'study-specific' signals. Such study-specific signals may support within study prediction as tested in CV but will impede cross-study prediction. From this point of view, cross-study and within-study prediction can be considered as two different types of problems with different characteristics.

(a) CSV vs. CV



(b) Glmnetridge:

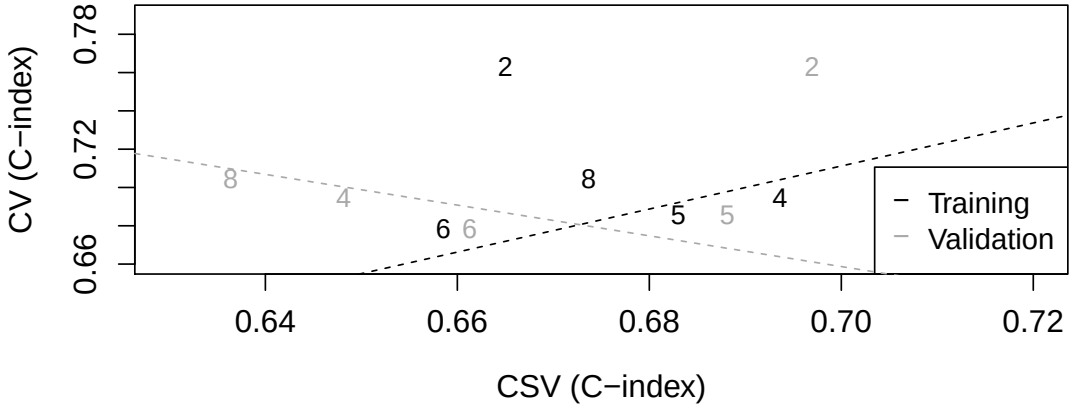


Figure 2.7: Comparison of $\overline{CSV}$ and $\overline{CV}$ on experimental data with two outlier datasets (MSK and CAL) removed. Panel (a) describes the distributions of $\overline{CSV}$ and $\overline{CV}$ estimates separately for each of the six considered algorithms. In contrast to Figure 2.5a, each boxplot represents the distribution of the values of a single matrix $\mathbf{Z}^k$ as depicted in Figure 2.1b. The pattern is similar to the one obtained on the simulated data. $\overline{CV}$ leads to substantially higher estimates of discrimination performance compared to $\overline{CSV}$. Panel (b) illustrates the relation between $\overline{CV}$ and $\overline{CSV}$ estimates by averaging over all datasets combinations with a fixed study used for training (black) or validation (gray). The illustration shows results for the learning algorithm *Glmnetridge* and the numbers refer to the study number given in Table 2.1 (outliers CAL and MSK have been removed). Cross-validation statistics on the y-axis are moderately correlated to CSV (row wise means [fixed training study] or column-wise means [fixed validation study] of the non-diagonal elements in the corresponding $\mathbf{Z}^k$-matrix) on the x-axis. For *Glmnetridge* the correlation is 0.2 when one plots CV versus the row wise means and -0.33 when one considers the column wise means. These correlations vary over the K algorithms, but the same pattern is predominant.

Finally, it should be mentioned that $\overline{CV}$ is a less suitable method for the detection of outlier studies, since it can estimate relatively good prediction performances even on studies where all algorithms fail in cross-study validation. In the example, this occurred for the MSK study with the algorithms *Superpc* and *Unicox*.

2.2.4 Specialist and generalist algorithms

This leads to the question of whether some algorithms might be considered as specialist algorithms according to the definition given in the introduction. It is obvious that the examples here are not exhaustive and additional examples will be required in order to determine 'specialist' or 'generalist' tendencies of these algorithms. However, the fact that *Glmnetridge*, *Glmnetlasso* and *CoxBoost* rank distinctly better for $\overline{CV}$ than for $\overline{CSV}$ in the simulation example suggests that these two algorithms may adapt more strongly to the specific properties of an individual data set than other algorithms. This problem can be analyzed in greater detail using the local performance criteria S_{self} and S_{across} which directly measure the correlation of the true and fitted linear predictors on the training study it*self* as well as *across* studies.

CoxBoost and *Glmnetridge* indeed achieve better ranks in R_{self} than in R_{across}. CoxBoost improves its position by 1 or 2 ranks, which is also the difference which could be found between CoxBoost's $\overline{CSV}$ and $\overline{CV}$ rankings. Thus, one may conclude that these two algorithms have the most pronounced tendency to specialize to the dataset at hand.

Nonetheless, it must be mentioned that all the algorithms share this tendency to some extent since S_{self} is consistently substantially higher than S_{across} (median: 0.6 vs. 0.2). In this sense, the higher $\overline{CV}$ values are in part justified by the fact that all algorithms perform better in within-study prediction than in cross-study prediction. To address this issue more deeply, one can also compare cross-validation to independent within-study validation for the simulated data. For the independent validation, two separate datasets are simulated using the true coefficients and gene expressions of a single study. Subsequently, a model is trained on the first dataset and evaluated on the second dataset. As can be seen in Figure B.5 in the appendix, $\overline{CV}$ values are slightly smaller than for independent validation. This matches the expectation because models are trained on less data in $\overline{CV}$. From this point of view, $\overline{CV}$ is not optimistic in the simulation study if it is used for estimating within-study performance, but it is optimistic for estimating performance in new studies of different patient cohorts.

2.3 Discussion

In applying genomic approaches to clinical problems, it is rarely safe to assume that the studies used in a research environment faithfully represent what will be encountered in clinical application, across a variety of populations and medical environments. From this standpoint, study heterogeneity can be a strength, as it allows to quantify the degree of generalizability of results, and to investigate the sources of the heterogeneity. This

aspect has long been recognized in meta-analysis of clinical trials (Moher and Olkin, 1995). Therefore, one can expect that an increased focus on quantifying cross-study performance of prediction algorithms will contribute to the successful implementation of the personalized medicine paradigm.

A conceptual framework, statistical approaches, and software tools for this quantification are provided here. As an illustrating example, cross-study validation is demonstrated on eight independent microarray studies of ER-positive breast cancer, with overall survival as the endpoint of interest. Moreover, a simulation procedure has been developed which involves two levels of non-parametric bootstrap (sampling of studies and sampling of observations within studies) in combination with parametric bootstrap, to simulate a compendium of independent datasets with characteristics of predictor variables, censoring, baseline hazards, prediction accuracy, and between-dataset heterogeneity realistically based on available experimental datasets.

Cross-validation is the dominant paradigm for assessment of future prediction performance and comparison of prediction algorithms. The perils of inflated prediction accuracy estimations by incorrectly or incompletely performed cross-validation are well known (Molinaro et al., 2005; Varma and Simon, 2006; Subramanian and Simon, 2010; Simon et al., 2011). However, it has been shown here that even strictly performed cross-validation can provide optimistic estimates relative to cross-study validation performance. All algorithms, in both the simulation and the application on real data, showed distinctly decreased performance in cross-study validation compared to cross-validation. This reflects the reality of clinical genomic study, and likely other applications, where it is impossible to control all sources of between-study heterogeneity or to ensure consistent application of new technologies in diverse laboratory settings.

In simulations, the ranking of algorithms by cross-study validation was closer to true rankings defined by cross-study prediction both within each sample of studies (R_{across}) and by global true ranking determined through Monte Carlo simulation. Surprisingly, cross-study validation was also competitive to cross-validation for recovering true rankings based on within-study prediction, even out-performing cross-validation in this application after the removal of outlier studies. Considering that the ultimate goal of prediction modeling is to provide predictions for independent observations, it may be claimed that assessing algorithms by cross-study validation may provide a more useful indicator of the performance than the current standard of cross-validation.

Systematic cross-study validation also provides a means to prioritize relevant sources of heterogeneity within the context of the prediction problem of interest. By simple inspection of the CSV matrix, two outlier studies have been identified that yielded prediction models no better than random guessing in new studies. This may be related to known differences in these studies: smaller numbers of observations, higher proportions of node positive patients, or larger tumors. However, differences in other important clinical variables, which are related to outcome, such as type of adjuvant therapy, had no obvious impact on cross-study prediction accuracy. Although one cannot verify indicators of poor cross-study prediction accuracy within the scope of this study, one can confidently conclude that these two datasets are outliers within the context of the compendium of datasets studied here.

It may be supposed that in practice it is neither possible nor even desirable to eliminate all sources of heterogeneity between study cohorts of complex diseases. The adoption of 'leave-one-in' cross-study validation, in settings where at least two comparable independent datasets are available, can provide more realistic expectations of future prediction model performance, identify outlying studies or clusters of studies, and help to develop "generalist" prediction algorithms with less tendency to fit to dataset-specific biases. Further work is needed to formalize the identification of clusters of comparable studies, to develop databases for larger-scale cross-study assessment of prediction algorithms, and to develop better "generalist" prediction algorithms. With the development of software and data resources, cross-study validation is in practice no more difficult or CPU-consuming than cross-validation, and should become an equally standard tool for assessment of prediction models and algorithms.

Chapter 3

Computational Aspects & Software

The analyses performed in this thesis would not have been possible without the usage of parallel HPC systems. The following sections describe several aspects of the software developed and used for the scientific work in this thesis in more detail. Apart from the **R**-package *survHD* which has been extensively used for the simulations in Chapter 2, these sections will also cover computational topics, which refer to the administration of computer resources as well as implementations of statistical analysis software on heterogeneous cluster or cloud infrastructures. The chapter also provides a short introduction to parallel computing in **R** as well as several approaches for the usage of cloud computing for statistical projects. It also tries to compare cloud and cluster resources and to show solutions for combining both types of resources in order to perform statistical analysis projects in less time and without large investments for in-house resources which might be idle most of the time. Finally, it presents state-of-the-art approaches to the handling of large datasets which will probably gain in importance in the near future since biological technology produces increasingly vast amounts of data.

3.1 Introduction

Nowadays, the structure and size of data are vastly gaining in complexity. One of the most prominent examples is the upcoming of next generation sequencing data which even after preprocessing can comprise several Gigabyte. Thus, these datasets are significantly larger than the micro array data used in this thesis. Additionally, the necessary preprocessing, which encompasses sequence alignment and normalization, requires a huge amount of computational resources. This demands a close cooperation of statisticians and bioinformaticians who usually perform most of the preprocessing steps as well as the database management involved in modern state-of-the-art biological data. In this context, the Bioconductor project can serve as a good interface between these two professional groups. On the one hand, its package-based system allows users from both domains to contribute their developed algorithms for data analysis and preprocessing to a research community. On the other hand, scientists can find assistance and useful tools for those parts of their

research which do not involve their actual domain of expertise. One of the most important aspects of Bioconductor is the usage of the statistical programming language **R**. Although it is not one of the fastest programming languages it basically serves as a good framework for embedding the different software tools available at Bioconductor. The relatively strict guidelines for structure and documentation of **R**-code and **R**-packages ensure minimal standards for usability and user friendliness. Additionally, the packages are all open source and thus the source code is accessible and can be checked by the users themselves. Another advantage of the **R** language is that it provides many flexible mechanisms for parallelization.

3.2 *survHD*: An application programming interface (API) for survival analysis on high dimensional data

The following section describes an **R**-package for survival analysis on high-dimensional data which has been developed during the simulations presented in Chapter 2. It is currently available at `https://bitbucket.org/lwaldron/survhd` whereas the separate feature package *survHDExtra* is available at `https://github.com/bernau/survHDExtra`. In general, survival analysis on high-dimensional data is frequently used in recent studies trying to link survival time to gene expression profiles (van Wieringen et al., 2009). One of the most important aspects in this context is the evaluation and assesment of the generalization errors of fitted prediction rules. In Waldron et al. (2013) the authors point out that within-study validation, which is usually based on resampling procedures is often opimistic. As mentioned above, this phenomenon has several causes including batch effects, optimal selection and incomplete cross-validation. The comparability of studies can be improved by methods like add-on normalization (Kostka and Spang, 2008) or frozenRMA (McCall et al., 2009).

3.2.1 General concept

Other reasons for the optimism in within-study validation can be addressed by investing more effort into the proper approach to within-study validation (e.g. using nested cross-validation). For the purpose of proper evaluation, the **R**-package *survHD* has been developed in cooperation with Levi Waldron and Markus Riester from the Harvard Medical School. Currently, this package encompasses the following algorithms for survival analysis on high-dimensional data:

(1) boosting for survival analysis based on the Cox model (CoxBoost Binder et al., 2009)

(2) supervised principal components analysis (superPC Blair and Tibshirani, 2004)

(3) Unicox: a method which is based on univariate cox models (Tibshirani, 2009)

(4) random survival forests (Ishwaran et al., 2008)

(5) PlusMinus (Zhao et al., 2013)

(6) penalized Cox regression (lasso, ridge and elastic net Goeman, 2010; Simon et al., 2011)

These methods are representative for the commonly used algorithms in recent literature on survival analysis. They include very established methods as penalized regression as well as more heuristic methods like the PlusMinus. This method is usually more applied by bioinformaticians and machine learners and a real theoretic foundation has just been developed recently (Zhao et al., 2013). One of the problematic aspects in the evaluation of these algorithms is the lack of an intuitive performance criterion which is established as gold standard. In binary classification, the misclassification rate or the area under the curve have already been established as such criteria. In survival analysis, however, a prediction error cannot be defined equally easily whereby the dependence on time as well as the handling of censored observations are the most delicate questions concerning this issue. For models which are subject to the proportional hazards assumption, one can resort to Harrel's C-Index (Harrell et al., 1996) which is mostly applied in practice. It is based on a pairwise comparison of survival observations and their corresponding linear predictor estimated by the respective model. Since the cumulative hazard of a patient with higher linear predictor is at all times higher than the cumulative hazard of the patient with lower linear predictor in a Cox-model, a pair of two uncensored observations is classified as concordant if the patient with the shorter survival time also has the higher value of the linear predictor. For details on the handling of censored observations, please see Harrell et al. (1996). Other criteria are more general and try to transfer common ideas of goodness of fit or likelihood to the case of survival regression. These criteria are usually more complex since they involve the estimation of the baseline-hazard as e.g. prediction error curves. Consequently, their computation is more difficult and the estimation of additional parameters renders them less reliable in the case of small datasets. However, they are also more gernerally applicable since they are not based on the proportional hazards assumption. The *survHD* package provides functions for a wide spectrum of performance criteria for survival models:

(1) Cross-Validated Partial Log-Likelihood (Verweij and Houweilingen, 1993)

(2) Uno's C-Index (Uno et al., 2011)

(3) Harrell's C-Index (Harrell et al., 1996)

(4) Prediction Error Curves (Porzelius et al., 2011)

(5) Net Reclassification Index (Uno and Cai, 2012)

(6) Integrated Discrimination Improvement Index (Uno and Cai, 2012)

Using these criteria, users can assess the perfomance of the different prediction models and optimize potential tuning parameters. For this purpose, *survHD* provides an incorporated tuning function which performs (nested) cross-validation on a grid of parameters in

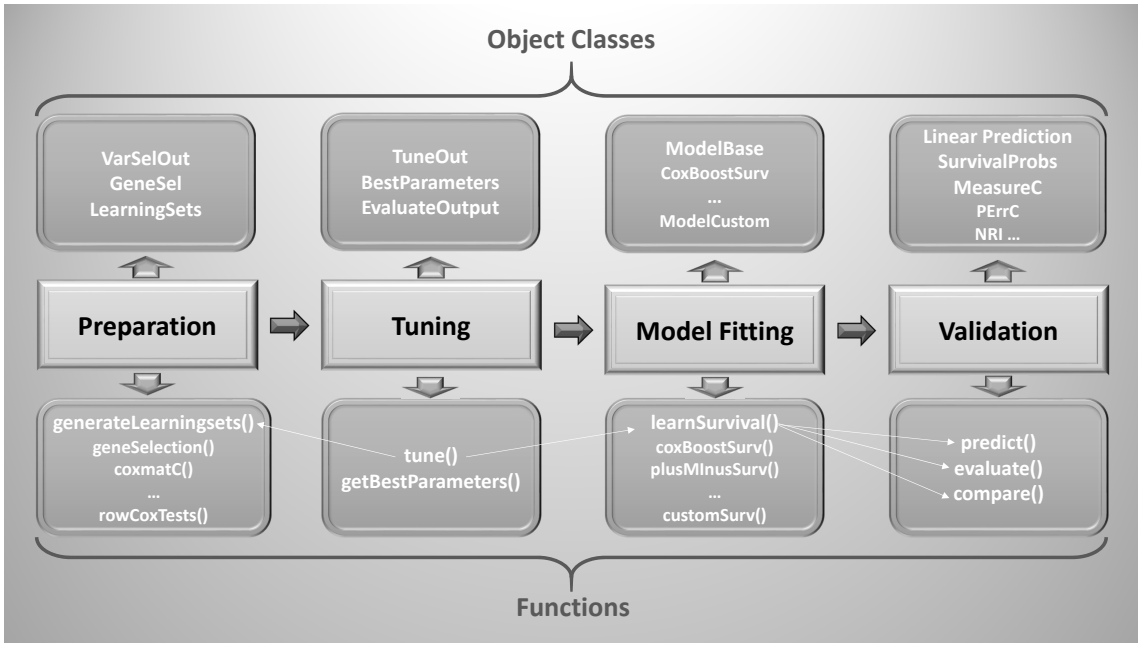


Figure 3.1: Most important functions and objects in *survHD*. A typical workflow consists of creating learning sets, variable selection, tuning, modelfitting and subsequent evaluation. Moreover, there are dedicated methods for the comparison of models.

order to find appropriate hyper parameter values based on an user-specified performance criterion. A complete workflow in *survHD* is depicted in Figure 3.1.

In contrast to *CMA* (Slawski et al., 2009), which provides similar functionality in the context of classification, *survHD* also allows for the usage of package-specific tuning mechanisms which are often more efficient than a simple grid search. The penalized package for example, has incorporated an analytic way for finding the optimum of the cross-validated partial log-likelihood in penalized Cox-regression (Goeman, 2010). Usually such analytic approaches are more efficient and thus preferable to the common grid search. In practice, however, the additional embedded grid search in *survHD* is an important alternative to the package specific tuning mechanisms since they suffered from convergence problems in several test setups which were used during the test phase of *survHD*.

Another important feature is the funcitonality for independent validation. As pointed out in Chapter 2, within-study validation is suboptimal and independent validation is always preferable if it can be performed. Apart from the typical split into training and validation set, *survHD* also has an implementation for the computation of the **Z**-matrix as proposed in Chapter 2. The implementation is based on a list of datasets which are used as training as well as independent validation set. Thus, users are easily able to check whether the estimated prediction rules are really generalizable on data that have been obtained in different studies. This is the basic precondition if a gene profile or gene expression based prediction model is intended to be applied in practice.

3.2.2 Custom Survival Models

Another decision was also motivated by the intention to improve user-friendliness. For better extensibility, customizability and maintenance, the API has been split into a stable core, which is supposed to be incorporated into the Bioconductor project in the future, and an extendable feature package (*survHDExtra*). The feature package has been moved to github and will serve as an opportunity for extending the package without modifying the core in an essential way. One desirable extension consists in the ammendment of further survival models for high-dimensional data. For that purpose, an interface for user-defined algorithms was implemented in *survHD*. This interface for user-defined survival algorithms can serve as a good opportunity to illustrate the basic structure of *survHD*. This interface will be presented in more detail here whereas a more extensive description of the remaining part of *survHD* can be found in Appendix C.1. The interface is based on two features:

(1) model fitting

(2) prediction from previously fitted models

The second feature is necessary to guarantee the correct computation of the diverse performance criteria in the package. The function is stored as a slot in the constructed model object which ensures that the model object can be embedded into the typical mechanism for obtaining predictions in *survHD*.

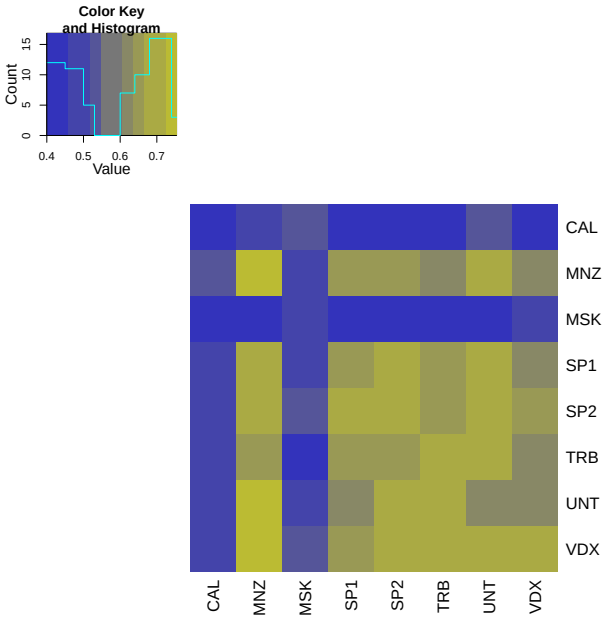


Figure 3.2: 'Leave-One-In-Validation matrix' for the algorithm 'Plusminus' on eight breast cancer studies. Each of the experimental microarray datasets is used for training (rows) and subsequently evaluated on all other datasets (columns). The main diagonal corresponds to performance estimates obtained by nested cross-validation.

Both functions are eventually implemented using a single object. In the following example the method *rsf* from the package *randomSurvivalForest* (Ishwaran et al., 2008) is added to *survHD* as a custom function *customRSF*. The user-defined function has to accept at least three predefined inputs:

(1) **Xlearn:** A **data.frame** of gene expressions for the current training data (columns are genes)

(2) **Ylearn:** the survival response for the current training data

(3) **learnind**: the indices of the current training observations inside the complete dataset **X** which is usually passed to functions like **learnSurvival**. Using this argument, one can e.g. extract the relevant observations from additional clinical covariates which can be passed using the ... argument.

```
customRSF <- function(Xlearn, Ylearn, learnind, ...){
               #load required packages
               require(randomSurvivalForest)
               #handle inputs
               ll <- list(...)
               datarsf <- data.frame(Xlearn, time=Ylearn[, 1], status=Ylearn[, 2])
               ll$data <- datarsf
               ll$formula <- as.formula('Surv(time, status)~.')
               #call actual model function rsf from randomSurvivalForest
               output.rsf <- do.call("rsf", args = ll)
               ...}
```

First, one typically has to load or source a function or package which performs the actual model fitting. The next step is the processing of the inputs. In this case one does not use any additional covariates, so one only has to convert **Xlearn** and **Ylearn** into the input format of the function *rsf*. As can be seen from the code, this function accepts a **formula** and a **data.frame** containing all necessary variables. Moreover, one passes all the arguments represented by the **...** argument and eventually calls the function *rsf* using *do.call*. After this step, the model fitting is complete.

In order to provide outputs which can be evaluated and processed by *survHD*, one also needs a *predict* function. This function must either provide an object of class **LinearPrediction** or **SurvivalProbs**. The prediction function has to accept four arguments:

(1) **object:** an object of class **ModelCustom** which is created by *survHD* automatically and contains the fitted model (here: **output.rsf**) in its slot **mod**.

(2) **newdata: data.frame** of gene expressions for the observations for which predictions will be performed.

(3) **type:** indicates whether linear predictors (**'lp'** or survival probabilities (**'SurvivalProbs'**) will be predicted

(4) **timegrid:** if **type='SurvivalProbs'** this argument specifies the time points at which predictions will be performed

```
predfun <- function(object, newdata, type, timegrid=NULL, ...){
require(randomSurvivalForest)
#either type lp or type SurvivalProbs must be implemented
#for typ lp the obligatory return class is LinearPrediction
       if (type == "lp") {
      stop("Random Forests don't provide linear predictors, sorry.")
       }
#for typ SurvivalProbs the obligatory return class is Breslow
       else if (type == "SurvivalProbs") {
           modelobj <- mod(object)
           if (is.null(timegrid)) {
               stop("No timegrid specified.")
           }
#create data for which predictions are to be performed
predsrsf <- predict.rsf(object = modelobj, test=data.frame(newdata,
time=rexp(n=nrow(newdata)), status=sample(c(0, 1), nrow(newdata), replace=T)))
curves <- exp(-t(apply(predsrsf$ensemble, 1, FUN=function(z)
approx(x=predsrsf$timeInterest, y=z, xout=timegrid)$y)))
#create breslow-object
pred <- new("breslow", curves = curves, time = timegrid)
#create SurvivalProbs-object embedding the breslow-object
           pred <- new("SurvivalProbs", SurvivalProbs = pred)
       }
else stop('Invalid "type" argument.')
return(pred)
}
```

This function is structured into two sections which correspond to predicting linear predictors and survival probabilities respectively. Since random survival forests are not capable of providing linear predictors the first section simply returns an error. In the survival probability section at first the input data have to be preprocessed for the function *predict.rsf* which can perform predictions on the basis of objects of class **randomSurvivalForest** created by the function *rsf*. Subsequently, these predictions are estimated and eventually an object of class **SurvivalProbs** is created which is the predefined class for survival probabilities in *survHD*. Its only slot **SurvivalProbs** is an object of class **Breslow** which not only stores the survival probabilities in the slot **curves** but also the time points in the slot **time**. This **survprob** object is finally returned by the custom prediction function.

The definition of this prediction function was the second part of the user-defined survival function *customRSF*. In the end, one still has to create the obligatory output object of class **ModelCustom** which primarily consists of the fitted model object **output.rsf** in slot **mod** and the user-defined prediction function in slot **predfun**. Additional information can be stored in the slot **extraData** which must be of class **list**:

```
#create customsurvhd-object (which is the obligatory output-object)
custommod <-new("ModelCustom", mod=output.rsf, predfun=predfun, extraData=list())
 return(custommod)
```

It is practical to make sure that the custom function is known to the global environment of **R** such that it is available for all functions of *survHD*. Of course, another opportunity consists in including the new survival algorithm into the addon package *survHDExtra*.

```
#define function in global environment
assign(x="customRSF", value=customRSF, envir=.GlobalEnv)
```

Using the custom function *customRSF* one can easily implement a complete workflow of generating learning sets, gene selection, tuning, resampling and a final evaluation.

```
#learningset
ls <- generateLearningsets(y=y[, 1], method='CV', fold=5)
#gene selection
gsel <- geneSelection(X=X, y=y, method='fastCox', LearningSets=ls,
                  criterion='coefficient')
#tune
tuneres <- tune(X=X, y=y, GeneSel=gsel, nbgene=30, survmethod='customSurv',
customSurvModel=customRSF, LearningSets=ls, grids = list(ntree = 20*(1:10)))
#use tuneres in learnSurvival
svaggr <-learnSurvival(X=X, y=y,GeneSel=gsel,nbgene=30,survmethod='customSurv',
customSurvModel=customRSF, LearningSets=ls, tuneres=tuneres, measure="PErrC",
timegrid=4:10, gbm=FALSE, addtune=list(GeneSel=gsel, nbgene=30))
```

The only essential difference to the call to *survHD*'s built-in algorithms consists in the argument **survmethod** which is set to **'customSurv'**.

Supporting reproducibility By providing this interface for user-defined functions, the evaluation framework of *survHD* can also support reproducibility. By integrating a new algorithm via the interface, evaluation criteria as well as validation on independent data can be achieved in a more objective and replicable way. Variable selection, tuning and evaluation on several datasets can be performed with a few lines of code. Ideally, it would be recommended that new methods are additonally evaluated by experts who have not been involved in the process of developing the new algorithm. Using *survHD*, one could easily set up a list of benchmark datasets like the selection of breast cancer microarray studies introduced in one of the previous sections and validate the proposed algorithm. The only prerequisites are the two interface functions explained in this section.

3.3 Short introduction to parallel programming in R

The statistical analyses performed in *survHD* and the previous chapters, are based on resampling methods like bootstrapping or cross-validation. In this sense, they are representative for many recently developed statistical methods which require more and more computational resources. An important aspect of these methods is their large number of independent subtasks, e.g. a single resampling iteration or one permutation step. This facilitates a parallel implementation since the different processes do not have to send large packages to each other.

3.3.1 The package *Rmpi*

The programming language **R** provides several mechanisms for processing such independent tasks without large modifications to the code. Parallelization is mostly performed for loops which are already efficiently implemented in **R** using wrapper functions like *lapply* or *sapply*. The package *multicore* provides the function *mcapply* which enables the use for shared memory parallelization on a single node. In the presence of a message passing interface (MPI) environment, the package *Rmpi* can be used. This package implements many parts of the common MPI standard (MPI-Forum, 2012) including point-to-point and collective communication like *MPI_Broadcast*. After setting up a MPI communicator, all necessary objects are broadcasted from a master process to all remaining MPI-processes and the independent subtasks of the analysis can be processed in parrallel. Let us take a look at the startup procedure in a typical *Rmpi* program as it could be run on the cluster at the IBE:

```
1  ###load Rmpi
2  library('Rmpi')
3  ###check size of parallel environment
4  mpi.universe.size()
5  mpi.comm.size()
6  ###get MPI-rank of the process
7  mpi.comm.rank()
8  ###try broadcast
9  x<-5
10 mpi.bcast.Robj2slave(obj=x)
11 ###create Communicator
12 mpi.spawn.Rslaves()
13 ###check communicator size
14 mpi.comm.size()
```

After loading the package *Rmpi*, one first checks the size of the MPI environment using the function *mpi.universe.size()*. The different processes of the MPI environment can be organized in different communicators which can be considered subgroups of the processes in the environment. These communicators are useful for collective calls, if for example all

processes of a communicator have to wait for a certain event or have to share data. In this example, however, only the standard communicator will be used here which includes all processes of the environment. This communicator must be created by the user at the beginning. Consequently, the first broadcast (l. 10) will fail, because the communicator is only created in the call to *mpi.spawn.Rslaves* (l. 12). This command starts a worker on each core in the MPI environment (also on the core hosting the master process) and establishes a communicator including all worker processes and the master process. Consequently, *mpi.comm.size* (l. 14) will be equal to *mpi.universe.size()*+1 whereas it returns 0 in line 5.

In line 4 one has initialized the variable **x** using the value 5. However, this variable is still known exclusively to the master process where this initialization was executed. Thus, *mpi.remote.exec* (l. 16), which runs an **R** expression on each worker process in the communicator returns a list of 4 errors reporting that object **x** could not be found.

```
15 #evaluate R-expression on each worker
16 mpi.remote.exec(cmd=x)
17 #Broadcast
18 mpi.bcast.Robj2slave(x)
19 mpi.remote.exec(x)
```

The **R**-object **x** has to be copied from the master process to the worker processes by a call to *mpi.bcast.Robj2slave*. The fact that the different worker processes are operating on separate environments, including different objects, must always be kept in mind in order to avoid errors. It is also possible that eponymous variables have different values or even different types in the environments of the different worker processes. After the broadcast of the object **x** in line 18, one will be returned a list of fives in line 19 because all worker processes have been sent the object **x** by now.

Now, a first simple example can be implemented. A vector of 2 million normally distributed random numbers is created and its mean is computed on each of the worker processes. For that purpose, one defines the following function.

```
20 est.mean<-function(seed){
21 set.seed(seed)
22 mean(rnorm(2000000))
23 }
```

The example already includes a very simple mechanism to ensure reproducibility. In parallel programming, it is not enough to set a seed on the master process only, but one has to ensure that the program is deterministic for all processes. Besides more sophisticated mechanisms, a potential solution for this problem consists in passing a seed parameter and setting the seed inside the function that is run in parallel. Consequently, it does not matter where and when the function body will be executed. To run the function on all worker processes, one uses the function *mpi.apply* or its load balanced counterpart *mpi.applyLB*:

```
24 set.seed(25)
25 ar.seed<-array(ceiling(runif(100)*200),dim=c(100))
26 result<-unlist(mpi.applyLB(x=ar.seed,fun=est.mean))
```

The function *mpi.applyLB* runs the specified function *est.mean* on the different processes whereby the values in **ar.seed** are passed as the first argument one at a time. At the end, the results are collected by the master process and combined to a list object. A second execution of these 3 lines would return exactly the same result. A seed has been set on the master process in line 24. This seed ensures that the array **ar.seed** contains the same values in each execution. Thus, always the same seeds are passed to the respective function calls on the worker processes leading to equal return values.

For the case where more tasks than MPI processes exist, the load balanced version can be recommended. The load balancing purely ensures that new tasks are scheduled to a worker process as soon as one has finished its previous task instead of scheduling the tasks to the processes always in the same order. More sophisticated load balancing has to be implemented by the user himself by distributing tasks of approximately equal total length to all available processes.

3.3.2 The packages *snow* and *doSNOW*

The package *snow* simplifies the programming and also extends the functionality of *Rmpi*. It provides even easier functions for sending **R**-objects to the worker processes and running independent tasks on them. A small example for gene interaction tests is presented here which illustrates the typical workflow in *snow*. Suppose that one wants to compute a model for each pair of genes in a gene expression matrix $\mathbf{X}$ (rows correspond to patients and columns correspond to genes) in which one wants to test for the corresponding interaction effect on a binary response $\mathbf{y}$. In order to reduce the number of pairs, one first preselects **nv** genes using the function *GeneSelection* in *CMA*:

```
1  #prepare gene expressions and response
2  library(CMA)
3  data(golub)
4  X<-as.matrix(golub[,-1])
5  y<-golub[,1]
6  #number of genes to be preselected
7  nv<-40
8  #preselection of variables (using CMA,X=matrix of gene expressions, y=response)
9  gs<-GeneSelection(X=X,y=y,method='t.test')
10 Ximp<-X[,gs@rankings[[1]][1:nv]]
```

In a second step, one defines a grid consisting of all pairwise indices of the preselected variables:

```
11 #define grid
12 todo<-c()
13 for(ii in 2:ncol(Ximp))
14 for(cc in 1:(ii-1))
15 todo<-rbind(todo,c(ii,cc))
```

The function, that has to be repeated for each pair, is the following. It first extracts the current pair of gene expressions from the matrix of the preselected variables **Ximp**. Subsequently, it computes a logistic model including the main and interaction effects of the two genes. Eventually, the coefficients are returned for further analysis.

```
16 #function to be run in parallel
17 varselpara<-function(k){
18 datc<-data.frame(y=y,reg1=Ximp[,todo[k,1]],reg2=Ximp[,todo[k,2]])
19 mod<-glm(y~reg1*reg2,family='binomial',data=datc)
20 return(coef(summary(mod)))
21 }
```

Now, this function is run on all processors in the MPI environment. One first loads the package *snow* and creates an MPI cluster (lines 20–22). Afterwards, one has to send the required objects for the computation to the workers. In this case these objects are specified in the parameter **list** of the function *clusterExport*. It includes the grid **todo** as well as the matrix of preselected genes **Ximp** and the response **y**. Note that the function *varselpara* is actually also an unknown object for the worker processes. The broadcasting of the function, however, will be performed by the function *clusterApplyLB* which also starts the execution. Apart from the cluster and the function *varselpara*, *clusterApplyLB* requires the parameter **x** which is usually a vector whose elements will be passed to *varselpara* as first argument one at a time.

```
22 library(snow)
23 #create cluster
24 cl<-makeMPIcluster(4)
25 #send necessary objects to worker processes
26 clusterExport(cl=cl,list=c('Ximp','todo','y'))
27 #parallel execution
28 results<-clusterApplyLB(cl=cl,x=1:nrow(todo),fun=varselpara)
```

The speedup curve for running this small example on different clusters can be seen in Figure 3.3

doSNOW: In the special case of loop parallelization, several helper functions have been developed which ease computation, e.g. by automatically sending necessary objects to all cores (Revolution-Analytics, 2013). The latter reference also provides an interesting mechanism for generalized 'parallelization'. It overwrites the operators *do* and *dopar* in order to process the loops. The parallelization type is specified by a previous call to *register** where the asterisk is a placeholder for MPI, multicore or other parallelization mechanisms. This way, the same code can lead to different implementations of parallelization, i.e. it is easily adopted to the resources that are currently available. The previous example could be continued from line 14 with *doSNOW* using the following lines.

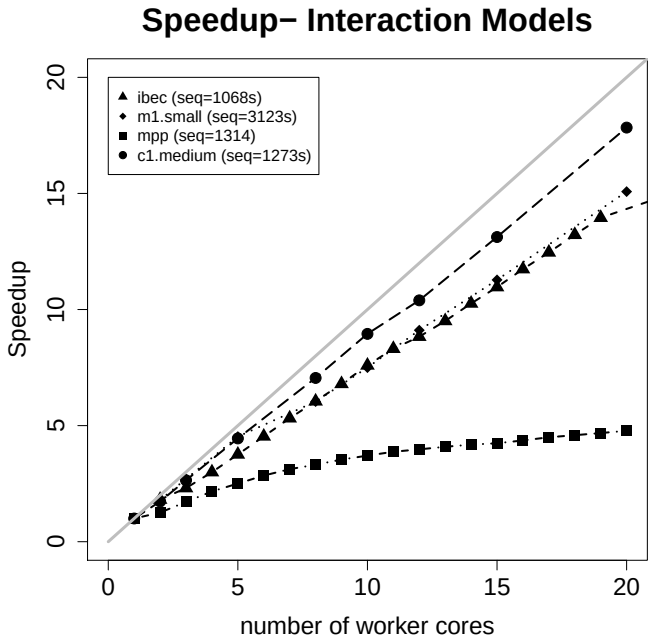


Figure 3.3: Speedup for pairwise logistic models on different computing clusters. The bad performance was probably caused by a problem in the **R** installation. Benchmarks with a newer installation did not suffer from such decreases in performance.

```
15 #create cluster and register it
16 library(doSNOW)
17 cl<-makeCluster(4)
18 registerDoSNOW(cl)
19 #parallel execution
20 results<-foreach(j=1:nrow(todo)) %dopar% varselpara(j)
```

After the startup of the package consisting of the creation of the cluster and the registering (lines 15–18), one can basically run common **R** code. The foreach-statement, which does not differ substantially from a classical loop statement, could also be run in serial by replacing **dopar** by **do**. One also does not have to consider the different environments of the processes because the package automatically broadcasts relevant objects wherever necessary. Moreover, one could use different types of parallelization, e.g. shared memory parallelization (**R** package *multicore*) or the Redis approach which will be introduced in 3.4.4. Thus, one can switch between different modes of execution without having to modify the code substantially.

3.3.3 pbdMPI: Overcoming the master-worker concept

Recently, a new **R** package – *pbdMPI* – has been developed and published by Chen et al. (2013). It overcomes the master-worker concept the other packages are based on. Its programming style resembles more the common MPI programs where each MPI process runs the same lines of code and process specific code parts are handled using *if*-statements. The key elements of the package are introduced here by implementing the previous example of pairwise tests. At first, one has to load the package and run a start-up mechainism using the *pbdMPI* function *init*. Basically, all programs using *pbdMPI* commonly start with these two lines:

```
1 library(pbdMPI)
2 init(set.seed=FALSE)
```

The argument *set.seed=FALSE* prevents that *pbdMPI* sets its own random seeds. If this argument is set to *TRUE*, deviations from **R**'s usual behaviour in random number generation can be observed which shall be avoided here. Afterwards, one has to prepare the data. The alert reader will recognize that these are exactly the same lines as in the previous section. Their meaning, however, is different and illustrates the 'philosophy' of the package quite well.

```
3 #prepare gene expressions and response
4 library(CMA)
5 data(golub)
6 X<-as.matrix(golub[,-1])
7 y<-golub[,1]
8 #number of genes to be preselected
```

```
9  nv<-40
10 #preselection of variables (using CMA,X=matrix of gene expressions, y=response)
11 gs<-GeneSelection(X=X,y=y,method='t.test')
12 Ximp<-X[,gs@rankings[[1]][1:nv]]
13 #function to be run in parallel
14 varselpara<-function(k){
15 datc<-data.frame(y=y,reg1=Ximp[,todo[k,1]],reg2=Ximp[,todo[k,2]])
16 mod<-glm(y~reg1*reg2,family='binomial',data=datc)
17 return(coef(summary(mod)))
18 }
19 #define grid
20 todo<-c()
21 for(ii in 2:ncol(Ximp))
22 for(cc in 1:(ii-1))
23 todo<-rbind(todo,c(ii,cc))
```

In contrast to the previous section, this code is run on all processes at the same time, i.e. all processes load the data and run the preselection of the genes separately. This might look like a waste of resources. But the runtime of the computation is smaller than a second, so having the results broadcasted by a master process might even take longer. The loading of the dataset by each process might be more questionable. Depending on the network file-system and the number of different MPI processes, the fact that all processes have to access a single file at the same time might cause a bottleneck. Here, however, one does not need to treat this issue in detail. The most important aspect is that all processes have the necessary objects (**todo**, **y**, **Ximp**) and also the funtion *varselpara* in their corresponding environment at the end of this code section. Hence, one does not need to broadcast them or wait for a master to give further instructions. The comparison of these workflows in *snow* and *pbdMPI* is illustrated in Figure 3.4.

After this preparation, one has to ensure that each task computes some of the pairwise logistic models. Again, each task determines separately what it has to do. In this case, it requests its own rank, the total number of MPI processes and (lines 24–26) before it computes its specific indices in the grid **todo** (lines 28–30).

```
24 #determine process specific tasks
25 rank<-comm.rank()+1
26 nranks<-comm.size()
27 ntodo<-floor(nrow(todo)/nranks)
28 remainder<-floor(nrow(todo)%%nranks)
29 tasks<-c(0,rep(ntodo,nranks)+c(rep(1,remainder),rep(0,nranks-remainder)))
30 myindices<-(sum(tasks[1:rank])+1):sum(tasks[1:(rank+1)])
```

The actual computation is performed in a code section which is absolutely identical to a serial implementation. It is a simple loop-statement calling the function *varselpara* iteratively in order to compute the corresponding logistic models. The coefficients are stored in a local matrix **myresults**. The matrix **myresults** is a good example for an eponymous

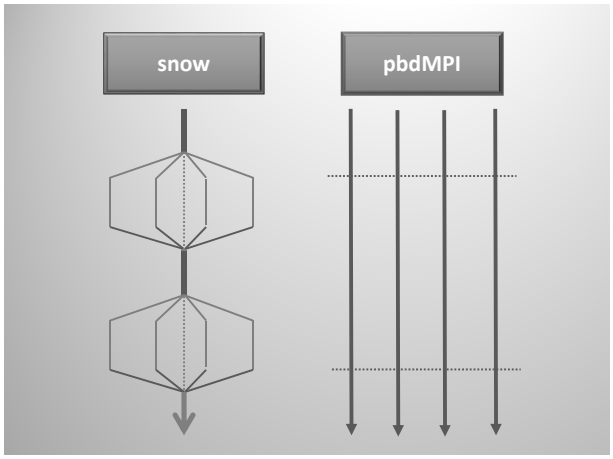


Figure 3.4: Comparison of parallelization in *snow* and *pbdMPI*. With *snow*, one has a master process which sends commands and objects to the worker processes and collects results at the end. During the actual computation parts, the master is idle (see dashed lines in the left diagram). The package *pbdMPI* follows a philosophy of independent workers. At certain time points, the workers have to communicate their results in order to benefit from the work performed by the others (see dashed lines in the diagram on the right-hand side).

object which is present in the environments of all processes but containts different values in each of these environments.

```
31 myresults<-c()
32 for(ind in myindices)
33 myresults<-rbind(myresults,varselpara(ind))
```

In order to obtain the results, one now has to collect these different **myresults** matrices from the processes. This is again done by all processes using the function *allgather*. This means that after line 34, which is again executed on all processes, each process will have an object **allresults** which is a list of the different **myresults** matrices that have been computed by the different MPI processes. In this case, it is not necessary that all the processes collect the results. However, if one would like to continue the analysis probably each of the processes would need these results. Moreover, this call to **allgather** does not take much longer than having a single process gather the **myresults** matrices.

```
34 allresults<-allgather(myresults)
35 if(rank==1){
36   save(allresults,file='testresults.RData')
37 }
38 finalize()
```

The only code line, that is indeed executed by a single MPI task, can be found in line 36. It saves the results and is encapsulated by a if-statement checking the MPI rank.

Afterwards, the computation is terminated by a call to the function *finalize*.

```
39 mpiexec -n 7 Rscript pbdtests.r
```

Another aspect of the package *pbdMPI* is that it has to be run in batch mode in contrast to *snow* which can also be run interactively. Thus, the save operation in the previous code section is indeed necessary to save the results for subsequent analysis. An exemplary command line for starting the **R**-script *pbdtests.r* can be found in line 38. It uses the command *Rscript* instead of *R* to start the script on $n = 7$ MPI processes in this case.

3.4 Comparison of Cluster and Cloud Computing

The next section uses some of these parallelization mechanisms in **R** in order to implement several statistical projects on different computing clusters and cloud resources and compares the implementations with regard to runtimes, scalability and efficiency. The following test of the usability and efficiency of the individually assembled computer clusters consists of four stages. At first, the basic computation and network capacity of the different instances is analyzed and compared to other computer clusters used by statistical departments, namely the computer cluster at the IBE (IBEC) and the mpp1-cluster (mpp) at the Leibniz Supercomputing Center in Munich. Subsequently, some important parts of two statistical projects which have already been realized on common computer clusters are implemented on cloud res-sources. This is a crucial part since it will answer the question, whether the statistical analysis of high-dimensional biological data can be reasonably realized in the cloud. The last experiment will even go one step further by combining machines from several computer clusters at different departments and EC2. This experiment will present an opportunity to let machines from very heterogeneous sources jointly compute different sub-tasks of a statistical analysis.

3.4.1 Simple Trend Benchmarks

Synthetically benchmarking computer systems is basically a science on its own, which shall not be treated here. But for an overview and estimation of runtime behavior of the real applications on the different EC2 instances, two simplistic synthetic benchmarks are used. Both are not benchmarking the complete (virtual) machine, but speed and memory aspects of the destination platform R. Overall system benchmarks like the Spec suite Evaluation (2011) are far more elaborated and precise, but lack the integration of the **R** interpreter in the results. Moreover, the Spec test is rather difficult to configure and time consuming to run (Spec results for some instance types can be found in Coffrey et al., 2011). The small benchmarks are using the arithmetic system of **R** – as desired for the biostatistical software used in the bigger tests afterwards. The performance of single cores of the CPU is tested by calculation of a specific subset of the Mandelbrot set (Mandelbrot, 2003), which is done using a recursive loop calculating whether a complex polynomial is bound or not. The inner loop is very small, so depending on the specific

Identifier/ API-Name	Provider	CPU [Ghz]	Cores/ Machine	RAM per core	Cost/h (*)	Benchmark 1 Mean (sd) [s]	Benchmark 2 Mean (sd) [s]
ibec	IBE	2.7	4	1.5 GB	-	79.1 (1.03)	71.8 (0.7)
mpp	LRZ	2.0	16	2 GB	-	98.6 (2.5)	91.9 (3.6)
gvs1/3	LRZ	2.3	16	8 GB	-	100.6 (1.8)	97.8 (0.23)
m1.small	AWS	2.7	1	1.7 GB	0.085$	309.1 (6.9)	317.6 (6.6)
c1.medium	AWS	2.3**	2	0.88 GB	0.17 $	132.6 (4.9)	135.1 (5.3)

Table 3.1: Results for the basic benchmarks on the various instance included in the study.* Prices from the Amazon US east data center in North Virginia (September 2011). Prices vary between the individual data centers. ** Although running on a 2.7GHz CPU this instance only features the performance of a 1.7GHz Xeon processor

R environment it can be held and executed inside the processor's cache. The memory test is even simpler. It copies memory blocks inside a matrix guaranteed bigger than the processors cache. Overall runtime of the benchmarks is short, between one and three minutes depending on the system. Therefore the benchmark can be repeated many times (at least 20 iterations), giving a measurement of the variance in execution time, which is only taken inside the **R** program and therefore no startup time is included. These short running benchmarks are giving a basic trend of the systems processor's speed using **R** and they provide an opportunity to quickly and therefore cheaply test the basic system performance, which is important due to the sheer amount of different EC2 instance types. Table 3.1 provides an overview over the instances and servers used for the analysis. The focus was on the instance types m1.small, m1.large and c1.medium, which are basically the cheapest instances. Nonetheless, they provide enough memory for the statistical projects which will be analyzed in the subsequent sections. The first impression is that all three instance types are slower than the computer cluster machines which were bought about four years ago. Especially the m1.small instance, which is EC2's standard instance, needs about two to three times more time for the first benchmark. For compatibility issues, this instance type is representing and running at the speed of a 1 Ghz CPU from 2006 when EC2 was introduced. Via virtualization, it only gets time shares of about 40% of a single physical core (Evangelinos and Hill, 2008). The m1.large and c1.medium instances are remarkably faster whereby both instances provide two virtual cores. Appendix C also provides the results of these two benchmarks for the larger, faster instances. For computations with low memory demands c1.medium is basically the instance of choice having almost the same price per core as the slower m1.small instance. The m1.large instance provides 3.75GB per core. However, its costs are four-fold. This price structure clearly indicates that writing memory efficient code can save money. Thus, it is quite probable that a statistical analysis in the cloud will produce some avoidable costs. For example, usually the computations had to be run using only one core per c1.medium instance. It is also worth mentioning that the performance of the AWS instances is subject to higher variability. Depending

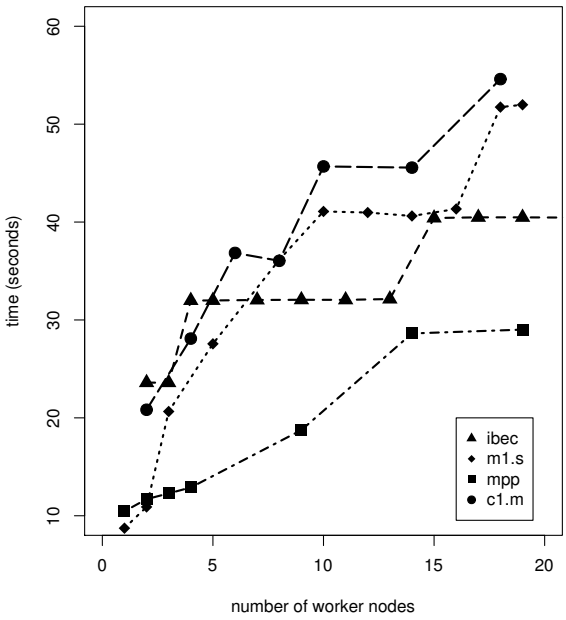


Figure 3.5: Time needed for the broadcast of a 2GB Affybatch. Here, the MPP-cluster with its Infiniband network is clearly superior to the other clusters.

on the structure of the underlying Xen virtualization (Systems, 2012), cores and memory are separated quite well on the instances. But network and storage I/O (Input/Output) cannot be separated this way and therefore all instances on a host can affect each other. In order to check the network capacity *Rmpi*'s broadcast command (Yu, 2011) was used for sending a 2 GB AffyBatch object (Gautier et al., 2004), which is the commonly used **R**-object for microarray batches, to a varying number of worker cores.

As shown in Figure 1 3.5, one can observe that this broadcast takes slightly longer on the EC2 instances and that their runtimes are growing faster if the number of worker cores is increased. In the case of the mpp-cluster, which is equipped with an Infiniband network connection, the broadcasting time basically does not increase after 10 cores. However, network performance seems to be sufficient for most statistical methods where larger amounts of data are only sent once. Further analysis of the network connection and description of technical details can be found in Appendix C. The result of these first tests are that computations might take longer on the chosen EC2 instances but the network capacity seems to be sufficient for the planned projects.

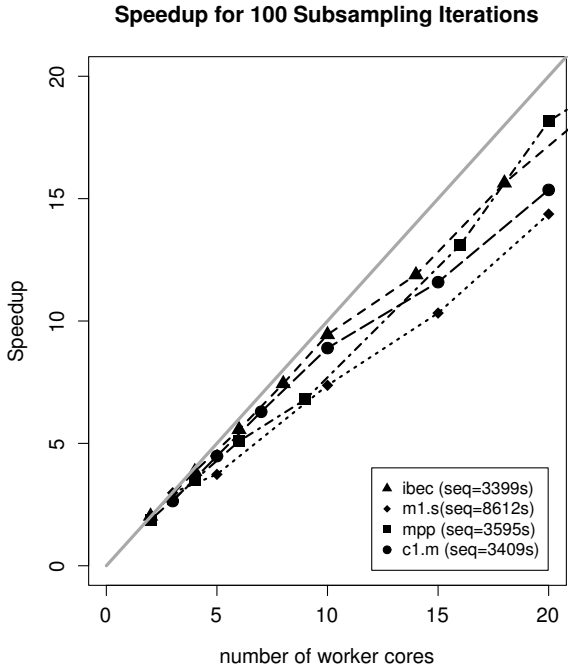


Figure 3.6: Speedup for Project 1 on cluster and cloud instances. The speedup is almost linear and only minor differences can be observed between cluster and cloud platforms.

3.4.2 Test Project: Optimization Bias

The first project to be conducted in the cloud deals with nested resampling procedures in the context of supervised classification with high-dimensional microarray data as predictors, as described in detail in Chapter 1. One goal of this project was to estimate the bias induced by simply trying several predefined cost parameter values and reporting the minimal resampling misclassification rate only. In order to analyze this bias, one had to reiterate many times the whole resampling approach with SVM (support vector machine Hastie et al., 2001) as a classification method. Additionally, the whole procedure has to be repeated for several microarray datasets because the optimization bias, just as the cost parameter itself, strongly depends on the classification task at hand. For financial reasons, just parts of the project were implemented on the EC2 instances but the efficiency of the parallelization for the most important part of this analysis can be illustrated this way, too. The central routine implements the resampling approach for a specific cost parameter value. If this part scales sufficiently, one can expect the whole project to be efficiently scaling as well.

Figure 3.6 shows the speedup for computer cluster and cloud instances, which consists of the sequential time divided by the parallel time for an increasing amount of workers. The gray line represents a perfect linear speedup on adding additional workers. Technically, this

is based on a parallelization in **R** using MPI. The code used for the analyses is provided on a publically available Amazon Machine Image (see Appendix C). The consumed time is only measured inside the **R** program, so the overhead of instance startup is not included. Starting up 20 EC2 instances can take several minutes (referring to the year 2011) whereby this start up time is quite variable. Including this overhead into the analysis would severely distort the results since only a small part of the project is run at all. Moreover, AWS charges fees per started hour and instance which also has to be considered appropriately. Both of these issues are negligible if the whole project is calculated. One can see that scaling is almost identical for cloud and computer cluster instances. Of course, the absolute computation times are remarkably higher for the m1.small instance. However, since the implementation scales well, the higher sequential runtime can be easily reduced by renting more instances. The main result of this experiment is that the main computation part of this project can also be efficiently parallelized on EC2 instances.

3.4.3 Cost Aspects: An example

This subsection presents a short cost estimation for the real data simulations performed in Chapter 1. It will be approximated how much money would have been necessary if these simulations had been run at AWS EC2.

Estimated Computation Hours

In this project, analyses on 4 real datasets as well as several simulated datasets were performed. For the real datasets, the resampling procedure had to be run for seven different classifiers. However, these classifiers are not as computationally intensive as SVMs. According to experience it takes approximately 3 times more time to compute the whole classifier pool than running SVM only. Additionally, the correct actual response as well as a randomly generated response was used in order to simulate non-informative data. It was also necessary to assess the impact of the size of the training dataset so the analyses were run for two different portions of training data. Finally, 50 replications for each of these setups had to be performed in order to analyze the variability of the method. This adds up to 1200 computing hours.
Real data study:

- four datasets (smaller than the Wang dataset) [1] 2
  ×
- at least seven classifiers (less computationally intensive than SVM) 3
  ×

[1] Since these datasets are smaller than the Wang dataset, it is expected that they require approximately half the runtime, needed for the Wang dataset. Thus, this runtime estimation uses a factor of 2 instead of 4.

- 50 replications 50
  ×
- informative and non-informative data 2
  ×
- two different portions of training data 2
  ×
- sequential runtime: 1 hour 1h

- Total: 1200h

In the simulation study, 5 data generating processes were applied for three different sample sizes. These datasets were smaller than the Wang dataset and the resampling procedure took approximately only half the time on these datasets. Here, the same classifier pool was analyzed. However, 200 replications on the simulated data were performed since the simulation provided the opportunity to use newly simulated data in each replication, i.e. each run was more informative. Concluding, this leads to an estimate of approximately 4500 computing hours.

Simulation study:

- 15 simulated datasets (smaller than than the Wang dataset) 7.5
  ×
- 200 replications 200
  ×
- at least seven classifiers (less computationally intensive than SVM) 3
  ×
- sequential runtime: 1 hour 1h

- Total: 4500h

Thus, the complete project requires at least 5700 computation hours if one uses the IBEC sequential runtime as a reference.

Estimated Costs

For approximating the costs, a simple formula can be applied:

$$\frac{f \times t}{s} n \times c = H \times n \times c, \tag{3.1}$$

Cores	1	3	5	7	10	15	20
Runtime [h]	5398	2042	1205	858	607	466	351
Costs [$]	918	1042	1024	1021	1032	1188	1195

Table 3.2: Estimated costs for the complete Project 1 on the EC2 instance *c1.medium*. Please note that due to the memory requirements the second *c1.medium* core was idle, i.e. its costs per core are 0.17$\h in this case.

Cores	1	3	5	10	15	20
Runtime [h]	13636	4574	3652	1850	1321	949
Costs [$]	1159	1166	1552	1572	1684	1613

Table 3.3: Estimated costs for the complete optimization bias project on the EC2 instance *m1.small*.

where s denotes the speedup factor, n the number of worker cores, c the costs per core and hour for the respective instance, t the sequential runtime at the respective instance and H the estimated number of hours the computation will take for the chosen number of instances. Moreover, f corresponds to the estimated factor that has to be used in order to obtain the project running time from the running time of its core part which has been analyzed here (compare to the two tables in the previous section), i.e. $4500 + 1200 = 5700$ in this case.

For this project the *c1.medium* instance is far more efficient. It is both faster and cheaper because the shorter computation time offsets its higher instance cost. In this case the second core was idle because of the memory requirements of the computation. If both cores can be used the efficiency of the *c1.medium* instance is substantially increased. Since speedups are almost linear one can achieve an substantial reduction of computation times for low additional costs.

3.4.4 A Hybrid Cloud Solution: Combining Computer Cluster & Cloud Resources

An advantage of the cloud is its scalability, i.e. the possibility to get more resources if more resources are needed. If a given infrastructure is fully utilized in peak times, the possibility to add machines from the cloud to existing computer clusters would be helpful. With some efforts, EC2 instances can be directly integrated into local computer clusters manually. Far more convenient is the usage of a system which can handle this automatically. The "data structure server" Redis (Sanfilippo and Noordhuis, 2012) can be used together with its **R** connector *doRedis* (Lewis, 2011) to combine local computer clusters and cloud instances. As networking is partly routed over the public Internet, it is clearly a bottleneck. But for embarrassingly parallel programs like the examples shown above, networking speed is not

essential. The concept of the Redis approach is that the server manages all the subtasks (e.g. a single resampling iteration) of a job and sends them to whichever worker node which connects to it.

3.4.5 Hybridcloud: Case study

For illustration purposes one of the analyses run for evaluation of the correction in Chapter 1 is used where 3000 instead of 100 resampling iteration are performed. The basic infrastructure is provided by a Redis-server on which the subtasks and their necessary data are stored. In the example, this server runs on a local machine at the IBE. The following code creates a job queue, **biascorrection1_3000**, on the Redis-server including the 3000 subtasks .

```
1 library(doRedis)
2 registerDoRedis('biascorrection1_3000')
3 results<-foreach(j=1:3000) %dopar% {cl2(j)}
```

Subsequently, one can connect local workers to the redis server which perform the subtasks. The required data are automatically sent to the workers by the redis server which is another convenient feature of this approach.

```
1 library(doRedis)
2 registerDoRedis('biascorrection1_3000',host='localpc@ibe.med.uni-muenchen.de')
3 startLocalWorkers(n=1, queue='project1_3000')
```

Now, the worker will perform the subtasks until the job queue is empty. If one wants the worker to stop after a certain number of subtasks the parameter **iter** can be specified in order to set a limit for the subtasks to be performed by the specific worker.

In an MPI environment several solutions are possible in order to connect multiple workers at the same time. In the example, the following code was run to connect 30 IBEC cores to the Redis-server:

```
1 join_doredis<-function(ind){
2 library(doRedis)
3 redisWorker('project1_3000',host='localpc@ibe.med.uni-muenchen.de')
4 }
5 library(Rmpi)
6 mpi.spawn.Rslaves(nslaves=30)
7 mpi.apply(1:30,join_doredis)
```

Of course, the same code can be applied on the EC2 nodes which eventually ensures a cooperation of cluster and cloud resources. An ambiguous feature of the approach is that each worker only connects to the Redis-server and thus does not know about the other workers. On the one hand, this alleviates certain problems because one does not need

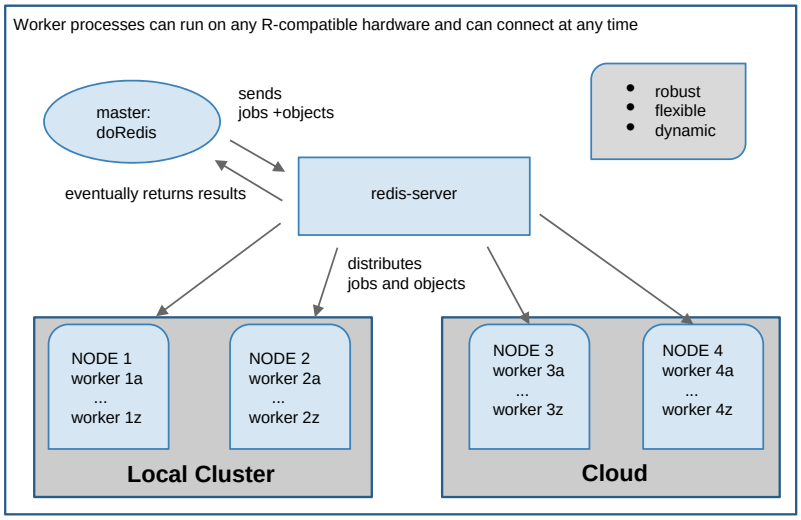


Figure 3.7: Hybrid-Cloud based on **R** and the doRedis.package. The jobs are sent from a master process to the redis server and subsequently performed by heterogeneous instances which can connect at any time. In the end, the results are sent from the workers to the redis server which finally returns them to the master process.

any interconnect between the worker nodes and different architectures can be combined in a very comfortable way. On the other hand, several hidden problems can occur in this approach. E.g. the different worker cores might use different versions of **R** or certain packages and the computation cannot be reproduced easily afterwards. A possible solution for this problem is a publically available AMI (Amazon Machine Images) which might be used to harmonize the different environments. It might be difficult, however, to use these AMIs at computer centers.

Another important advantage of this separation of job administration and computation is that one does not encounter any problems if one temporarily does not have any workers at one's disposal, i.e. the job can be paused. Results and the description of the individual subtasks of the job are stored independently from the workers. For example, if one wants to use the staff's computers for larger computation tasks at night, one can use a small script connecting the respective computer to the Redis-server as soon as the staff leaves the office. In the example, resources from the LRZ (gvs1, gvs3), IBE and EC2 were combined. Since this example was intended to demonstrate the sheer possibility of this approach the pool of worker machines primarily consisted of free computing resources. Figure 3.8 shows how many tasks have been performed by the different machines. Of course, the main part of the tasks has been processed by the 30 IBEC cores. However, one can see that the c1.medium instances processed more tasks per core than the IBEC machines although their connection to the Redis-server is slower. In practice, the actual number of integrated cloud instances can be chosen depending on the availability of free computing resources and the urgency of the job.

3.4.6 Parallelization for *survHD*

Most of the parallel programming approaches, which have been used in this comparison of cloud and cluster platforms, can also be applied in the context of *survHD* which has been introduced in the previous section. A small example will be presented which uses the *doRedis* approach. The important feature here is the independence of the different iterations in the resampling-based evaluation process which is the most computationally intensive component in the analysis. The following Gantt chart illustrates the parallelization of a cross-validation with nested hyper-parameter optimization for the CoxBoost algorithm using a combination of *doRedis* and shared memory parallelization.

Apparently, most of the time is spent in the green regions representing the actual computation time. Since the tasks in the tuning procedure are equally long, load balancing is not an important issue in this case. In comparison to *CMA*, parallelization is similarly effective. For the survival algorithms, however, parallelization is even more urgently needed, since runtimes are significantly longer especially if hyperparameter tuning is involved. Another important aspect is the robustness of the *doRedis* approach. As experienced in practice, the survival algorithms suffer from convergence problems to a much higher extend than most classification algorithms. Consequently, many tasks have to be rerun and errors have to be caught without breaking the complete computation.

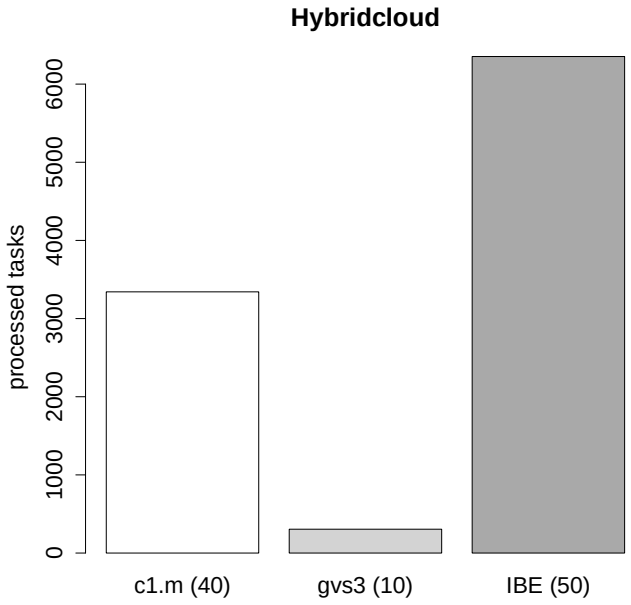


Figure 3.8: Distribution of the 1000 subtasks over the involved clusters in the hybrid cloud at AWS EC2, IBE and LRZ. Although the connection of the AWS instances (c1.m) to the Redis-server is distinctly slower, they perform approximately as many subtasks as one would expect from their number and processor speed.

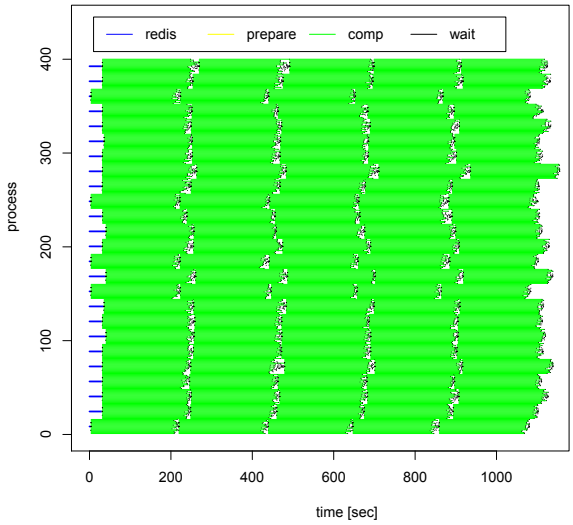


Figure 3.9: Gantt chart of *doRedis* / *multicore*-implementation of a resampling-based hyperparameter optimization on SuperMUC using 512 cores on 32 nodes. One can see that after a small loss of time at the startup, all processes are mainly performing the computations (green). Additionally, some time is lost at the end of each computation parts when the threads of the shared memory parallelization have to wait for each other.

3.5 Working with large datasets in R

Modern biological data are growing in size from generation to generation. The current next generation sequencing datasets often comprise Terabytes of data. The software engineering community has already devoted much attention to this problem in its discussion about 'Big Data'. Many distributed approaches to the handling of large datasets have been invented in this context. In some cases, the raw data can be preprocessed and the statistical analysis can be performed on distinctly smaller datasets. For microarrays as well as for many other data types, however, this preprocessing already involves statitistical methods and thus the handling of large datasets evolves into a problem for statisticians, too. At first, a small example is described where the need for new approaches to the handling of large datasets becomes evident. Subsequently, a recent approach to the processing of large datasets is presented which is already implemented in the programming language **R** and is thus easily deployable for statisticians who need to analyse such large datasets. Furthermore, this implementation is compared to a classical approach which is not available in **R**.

3.5.1 Motivating Example: Normalization of microarrays

In order to illustrate the increasing importance of handling large datasets in statistics, this subsection presents a motivating example based on the microarray preprocessing step. This preprocessing step has already been discussed in previous sections, for example in the context of cross-study validation in Chapter 2 as well as in the description of the workflows in *survHD*. As already mentioned in Chapter 2, the comparability of microarray studies and the transfer of prediction rules from one study to another is a delicate problem. Microarray raw data comprise several Gigabyte and have to be preprocessed whereby mainly statistical approaches are used. This preprocessing step was probably one of the first common statistical approaches where the raw data did not fit into the typical main memory of a modern desktop PC. Schmidberger et al. (2011) have even developed a Bioconductor package using a distributed data approach for the preprocessing step.

Gene expression microarrays usually consist of approximately 500000 measurements whereby between 30 and 50 measurements belong to a probeset. The gene expression measurements are influenced by several laboratory conditions. Additionally, biological phenomena just as non-specific binding can distort the gene expression values obtained using the arrays. Therefore, the microarrays have to be preprocessed before statistical high-level analysis. Due to the large number of values, this preprocessing is computationally intensive and requires large amounts of main memory. With regard to preprocessing, so-called batch effects play an important role. Microarrays obtained under equal laboratory conditions are sometimes more similar to each other and share common patterns. McCall et al. (2009) have shown that this effect can even lead to the problem that 2 microarrays from exactly the same tissue but processed under different laboratory conditions can be less similar than two arrays from different tissues but processed under the same laboratory conditions. Most papers (e.g. Kostka and Spang, 2008) treating this issue claim that multi-array approaches, which use the information from several arrays at the same time, can even

exacerbate this problem.

An experiment with 'artificial' batches

In order to analyze this concern, 'artificial' batches are created in a small simulation study. The following procedure was performed in each iteration b, $b = 1, \ldots, 50$, on a microarray dataset consisting of 47 microarrays collected in a colon cancer study (Ancona et al., 2006):

(1) A random subset $\mathbf{G}^b$ of 10 arrays was sampled without replacement. This sample represents an 'artificial' batch.

(2) The random subset $\mathbf{G}^b$ was preprocessed using each of the following methods m, $m = 1, \ldots, 4$ leading to four preprocessed random subsets $\tilde{\mathbf{G}}^b_m$, $m = 1, \ldots, 4$:

   1) robust multi-array average (RMA, Irizary et al., 2003)
   2) variance stabilizing normalization (Huber et al., 2002)
   3) addon RMA as proposed in Kostka and Spang (2008). Here, the preprocessing on $\mathbf{G}^1$ was used as reference batch whereas the batches $\mathbf{G}^b$, $b = 2, \ldots, 50$ were treated as addon batches. On these batches, the addon RMA method was applied which uses the information from the preprocessing on the reference batch.
   4) frozen RMA (multi-array version, McCall et al., 2009)

Thus, one obtains a total of 200 preprocessed artificial batches $\tilde{\mathbf{G}}^b_m$, $m = 1, \ldots, 4$, $b = 1, \ldots, 50$. Subsequently, for each method m, the corresponding preprocessed artificial batches $\tilde{\mathbf{G}}^b_m$, $b = 1, \ldots, 50$, were merged over the 50 iterations. This resulted in four multi-batch datasets including 50 preprocessed microarrays.

The actual data all belonged to a single 'biological batch', i.e. they were all processed under the same laboratory conditions. The only difference between the preprocessed arrays of an individual patient in the different $\tilde{\mathbf{G}}^b_m$, $b = 1, \ldots, 50$, consisted in the set of additional arrays which were normalized together with the respective microarray by a certain method m. The next step consisted in a cluster analysis which was performed separately on each of the 4 multi-batch datasets.

The two illustrations show the results for the cluster analysis on the merged dataset for $m = 2$ (RMA, Figure 3.10) and $m = 3$ (addon RMA, Figure 3.11). They provide evidence that the batch effect caused by multi-array normalization indeed exists.

The figures illustrate the results for a subset of 15 patients in order to enhance clearance. The patient number can be found on the left side of the heatmap and a color coded version is presented on the right side. Raws with the same patient number refer to exactly the same raw microarrays which have been normalized together with different additional arrays within a randomly assembled artificial batch $\mathbf{G}^b$. Consequently, microarrays belonging to the same patient can be expected to be assigned to the same cluster by the corresponding cluster analysis (these clusters are not of the same size since the different patient arrays were

not equally often part of the normalized subset). Particularly in the lower part of Figure 3.10, the small colored rectangles indicate that the patient clusters are not clearly separated but mixed up. This means that the cluster analysis classified preprocessed microarrays belonging to different patients as more similar than preprocessed microarrays belonging to the same patient.

This problem usually does not occur in the case of the different addon-normalization techniques, which have been introduced in Kostka and Spang (2008), or at least to a much lesser extent. These addon-techniques try to save the parameters of the preprocessing on a first dataset so that a second dataset can be preprocessed in an analogous manner. Instead of recomputing the preprocessing parameters on the second dataset, the archived parameters from the first dataset are used. Likewise, with the established frozenRMA approach (McCall et al., 2012), which integrates information from large datasets of the same architecture into the normalization in order to stabilize it, this phenomenon is observed only rarely. This is exemplified by the second figure where the cluster analysis mainly assigns microarrays from the same patient to the same cluster.

3.5.2 Implementation

As described in more detail in Appendix C, the microarray preprocessing step can be implemented using two different ways. One can either parallelize the preprocessing function itself, as implemented in the Bioconductor package *affyPara* (Schmidberger et al., 2011), or one can parallelize the large number of resampling iterations. The first strategy scales significantly poorer regardless of the platform used (see Appendix C). Usually, an efficient scalability can only be achieved up to 10 cores unless really large datasets (>300 microarrays) are normalized together. The most important advantage of this strategy, however, consists in its clearly lower main memory requirements, which in some cases renders the computation possible on clusters with less RAM where a normalization of all arrays at the same time cannot be performed at all. On these clusters, a combination of both strategies, resulting in a hybrid parallelization using the packages *doRedis* / *doSNOW* and multicore, can be very useful.

3.5.3 Parallel file access in R

The microarray preprocessing step provides the opportunity to analyze an important functionality with respect to the efficient implementation of statistical analyses of large datasets using a large number of processes, namely the parallel access of many **R**-processes to a single large dataset. In this context, the **R**-package *bigmemory* (Kane and Emerson, 2012) plays a crucial role because it implements a parallel access using memory attached files. The package was originally developed in order to alleviate a general problem concerning extremely large datasets in **R**. The problem is related to a limit of row or column indices in the **R** programming language which renders impossible to operate on datasets with more than $2^{31} - 1 \approx 2.15BN$ elements. However, it also provides the basic functionalities that

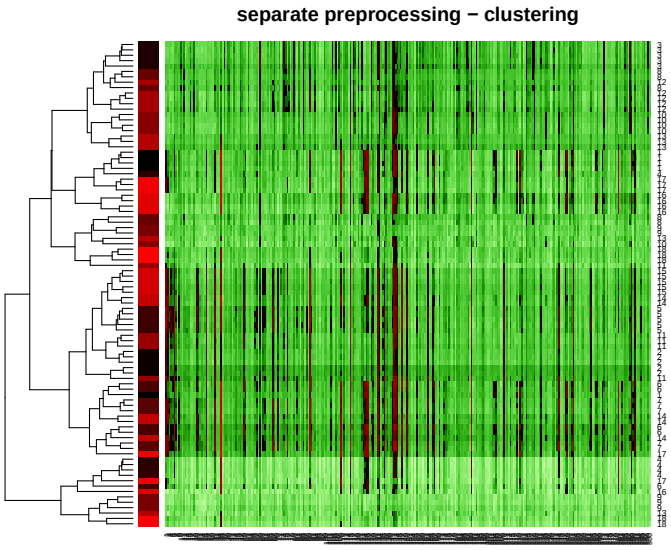


Figure 3.10: Cluster analysis of separately preprocessed microarrays (using RMA) in artificially created batches. Partially, microarrays belonging to the same patient are assigned to different clusters and often assigned to clusters containing arrays of the same artificial batch.

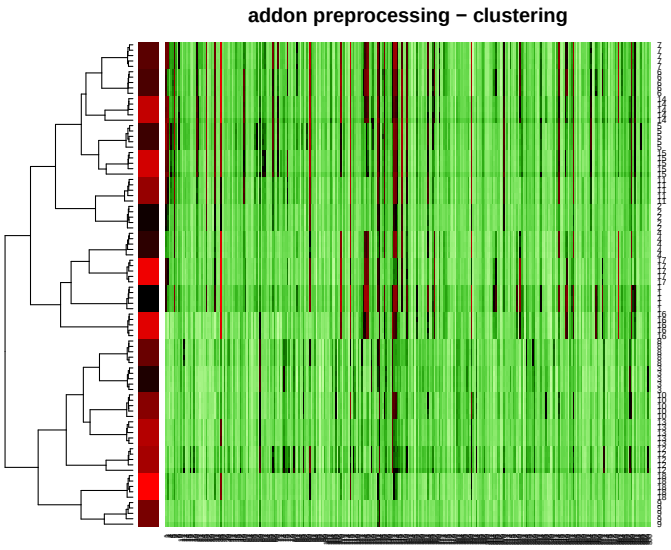


Figure 3.11: Cluster analysis of addon-preprocessed microarrays (using addon RMA, Kostka and Spang, 2008) in artificially created batches. Microarrays belonging to the same patient are always assigned to the same cluster.

are commonly implemented by parallel I/O standards like MPI-IO (see MPI-Forum, 2012) for C or Fortran.

The *bigmemory* implementation is more flexible in the sense that it does not rely on collective calls to *open* or *close* which enables a dynamic worker pool to access the dataset. This feature is crucial for the usage in combination with the *doRedis* approach described above since the workers are dynamically added or removed during the computation. The common MPI-IO approach would not be possible in this situation because here a synchronization of the different MPI tasks and collective calls are necessary.

The following illustration shows a comparison of two Gantt charts whereby the first figure refers to a *bigmemory* implementation in **R**, whereas the second one depicts an implementation using MPI-IO. The runtime is not exactly comparable since the second program was not run in **R** which currently lacks a feasible MPI-IO-implementation. Therefore, the second benchmark has been performed based on a Fortran 90 implementation of MPI-IO.

The results in Figures3.12 and 3.13 have been obtained during tests on the LRZ (Leibniz Supercomputing Center) SuperMIG HPC (high performance computing) system. They provide evidence for the good scalability of the approach and the significantly more flexible access with the *bigmemory* approach in comparison to the MPI-IO standard.

On the contrary, the same benchmarks look quite different at the IBE cluster (see Appendix C.2) which also uses a network file system which is, however, not dedicated to parallel file access. An experiment was run where each process at first had to *read* a 500x5000 matrix which was then used for constructing a classification model. Subsequently the error rates of these models on a test dataset had to be *written* into a common memory attached file (small write). Moreover the used dataset was written back into another file (large write). One can observe that the access to the common files is quite poorly coordinated. The first read operation lasts very long for several processses since the network file system is not capable of delivering the data to all processes at the same time. As soon as the first read operation is completed, the remaining ones are performed within seconds since the datasets are already in the main memory of the respective cluster node. The obvious pattern of groups of four similar tasks can be traced back to the fact that at the IBE cluster each computing node consists of four cores. For the writing operation, the problem is even more pronounced. Although the size of the file written is equal for all tasks, there is a strong variation between the large write operations.

If one compares these benchmarks to the one obtained on SuperMIG, it can be recognized that this behavior is not caused by processes locking the file since on both systems the same code has been used. The problem consists in the lower coordination capability of the network file system and the lower network capacity at the IBE cluster. This bottleneck becomes even more obvious in an experiment using matrices which are ten times larger than in the current test (see Appendix C.2). In this test the write operation was omitted since they caused severe disturbances in the operation of the cluster. One can observe, however, that the real computation tasks (green) hardly overlap in the corresponding Gantt chart. The first node (each node hosted two processes in this test) has already completed its ten computations and the corrresponding ten read operations when the second node started to construct the first model. In this case the actual computation times where quite small

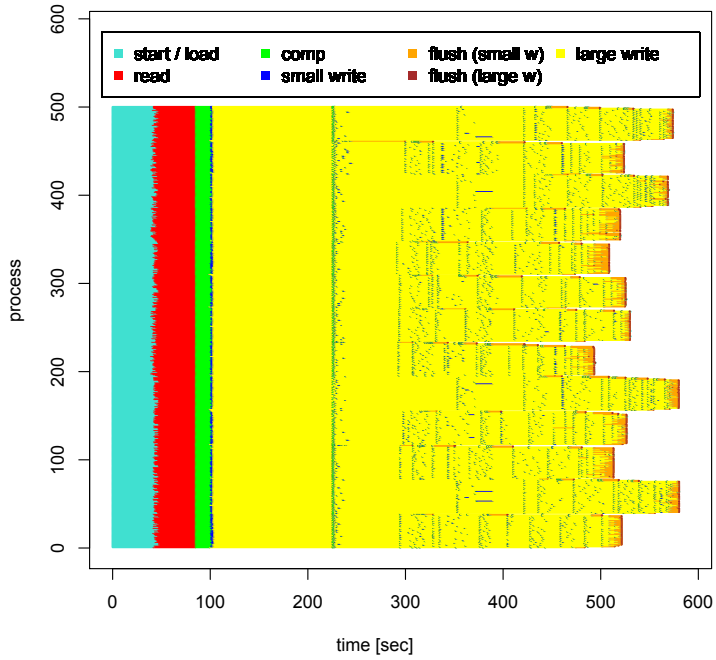


Figure 3.12: Gantt chart for a *bigmemory*-experiment with 500 processes at LRZ SuperMig reading and writing 500×5000 matrices using a 6GB memory-attached file.

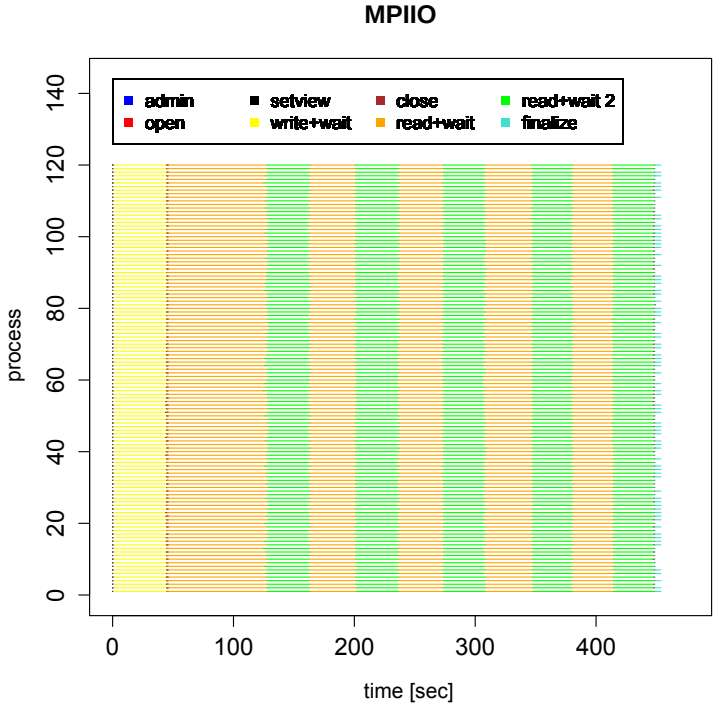


Figure 3.13: Gantt chart for MPI-IO with 160 processes at LRZ SuperMig writing and subequently reading 2 million integers in a single shared file.

in comparison to the time needed for reading the memory attached file. Consequently, a parallelization would not be necessary. Nonetheless, the networking problems, which would occur in a large parallel computation, become evident in this test.

In another minor test, the combination of the *doRedis* approach and *bigmemory* was tested in the context of a hybrid. Hereby, several instances from AWS EC2 (Amazon Web Services Elastic Compute Cloud 2) were integrated into the file system using sshfs. This 'Hybrid-File system' as well as the common access of cloud and cluster instances to a memory attached file could be implemented using these two **R**-packages. The performance, however, was rather poor as far as latency and connectivity were concerned.

Combining *bigmemory* and *doRedis*

In the context of microarray preprocessing, a combined approach of *doRedis* and *bigmemory* was tested. Instead of reading the required microarray data from disk, a large *bigmemory* matrix containing all the array data (approximately 4.2GB) was created which could be accessed by all processes at the same time. Each process normalized other parts of the matrix in this experiment, whereby these parts were randomly assembled and were not of equal size. The normalized microarrays were written to another memory attached file (ca. 6.7 GB) which was accessed in parallel as well. Each process performed this cycle of reading the raw data, normalizing them and writing the resulting arrays ten times. For the reading operations between 40MB and 433MB had to be read, whereas the write operations required between 3MB and 200MB. As illustrated Figure 3.14, the access to the two memory attached files is clearly less structured than in the previous examples. One notices that for several calls to the function flush (brown), processes, hosted on the same node, are synchronized after write operations. Additionally, it becomes evident that only the first read and write operations require relevant amounts of time whereas the later operations are processed significantly faster. Especially in the case of the read operations, only the first operation can be seen in the illustration. Since each process normalizes microarray subsets of different size, some processes have completed their task after 650 seconds whereas the last processes on the same node finish after approximately 1080 seconds. In this case, this finding can not be explained by the fact that certain processes block the data files when they read or write. It is merely caused by the different runtimes of the computation steps (green) which simply take less time with the processes that finish first.

3.6 Discussion

The new types of biological data which have emerged in the last two decades require interdisciplinary approaches and the efficient usage of computational resources. In this context, it is reasonable to focus more on the cycle time, consisting of preparation, data preprocessing, runtime and queuing time, as well as cost efficiency. Cloud computing can help to distinctly decrease two of these quantities. Cloudbursting, i.e. outsourcing simulation runs into the cloud if local resources are temporarily busy, is a good opportunity

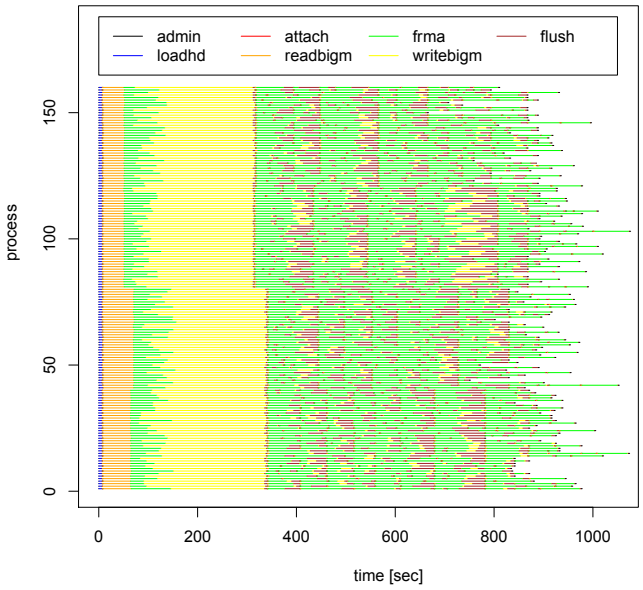


Figure 3.14: Parallel normalization of microarrays using a separate memory attached file for reading the raw data and writing the normalized arrays. The test was run on LRZ SuperMIG using the **R**-package *bigmemory* for 400 processes.

to avoid long queuing times. Moreover, it provides the opportunity to reduce the investments into new in-house resources by curtailing one's hardware stack to the requirements of the average workload whereas peak work loads can be handled using cloud resources. Hereby, the **R** language, for which implementations even for tablet PCs and iPods exist, can be considered a great advantage. As shown in the previous sections, it is quite easy to build a flexible worker pool of heterogeneous cloud resources in order to perform a large number of independent jobs as they can be found with most modern statistical methods, e.g. resampling or permutation tests. The combination of a Redis-server and an **R** Redis-connector, which has been proposed in section 3.4.5, represents a light-weight, easy-to-use opportunity to realize a cloudbursting strategy. Moreover, it can also be used on classical cluster computer resources. On the LRZ Supermuc HPC system, benchmarks running on up to several thousands of cores have been successfully performed with this approach, for example in the simulations for Chapter 2. This specific case also featured the good property that only small data files had to be read from disk and that these data were quite small. Thus, this project would have been certainly realizable in the cloud, too. Here, some experiences concerning the workflow in the cloud and on computer clusters shall be discussed as well since they might also have an impact on one's individual preference of cluster or cloud computing. A very convenient feature of utility computing in the cloud is its high flexibility. One can build one's own computer cluster at EC2 whenever necessary and one can customize it accordingly to the needs of specific analyses. Virtualization and virtual images also provide the opportunity to use different operating systems for different jobs which can be very useful if for certain workflows specific codes or software has to be used which is dependent on certain operating systems or even certain versions of an operating system. Moreover one has the opportunity to rent single instances for sequential code, e.g. if the memory requirements of the computations might exceed the capacity of the in-house machines. Another advantage is that one has full control over the ordered resources and one does not have to resort to job scheduling systems coordinating the needs of the different computer cluster users. On the contrary, the workflow on cloud resources is remarkably changed by the fact that one has to pay for each computation. One always reflects twice whether a specific computation is really necessary which limits the opportunity to test and experiment. Of course, this will induce researchers to plan their computations more elaborately and avoid unnecessary tests which might also be considered a positive side-effect. Cloud computing, however, also has some hurdles for starters. Simple mistakes like forgetting to shut down instances or moving results to the own computer can easily produce unnecessary costs. Moreover, one must always estimate time and memory needs in order to decide which instances are adequate for the task at hand. Consequently, AWS and similar services are currently probably not the best place for the first steps in parallel computing. Another problem concerns experienced users as well. After finding an error or recognizing that a certain part of one's algorithm has to be changed, all the computations have to be repeated. Also, discovered problems in used packages can force a recalculation. In an advanced stage of a project this means that the computation costs will effectively multiply. After all, utility computing in the cloud seems to be a tool which has to be applied in a considered way.

In contrast to the computations discussed above, I/O intensive projects, as for example the microarray normalization project in Section 3.5.1, pose a more difficult challenge for the cloud approach. The corresponding section has illustrated that such analyses can distinctly benefit from a strong network file system which is usually not available on cloud resources. Even the network file system at the IBE-cluster faced clear difficulties and performance losses in these benchmarks. Some of these problems might be overcome using distributed data approaches e.g. using Hadoop. However, such approaches are more difficult to implement and only realizable if one invests larger amounts of time into the programming of such analyses. Hence, for I/O intensive projects, the classical cluster approach still seems to be the reasonable choice.

Finally, **R**-packages can support running statistical analyses in cluster or cloud by embedding mechanisms for parallelization directly into the packages. The API-package for survival analysis proposed in Section 3.2, constitutes an example in this context. It includes several functions where parallelization can be easily embedded into the package. Either more cluster-based solutions, like *snow* or the flexible *doRedis* approach can be used for the parallelization whereby combinations are possible which can also pave the way to meta-clouds, i.e. to running jobs on cloud resources from several providers as AWS EC2 or OpenStack. The simulations presented in Chapter 2 could easily be implemented in such a meta-cloud using the *doRedis* approach in combination with the package *survHD*.

Summary & Outlook

The first chapter has focused on the term wrapper algorithm as a combination of algorithm and an appropriate procedure for tuning its hyper parameters. This definition can also be extended to the case where the 'tuning parameters' represent different algorithms, i.e. it can be used in all types of model selection. Further, this chapter has tried to shed light on the underlying conflict of conditional and unconditional error rate which seems to be caused by a discrepancy of interests between physicians and statisticians or machine learners. The review process of the corresponding paper (Bernau et al., 2013), in which it was barely possible to convince one of the reviewers that the unconditional error rate might be of any interest at all, has shown that this conflict has not been covered in current research to an appropriate extent. This is all the more understandable because for most traditional statistical approaches the difference between conditional and unconditional error can be expected to be small. Especially in the model selection setup, however, ignoring the effects of retraining and retuning the model is not adequate. The models, chosen on different training datasets, originate in different algorithms and cannot be assumed to lead generally to very similar prediction performances. It has been shown that quantities like the unconditional error rate of a wrapper algorithm can be paralleled to decision theory and the risk of strategies. Moreover, an asymptotic consideration has affirmed the hypothesis that the unconditional error rate of such a wrapper algorithm can be considered the estimation target of the well established nested cross-validation estimator. This hypothesis has already been formulated in Varma and Simon (2006) but has not been justified in more detail there.

Furthermore, a method for the estimation of this quantity has been proposed, avoiding the high computational cost of nested cross-validation, which also approaches the correction of the optimization addressed by NCV. Although one might discuss whether the unconditional error rate might be of any value for a physician, the two WMC variants as well as NCV are certainly useful for this professional group, too. Eventually, both methods try to correct for the tuning bias which has been a source for overoptimism, especially in the early days of microarray studies (see e.g. Ambroise and McLachlan, 2002; Varma and Simon, 2006). A proper assessment of the real prediction quality certainly adds some value for physicians although independent validation on future studies will always be the best choice for an estimation of the prediction quality of a concrete model.

The second chapter covered a recent validation scheme, cross-study validation, which is especially useful in the context of wrapper algorithms and microarray studies. Its design can safeguard against the optimization bias due to its strict separation of training / tuning

dataset on the one hand and validation datasets on the other hand. The conflict between evaluating concrete models or wrapper algorithms, i.e. the discrepancy between conditional error of a model or unconditional error of a wrapper algorithm, has again played a crucial role because the performance scores have included estimates for different tuning parameters also in this case. However, there have also been other remarkable findings. Even with the application of several tools for the increase of comparability between studies, within-study validation has lead to distinctly higher performance estimates. Moreover the resulting ranking of the algorithms has been clearly affected by the definition of the scope of the prediction. In the simulation study, some algorithms like CoxBoost have performed better in within-study validation whereas other algorithms like the Plusminus have been better in cross-study-validation. This has provided evidence for the hypothesis that there exist specialist and generalist algorithms, as proposed in the introduction to this chapter. Finally, the analyses have provided results suggesting that in this validation scheme, the usage of scores like the third quartile, which in a certain sense account for the presence of 'bad studies', is recommendable. Additionally, it has also been found that within-study validation and cross-study validation have been by far less correlated in this case than one might have assumed. For example, for no algorithm a distinctly lower level for the performance estimates has been found on those studies where cross-study validation performance had suggested that the prediction methods are merely better than random guessing. Of course, these results still have to be affirmed by future work. Nonetheless, they have helped to gain insights into this complex problem of cross-study prediction.

Finally, the third chapter has covered several computational aspects which have been important for the realization of the research projects in this thesis. The chapter has started with the description of the R-package *survHD* (survival analysis on high-dimensional data) which has strongly supported the realization of the simulations performed in the second chapter. Its main objective is to provide a common interface to various methods for survival analysis and their evaluation. Right now it encompasses six different algorithms and six evaluation scores. Its split into a thin, easily maintainable core package and the add-on package *survHDExtra* already underlines the intention to extend the package in the future. For this purpose, an interface for user-defined algorithms has been embedded into *survHD* which will hopefully enable the package to continually improve by the inclusion of algorithms developed by its users.

The second part of the chapter has provided a short introduction to parallel computing in **R** using the packages *Rmpi*, *snow* and *pbdMPI*. The latter package has abandoned the master and worker concept of the previous packages and thus provides new opportunities for programming which more resemble the one used in typical MPI programs. These parallel methods have subsequently been tested in a case study which has provided evidence that the analysis of preprocessed microarray data can be efficiently performed in cloud environments like AWS EC2. With the combination of a Redis-server and its **R**-connector *doRedis*, it has also introduced a simple way for the implementation of a hybrid cloud in which computational resources with heterogeneous hardware can be combined even without the use of virtualization in the context of statistical analysis. Finally, the last part of the chapter has described several techniques which can be helpful when handling data as large

as microarray raw data. It has presented a combination of the packages *bigmemory* and *doRedis*, and has compared this approach to a typical MPI-IO implementation.

Although *survHD* and this entire thesis has primarily emerged in the context of microarray data analysis, most of the treated subjects can be considered to be useful for high-dimensional statistical analysis in general. The upcoming of next generation sequencing will ensure that this topic will probably even gain in importance in the near future. This will likely affect the computational side with the already well-established discussion about 'big data', as well as the analysis part where proper validation and the tuning of wrapper-algorithms will play an important part in future research. This thesis has hopefully contributed to this quickly evolving field of high-dimensional analysis.

Appendix A

Correcting the optimal resampling-based error rate by estimating the error rate of wrapper algorithms

A.1 Additional results on real datasets of Chapter 1

Study design

The new estimator $\widehat{Err}_{WMC}$ (Eq. (1.10)) and its shrunken version $\widehat{Err}_{WMCS}$ (Eq. (1.12)) are first evaluated on several experimental microarray datasets and compared to the widely established estimator $\widehat{Err}_{ICV}$ (Eq. (1.4)) and to the naive raw mean estimator $\widehat{Err}_{RawMean}$ (Eq. (1.9)). The study is based on four microarray datasets: a colon cancer dataset (Alon et al., 1999) included in Bioconductor package *colonCA* with $n = 62$ diseased or healthy tissues and $p = 1991$ variables, a prostate cancer dataset (Singh et al., 2002) with $n = 102$ diseased or healthy patients and $p = 12625$ variables, a leukemia dataset (Golub et al., 1999) included in Bioconductor package *CMA* with $n = 38$ patients with two different leukemia subtypes and $p = 3051$ variables, and an ALL-leukemia dataset included in Bioconductor package *ALL* (Chiaretti et al., 2004) with $n = 100$ patients with and without relapse and $p = 12625$ variables. Moreover, modified versions of these four datasets are considered which are obtained by replacing the response Y by a randomly generated Bernoulli distributed variable $Y' \sim \mathcal{B}(1, 0.5)$. These modified datasets are denoted as "non-informative" setup, in contrast to the original version of the datasets including "informative" predictors. More details on the study design can be found in Section 1.4.1.

A.1.1 Alon data

Results

knn-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.170	0.180	0.172	0.180	0.163	0.240
absdicv	0.000	0.010	0.006	0.010	0.008	0.070
sd	0.011	0.008	0.009	0.008	0.009	0.009

knn-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.499	0.495	0.487	0.495	0.473	0.517
absdicv	0.000	0.009	0.013	0.009	0.026	0.018
sd	0.062	0.062	0.060	0.062	0.058	0.058

knn-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.184	0.193	0.187	0.193	0.175	0.239
absdicv	0.000	0.009	0.005	0.009	0.009	0.055
sd	0.007	0.007	0.006	0.006	0.007	0.008

knn-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.499	0.495	0.491	0.495	0.478	0.514
absdicv	0.000	0.006	0.009	0.006	0.021	0.016
sd	0.053	0.053	0.053	0.053	0.056	0.046

pls-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.176	0.183	0.168	0.186	0.149	0.210
absdicv	0.000	0.007	0.009	0.010	0.028	0.034
sd	0.011	0.008	0.009	0.008	0.009	0.008

pls-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.502	0.497	0.483	0.504	0.465	0.538
absdicv	0.000	0.011	0.019	0.014	0.037	0.037
sd	0.052	0.050	0.053	0.050	0.053	0.046

pls-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.185	0.197	0.180	0.199	0.158	0.226
absdicv	0.000	0.012	0.006	0.015	0.027	0.041
sd	0.010	0.007	0.007	0.006	0.008	0.007

pls-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.508	0.506	0.498	0.509	0.481	0.533
absdicv	0.000	0.008	0.012	0.010	0.027	0.025
sd	0.044	0.043	0.046	0.042	0.049	0.042

sel-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.164	0.179	0.163	0.190	0.142	0.257
absdicv	0.000	0.016	0.005	0.026	0.021	0.093
sd	0.010	0.010	0.009	0.007	0.009	0.014

sel-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.492	0.482	0.462	0.488	0.440	0.531
absdicv	0.000	0.016	0.030	0.018	0.051	0.042
sd	0.049	0.048	0.048	0.047	0.049	0.054

sel-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.178	0.204	0.185	0.205	0.159	0.264
absdicv	0.000	0.025	0.007	0.027	0.020	0.085
sd	0.008	0.007	0.007	0.006	0.007	0.011

sel-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.497	0.494	0.480	0.496	0.460	0.531
absdicv	0.000	0.009	0.017	0.011	0.037	0.035
sd	0.051	0.046	0.049	0.044	0.053	0.039

Table A.1: Average corrected errors, mean absolute difference to ICV, and standard deviations (over $T = 50$ replications) for all setups on the Alon dataset. The tables include the kNN (knn) and PLSLDA (pls) setups as well as the selection setups (sel). For each setup two informative (info) and two non-informative (noif) setups are considered with two different proportions of observations in the learning sets (0.63 and 0.8).

Shrinkage factor

The following histograms display the values of the shrinkage factors computed within the WMCS procedure for sample sizes $n = 40$, $n = 60$ and $n = 80$.

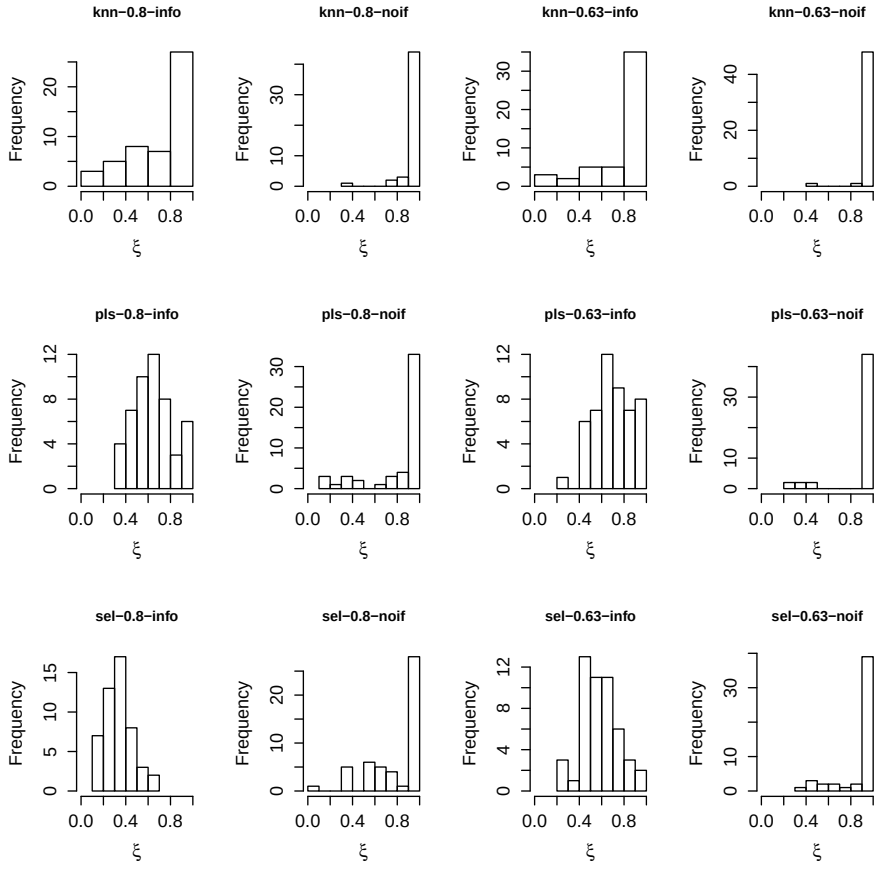


Figure A.1: Distribution of the shrinkage factor ξ for all simulation setups on the Alon dataset. The figure includes the kNN (knn) and PLSLDA (pls) setups as well as the selection setup (sel). For each setup two informative (info) and two non-informative (noif) setups are considered with two different proportions of observations in the learning sets (0.63 and 0.8).

Summary

In the informative setups, one can observe several cases in which WMC yields corrected errors higher than those obtained by ICV which is not in accordance with its general optimistic tendency. Consequently, WMCS, which is in practice lower-bounded by WMC (exceptions are theoretically possible), performs even worse. If one takes a look at the distribution of the shrinkage ξ in this case one can observe a clear difference to its characteristic distribution in non-informative setups. Nonetheless, the estimates are quite close

to the raw mean. This is the main problem of WMCS: in intermediate cases the measure ξ has not enough power to push the WMCS results closer to those of WMC which performs well in these cases. This problem seems to be caused by an overestimation of the expected bias ζ which is itself caused by a less accurate variance estimation especially in the 0.63-setups. Also in the tuning setups, the shrinkage factor is overestimated, which shrinks the WMCS estimates close to the raw mean. Again the differences among the different tuning parameters seem to be too small in comparison to the estimated variance. WMCS is clearly better than WMC in non-informative setups where WMC is too optimistic.

A.1.2 Singh data

Results

knn-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.090	0.092	0.086	0.095	0.079	0.111
absdicv	0.000	0.004	0.006	0.006	0.011	0.020
sd	0.007	0.006	0.005	0.005	0.005	0.006

knn-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.498	0.496	0.492	0.496	0.482	0.510
absdicv	0.000	0.005	0.007	0.005	0.016	0.013
sd	0.042	0.042	0.041	0.042	0.041	0.042

knn-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.102	0.104	0.097	0.106	0.090	0.121
absdicv	0.000	0.005	0.005	0.006	0.012	0.019
sd	0.006	0.005	0.005	0.005	0.005	0.006

knn-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.493	0.493	0.491	0.493	0.482	0.504
absdicv	0.000	0.004	0.004	0.004	0.011	0.011
sd	0.035	0.035	0.034	0.036	0.033	0.035

pls-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.087	0.092	0.080	0.091	0.075	0.180
absdicv	0.000	0.005	0.007	0.005	0.012	0.093
sd	0.006	0.005	0.005	0.005	0.005	0.009

pls-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.494	0.490	0.482	0.492	0.468	0.524
absdicv	0.000	0.009	0.013	0.011	0.026	0.032
sd	0.046	0.042	0.042	0.043	0.042	0.047

pls-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.092	0.099	0.089	0.098	0.081	0.181
absdicv	0.000	0.007	0.004	0.007	0.010	0.089
sd	0.005	0.004	0.004	0.004	0.004	0.009

pls-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.497	0.494	0.488	0.497	0.475	0.517
absdicv	0.000	0.007	0.009	0.010	0.023	0.020
sd	0.031	0.028	0.031	0.026	0.033	0.027

sel-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.097	0.104	0.092	0.133	0.083	0.319
absdicv	0.000	0.008	0.005	0.037	0.014	0.223
sd	0.006	0.006	0.005	0.005	0.005	0.011

sel-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.504	0.496	0.479	0.503	0.463	0.541
absdicv	0.000	0.012	0.025	0.015	0.041	0.037
sd	0.044	0.039	0.040	0.037	0.041	0.037

sel-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.107	0.123	0.106	0.141	0.095	0.312
absdicv	0.000	0.016	0.003	0.035	0.012	0.205
sd	0.004	0.005	0.004	0.004	0.004	0.012

sel-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.498	0.496	0.484	0.503	0.466	0.528
absdicv	0.000	0.008	0.014	0.011	0.032	0.030
sd	0.045	0.044	0.047	0.038	0.050	0.034

Table A.2: Average corrected errors, mean absolute difference to ICV, and standard deviations (over $T = 50$ replications) for all setups on the Singh dataset. The tables include the kNN (knn) and PLSLDA (pls) setups as well as the selection setups (sel). For each setup two informative (info) and two non-informative (noif) setups are considered with two different proportions of observations in the learning sets (0.63 and 0.8).

Shrinkage factor

The following histograms display the values of the shrinkage factors computed within the WMCS procedure for sample sizes $n = 40$, $n = 60$ and $n = 80$.

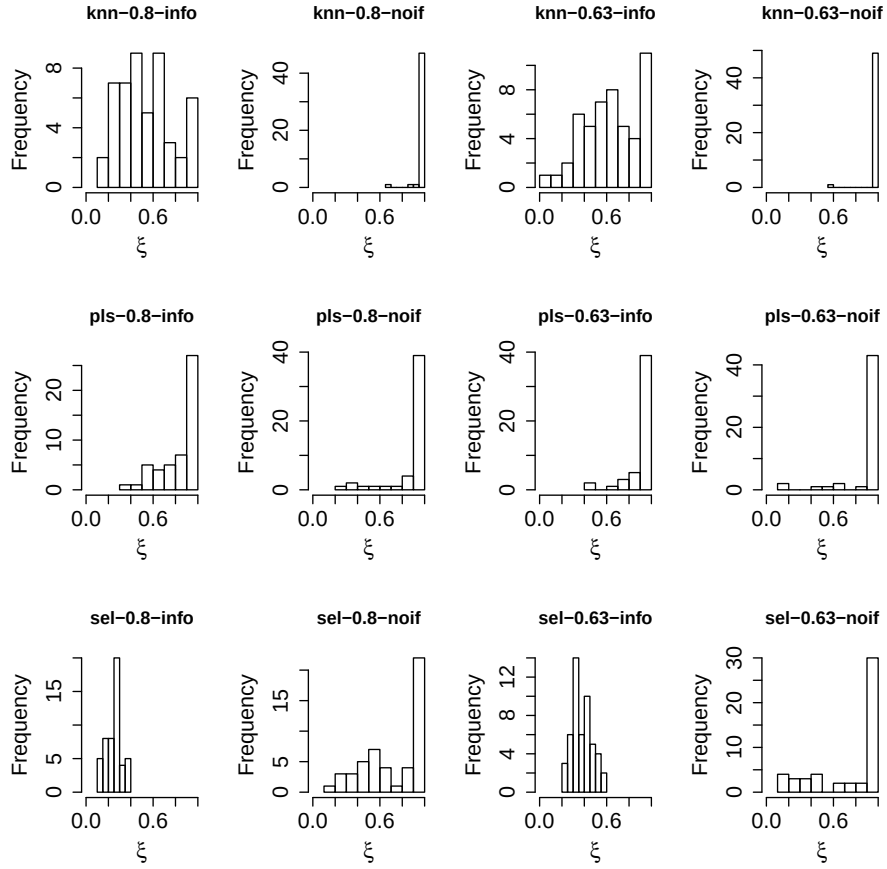


Figure A.2: Distribution of the shrinkage factor ξ for all simulation setups on the Singh dataset. The figures includes the kNN (knn) and PLSLDA (pls) setups as well as the selection setup (sel). For each setup two informative (info) and two non-informative (noif) setups are considered with two different proportions of observations in the learning sets (0.63 and 0.8).

Summary

WMC consistently produces smaller estimates than ICV whereas the WMCS method usually slightly overestimates the optimization bias. In the non-informative setups one can observe an overoptimistic tendency for WMC which is almost completely corrected by the shrinkage approach. The distribution of the shrinkage factor ξ is remarkably different for informative and non-informative setups which is probably the main reason for the good results of the WMCS method. In the pls setups this difference between shrinkage factors of informative and non-informative setups cannot be observed and indeed the raw mean is a good approximation of ICV results in these cases. In the kNN-setups, WMCS estimates are close to the raw mean although one can see that the distribution of the shrinkage factor ξ is different from the non-informative case. This indicates that a nonlinear transformation of

ξ, probably a sigmoidal transformation, might be useful to direct the WMCS more towards the WMC estimates which seems to be sensible in almost all cases where the mean of its distribution is approximately 0.5 or 0.6.

A.1.3 Golub data

Results

knn-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.071	0.071	0.064	0.074	0.048	0.115
absdicv	0.000	0.006	0.008	0.006	0.023	0.044
sd	0.010	0.009	0.009	0.009	0.009	0.014

knn-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.488	0.479	0.472	0.479	0.456	0.503
absdicv	0.000	0.014	0.018	0.014	0.032	0.021
sd	0.095	0.099	0.097	0.099	0.096	0.096

knn-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.072	0.095	0.083	0.107	0.057	0.222
absdicv	0.000	0.023	0.011	0.035	0.015	0.151
sd	0.009	0.009	0.007	0.010	0.008	0.018

knn-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.499	0.491	0.487	0.491	0.474	0.508
absdicv	0.000	0.011	0.013	0.011	0.025	0.014
sd	0.067	0.071	0.071	0.071	0.072	0.069

pls-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.048	0.034	0.030	0.035	0.024	0.041
absdicv	0.000	0.014	0.018	0.014	0.024	0.008
sd	0.009	0.006	0.005	0.006	0.005	0.007

pls-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.500	0.489	0.479	0.495	0.463	0.533
absdicv	0.000	0.016	0.022	0.016	0.037	0.037
sd	0.083	0.083	0.084	0.083	0.084	0.082

pls-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.047	0.040	0.036	0.040	0.030	0.043
absdicv	0.000	0.008	0.011	0.007	0.017	0.004
sd	0.006	0.005	0.004	0.005	0.005	0.005

pls-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.524	0.520	0.516	0.522	0.503	0.544
absdicv	0.000	0.011	0.012	0.012	0.022	0.023
sd	0.075	0.075	0.077	0.075	0.078	0.075

sel-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.026	0.021	0.018	0.061	0.004	0.226
absdicv	0.000	0.006	0.008	0.035	0.022	0.200
sd	0.006	0.005	0.003	0.005	0.002	0.021

sel-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.498	0.487	0.466	0.493	0.443	0.536
absdicv	0.000	0.014	0.031	0.015	0.055	0.040
sd	0.071	0.072	0.090	0.067	0.072	0.070

sel-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.036	0.041	0.028	0.075	0.015	0.279
absdicv	0.000	0.008	0.008	0.039	0.020	0.244
sd	0.005	0.009	0.004	0.005	0.004	0.013

sel-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.504	0.501	0.488	0.502	0.464	0.538
absdicv	0.000	0.011	0.017	0.012	0.040	0.034
sd	0.063	0.064	0.067	0.063	0.071	0.058

Table A.3: Average corrected errors, mean absolute difference to ICV, and standard deviations (over $T = 50$ replications) for all setups on the Golub dataset. The tables include the kNN (knn) and PLSLDA (pls) setups as well as the selection setups (sel). For each setup two informative (info) and two non-informative (noif) setups are considered with two different proportions of observations in the learning sets (0.63 and 0.8).

Shrinkage factor

The following histograms display the values of the shrinkage factors computed within the WMCS procedure for sample sizes $n = 40$, $n = 60$ and $n = 80$.

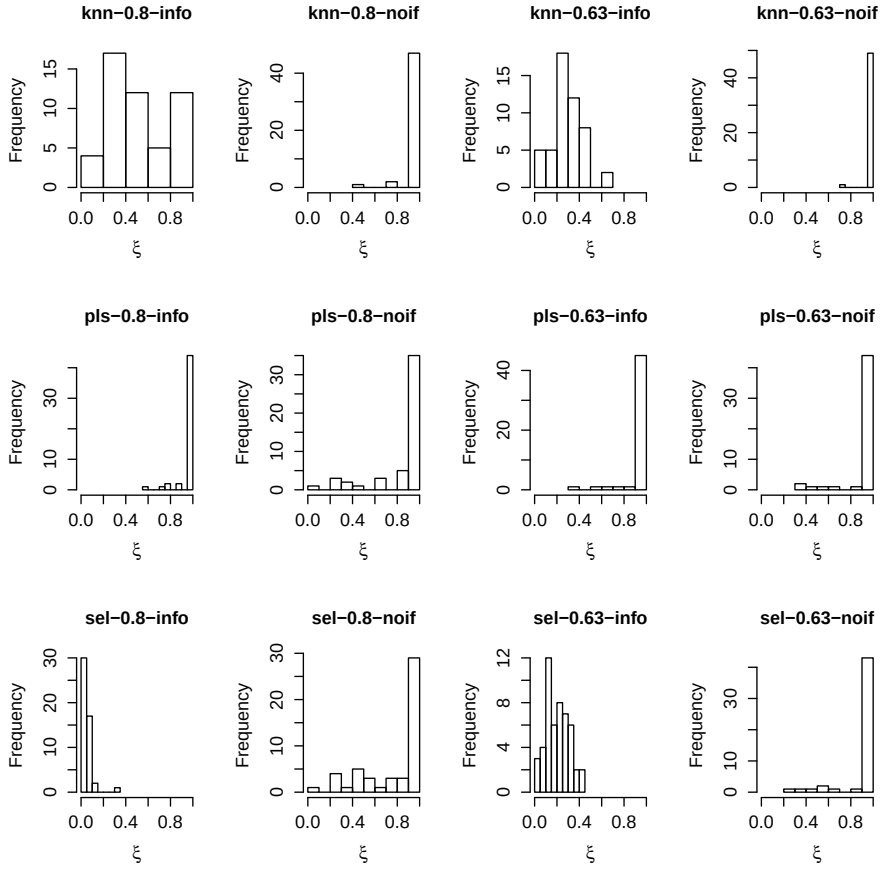


Figure A.3: Distribution of the shrinkage factor ξ for all simulation setups on the Golub dataset. The figures include the kNN (knn) and PLSLDA (pls) setups as well as the selection setup (sel). For each setup two informative (info) and two non-informative (noif) setups are considered with two different proportions of observations in the learning sets (0.63 and 0.8).

Summary

The results on the Golub dataset are quite heterogeneous and difficult to interpret. The largest difference between WMCS and ICV can be observed in the knn-0.63-info setup where WMCS is clearly too pessimistic. Interestingly, WMCS yields very good results in the corresponding 0.8-setup. In the informative pls setups ICV estimates are higher than the maximum error rate which is a theoretically questionable estimate. In the informative selection setups both WMCS variants have a mean absolute difference to ICV smaller than 0.01. With the exception of the pls setups the distributions of the shrinkage factor ξ have clearly different characteristics for informative and non-informative setups. Nonetheless, WMCS estimates are quite close to the raw mean in the tuning setups, whereby the knn-0.8 setup epitomizes an exception.

A.1.4 Chiaretti data

Results

knn-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.398	0.394	0.393	0.394	0.383	0.414
absdicv	0.000	0.006	0.007	0.006	0.015	0.017
sd	0.010	0.008	0.008	0.008	0.008	0.011

knn-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.498	0.497	0.492	0.497	0.481	0.511
absdicv	0.000	0.006	0.008	0.007	0.017	0.014
sd	0.044	0.044	0.043	0.044	0.043	0.043

knn-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.399	0.399	0.399	0.399	0.388	0.417
absdicv	0.000	0.004	0.032	0.004	0.011	0.018
sd	0.008	0.006	0.009	0.006	0.007	0.007

knn-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.504	0.504	0.500	0.504	0.487	0.519
absdicv	0.000	0.007	0.006	0.007	0.017	0.016
sd	0.051	0.051	0.050	0.051	0.050	0.046

pls-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.431	0.434	0.417	0.441	0.398	0.459
absdicv	0.000	0.007	0.014	0.010	0.033	0.028
sd	0.009	0.008	0.008	0.009	0.008	0.009

pls-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.495	0.492	0.482	0.498	0.469	0.523
absdicv	0.000	0.006	0.014	0.011	0.026	0.028
sd	0.042	0.041	0.042	0.042	0.043	0.043

pls-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.438	0.443	0.431	0.444	0.414	0.456
absdicv	0.000	0.006	0.007	0.007	0.025	0.018
sd	0.007	0.004	0.005	0.004	0.006	0.006

pls-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.501	0.499	0.495	0.500	0.483	0.518
absdicv	0.000	0.007	0.007	0.008	0.018	0.018
sd	0.043	0.040	0.042	0.040	0.044	0.043

sel-0.8-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.398	0.403	0.383	0.420	0.365	0.452
absdicv	0.000	0.008	0.015	0.022	0.033	0.054
sd	0.010	0.010	0.009	0.007	0.009	0.010

sel-0.8-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.505	0.498	0.484	0.501	0.468	0.532
absdicv	0.000	0.011	0.021	0.012	0.037	0.028
sd	0.055	0.052	0.053	0.049	0.055	0.051

sel-0.63-info	ICV	WMCS	WMC	Raw	Min	Max
mean	0.399	0.407	0.387	0.419	0.365	0.455
absdicv	0.000	0.009	0.012	0.020	0.033	0.056
sd	0.007	0.006	0.006	0.004	0.005	0.007

sel-0.63-noif	ICV	WMCS	WMC	Raw	Min	Max
mean	0.498	0.497	0.487	0.498	0.470	0.521
absdicv	0.000	0.005	0.011	0.006	0.028	0.024
sd	0.041	0.038	0.040	0.038	0.042	0.038

Table A.4: Average corrected errors, mean absolute difference to ICV, and standard deviations (over $T = 50$ replications) for all setups on the Chiaretti dataset. The tables include the kNN (knn) and PLSLDA (pls) setups as well as the selection setups (sel). For each setup two informative (info) and two non-informative (noif) setups are considered with two different proportions of observations in the learning sets (0.63 and 0.8).

Shrinkage factor

The following histograms display the values of the shrinkage factors computed within the WMCS procedure for sample sizes $n = 40$, $n = 60$ and $n = 80$.

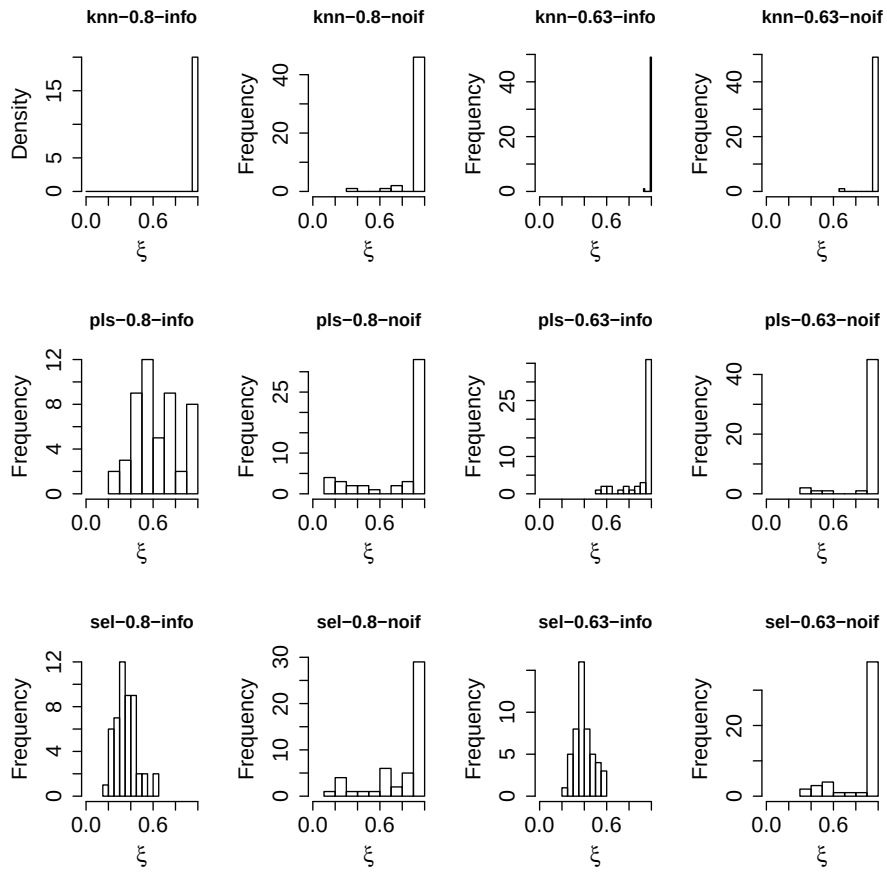


Figure A.4: Distribution of the shrinkage factor ξ for all simulation setups on the Chiaretti dataset. The figures include the kNN (knn) and PLSLDA (pls) setups as well as the selection setup (sel). For each setup two informative (info) and two non-informative (noif) setups are considered with two different proportions of observations in the learning sets (0.63 and 0.8).

Summary

On the ALL data WMCS clearly performs best. The only exceptions are both informative knn setups where the shrinkage factor is overestimated, thus producing results almost identical to the raw mean. WMC has an optimistic tendency which can be almost completely corrected on the non-informative setups by applying the shrinkage approach.

A.2 Additional results of the simulations in Chapter 1

In the simulation study, five different data generating processes (DGPs) have been used, including DGPs with correlated and uncorrelated predictors as well as different signal strengths and sample sizes. $T = 200$ datasets are randomly drawn from each setting. The first of the five DGPs has already been presented in Section 1.4.2 whereas the remaining four DGPs will be described here. For each dataset the true error rate of the wrapper algorithm is approximated on a large validation set. Thus, the methods (WMC and WMCS) can be compared to the existing methods (ICV and raw mean) based on data, where the true value of the parameter of interest $Err = \mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi)$ –the unconditional error rate of the wrapper algorithm ϕ– is known. For simulating the predictors, a block design is used because drawing data from a 2000-dimensional multivariate normal distribution is computationally intractable. That is why blocks of 100 or 200 correlated predictors are simulated and subsequently combined. The correlations between variables from different blocks are thus zero. Since the selection setups have been found to produce larger optimization biases and are generally more challenging, the simulation focuses on this setup. See Section 1.4.1 for details on the considered methods. The proportion of observations in the training data is set to $q = 0.8$. Note that the true parameter to be estimated primarily depends on $n_L = q \cdot n$ (not n itself), i.e. changing the proportion q also changes the true parameter. In order to analyze the behavior of the different estimators for different sample sizes, n is set successively to $n = 40$, $n = 60$ or $n = 80$ observations.

Approximating the true value of $Err = \varepsilon^{n_L}(\phi)$

The estimates produced by the investigated methods (WMC, WMCS, ICV, raw mean) are compared to the true value of the parameter of interest $Err = \mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi)$. In the simulation, this true value is approximated based on 1000 independent randomly generated datasets S of size $n_L = 0.8n$ (corresponding to the number of training observations in the outer subsampling loop). Each dataset S is used both for determining $k^*(S)$ (based on cross-validation with folds of approximate size 5) and for constructing the prediction rule using method $k^*(S)$, i.e. the wrapper algorithm is applied on each of these datasets. The 1000 constructed prediction rules are subsequently evaluated using a large independent test dataset of size 20000 generated from the same data generating process, and $\varepsilon^{n_L}(\phi)$ is estimated as the average error of the 1000 prediction rules on this test dataset.

Correlated Gaussian data with moderate signal

Study design

In the following setup, the strength of the signal is decreased in comparison to the setup in Section 1.4.2 by multiplying the vector of mean differences $\boldsymbol{\delta}$ used in the last section by 0.7

and by considering only 100 informative predictors instead of 200. The block size is also decreased for the non-informative predictors to 100, i.e. for the first block the covariance is estimated from the predictors ranked 101 to 200 in the Singh dataset (according to their between group t-statistic), and so on.

Results

The boxplots in Figure A.5 display the estimated errors obtained for the 200 simulated datasets with the five considered methods as well as the minimal and maximal error over the K investigated methods (top: $n = 40$, middle: $n = 60$, bottom: $n = 80$). The (approximated) target value $\varepsilon^{n_L}(\phi)$ is represented as an horizontal gray line. The exact values of $\varepsilon^{n_L}(\phi)$ can be found in A.6. The histograms in Figure A.6 display the values of the shrinkage factors computed within the WMCS procedure for sample sizes $n = 40$, $n = 60$ and $n = 80$.

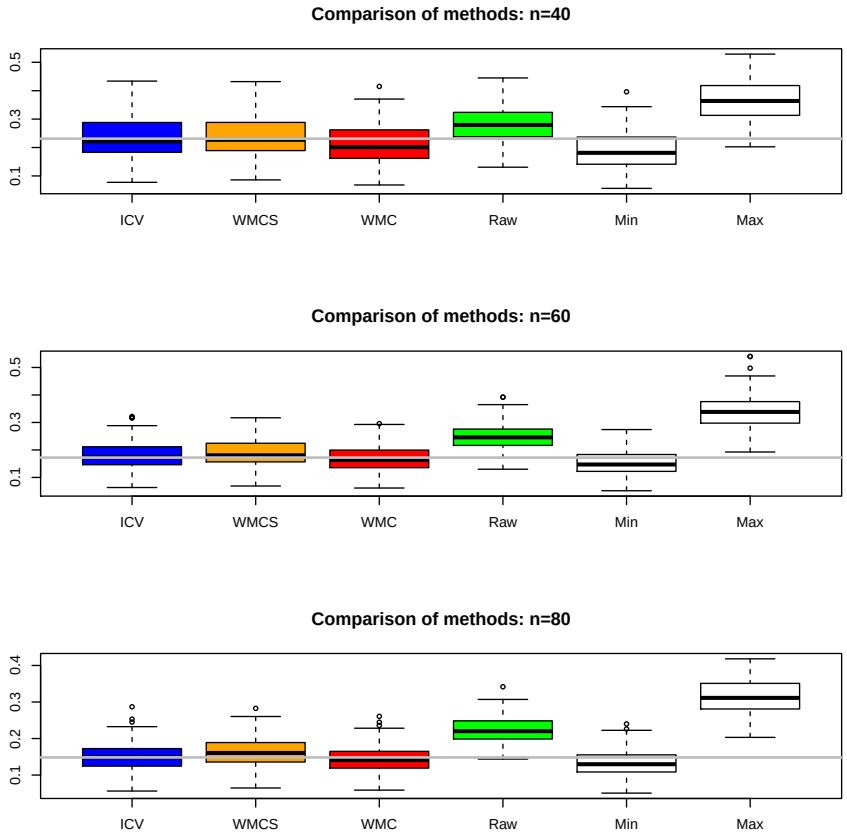


Figure A.5: Comparison of ICV (Internal Cross-Validation), WMC (Weighted Mean Correction) and WMCS (Weighted Mean Correction with Shrinkage) for the setup with correlated Gaussian data and weaker signal. The gray line represents the approximation of the true value of $\varepsilon^{n_L}(\phi)$ as described in A.2.

The mean, standard deviation and average absolute difference to the reference method ICV are displayed in the following table for all methods with the three considered sample sizes.

	$n = 40$			$n = 60$			$n = 80$		
	mean	absdicv	sd	mean	absdicv	sd	mean	absdicv	sd
ICV	0.234	0.000	0.069	0.177	0.000	0.049	0.151	0.000	0.037
WMCS	0.239	0.014	0.069	0.188	0.013	0.050	0.162	0.012	0.039
WMC	0.213	0.022	0.066	0.168	0.011	0.046	0.144	0.008	0.035
Raw	0.283	0.066	0.060	0.248	0.066	0.045	0.222	0.036	0.036
Min	0.192	0.042	0.064	0.153	0.024	0.045	0.133	0.018	0.034
Max	0.365	0.131	0.070	0.340	0.163	0.061	0.313	0.162	0.050

Table A.5: Mean, average absolute difference to ICV (absdicv) and standard deviation of the different correction methods for datasets of size $n = 40$, $n = 60$ and $n = 80$ in the setup with correlated Gaussian data and weaker signal.

The true parameter $Err = \varepsilon^{n_L}(\phi)$

	n=40	n=60	n=80
$\varepsilon^{n_L}(\phi)$	0.231	0.172	0.148

Table A.6: Estimation of $\varepsilon^{n_L}(\phi)$ based on 1000 independent training sets and a large validation set containing 20000 observations (see A.2) in the setup with correlated Gaussian data and weaker signal.

Shrinkage factor

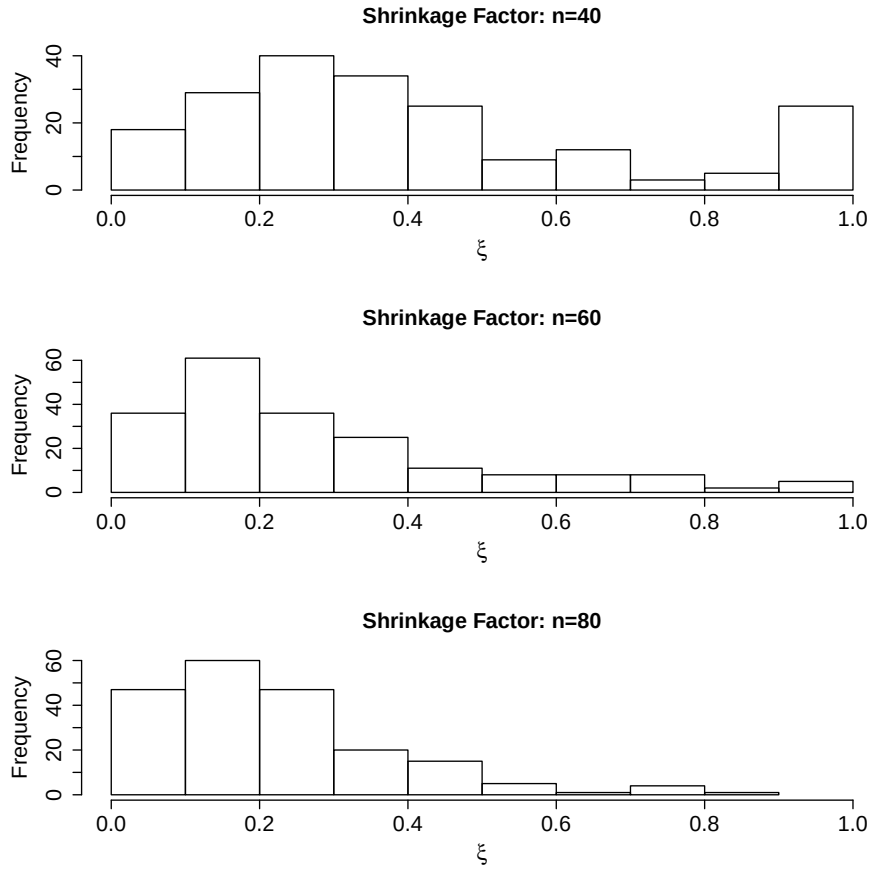


Figure A.6: Distribution of the shrinkage factor ξ for different numbers of observations for the setup with correlated Gaussian data and weaker signal. The distribution is shifted closer to 0 as the size of the dataset is increased. This shift is less distinct than the one observed in the last setup in which the signal is stronger.

Summary

The WMC method underestimates the bias whereby this optimistic bias decreases with increasing sample size. WMCS yields slightly pessimistic estimates. For small datasets WMCS performs better than WMC whereas this tendency is reversed for larger datasets where WMC performs best. The shrinkage factor becomes smaller with increasing sample size, which explains why WMC and WMCS become more similar.

A.2.1 Correlated non-informative data

Study design

In this setup, all 2000 predictors are simulated as non-informative. The predictors are simulated exactly in the same way as the non-informative predictors in 1.4.2.

Results

The following boxplots display the estimated errors obtained for the 200 simulated datasets with the five considered methods as well as the minimal and maximal error over the K investigated methods (top: $n = 40$, middle: $n = 60$, bottom: $n = 80$). The (approximated) target value $\varepsilon^{n_L}(\phi)$ is represented as an horizontal gray line.

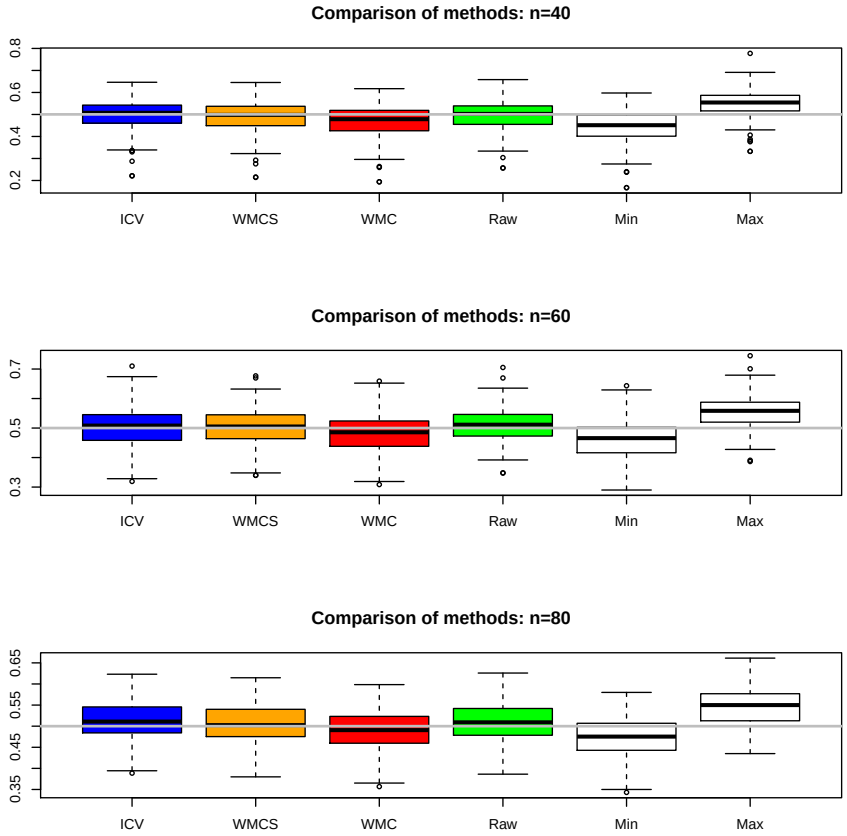


Figure A.7: Comparison of ICV (Internal Cross-Validation), WMC (Weighted Mean Correction) and WMCS (Weighted Mean Correction with Shrinkage) for the setup with correlated non-informative Gaussian data. The gray line represents the approximation of the true value of $\varepsilon^{n_L}(\phi)$ as described in A.2.

	$n = 40$			$n = 60$			$n = 80$		
	mean	absdicv	sd	mean	absdicv	sd	mean	absdicv	sd
ICV	0.496	0.000	0.071	0.504	0.000	0.063	0.510	0.000	0.046
WMCS	0.491	0.013	0.073	0.500	0.012	0.060	0.506	0.011	0.046
WMC	0.467	0.029	0.074	0.480	0.024	0.062	0.489	0.021	0.047
Raw	0.498	0.059	0.067	0.507	0.052	0.056	0.510	0.053	0.045
Min	0.444	0.052	0.075	0.461	0.043	0.062	0.472	0.038	0.048
Max	0.549	0.053	0.066	0.552	0.049	0.056	0.546	0.036	0.044

Table A.7: Mean, average absolute difference to ICV (absdicv) and standard deviation of the different correction methods for datasets of size $n = 40$, $n = 60$ and $n = 80$ in the setup with correlated non-informative Gaussian data.

The true parameter $Err = \varepsilon^{n_L}(\phi)$

The (approximated) true value of the parameter $\mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi)$ is given in the following table for the three considered sample sizes $n = 40$, $n = 60$ and $n = 80$ (values are rounded to three digits and not exactly 50.0%).

	n=40	n=60	n=80
$\varepsilon^{n_L}(\phi)$	0.500	0.500	0.500

Table A.8: Estimation of $\varepsilon^{n_L}(\phi)$ based on 1000 independent training sets and a large validation set containing 20000 observations (see A.2) in the setup with correlated non-informative Gaussian data.

Shrinkage factor

The following histograms display the values of the shrinkage factors computed within the WMCS procedure for sample sizes $n = 40$, $n = 60$ and $n = 80$.

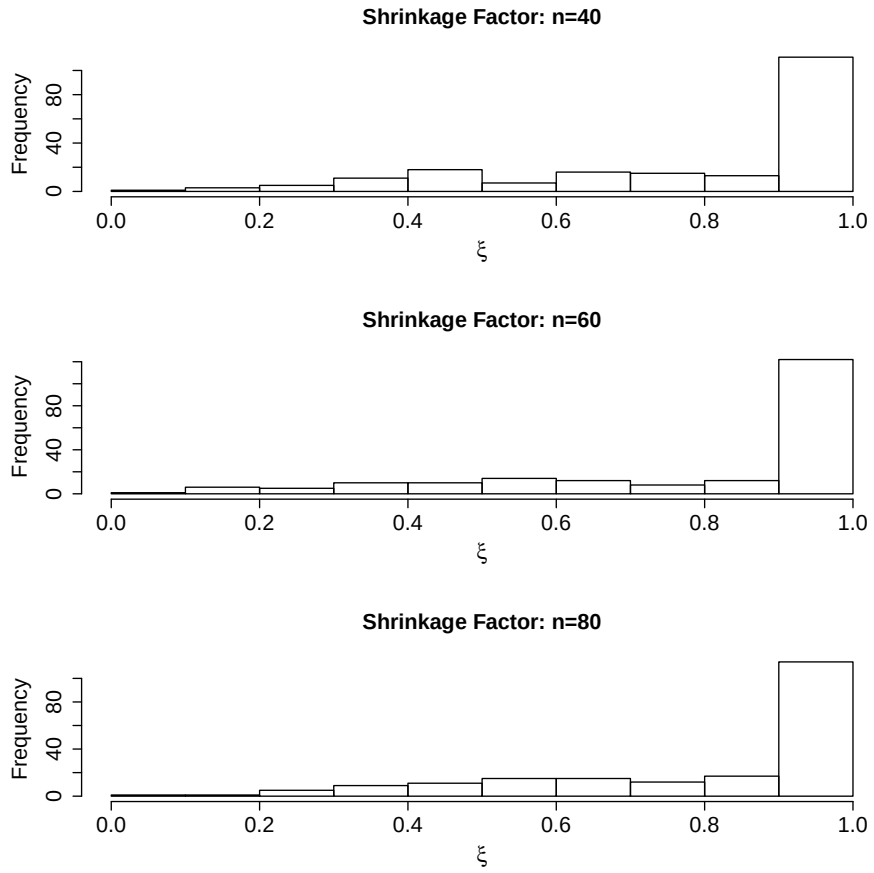


Figure A.8: Distribution of the shrinkage factor ξ for different numbers of observations for the setup with correlated non-informative Gaussian data. The distribution of ξ concentrates around 1 and is not affected by the sample size.

Summary

WMCS yields the average estimate closest to ICV, whereby the mean absolute difference between both methods is only 1.3%. WMCS usually relies on the raw mean estimate since $\zeta > \widehat{Err}_{RawMean} - e(k^*(s_0) \parallel s_0)$ most often holds. Again, the WMC procedures underestimates the bias. This underestimation becomes less severe with increasing sample size (which coincides with a smaller bias).

A.2.2 Uncorrelated Gaussian data

Simulation Design

In this setup, $p = 2000$ uncorrelated predictors are used. 2% of these predictors are informative, i.e. associated with the binary response. In the second response group all informative predictors follow the distribution $\mathcal{N}(0, 1)$, while in the first response group,

the d-th informative variable is simulated according to $\mathcal{N}(\boldsymbol{\delta}_d, 1)$, where δ_d is itself drawn from the distribution $\mathcal{N}(0.2, 0.5)$. Consequently, the difference between the group means is random. The non-informative variables are simulated according to $\mathcal{N}(0, 1)$ regardless of the group they belong to.

Results

The following boxplots display the estimated errors obtained for the 200 simulated datasets with the five considered methods as well as the minimal and maximal error over the K investigated methods (top: $n = 40$, middle: $n = 60$, bottom: $n = 80$). The (approximated) target value $Err = \varepsilon^{n_L}(\phi)$ is represented as an horizontal gray line.

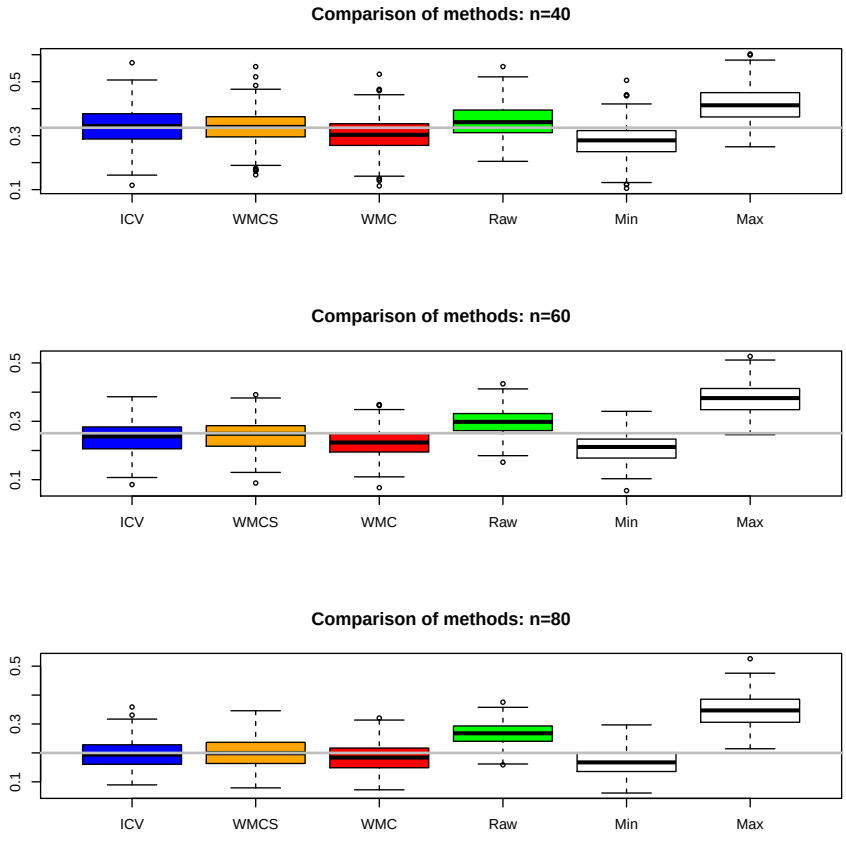


Figure A.9: Comparison of ICV (Internal Cross-Validation), WMC (Weighted Mean Correction) and WMCS (Weighted Mean Correction with Shrinkage) for the setup with uncorrelated Gaussian data. The gray line represents the approximation of the true value of $\varepsilon^{n_L}(\phi)$ as described in A.2.

The mean, standard deviation and average absolute difference to the reference method ICV are displayed in the following table for all methods with the three considered sample sizes.

	$n = 40$			$n = 60$			$n = 80$		
	mean	absdicv	sd	mean	absdicv	sd	mean	absdicv	sd
ICV	0.331	0.000	0.075	0.245	0.000	0.056	0.195	0.000	0.051
WMCS	0.329	0.017	0.069	0.252	0.013	0.056	0.201	0.011	0.052
WMC	0.300	0.031	0.072	0.226	0.019	0.052	0.184	0.012	0.047
Raw	0.349	0.079	0.061	0.299	0.053	0.045	0.268	0.070	0.042
Min	0.277	0.054	0.071	0.208	0.037	0.050	0.169	0.027	0.045
Max	0.419	0.090	0.068	0.379	0.134	0.056	0.347	0.152	0.056

Table A.9: Mean, average absolute difference to ICV (absdicv) and standard deviation of the different correction methods for datasets of size $n = 40$, $n = 60$ and $n = 80$ in the setup with uncorrelated Gaussian data.

The true parameter $Err = \varepsilon^{n_L}(\phi)$

The (approximated) true value of the parameter $\mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi)$ is given in the following table for the three considered sample sizes $n = 40$, $n = 60$ and $n = 80$.

	n=40	n=60	n=80
$\varepsilon^{n_L}(\phi)$	0.329	0.259	0.200

Table A.10: Approximated value of $\varepsilon^{n_L}(\phi)$ computed from 1000 independent training sets and a large validation set containing 20000 observations (see A.2) in the setup with uncorrelated Gaussian data.

Shrinkage factor

The following histograms display the values of the shrinkage factors computed within the WMCS procedure for sample sizes $n = 40$, $n = 60$ and $n = 80$.

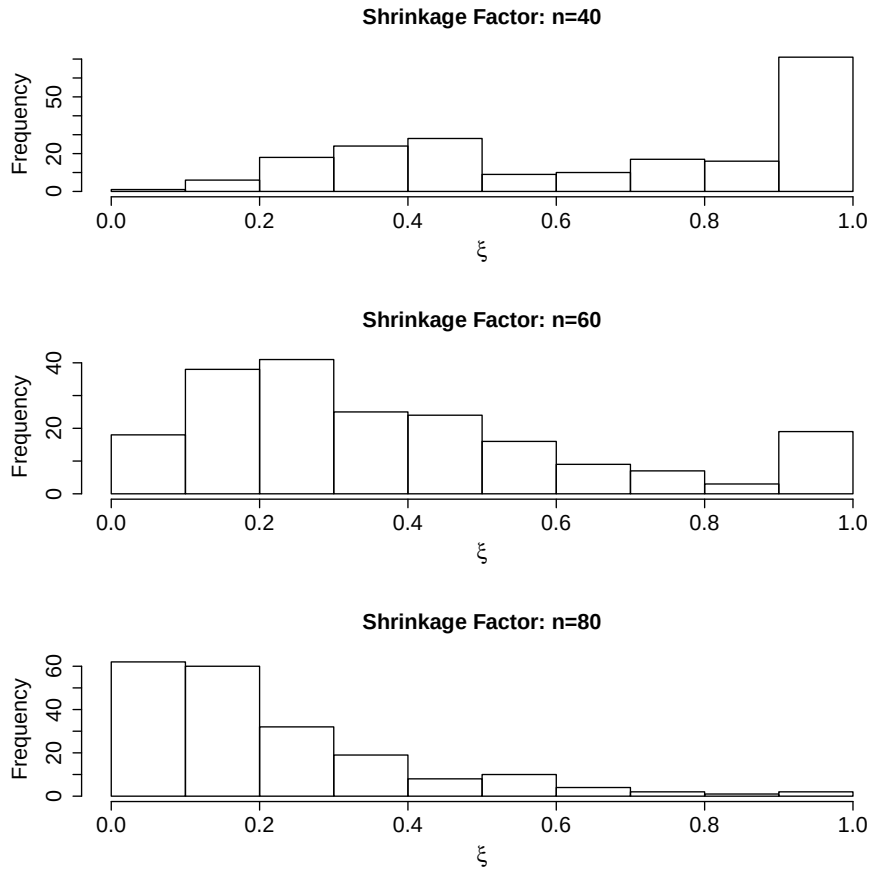


Figure A.10: Distribution of the shrinkage factor ξ for different numbers of observations for the setup with uncorrelated Gaussian data. The distribution is shifted to 0 as the size of the dataset is increased.

Summary

ICV and WMCS yield almost identical results as far is their average is concerned. WMC, however, underestimates the bias. The WMCS method often uses a shrinkage factor close to 1 in the case of smaller datasets, which means that the error rates are strongly shrunken towards the raw mean. For $n = 80$ shrinkage factors close to 1 are rare.

A.2.3 Differences in covariances between groups

Again, $p = 2000$ predictors and a block size of 200 are used. However, this time the vector of mean differences between groups is set to $\boldsymbol{\delta} = \mathbf{0}$ for all p predictors (both informative or non-informative). The only difference between informative and non-informative predictors is the covariance matrix. The 200 informative predictors are simulated from the multivariate normal distribution with mean zero and a covariance matrix chosen as the corresponding

within-group empirical covariance matrix of the 200 first variables from the Singh dataset. The non-informative predictors are simulated in exactly the same way as described in 1.4.2.

Results

The following boxplots display the estimated errors obtained for the 200 simulated datasets with the five considered methods as well as the minimal and maximal error over the K investigated methods (top: $n = 40$, middle: $n = 60$, bottom: $n = 80$). The (approximated) target value $\varepsilon^{n_L}(\phi)$ is represented as an horizontal gray line.

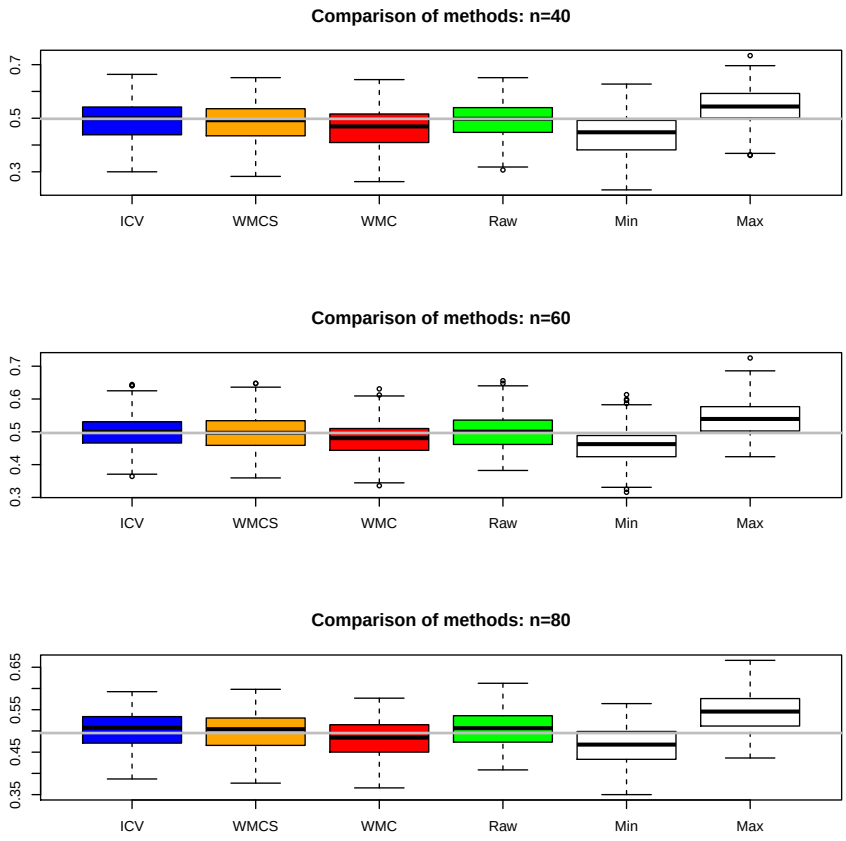


Figure A.11: Comparison of ICV (Internal Cross-Validation), WMC (Weighted Mean Correction) and WMCS (Weighted Mean Correction with Shrinkage) for the setup with differences in covariances between response groups. The gray line represents the approximation of the true value of $Err = \varepsilon^{n_L}(\phi)$ as described in A.2.

The mean, standard deviation and average absolute difference to the reference method ICV are displayed in the following table for all methods with the three considered sample sizes.

	$n = 40$			$n = 60$			$n = 80$		
	mean	absdicv	sd	mean	absdicv	sd	mean	absdicv	sd
ICV	0.489	0.000	0.077	0.501	0.000	0.053	0.504	0.000	0.043
WMCS	0.485	0.013	0.079	0.498	0.011	0.054	0.501	0.010	0.044
WMC	0.462	0.028	0.080	0.479	0.023	0.055	0.483	0.021	0.043
Raw	0.493	0.069	0.073	0.502	0.058	0.052	0.505	0.041	0.042
Min	0.438	0.051	0.081	0.459	0.042	0.056	0.466	0.038	0.044
Max	0.545	0.056	0.072	0.543	0.042	0.053	0.544	0.040	0.046

Table A.11: Mean, average absolute difference to ICV (absdicv) and standard deviation of the different correction methods for datasets of size $n = 40$, $n = 60$ and $n = 80$ in the setup with differences in covariances between response groups.

The true parameter $Err = \varepsilon^{n_L}(\phi)$

The (approximated) true value of the parameter $\mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi)$ is given in the following table for the three considered sample sizes $n = 40$, $n = 60$ and $n = 80$.

	n=40	n=60	n=80
$\varepsilon^{n_L}(\phi)$	0.498	0.496	0.495

Table A.12: Estimation of $\varepsilon^{n_L}(\phi)$ based on 1000 independent training sets and a large validation set containing 20000 observations (see A.2) in the setup with differences in covariances between response groups.

Shrinkage factor

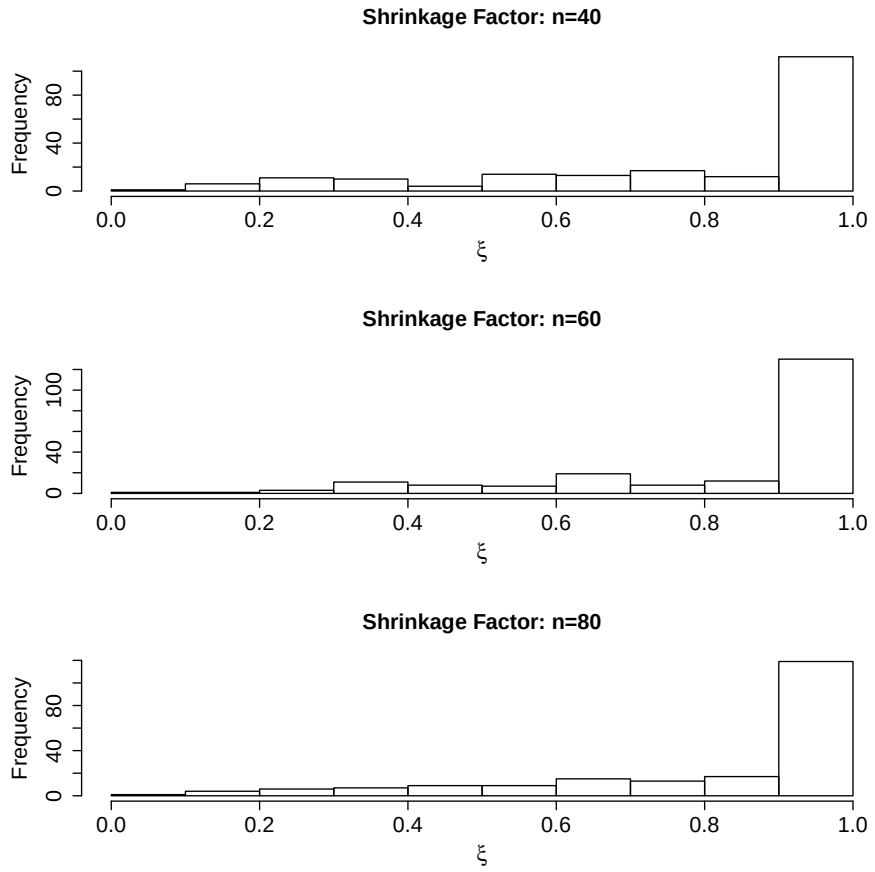


Figure A.12: Distribution of the shrinkage factor ξ for different numbers of observations for the setup with differences in covariances between response groups. The distribution of ξ is concentrated around 1 and does not change if the number observations is increased.

Summary

On average, WMCS yields results closest to the ICV results. The mean absolute difference between both methods is only 1.3%. Again, WMCS uses the raw mean as an estimate in most cases. WMC underestimates the bias. This underestimation becomes less severe with increasing sample size (which coincides with a smaller bias).

A.3 Normality assumption

This section supplements the discussion on the normality assumption of the WMC estimator in Section 1.5.1.

With $n = 80$ the deviation from normality is much lower, as illustrated in the following plots showing the *largest* deviations observed in the case $n = 80$. They are obtained from the simulation setup described in 1.4.2.

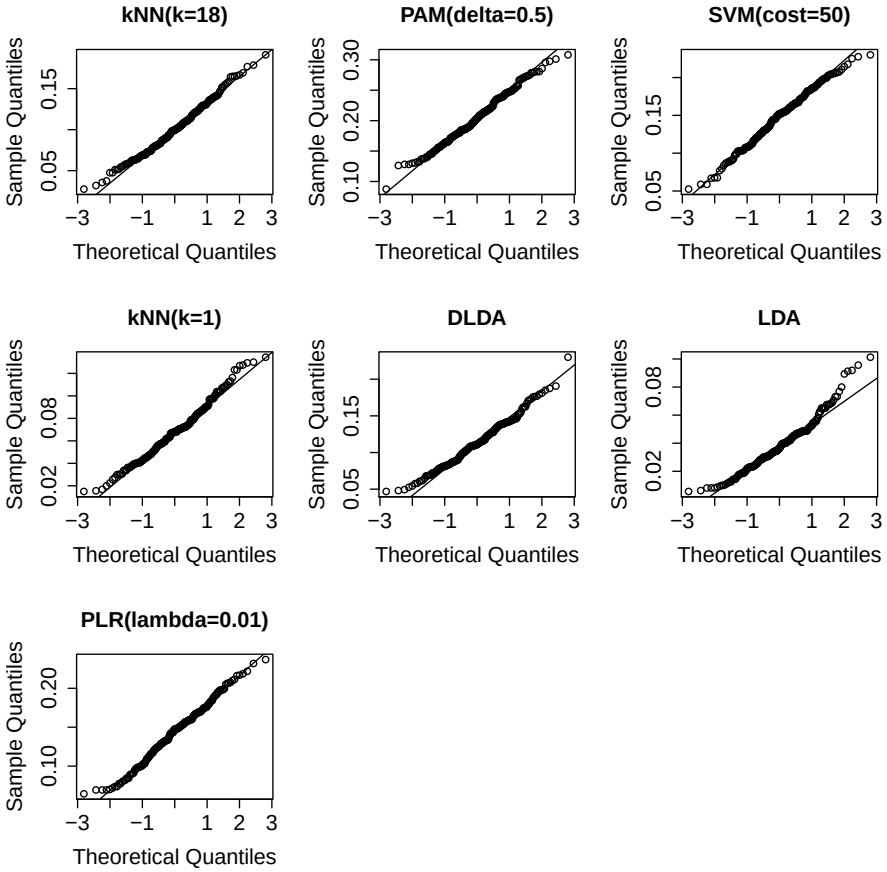


Figure A.13: Normal quantile plots of $e(k\|S)$ for the case $n = 80$ in simulation setup 1.4.2 illustrating the largest deviations from normality observable in the simulation study for datasets of size $n = 80$.

Only the LDA-plot shows deviations which are larger than the deviations observed in most of the 500 plots simulated under perfect normality (data not shown). This matches the expectation that the normal assumption (as with the Central Limit Theorem) is more adequate for larger datasets.

A.4 Further information on runtimes

This table is a supplement to Section 1.5.2 and presents the runtimes in the case of the $PLSLDA$ setup.

$PLSLDA$ setup

Runtimes	WMC	WMCS	ICV
Alon	3.3	3.8	35.4
Singh	16.7	16.9	310.7
Golub	2.9	3.4	21.8
Chiaretti	21.1	19.6	340.0

Table A.13: Average runtimes (in seconds) WMC (Weighted Mean Correction), WMCS (Weighted Mean Correction with Shrinkage) and Internal Cross-Validation for the $PLSLDA$-setups on the different datasets. The difference between WMC and WMCS is negligible in comparison to the absolute runtimes.

Appendix B

Ranking High-Dimensional Wrapper Algorithms using multiple studies

B.1 Correlation of the rankings defined by CV and CSV in the simulation example

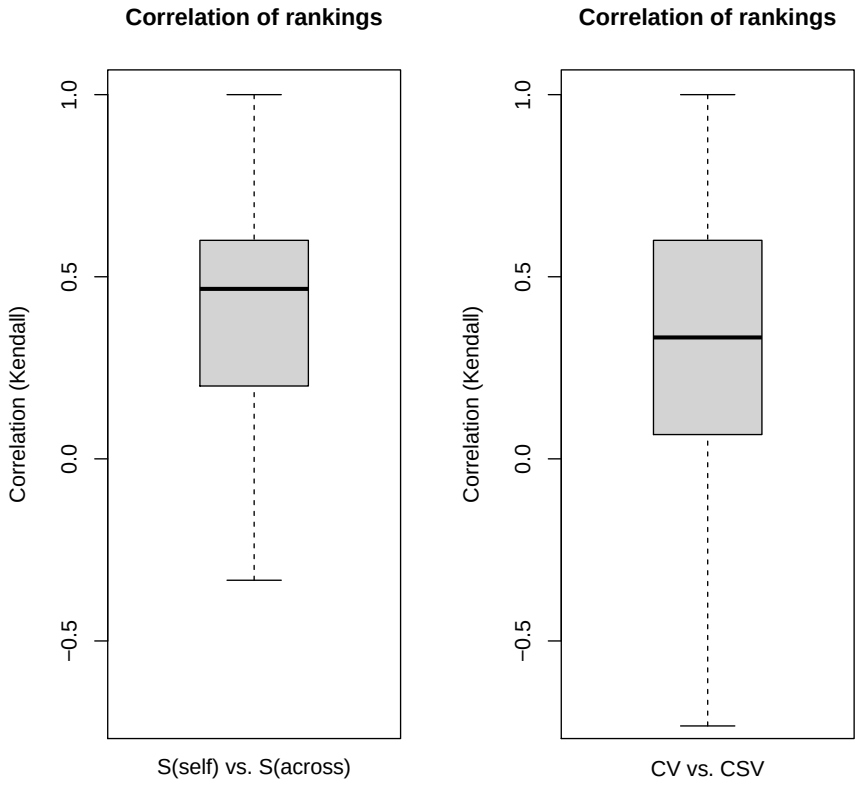


Figure B.1: The left panel illustrates the distribution of the Kendall correlations between the rankings as defined by S_{self} and S_{across}. The boxplot in the right panel corresponds to the distribution of the Kendall correlations of the rankings as defined by $\overline{CV}$ and $\overline{CSV}$. Whereas S_{self} and $\overline{CV}$ are more dedicated to measuring the within-study performance S_{across} and $\overline{CSV}$ assess cross-study performance. The medians below 0.5 in both boxplots suggest that rankings in cross-study prediction and within-study prediction differ distinctly.

B.2 Simulation example with removal of outlier studies

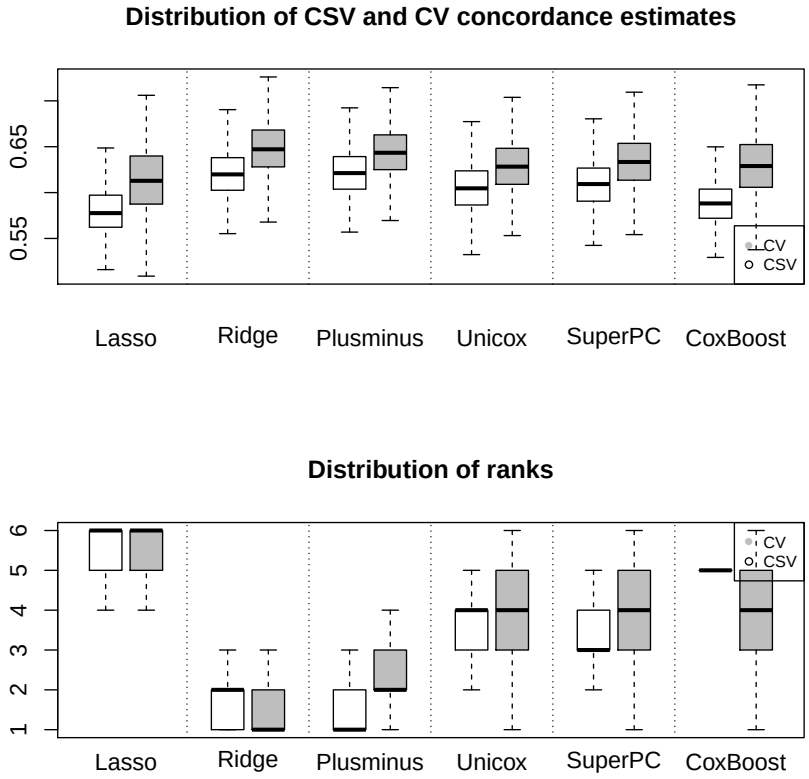


Figure B.2: This illustration corresponds to Figure 2 in the main paper after the removal of the outlier studies CAL and MSK. Discrimination performance increases distinctly in the case of $\overline{CSV}$ as can be seen from the top panel. The values of $\overline{CV}$ are almost unaltered. Nonetheless, $\overline{CV}$ still estimates higher discrimination performance although the difference to $\overline{CSV}$ is smaller now. The bottom panel illustrates the distribution of the ranks of the competing algorithms. There are still differences between the rankings defined by $\overline{CSV}$ and $\overline{CSV}$. As can be seen from Figure B.4, the median Kendall correlation between both rankings amounts to approximately 0.5.

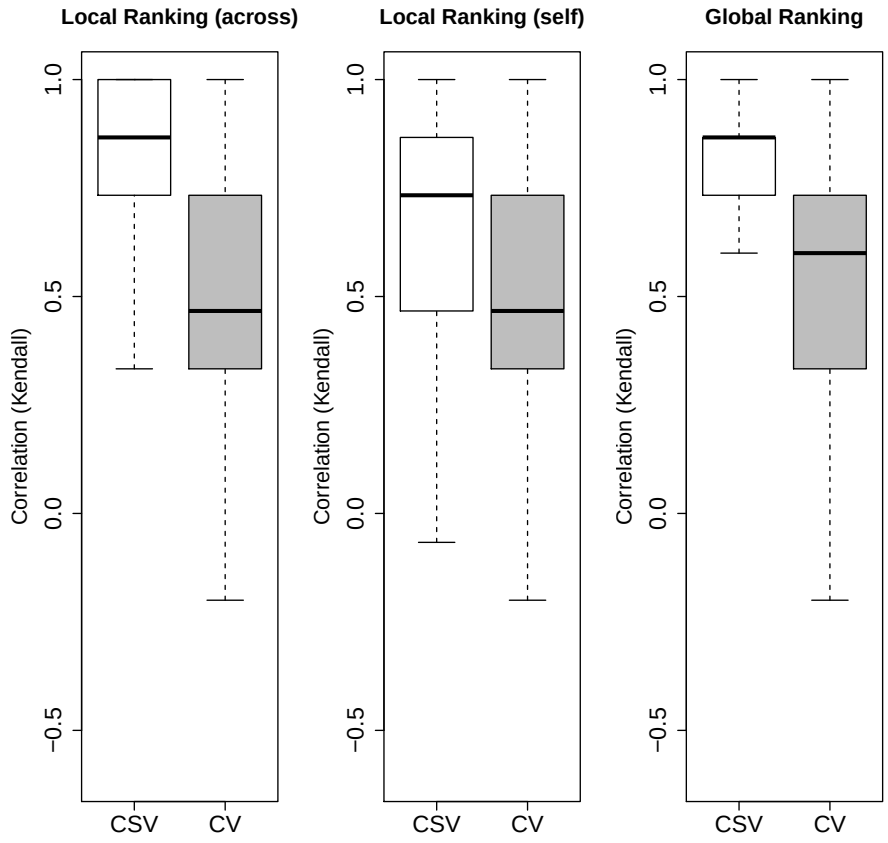


Figure B.3: This illustration corresponds to Figure 3 in the main paper after removal of the outlier studies. Also after this removal, the ranking defined by $\overline{CSV}$ features higher correlations to the global ranking and the ranking defined by S_{across}. Here it even achieves a higher correlation than $\overline{CV}$ in the case of the ranking according to S_{self}. This is not expected because both $\overline{CV}$ and S_{self} measure within-study performance in contrast to $\overline{CSV}$. However, one also has to keep in mind that $\overline{CS}$ is less closely related to the model which has actually been fit on the respective dataset because it represents an average of four models which have been fitted on subsets of the actual dataset. Generally, Kendall correlations to the respective true rankings are distinctly higher than before the removal of the outlier studies.

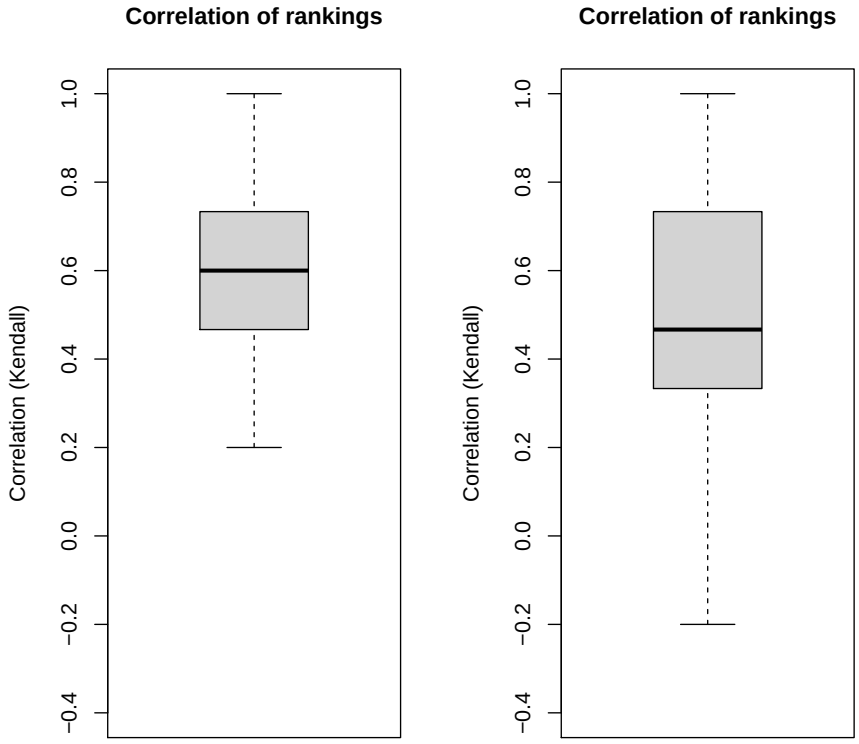


Figure B.4: This illustration corresponds to Figure B.1 after the removal of the outlier studies. Whereas S_{self} and $\overline{CV}$ are more dedicated to measuring the within-study performance S_{across} and $\overline{CSV}$ assess cross-study performance. The medians around 0.5 in both boxplots suggest that rankings in cross-study prediction and within-study prediction still differ distinctly also if the outlier studies are removed.

B.3 CV as an estimator of independent within-study discrimination performance

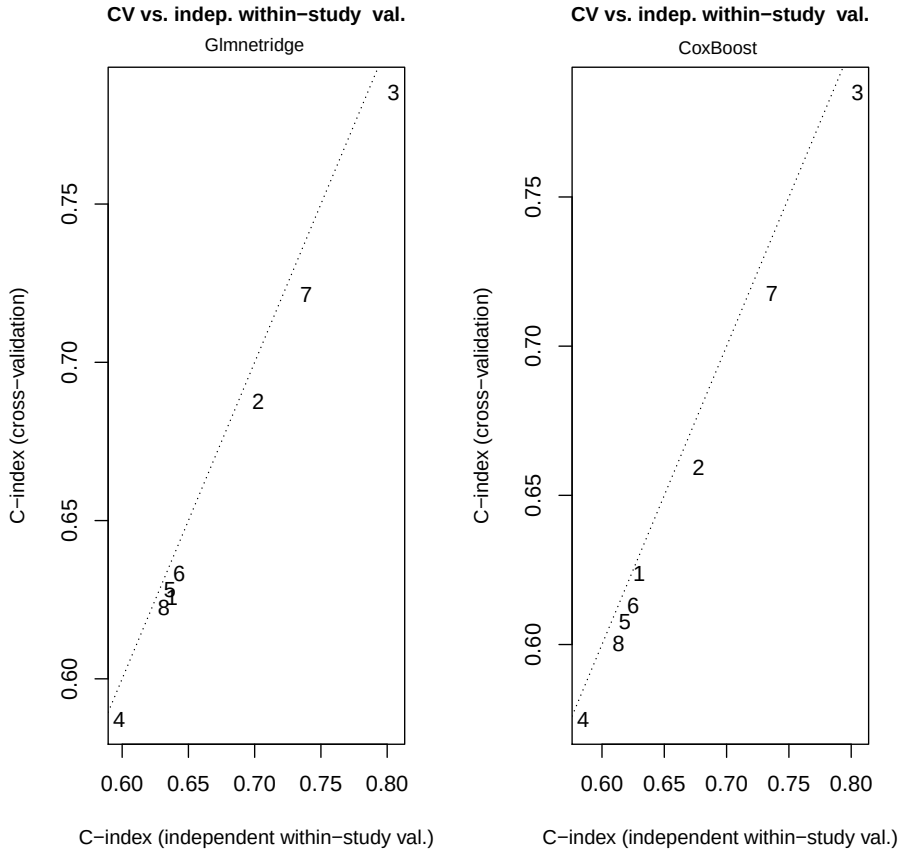


Figure B.5: In this Figure, the average discrimination performance as estimated by CV are depicted against the corresponding average in independent within-study validation for *Glmnetridge* (left panel) and *CoxBoost* (right panel). The averages are taken over values from the same study as indicated by the different study numbers. In the simulation example (including outlier studies), CV estimates are slightly smaller which matches the expectation since one fold of observations is left out CV when training the models.

B.4 Rankings on experimental microarray data (outlier studies included)

Algorithm	CSV			CV		
Glmnetridge	4	1	3	5	1	1
Plusminus	3	2	1	4	2	3
Superpc	1	3	4	1	3	4
Unicox	2	4	2	2	4	5
CoxBoost	5	5	5	6	5	6
Glmnetlasso	6	6	6	3	6	2
criterion	Av.	Med.	3 Q.	Av.	Med.	3 Q.

Table B.1: Rankings as estimated by CV and CSV on the experimental microarray data with outlier studies CAL and MSK included. Here, rankings are based on a single matrix Z^k for each of the $K = 6$ algorithms. For both validation concepts rankings are presented for the aggregation methods simple average (Av.), median (Med.) and third quartile (3 Q.). CSV ranking is more consistent over the different aggregation methods assigning the last ranks always to *CoxBoost* and *Glmnetlasso*. For the top and middle ranks the ranking is less consistent. CV features large inconsistencies over the aggregation methods. For example, *Glmnetridge* and *Glmnetlasso* are both assigned top and bottom ranks depending on the aggregation method. The Kendall's correlations between CSV and CV rankings are also strongly varying amounting to 0.6 (Av.), 1 (Med.) and 0.07 (3Q.) suggesting distinct differences between the estimated rankings. In summary, rankings are less similar to the simulated data than the rankings obtained after the removal of outlier studies (see Table B.2).

B.5 Rankings on experimental microarray data (outlier studies removed)

Algorithm	CSV			CV		
Glmnetridge	2	1	3	3	1	2
Plusminus	1	2	1	1	2	3
Superpc	3	4	2	2	3	4
Unicox	4	3	4	6	5	5
CoxBoost	5	5	5	4	6	6
Glmnetlasso	6	6	6	5	4	1
criterion	Av.	Med.	3 Q.	Av.	Med.	3 Q.

Table B.2: Rankings as estimated by CV and CSV on the experimental microarray data after the removal of outlier studies CAL and MSK. Here, rankings are based on a single matrix Z^k for each of the $K = 6$ algorithms. For both validation concepts rankings are presented for the aggregation methods simple average (Av.), median (Med.) and third quartile (3 Q.). CSV ranking is more consistent over the different aggregation methods assigning top ranks to *Glmnetridge* and *Plusminus*, and the last ranks to *CoxBoost* and *Glmnetlasso*. This ranking is also close to the true global ranking on simulated data. CV is clearly less consistent assigning e.g. the ranks 5,4, and 1 to *Glmnetlasso*. The Kendall's correlations between CSV and CV rankings are also strongly varying. They amount to 0.6 (Av.), 0.47 (Med.) and 0.07 (3Q.) suggesting distinct differences between the estimated rankings.

Appendix C

Computational Aspects & Software

C.1 Package Vignette: SurvHD - Survival Analysis for High-Dimensional Data

The packages *survHD* and *survHDExtra* have been developed in cooperation with Levi Waldron (Harvard Medical School and Dana-Faber Cancer Institute) and Markus Riester (Dana-Faber Cancer Institute). Levi Waldron had the initial idea of developing an **R**-package parallel to *CMA* which can be used for survival analysis on high-dimensional data. He has implemented the algorithm functions *penalizedSurv* and *penalizedSurv* and also some of the functions implementing certain performance criteria. Moreover he provided many comments and suggestions for changes on the basic workflow and concept of the package. Finally, Markus Riester contributed the functions for gene set analysis and *customSuperPc* as well as several graphical tools. Additionally, he supported the developement of the package by providing feedback and helpful suggestions concerning the class structure and the general workflow. The pages 20–24 of this vignette have also been created by him.

survHD
package vignette

Christoph Bernau *
Levi Waldron †
Markus Riester‡

2012

1 SurvHD - Survival Analysis for High-Dimensional Data (Gene Expression Data)

Survival analysis on high-dimensional data is a complex and computationally intensive task. Many standard algorithms for survival data are not applicable in the "$p >> n$"-case, e.g. with several thousands of variables and only hundreds of samples. Many adaptions of the Cox-Model and other standard approaches for high-dimensional data have been developed in the last years. Furthermore, methods for tuning and evaluation of models are distinct from classification problems. Our R-package **survHD** provides a framework for the application, optimization, and evaluation of high-dimensional survival models. In this vignette, we will introduce the basic work flow of the package using a well-known cancer microarray data set. Additionally, we will explain the built-in interface for including user-defined custom survival model functions into the **survHD** framework.

1.1 Input Data

The package accepts three different formats for predictor data **X**: data-frame, matrix, or the Bioconductor-defined **ExpressionSet** class. The corresponding survival outcome can be either a **Surv** object from the package **survival**, or in the case of the **ExpressionSet**-format, a character vector specifying the name of the **surv** object in the **phenoData**-slot of the **ExpressionSet** object. In our example, we use the matrix format for analyzing the gene expressions of 86 lung adenocarcinoma patients published by Beer et al. (2002), which can be found in the data folder of the package (here we just use the first 500 probe sets for illustration purposes):

```
set.seed(321)
data(beer.exprs)
data(beer.survival)
X <- t(as.matrix(beer.exprs))
y <- Surv(beer.survival$os, beer.survival$status)
```

*bernau@ibe.med.uni-muenchen.de
†lwaldron.research@gmail.com
‡markus@jimmy.harvard.edu

1.2 Fitting Models

The most important function in **survHD** is called **learnSurvival**. It is used to fit the survival model and store its essential information for later evaluation on test data. Apart from the input data mentioned in the last section, the only additional required argument is the name of the survival method that shall be performed: in this example we will use **coxBoostSurv**, a wrapper to the **CoxBoost** package. As a first example of training on a full dataset followed by independent evaluation on a pre-defined dataset, we exclusively use the first 60 observations for model fitting because we want to evaluate our model on the last 26 observations of our data. The parameter **stepno** is a method specific tuning parameter (the number of Boosting steps) which is simply passed to the **CoxBoost** function.

```
ytrain <- y[1:60,]
Xtrain <- X[1:60,]
coxboostmodel <- learnSurvival(y=ytrain,
                               X=Xtrain,
                               survmethod='coxBoostSurv',
                               stepno=20)

printl('str(coxboostmodel, max.level=2)')
```

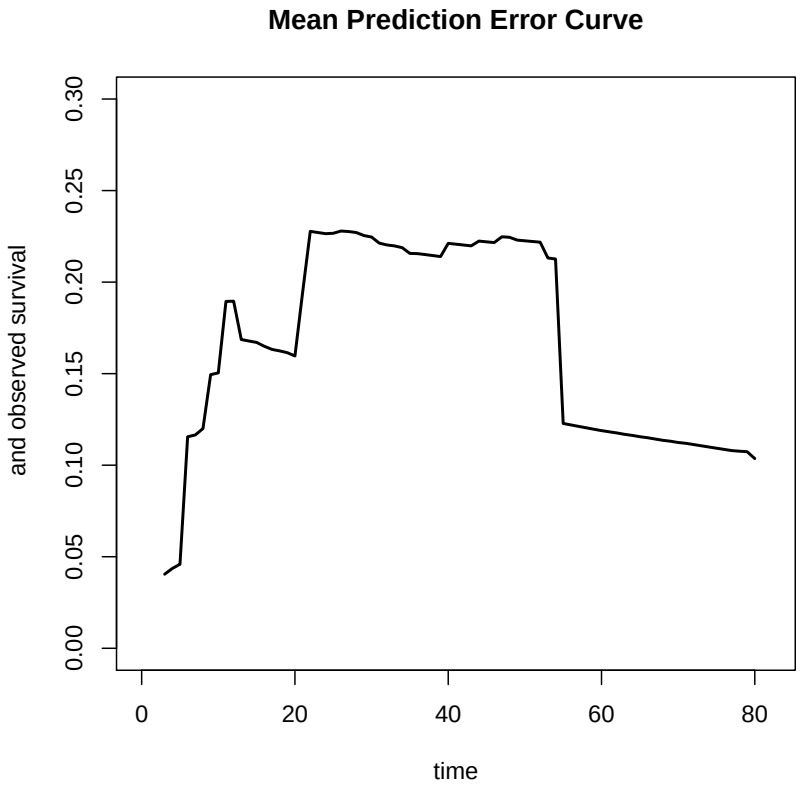
```
Formal class 'LearnOut' [package "survHD"] with 7
slots
..@ ModelLearnedlist:List of 1
..@ X :'data.frame': 60 obs. of 7129 variables:
.. .. [list output truncated]
..@ y : Surv [1:60, 1:2] 84.1 91.8+ 93.7+ 108.2+ 34.6
68.1+ 34.2+ 47.0+ 39.1+ 45.8 ...
.. ..- attr(*, "dimnames")=List of 2
.. ..- attr(*, "type")= chr "right"
..@ GeneSel :Formal class 'GeneSel' [package
"survHD"] with 4 slots
..@ nbgene : int 7129
..@ LearningSets :Formal class 'LearningSets'
[package "survHD"] with 4 slots
..@ threshold : num -1
```

The call to **str()** shows the slot of the output object which stores all the information necessary to perform predictions for new data and evaluate the model.

1.3 Evaluation on independent Data

Evaluation of survival models is a complicated task for which no gold standard exists. One of the biggest problems is the time-dependency of the model fit which is difficult to include or weight appropriately. Many reasonable measures for evaluating survival models have been proposed in the literature. Currently, three of these measures are implemented in the function **evaluate**: Harrell's C-Index, Uno's C-Index, and Prediction Error Curves. Prediction error curves compare the model-based estimated survival curves with the actual survival indicator function of the test observations. For evaluating our model with regard to this measure ("PErrC") we have to pass the output of **learnSurvival** (coxboostmodel) and the validation data to the function **evaluate**. Additionally, we must define a grid of time points for which the comparison of estimated and observed survival shall be performed:

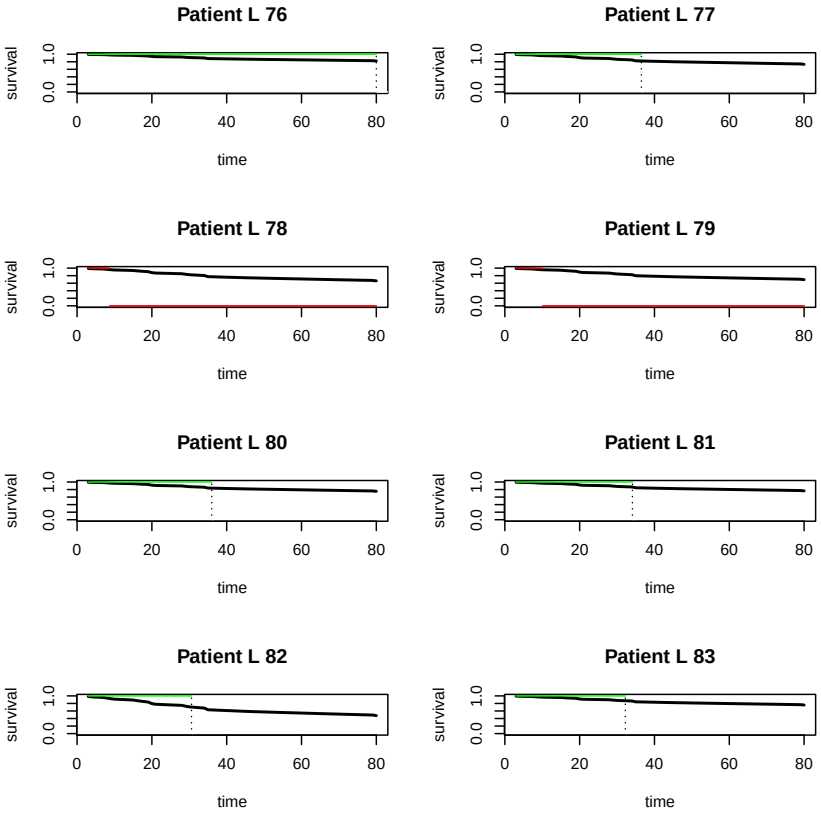
```
ytest <- y[61:86, ]
Xtest <- X[61:86, ]
timegrid <- 3:80
ev1 <- evaluate(coxboostmodel, measure = "PErrC", newy = ytest,
     newdata = Xtest, timegrid = timegrid, addtrain = F)
plot(timegrid, result(ev1)[[1]], type = "l", lwd = 2,
     col = "black", xlim = c(0, 82), ylim = c(0, 0.3),
     xlab = "time", ylab = "mean difference between estimated
     and observed survival", main = "Mean Prediction Error Curve")
```



We can see that the prediction performance for the validation set is worse in the time interval between 20 and 55 months. For a more detailed evaluation of the model fit we can compare the observed and estimated survival for each patient separately. For that purpose we can use the **predict** function on the **LearnOut** object in order to obtain observationwise predictions and compare them to the survival indicator function.

```
probs <- predict(coxboostmodel, newdata = Xtest,
          type = "SurvivalProbs", timegrid = 3:80)
par(mfrow = c(4, 2))
for (i in 6:13) {
     followup <- xlim2 <- 80
     col1 <- "red"
     col2 <- "red"
     if (ytest[i, 2] == 0) {
         followup <- min(80, ytest[i, 1])
         col1 <- "green"
         col2 <- "white"
     }
     plot(timegrid, as.matrix(SurvivalProbs(probs[[1]]))[i,
```

```
        ], col = "black", lwd = 2, main = paste("Patient L",
        i + 70, sep = " "), xlab = "time", ylab = "survival",
        type = "l", ylim = c(0, 1), xlim = c(3, xlim2))
    segments(c(3, ytest[i, 1]), c(1, 0), c(min(followup,
        ytest[i, 1]), followup), c(1, 0), col = c(col1,
        col2))
    if (ytest[i, 2] == 0)
        abline(v = followup, lty = 3)
}
```



This plot illustrates a moderate prediction accuracy on this patient selection. For the two patients with events, the survival curves are indeed lower than e.g. in the first graph representing the survival curve of a patient who was still alive after 80 months when he was censored.

Using **predict** (*type* =**'lp'**), we can also see that the linear predictors are indeed higher for the two patients with events. Please note the (maybe unexpected) structure of the output object. Since **survHD** is actually designed to fit several models at a time – e.g. in a cross-validation – the output object is a list containing a single object of class **LinearPrediction** whose slot **lp** contains the linear predictor which we wanted to obtain:

```
List of 1
$ :Formal class 'LinearPrediction' [package "survHD"]
with 1 slots
.. ..@ lp: Named num [1:26] 1.952 -0.1976 -0.0529
-0.1154 -0.5334 ...
.. .. ..- attr(*, "names")= chr [1:26] "L59" "L61"
"L62" "L64" ...
```

```
       L76        L78        L79        L80        L81
-1.0816411 -0.6246998 -0.3415224 -0.4665002 -0.7211664
       L82        L83        L84
-0.7733700  0.2401477 -0.7437558
```

Based on the linear predictor one can compute other evaluation metrics like Harrell's C-Statistic. In **survHD**, it can be computed by specifying **measure="HarrellC"** in the call to **evaluate**:

```
ev2 <- evaluate(coxboostmodel, measure='HarrellC', newy=ytest,
                newdata=Xtest)
printl('result(ev2)')
```

```
[[1]]
C Index
0.6612903
```

```
ev3 <- evaluate(coxboostmodel, measure='HarrellC', newy=ytrain,
                newdata=Xtrain)
printl('result(ev3)')
```

```
[[1]]
C Index
0.918429
```

Similarly to the prediction error curves this measure suggests a moderate prediction performance on the independent validation set. As we can see from the second call to **evaluate**, the performance on the independent test data is clearly worse than on the training data, indicating overfitting on the training data. Consequently, evaluation should always be performed using independent test observations.

Two other standard evaluation measures are available in **survHD** - Uno's C-Statistic (**measure='UnoC'**), which models the censoring distribution in the validation set. Cross-validated partial log likelihood (**measure='CvPLogL'**) is also available, and normally used during cross-validation rather than independent validation. Cross-validation will be covered in the next section on "resampling based evaluation."

```
ev4 <- evaluate(coxboostmodel, measure='UnoC', newy=ytest,
                newdata=Xtest, tau=9)
printl('result(ev4)')
```

```
[[1]]
[1] 0.6761186
```

```
ev5 <- evaluate(coxboostmodel, measure='CvPLogL', newy=ytrain,
                newdata=Xtrain)
printl('result(ev5)')
```

```
[[1]]
[1] -66.91228
```

1.4 Resampling based evaluation

Often, there are not enough data to set aside an independent validation set, and in examples such as above, evaluation estimates using a single validation set are sensitive to the selection of that validation set. Cross-validation or the bootstrap are better alternatives where data for all samples are available.

The work flow for resampling is almost the same as for evaluation on independent data. However, one first has to create an object of class **LearningSets** using the function **generateLearningsets**, which defines the validation scheme. In our example we will use 10 fold cross-validation:

```
ls <- generateLearningsets(y=y, method='CV', fold=10, strat=TRUE)
printl('str(ls,  max.level=2)')
```

```
Formal class 'LearningSets' [package "survHD"] with 4
slots
..@ learnmatrix: num [1:10, 1:78] 2 2 2 0 3 2 2 2 2 0
...
..@ method : chr "stratified CV"
..@ ntrain : int 78
..@ iter : int 10
```

By setting the argument **strat** to **TRUE**, we cause **survHD** to create folds with similar proportions of censored to uncensored observations. The most important slot of the returned object (**ls**) is called **learnmatrix** which contains the indices of the training folds in its rows.

The actual model fitting is again performed by the function **learnSurvival** by passing the return-object of **generateLearningsets (ls)** as argument **LearningSets**:

```
outcv <- learnSurvival(y=y, X=X, survmethod='coxBoostSurv',
                       LearningSets=ls, stepno=20)
```

This call automatically fits a CoxBoost model on each training fold. Its return object is of class **LearnOut**. The fitted models can be found in the slot **ModelLearnedlist** and will be used by **survHD** for model evaluation later-on.

```
printl('str(outcv, max.level=2)')
```

```
Formal class 'LearnOut' [package "survHD"] with 7
slots
..@ ModelLearnedlist:List of 10
..@ X :'data.frame': 86 obs. of 7129 variables:
.. .. [list output truncated]
..@ y : Surv [1:86, 1:2] 84.1 91.8+ 93.7+ 108.2+ 34.6
68.1+ 34.2+ 47.0+ 39.1+ 45.8 ...
.. ..- attr(*, "dimnames")=List of 2
.. ..- attr(*, "type")= chr "right"
..@ GeneSel :Formal class 'GeneSel' [package
"survHD"] with 4 slots
..@ nbgene : int 7129
..@ LearningSets :Formal class 'LearningSets'
[package "survHD"] with 4 slots
..@ threshold : num -1
```

The call to **evaluate** is exactly the same as for independent evaluation. The only difference is that the slot **result** of the output object is now a list of the foldwise evaluation statistics as here for measure **HarrellC**:

```
evcv <- evaluate(outcv, measure='HarrellC')
printl('unlist(result(evcv))')
```

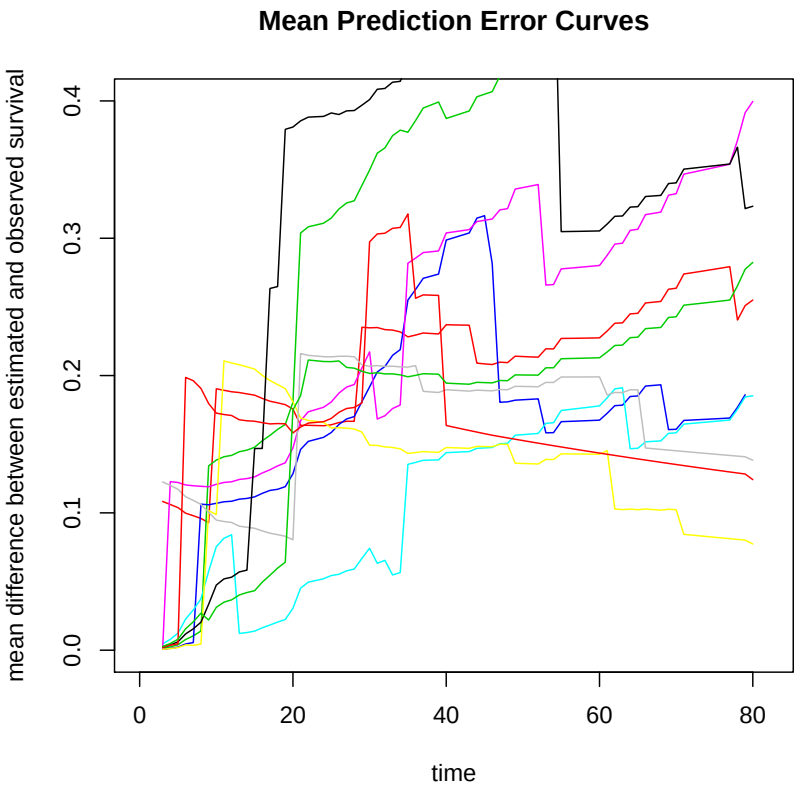
```
C Index C Index C Index C Index C Index C Index
0.8235294 0.2307692 0.8000000 0.0000000 0.2222222
0.9090909
C Index C Index C Index C Index
0.6363636 0.3125000 0.5555556 0.6000000
```

```
printl('mean(unlist(result(evcv)))')
```

```
[1] 0.5090031
```

As can be seen from the vector of fold statistics the variance of the C-Index is large, i.e. the performance was quite different on the ten folds.

```
evcv2 <- evaluate(outcv, measure='PErrC', timegrid=3:80)
plot(0, 0, col='white', xlim=c(0, 82), ylim=c(0, 0.4), xlab='time',
      ylab='mean difference between estimated and observed survival',
      main='Mean Prediction Error Curves')
for(i in 1:10){
 lines(3:80, result(evcv2)[[i]], col=i+1)
 }
```



Also here, one can observe that the prediction performance of the models on the different training folds decreases after 20 months.

1.5 Univariate feature selection

survHD allows the incorporation of a univariate pre-filter with any of the learning algorithms. This can help interpretability of models generated from algorithms which otherwise do not perform feature selection, as well as dramatically reducing computation times. **survHD** also provides a fast version of coxph **rowCoxTests**, for fitting the univariate Cox models to many features.

In **survHD**, variable selection is implemented in the function **geneSelection**. Parallel to **learnSurvival**, this function performs a variable selection for each resampling step if the argument **LearningSets** is passed. In our example, we use the method **fastCox**, i.e. univariate Cox models for each gene expression in **X**:

```
gsel <- geneSelection(X=X, y=y, LearningSets=ls, method='fastCox',
                      crit='coefficient')
```

If the argument **crit** is set to **pvalue survHD** will store the p-values instead of the coefficients in the return-object. This might be useful if one wants to set a certain threshold for the p-value of the variables included in a model later-on.

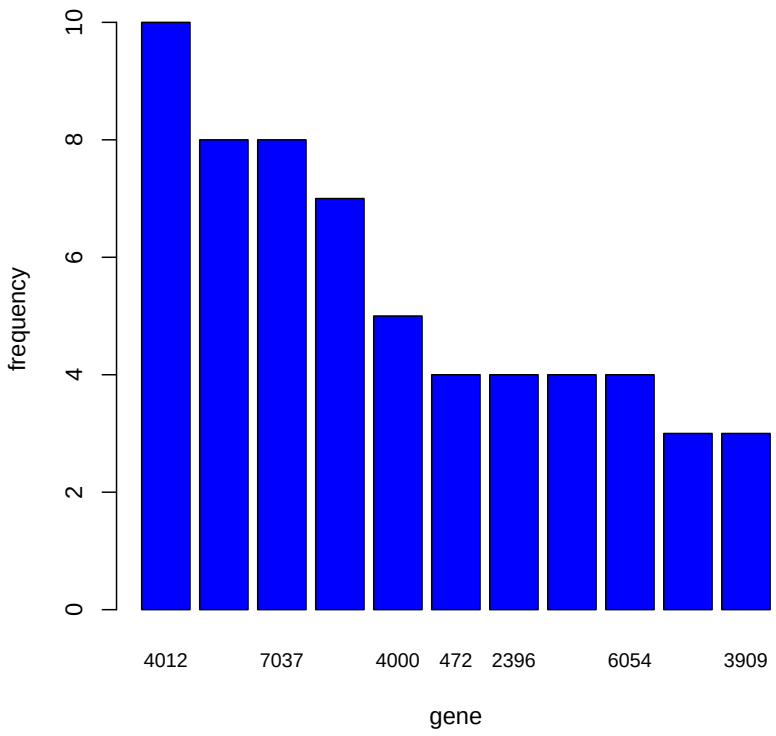
```
printl('str(gsel, max.level=2)')
```

```
Formal class 'GeneSel' [package "survHD"] with 4
slots
..@ rankings :List of 1
..@ importance:List of 1
..@ method : chr "fastCox"
..@ criterion : chr "coefficient"
```

As can be seen from the previous code, **geneSelection** returns an object of class **GeneSel** which contains the rankings of the different gene expressions as well as their importance. Using the slot **rankings** we can easily see that there is only a single gene which belongs to the Top 10 in each fold.

```
ranks <- c()
for(i in 1:10)
   ranks <- c(ranks, rankings(gsel)[[1]][i, 1:10])
genetab <- sort(table(ranks), decreasing=T)
bp <- barplot(genetab[genetab>=3], col='blue', xlab='gene',
              ylab='frequency',  cex.names=0.8)
printl('abs(importance(gsel)[[1]][1, 1:10])')
```

```
gene4729 gene4012 gene472 gene2044 gene7037 gene4885
6.491064 5.976582 5.692645 5.363823 5.344502 5.200338
gene3841 gene1476 gene3041 gene3267
5.183888 5.180810 5.098753 5.091743
```



The last line of code illustrates the absolute coefficients for the top 10 genes in the first cross-validation fold.

1.6 Hyperparameter Tuning

Survival models for high-dimensional data usually incorporate a hyperparameter in order to adjust model complexity to the requirements of the data at hand. For most of these hyperparameters no good rule of thumb exists, and one has to resort to resampling approaches to optimize or tune the hyperparameter(s). As well as variable selection, the hyperparameter optimization has to be performed on each training set, resulting in a nested cross-validation or comparable resampling schemes (an inner or nested layer for tuning, and an outer layer for evaluation).

The function **tune** performs hyperparameter optimization for each training set of an (outer) resampling procedure by conducting an internal cross-validation loop. The hyperparameter grid is passed as a list of vectors containing the candidate values. Since we solely tune the number of boosting steps here the argument **grids** is a one-element list with the vector of candidate numbers for the boosting steps. Please note, that the name of the vector has to exactly match the argument name of the tuned hyperparameter (here **stepno** as in the corresponding **CoxBoost**-function).

```
tunecoxboost <- tune(y=y, X=X, survmethod='coxBoostSurv',
                     LearningSets=ls, GeneSel=gsel, nbgene=30,
                     grids=list(stepno=(1:5)*20))
```

By additionally specifying the **GeneSel** argument and **nbgene=100**, **survHD** will tune the models based on the top 100 genes in each fold.

The best parameter of each fold **(res.ind)** as well as its index in the tuning grid can be obtained using the function **getBestParameters**. Parallel to the function **evaluate**,

one can choose between several measures according to which the best parameter shall be determined. Here we use the cross-validated partial log likelihood.

```
gb <- getBestParameters(tunecoxboost, res.ind=1, measure='CvPLogL')
printl('param(gb)')
```

```
$stepno
[1] 20
```

```
printl('bestind(gb)')
```

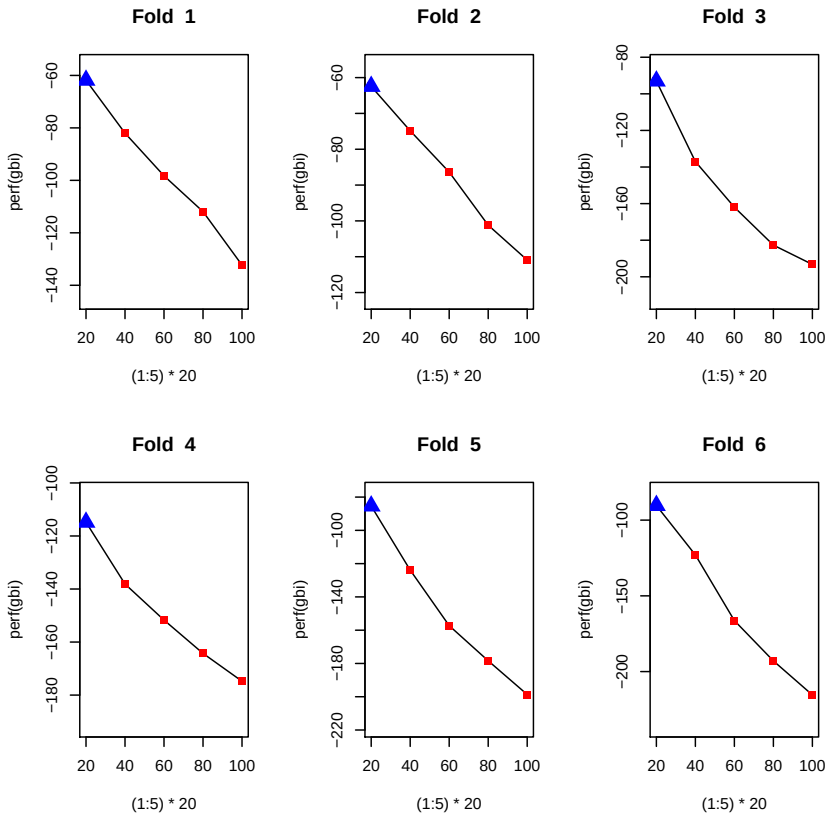
```
[1] 1
```

```
printl('str(gb, max.level=2)')
```

```
Formal class 'BestParameters' [package "survHD"] with
4 slots
..@ param :List of 1
..@ bestind: int 1
..@ perf : num [1:5] -61.8 -82 -98.3 -111.9 -132.3
..@ measure:Formal class 'CvPLogL' [package "survHD"]
with 2 slots
```

We can also check how stable and distinct the tuning process has been, since **getBestParameters** returns the performance of all hyperparameter values:

```
par(mfrow = c(2, 3))
for (i in 1:6) {
     gbi <- getBestParameters(tunecoxboost, res.ind = i,
         measure = "CvPLogL")
     plot((1:5) * 20, perf(gbi), type = "l", col = "black",
         ylim = range(perf(gbi)) * c(1.1, 0.9), main = paste("Fold ",
             i, sep = " "))
     points((1:5) * 20,perf(gbi), col = "red", pch = 15)
     points(((1:5) * 20)[bestind(gbi)], perf(gbi)[bestind(gbi)],
         col = "blue", pch = 17, cex = 2)
 }
```



In our case, the lowest number of boost-steps achieves the best result in all of the first 6 folds.

1.7 Complete Workflow

So, let us summarize the steps needed for a complete work flow. After loading the data, we have to choose the resampling scheme and create the corresponding LearningSets:

```
ls <- generateLearningsets(y=y, method='CV', fold=10, strat=TRUE)
```

If necessary, the next step is variable selection using a filter method like **fastCox**. The generated resampling folds are passed via the argument **LearningSets**:

```
gsel <- geneSelection(X=X, y=y, LearningSets=ls, method='fastCox',
                      crit='coefficient')
```

Subsequently, for each training set a suitable hyperparameter has to be determined. If a filter method has been applied, the corresponding **GeneSel** object must be passed as well as the **LearningSets**:

```
tunecoxboost <- tune(y=y, X=X, survmethod='coxBoostSurv',
                LearningSets=ls, GeneSel=gsel, nbgene=100,
                grids=list(stepno=(1:5)*20))
```

Finally, all three previously obtained outputs can be combined in a call to **learnSurvival**, which will fit the requested survival model on each learningset using the variables selected in **GeneSel** and tune the hyperparameters according to their performance in **tuneres**. Regarding the incorporation of a **tuneres** argument, one must

additionally specify by which measure to evaluate the tuning results. This measure is specified in the argument **addtune** which is a list of arguments which is internally passed to function **evaluate**. Cross-validated partial log-likelihood is standard, but any available evaluation metrics may be used.

```
cv10 <- learnSurvival(y=y, X=X, tuneres=tunecoxboost,
           survmethod='coxBoostSurv', LearningSets=ls, GeneSel=gsel,
                     nbgene=30, addtune=list(measure='CvPLogL'))
```

The function **evaluate** can be used to assess the performance of the survival method. The performance is returned separately for each fold, and here we combine these using a simple average:

```
printl("mean(unlist(result(evaluate(cv10,
           measure='CvPLogL'))))")
```

```
[1] -12.75058
```

```
printl("mean(unlist(result(evaluate(cv10,
           measure='HarrellC'))))")
```

```
[1] 0.3828795
```

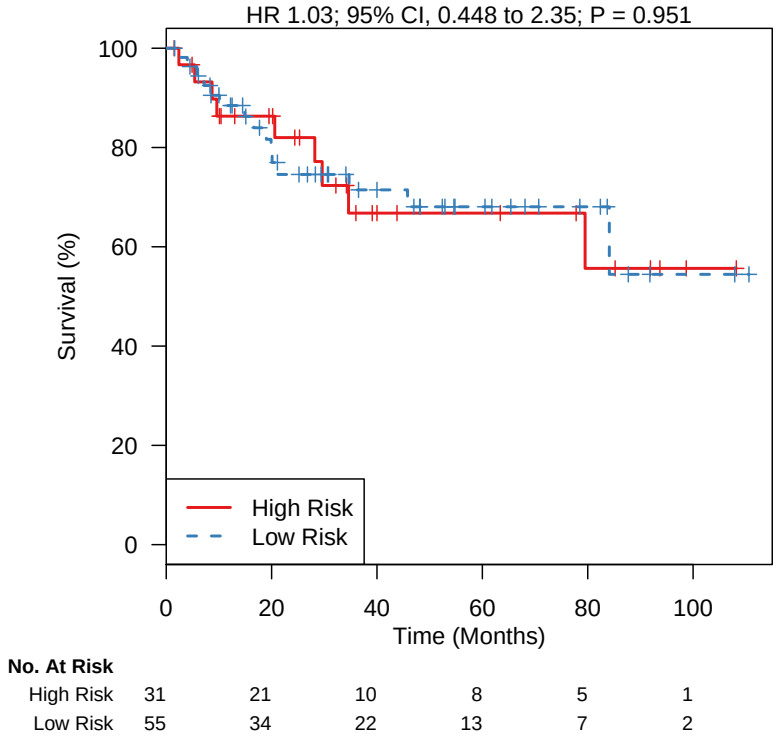
We also illustrate the performance of the models computed in the learnSurvival-step. Using the **predict** function on an object of class **LearnOut (with type='cp')** causes **survHD** to use the linear predictors –obtained for all test observations– to categorize samples into two risk classes.

```
preds <- predict(cv10, type='cp', voting_scheme='first')
printl('table(preds)')
```

```
preds
High Risk Low Risk
31 55
```

Subsequently, one can plot the corresponding Kaplan Meier curves. In **survHD** this can be achieved by simply using the function **plot** on the previous **LearnOut**-object.

```
plot(cv10)
```



1.8 Comparison of two models

In **survHD** two models can also be compared in a more direct way than only evaluating them according to a certain measure. The function **IDI.INF** from the package **survIDINRI** –among other measures– computes the median improvement in riskscore of a model in comparison with a null model. This feature is implemented in the function **compare**. We first have to fit two models using **learnSurvival**. The first one is the CoxBoost model described above whereas the second one is constructed using the plusMinus-algorithm:

```
Xtrain <- Xtrain[1:60, ]
ytrain <- y[1:60, ]
coxboostmodel1 <- learnSurvival(y = ytrain, X = Xtrain,
     survmethod = "coxBoostSurv", stepno = 20)
coxboostmodel2 <- learnSurvival(y = ytrain, X = Xtrain,
     survmethod = "coxBoostSurv", stepno = 40)
```

After fitting the models, they can be compared with regard to certain test data. Here we use the Integrated Discrimination Improvement Index (**IDI**):

```
ytest <- y[61:86, ]
Xtest <- X[61:86, ]
comp1 <- compare(coxboostmodel1, coxboostmodel2, measure = "IDI",
     newdata = Xtest, newy = ytest, t0 = 50)
printl('estimate(comp1[[1]])')
```

```
0.003753648
```

```
 comp2 <- compare(coxboostmodel2, coxboostmodel1, measure = "IDI",
      newdata = Xtest, newy = ytest, t0 = 50)
 estimate(comp2[[1]])
```

```
-0.003753648
```

```
 printl('str(comp2[[1]], max.level = 2)')
```

```
Formal class 'IDI' [package "survHD"] with 2 slots
..@ estimate: Named num -0.00375
.. ..- attr(*, "names")= chr ""
..@ conf.int: Named num [1:2] -0.0179 0.0258
.. ..- attr(*, "names")= chr [1:2] "2.5%" "97.5%"
```

The class bf MeasureOut also contains a confidence interval for the computed measure as can be seen from the output. Other measures for comparison of two survival models in **survHD** are:

- **NRI** Category-less Net Reclassification Index
- **MIRS** Median Improvement in Risk Score
- **CDelta** Difference in C-Statstic

2 Cross-Study Evaluation

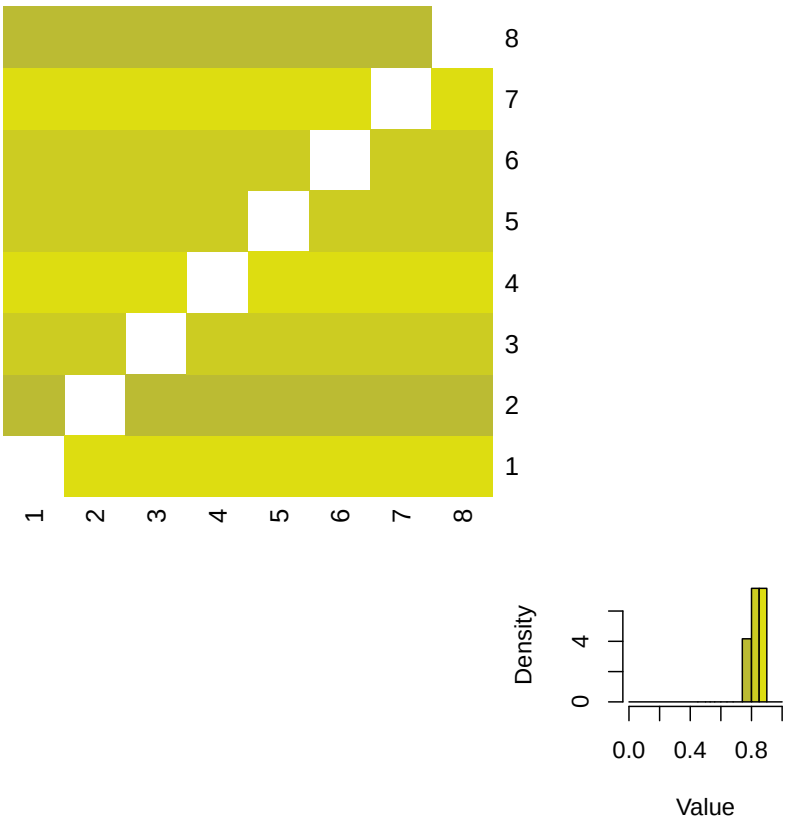
The package also provides a simple function for leave-one-in cross-study validation. This procedure is based on a list of experimental data sets. Each of the data sets is once used for training. Each fitted model is subsequently evaluated on all other data sets. In the following, we first create the main input parameters of the corresponding function **computeZ**, a list of gene expression matrices **Xlist** and a list of the corresponding survival observations **ylist**.

```
 set.seed(2)
 nsamples <- 100
 X <- matrix(rnorm(nsamples*200),nrow=nsamples)
 colnames(X) <- make.names(1:ncol(X))
 rownames(X) <- make.names(1:nrow(X))
 time <- rexp(nsamples)
 cens <- sample(0:1,size=nsamples,replace=TRUE)
 y <- Surv(time,cens)
 Xlist<-list()
 ylist<-list()
 for(i in 1:8){
  Xlist[[i]]<-X
  ylist[[i]]<-y
  }
```

Once these lists of gene expressions and survival observations are available, the procedure can be performed using a single line. We just have to specify the **measure** to be used and the algorithm (**survmethod**) which shall be applied. In our case, we also specify that we do not want the method **CoxBoost** to standardize its inputs which we can achieve via the argument **add.surv** which is a **list** of arguments passed to the

actual survival algorithm. Moreover, we use **survHD**'s tuning mechanism instead of a **CoxBoost** specific tuning mechanism by setting **packagetune=FALSE** and passing an empty hyper parameter grid via the argument **add.tune**, i.e. we also use **survHD**'s default hyper parameter grid for **CoxBoost**. The argument **plot=TRUE** tells **survHD** that a heatmap of the resulting **Z**-matrix shall be produced.

```
add.tune<-list(grids=list())
add.surv<-list(standardize=FALSE)
Zmat<-computeZ(Xlist=Xlist,ylist=ylist,survmethod='coxBoostSurv',
               measure='HarrellC',add.surv=add.surv,
               add.tune=add.tune,packagetune=FALSE,plot=TRUE,seed=222)
```




```
printl('Zmat[1,]')
```

```
1 2 3 4 5 6
NA 0.8818582 0.8818582 0.8818582 0.8818582 0.8818582
7 8
0.8818582 0.8818582
```

In this example, we have just reproduced the same data set eight times and we have performed the leave-one-in cross-study validation on this list of data sets. Since all the data sets are the same, all values in a row (rows correspond to the training data set) are equal because one and the same model is evaluated on identical data sets. However, the values in a column are not equal because the training process is subject to randomness in our example. The tuning process is based on cross-validation with ranomized partitioning which can lead to different results even for identical training data sets. Since training and validation sets are equal in our example, we observe very high C-Indices since we are actually computing resubstitution concordances.

3 Custom Survival Models

The **survHD** package provides an interface for incorporating user defined survival methods into its framework. For that purpose, two functionalities are needed:

- model fitting
- prediction from previously fitted models

Both features are implemented by defining a single function. In our example we will add the method **rsf** from the package **randomSurvivalForest** to **survHD** as a custom function **customRSF**. The user-defined function has to accept at least three predefined inputs:

- **Xlearn:** A **data.frame** of gene expressions for the current training data(columns are genes)
- **Ylearn:** the survival response for the current training data
- **learnind**: the indices of the current training observations inside the complete data set **X** which is usually passed to functions like **learnSurvival**. Using this argument, one can e.g. extract the relevant observations from additional clinical covariates which can be passed using the ... argument.

```
customRSF <- function(Xlearn, Ylearn, learnind, ...) {
    ### load required packages
    require(randomSurvivalForest)
    ### handle inputs
    ll <- list(...)
    datarsf <- data.frame(Xlearn, time = Ylearn[, 1],
        status = Ylearn[, 2])
    ll$data <- datarsf
    ll$formula <- as.formula("Surv(time, status)~.")

    ## call actual model function rsf from
    ## randomSurvivalForest
    output.rsf <- do.call("rsf", args = ll)
    ...
}
```

First, one typically has to load or source a function or package which performs the actual model fitting. The next step is the processing of the inputs. In our case we do not use any additional covariates, so we only have to convert **Xlearn** and **Ylearn** into the input format of the function **rsf**. As can be seen from the code, this function accepts a **formula** and a **data.frame** containing all necessary variables. Moreover, we pass all the arguments represented by the **...** argument and eventually call the function **rsf** using **do.call**. After this the model fitting is complete.

In order to provide outputs which can be evaluated and processed by **survHD** we also need a **predict** function. This function should either provide an object of class **LinearPrediction** or **SurvivalProbs** whose slots can be found below:

```
LinearPrediction <- predict(coxboostmodel, newdata=Xtest,
type='lp')[[1]]
str(LinearPrediction, max.level=2)
```

```
Formal class 'LinearPrediction' [package "survHD"] with 1 slots
  ..@ lp: Named num [1:26] 1.952 -0.1976 -0.0529 -0.1154 -0.5334 ...
  .. ..- attr(*,  "names")= chr [1:26] "L59" "L61" "L62" "L64" ...
```

```
SurvivalProbs <- predict(coxboostmodel, newdata=Xtest, type=
   'SurvivalProbs', timegrid=3:80)[[1]]
str(SurvivalProbs, max.level=2)
```

The prediction function has to accept four arguments:

- **object:** an object of class **ModelCustom** which is created by **survHD** automatically and contains the fitted model (here: **output.rsf**) in its slot **mod**.
- **newdata: data.frame** of gene expressions for the observations for which predictions shall be performed.
- **type:** indicates whether linear predictors (**'lp'** or survival probabilities (**'SurvivalProbs'**) shall be predicted
- **timegrid:** if **type='SurvivalProbs'** this argument specifies the time points at which predictions shall be performed

```
## define prediction function which will be
## stored in slot predfun and called by
## predictsurvhd (signature(ModelCustom))
predfun <- function(object, newdata, type, timegrid = NULL,
    ...) {
    require(randomSurvivalForest)
    # either type lp or type SurvivalProbs must be
    # implemented for typ lp the obligatory return
    # class is LinearPrediction
    if (type == "lp") {
        stop("Random Forests don't provide linear predictors,  sorry.")
    } else if (type == "SurvivalProbs") {
        # for typ SurvivalProbs the obligatory return
        # class is Breslow
        modelobj <- mod(object)
        if (is.null(timegrid)) {
            stop("No timegrid specified.")
        }

        ### create data for which predictions are to be
        ### performed function checks for response but
        ### does not use it (fake response)
        predsrsf <- predict.rsf(object = modelobj,
            test = data.frame(newdata, time = rexp(n = nrow(newdata)),
                status = sample(c(0, 1), nrow(newdata),
                  replace = T)))
        ### predict-function provides predictions for
        ### training times only ->interpolate for
        ### timepoints in timegrid
        curves <- exp(-t(apply(predsrsf$ensemble, 1,
            FUN = function(z) approx(x = predsrsf$timeInterest,
                y = z, xout = timegrid)$y)))
        ### create breslow-object
```

```
        pred <- new("breslow", curves = curves, time = timegrid)
        ### create SurvivalProbs-object embedding the
        ### breslow-object
        pred <- new("SurvivalProbs", SurvivalProbs = pred)
    } else stop("Invalid \"type\" argument.")
    return(pred)
}
```

This function is structured into two sections which correspond to predicting linear predictors and survival probabilities respectively. Since random survival forests are not capable of providing linear predictors the first section simply returns an error. In the survival probability section at first the input data have to be preprocessed for the function **predict.rsf** which can perform predictions on the basis of objects of class **randomSurvivalForest** created by the function **rsf**. Subsequently, these predictions are estimated and eventually an object of class **SurvivalProbs** is created which is the predefined class for survival probabilities in **survHD**. Its only slot **SurvivalProbs** is an object of class **Breslow** which not only stores the survival probabilities in the slot **curves** but also the time points in slot **time**. This **survprob** object is finally returned by the custom prediction function.

The definition of this prediction function was the second part of the user-defined survival function **customRSF**. In the end, we still have to create the obligatory output object of class **ModelCustom** which primarily consists of the fitted model object **output.rsf** in slot **mod** and the user-defined prediction function in slot **predfun**. Additional information can be stored in the slot **extraData** which must be of class **list**:

```
 ###now create customsurvhd-object (which is the obligatory output-object)
 custommod <- new("ModelCustom", mod=output.rsf, predfun=predfun,
             extraData=list())
 return(custommod)
```

Just for the sake of completeness, the complete user defined survival method looks like follows. It basically consists of three parts: model fitting, prediction function, creating of the **customsurvhd** object:

```
 ### random survival forest as a custom survival
### model function Xlearn and Ylearn are
### obligatory inputs
customRSF <- function(Xlearn, Ylearn, learnind, ...) {
    ### load required packages
    require(randomSurvivalForest)
    ### handle inputs

    ll <- list(...)
    datarsf <- data.frame(Xlearn, time = Ylearn[, 1],
        status = Ylearn[, 2])
    ll$data <- datarsf
    ll$formula <- as.formula("Surv(time, status)~.")

    ## call actual model function rsf from
    ## randomSurvivalForest
    output.rsf <- do.call("rsf", args = ll)

    ## define prediction function which will be
```

```
    ## stored in slot predfun and called by
    ## predictsurvhd (signature(ModelCustom))
    predfun <- function(object, newdata, type, timegrid = NULL,
        ...) {
        require(randomSurvivalForest)
        # either type lp or type SurvivalProbs must be
        # implemented for typ lp the obligatory return
        # class is LinearPrediction
        if (type == "lp") {
            stop("Random Forests don't provide linear predictors,  sorry.")
        } else if (type == "SurvivalProbs") {
            # for typ SurvivalProbs the obligatory return
            # class is Breslow
            modelobj <- object@mod
            if (is.null(timegrid)) {
                stop("No timegrid specified.")
            }

            ### create data for which predictions are to be
            ### performed function checks for response but
            ### does not use it (fake response)
            predsrsf <- predict.rsf(object = modelobj,
                test = data.frame(newdata, time = rexp(n = nrow(newdata)),
                  status = sample(c(0, 1), nrow(newdata),
                    replace = T)))
            ### predict-function provides predictions for
            ### training times only ->interpolate for
            ### timepoints in timegrid
            curves <- exp(-t(apply(predsrsf$ensemble,
                1, FUN = function(z) approx(x = predsrsf$timeInterest,
                  y = z, xout = timegrid)$y)))
            ### create breslow-object
            pred <- new("breslow", curves = curves,
                time = timegrid)
            ### create SurvivalProbs-object embedding the
            ### breslow-object
            pred <- new("SurvivalProbs", SurvivalProbs = pred)
        } else stop("Invalid \"type\" argument.")
        return(pred)
    }


    ### now create customsurvhd-object (obligatory
    ### output-object)
    custommod <- new("ModelCustom", mod = output.rsf,
        predfun = predfun, extraData = list())

    return(custommod)
}
```

It is practical to make sure that the custom function is in the global environment of **R** such that it is available for all functions of **survHD**.

```
 ###define function in global envir
 assign(x="customRSF", value=customRSF, envir=.GlobalEnv)
```

Using the custom function **customRSF** we can easily implement a complete work flow of generating LearningSets, gene selection, tuning, resampling and a final evaluation.

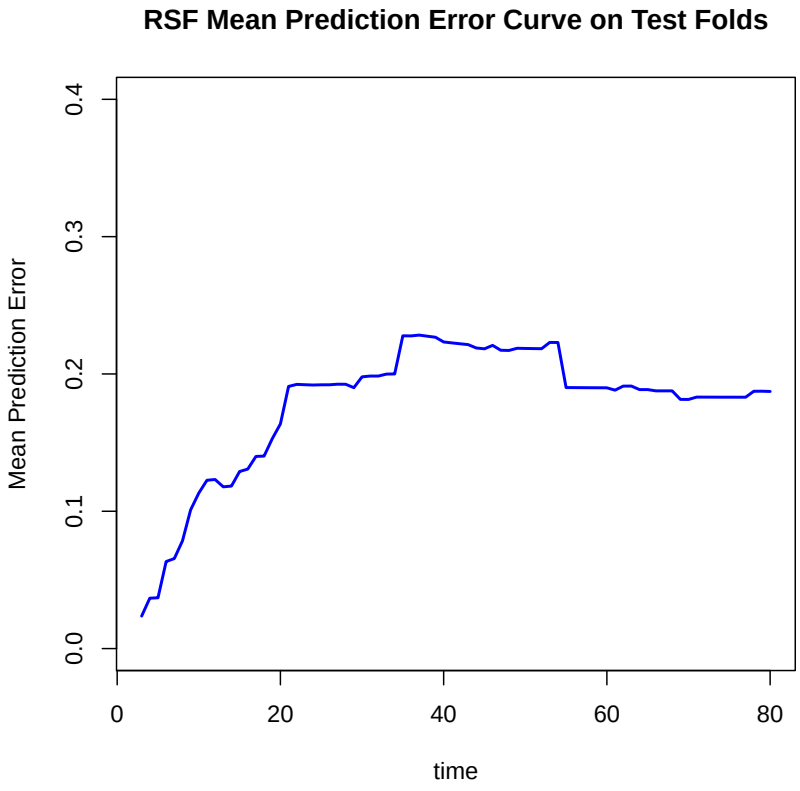
```
## learningset
ls <- generateLearningsets(y = y[, 1], method = "CV",
     fold = 5)
# gene selection
gsel <- geneSelection(X = X, y = y, method = "fastCox",
     LearningSets = ls, criterion = "coefficient")
### tune
tuneres <- tune(X = X, y = y, GeneSel = gsel, nbgene = 30,
     survmethod = "customSurv", customSurvModel = customRSF,
     LearningSets = ls, grids = list(ntree = 20 * (1:10)))
### use tuneres in learnSurvival
svaggr <- learnSurvival(X = X, y = y, GeneSel = gsel,
     nbgene = 30, survmethod = "customSurv", customSurvModel =
     customRSF, LearningSets = ls, tuneres = tuneres, measure =
     "PErrC", timegrid = 4:10, gbm = FALSE, addtune = list(GeneSel =
     gsel, nbgene = 30))
```

The only essential difference is the argument **survmethod** which is set to **'customSurv'**.

Note that even tuning does not need any additional code apart from defining a reasonable tuning grid (argument **grids**) for the **rsf**-specific argument **ntree**. The last step is the evaluation where we have to resort to **measure='PErrC'** since it is the only one which can be computed without a linear predictor:

```
###error expected because rsf does not provide lp
try(evaluate(svaggr, measure='CvPLogL'))
###error expected because rsf does not provide lp
try(evaluate(svaggr, measure='PErrC', timegrid=3:80, gbm=T))
###works without lp
evcust <- evaluate(svaggr, measure='PErrC', timegrid=3:80, gbm=F)

perrcs <- c()
for (i in 1:5) {
     perrcs <- cbind(perrcs, result(evcust)[[i]])
 }
plot(3:80, rowMeans(perrcs, na.rm = TRUE), , col = "blue",
     lwd = 2, main = "RSF Mean Prediction Error Curve on Test Folds",
     xlab = "time", ylab = "Mean Prediction Error",
     type = "l", ylim = c(0, 0.4), xlim = c(3, xlim2))
```



The lower mean prediction error curve suggests a better performance on the data compared to the previous results for **CoxBoostSurv**.

4 Gene Set Analysis

It is often of interest to test whether the expression of a particular group of genes is associated with some phenotype. This is commonly a phenotype that differs between two groups, for example between a test and a control group. This test is easily generalized to different phenotype classes like survival outcome by first ranking all genes in a univariate fashion and then testing for a non-random ranking of gene sets. We first create a random data set:

```
set.seed(100)
x  <- matrix(rnorm(1000*20), ncol=20)
dd <- sample(1:1000, size=100)
u  <- matrix(2*rnorm(100), ncol=10, nrow=100)
x[dd, 11:20] <- x[dd, 11:20]+u
y <- Surv(c(rnorm(10)+1, rnorm(10)+2),  rep(TRUE,  20))
genenames   <- paste("g", 1:1000, sep="")
rownames(x) <- genenames
```

Now we need some sets of genes which we will test for association with the outcome. Here we just create some random ones:

```
genesets <- vector("list", 50)
for(i in 1:50){
     genesets[[i]] <- paste("g", sample(1:1000, size=30), sep="")
 }
geneset.names <- paste("set", as.character(1:50), sep="")
```

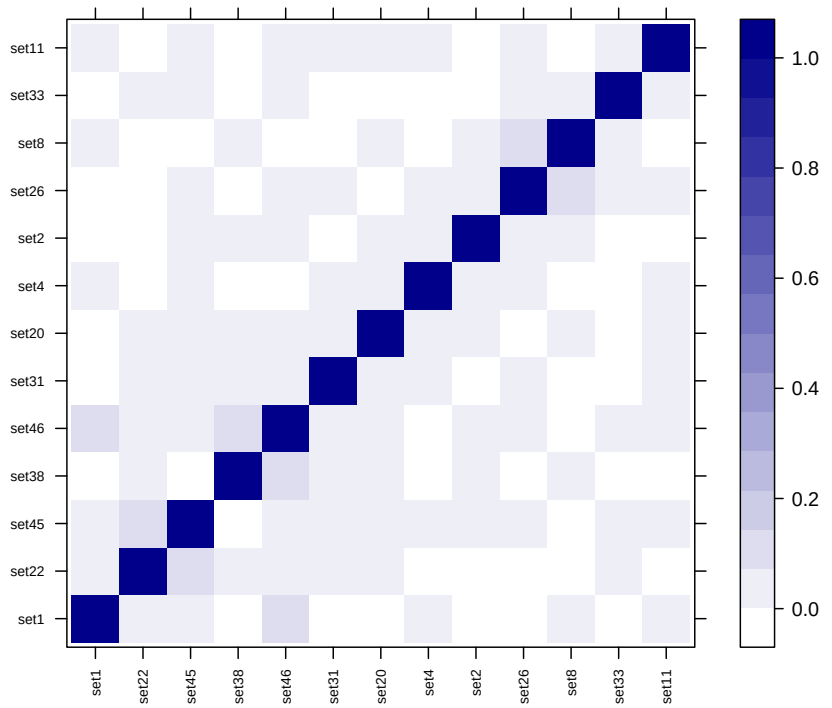


Figure 1: GSA overlap plot. This plot visualizes the similarity of gene sets that are associated with the phenotype of interest. Note that the p-values are not adjusted for multiple testing.

A well curated resource of gene sets is the MSigDB. It provides gene sets in tab separated files with the suffix gmt. These files can be imported with the `gsaReadGmt` function. For now, we just create such a gmt data structure from the list of random sets:

```
gmt <- new("gsagenesets",  genesets=genesets,
                  geneset.names=geneset.names)
```

The focus of this package is survival outcome, so testing gene sets for association with survival is simple:

```
gsa.res <- gsaWilcoxSurv(gmt, X=x, y=y, cluster=FALSE, p.value=0.3)
```

The `cluster` command can be used to cluster similar gene sets in the ranking. The similarity of gene sets can also be explored with the `plot` function and the results are shown in Figure 1:

```
plot(gsa.res)
```

Another useful plot is a barplot, which visualizes the ranking of genes in up to two gene sets (Figure 2):

```
plot(gsa.res,  type="barcode", geneset.id1="set22",
     geneset.id2="set33")
```

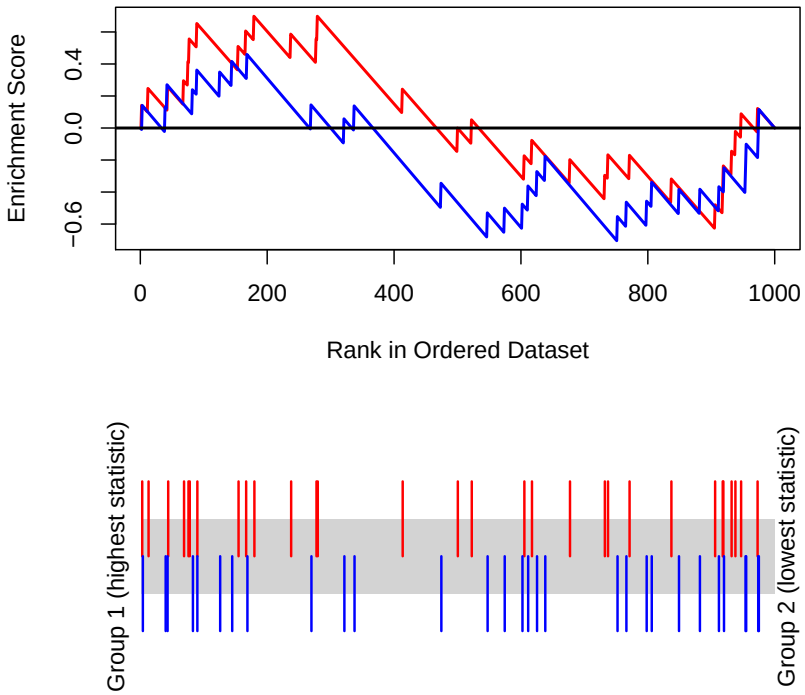


Figure 2: GSA barplot. The horizontal lines on the bottom visualize the positions of up to two different gene sets in a univariate ranking, colored here in red and blue. The plot on the top visualizes the local enrichment. It makes sense to compare two sets in one plot if the gene sets are devided in up- and down-regulated genes. This has the advantage that not only non-randomness is shown, but also that the gene expression direction is consistent. Red are typically the up-regulated genes, blue down-regulated genes. Genes with high statistic have a high hazard ratio and vice versa.

For phenotypes other than survival, `gsaWilcoxSurv` accepts a ranking of genes. In our example, we split the data in two groups and test gene sets for association with this splitting:

```
library(genefilter)
genes.ttest <- rowttests(x,  as.factor(c(rep(1, 10), rep(2, 10))))
gsa.res.tt  <- gsaWilcoxSurv(gmt,  genenames=rownames(x),
     statistics=genes.ttest[, 1])
```

All the examples above assume that the rownames of `X` correspond to the names of the genes in the gene sets. Here an example in which the rownames of `X` are Affymetrix probe ids and the genes in the gene set official gene symbols:

```
data(beer.exprs)
data(beer.survival)
library(hu6800.db)
library(annotate)
gmt <- gsaReadGmt(system.file("extdata/ovarian_gene_signatures.gmt",
     package = "survHD"))
```

We convert the symbols to Affymetrix ids with the `gsaTranslateGmt` function and Bioconductor's annotate package:

```
genes    <- getSYMBOL(rownames(beer.exprs),  "hu6800")
gmt.affy <- gsaTranslateGmt(gmt,  beer.exprs,  genes)
gsa.res  <- gsaWilcoxSurv(gmt.affy,  beer.exprs,
                Surv(beer.survival[, 2],  beer.survival[, 1]))
```

Note that if a gene is represented by more than one probe set, this code will use the first match in the GSA. It is recommended to use for example the `collapseRows` function of the WGNCA package to pick reasonable probe sets.

C.2 Bigmemory Benchmark at the IBE-Cluster

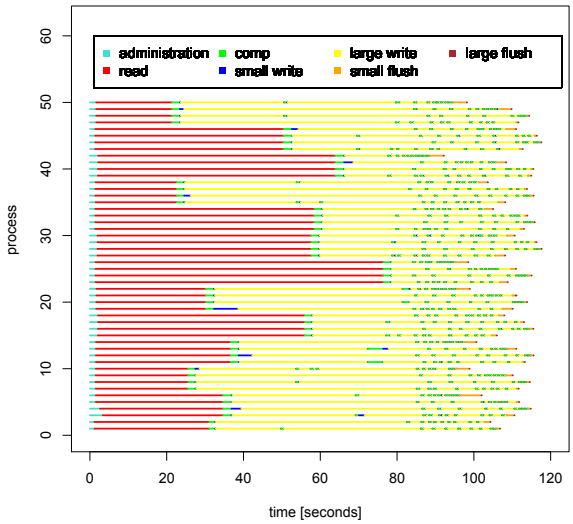


Figure C.1: Ganttchart for bigmemory-experiment with 50 processes at the IBE-cluster writing and reading 50x5000 matrices using a memory attached file.

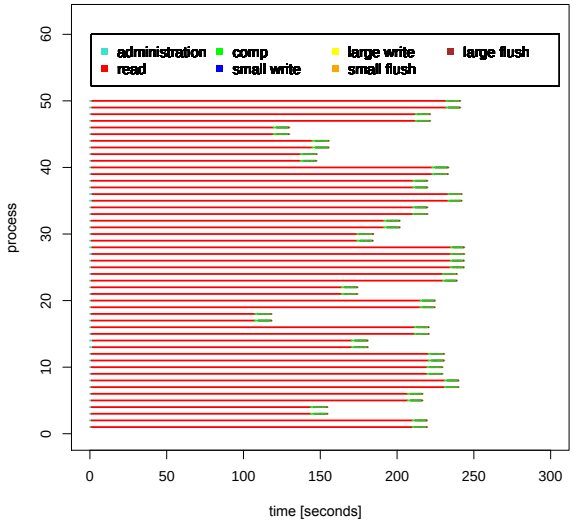


Figure C.2: Ganttchart for bigmemory-experiment with 50 processes at the IBE-cluster reading 500x5000 matrices from a memory attached file.

Appendix D

Papers containing parts of the work presented in this thesis

D.1 Correcting the Optimal Resampling-Based Error Rate by Estimating the Error Rate of Wrapper Algorithms (Bernau et al., 2013)

This paper has been written in cooperation with Thomas Augustin (Department of Statistics, Ludwig-Maximilians-Universität Munchen) and Anne-Laure Boulesteix (Department for Medical Informatics, Biometry and Epidemiology) in the context of the work presented in Chapter 1. Thomas Augustin has primarily supported the creation of the section related to decision theory (Section 1.5.3 in Chaper 1) as well as the abstract. He also helped revising the paper and its notation, and provided substantial feedback and comments as well as support in the context of the submission process. Anne-Laure Boulesteix proposed the idea to use an approximation based on the normal distribution for estimating the probabilities $\mathrm{P}\left(k^*(S) = k\right)$, $k = 1, \ldots, K$, in Eq. 1.5 as well as the general idea for developing a method for the correction of the optimal error rate. Furthermore, she has substantially contributed to *Sections 1, 2.2, 2.4, 3.1, 3.2* and *4.1* of the paper and the corresponding sections in Chaper 1. Additionally, she provided helpful comments, remarks and suggestions for improvement.

Biometrics 69, 693–702
September 2013

DOI: 10.1111/biom.12041

Correcting the Optimal Resampling-Based Error Rate by Estimating the Error Rate of Wrapper Algorithms

Christoph Bernau,[1,*] Thomas Augustin[2], Anne-Laure Boulesteix[1]

[1]Department for Medical Informatics, Biometry and Epidemiology, Marchioninistr. 15, D-81377 Munich, Germany

[2]Department of Statistics, University of Munich, Ludwigstr 33, D-80539 Munich, Germany

[*]*email:* bernau@ibe.med.uni-muenchen.de

Summary. High-dimensional binary classification tasks, for example, the classification of microarray samples into normal and cancer tissues, usually involve a tuning parameter. By reporting the performance of the best tuning parameter value only, over-optimistic prediction errors are obtained. For correcting this tuning bias, we develop a new method which is based on a decomposition of the unconditional error rate involving the tuning procedure, that is, we estimate the error rate of wrapper algorithms as introduced in the context of internal cross-validation (ICV) by Varma and Simon (2006, BMC Bioinformatics 7, 91). Our subsampling-based estimator can be written as a weighted mean of the errors obtained using the different tuning parameter values, and thus can be interpreted as a smooth version of ICV, which is the standard approach for avoiding tuning bias. In contrast to ICV, our method guarantees intuitive bounds for the corrected error. Additionally, we suggest to use bias correction methods also to address the conceptually similar method selection bias that results from the optimal choice of the classification method itself when evaluating several methods successively. We demonstrate the performance of our method on microarray and simulated data and compare it to ICV. This study suggests that our approach yields competitive estimates at a much lower computational price.

Key words: Classification; High-dimensional data; Method selection bias; Repeated subsampling; Tuning bias.

1. Introduction

Resampling-based procedures are routinely applied in order to assess the performance of statistical learning methods by estimating their prediction error. If the available data set were large enough, it would be recommended to partition the data into learning and validation data, to fit a model using the learning data, and to estimate its error based on the validation data. In the common case of small sample high-dimensional data considered in the present article, however, the available data set is usually too small for such a partitioning. Resampling-based procedures are thus particularly useful in the context of "$n \ll p$" data analysis, that is, when the number of predictors exceeds the number of observations.

In practice, most common classification methods for high-dimensional data involve a tuning parameter, for example, the cost parameter in linear support vector machines (SVM) or the number of neighbors in k-nearest-neighbors (kNN). If the error of a classification method is estimated by a resampling method several times with different values of the tuning parameter successively, each parameter value possibly yields a different estimated error. The approach consisting in selecting the parameter value yielding the smallest resampling error estimate and only reporting this resampling estimate is biased (Dupuy and Simon, 2007). That is because the minimal resampling error can be seen as the result of an optimal selection. As such, it is a biased estimate of the generalization error rate, that is, of the error that would be obtained with this parameter value on independent data. This bias, that is quantitatively assessed by Varma and Simon (2006) in the "$n \ll p$" setting, is denoted here as *tuning bias*. Note that the term "tuning" may be ambiguous since researchers from different fields might have different understandings of tuning. In this article, we consider a parameter as a tuning parameter if it is not optimized by an analytical method (like the least squares criterion for the coefficients in linear regression) but rather by trying several values successively and using the value yielding the best prediction performance on test data. When choosing the parameter value based on the performance yielded by different candidate values, one indirectly uses the test data for learning the decision function, leading to an optimistic bias.

A similar bias, called method selection bias in the sequel, occurs if a researcher tries out several classification methods successively and reports only the results of the method yielding the minimal error rate. For instance, suppose we compute the resampling error rate of SVM, Random Forests (RF), kNN, and L_1-penalized regression for a particular data set. Suppose further that kNN yields the smallest error rate in the resampling approach. This error rate is likely to be smaller than the error rate of kNN on independent data, because it was *optimally* selected across four error estimates that all show some variability. The resulting bias may be considerable, as illustrated by Boulesteix and Strobl (2009) and Jelizarow et al. (2010).

To correct for the tuning bias outlined above in the context of microarray-based classification, Varma and Simon (2006)

recommend to embed an *internal* cross-validation (ICV) into the *external* resampling-based error estimation procedure. If the external resampling procedure is also CV as considered by Varma and Simon (2006), it is usual to denote the whole procedure as "nested CV." In this article, however, we stick to the more general terminology "ICV." No matter which resampling scheme is used externally for error estimation, the principle of ICV for parameter tuning is as follows. In each external resampling iteration, an internal CV is performed based on the current learning set for different tuning parameter values. The parameter value yielding the smallest error is selected and then evaluated by predicting test observations. In this way, for each external iteration the choice of the parameter value is performed without using information from the test set. Note that, as Varma and Simon (2006) have already stressed in their article, the ICV procedure estimates the error rate of a so-called wrapper algorithm including the tuning process and not the error rate of a specific tuning parameter value. A similar procedure might also be used to address the method selection bias induced by the optimal choice of the classification method. However, the ICV technique is computationally expensive, since it requires an additional CV loop on each learning set. The computational burden might rapidly become intractable. Furthermore, ICV tends to yield highly variable results, sometimes leading to obviously inappropriate "corrected" errors outside the range of the original errors of the considered methods.

In this article, we suggest an alternative bias correction approach, which can be applied to address both the tuning and the method selection bias. Our procedure, which can be interpreted as a smooth, analytical variant of ICV, guarantees intuitive bounds, increases stability compared to ICV, and reduces the computation time drastically since it does not rely on an internal CV loop.

Apart from ICV, the literature on the bias of the optimally selected error rate is scarce. Tibshirani and Tibshirani (2009) introduce an approach addressing several of the mentioned inconveniences, first and foremost the computational burden. In contrast to ICV, however, it does not target the unconditional error rate of wrapper algorithms but the conditional error rate of the optimal method/tuning parameter. Therefore, we do not include this approach in our following study because the differing estimation targets impede a fair comparison.

The rest of the article is structured as follows. Section 2 introduces the settings and notations and recalls the different types of error rates in this framework. This section also revisits ICV. Section 3 presents our new correction method. In Section 4, this method is illustrated and compared to ICV based on a simulation study. Finally, Section 5 summarizes and discusses some characteristics of our method.

2. Internal CV and the Unconditional Error Rate of Wrapper Algorithms

2.1. *Settings and Notations*

From a statistical point of view, binary supervised classification can be described in the following way. On the one hand we have a response variable taking values in $\mathcal{Y} = \{0, 1\}$. On the other hand we have predictors taking values in $\mathcal{X} \subset \boldsymbol{R}^p$ that will be used for constructing a classification rule. Predictors and response follow an unknown joint distribution on $\mathcal{X} \times \mathcal{Y}$ denoted by $P(\boldsymbol{x}, y)$. The observed i.i.d. sample of size n is denoted by $s_0 = \{(\boldsymbol{x}_1, y_1) \cdots (\boldsymbol{x}_n, y_n)\}$. The classification task consists in building a decision function $\hat{f}^S : \mathcal{X} \mapsto \mathcal{Y}$, $\boldsymbol{x} \mapsto \hat{f}^S(\boldsymbol{x})$, which maps elements $\boldsymbol{x}$ of the predictor space $\mathcal{X}$ into the response space $\mathcal{Y}$. The superscript S indicates that the decision function is built using the sample S.

From now on, we denote by method k (with $k \in 1, \ldots, K,$) the considered combination of method and tuning parameter values. As an example, method $k = 1$ may stand for SVM with cost= 1, method $k = 2$ for kNN with 10 nearest neighbors, and so on. As a special case, $1, \ldots, K$ might represent different parameter values of the same method. The decision function obtained by fitting the prediction method k to the sample s_0 is denoted as $\hat{f}_k^{s_0}$. Using this notation, each method k can be defined as a function $k : \mathcal{S} \mapsto \mathcal{F}_{\mathcal{X}}, \quad S \mapsto k(S) = \hat{f}_k^S$, which maps any possible sample S to the prediction function $\hat{f}_k^S$. Here, $\mathcal{S}$ denotes the space of possible samples S and $\mathcal{F}_{\mathcal{X}}$ denotes the space of decision functions on $\mathcal{X}$.

2.2. *Conditional and Unconditional Error Rate*

For a decision function $\hat{f}_k^{s_0}$, the true prediction error $\varepsilon[\hat{f}_k^{s_0}]$ depends on the expectation $\mathbf{E}_P$ over the joint distribution P and an adequately chosen loss function $L(\cdot, \cdot)$, for example, the indicator loss considered in our article. Abbreviating the true error $\varepsilon[\hat{f}_k^{s_0}]$ of method k constructed from sample s_0 by the simplified notation $\varepsilon(k \parallel s_0)$, one obtains

$$\varepsilon(k \parallel s_0) = \mathbf{E}_P \left[L \left(\hat{f}_k^{s_0}(\boldsymbol{x}), y \right) \right] = \int_{\mathcal{X} \times \mathcal{Y}} L \left(\hat{f}_k^{s_0}(\boldsymbol{x}), y \right) \mathrm{d}P(\boldsymbol{x}, y), \tag{1}$$

$\varepsilon(k \parallel s_0)$ is commonly referred to as *conditional* error since it refers to the decision function constructed from the specific sample s_0. The corresponding conditional error rate $\varepsilon(k \parallel S)$ should be seen as a random variable, where S stands for a random sample that follows the distribution P^n.

The expectation $\varepsilon^n(k) = \mathbf{E}_{P^n} [\varepsilon(k \parallel S)]$ of this random variable $\varepsilon(k \parallel S)$ is usually referred to as the *unconditional* true error rate of method k. It depends only on the method k, on the size n of the sample S and on the joint distribution P, and can be seen as a fixed quantity for every method k. Since the joint distribution $P(\boldsymbol{x}, y)$ is unknown, the conditional errors $\varepsilon(1 \parallel s_0), \ldots, \varepsilon(K \parallel s_0)$ and the unconditional errors $\varepsilon^n(1), \ldots, \varepsilon^n(K)$ have to be estimated. Standard estimation approaches are based on CV or repeated subsampling. We focus on the repeated subsampling method in this article because our new correction method involves the estimation of the unconditional variance of the estimated error. To our knowledge the estimator proposed by Nadeau and Bengio (2003)—which is used here—works for repeated subsampling only, and we are not aware of any convincing alternative estimator applicable to CV.

In repeated subsampling the whole data set is randomly split into learning and test sets several times. Each learning set L_b, $b = 1, \ldots, B$ of size n_L (with $n_L < n$) is used to estimate a decision function that is subsequently evaluated on the corresponding test set $S \setminus L_b$. For each iteration $b = 1, \ldots, B$

and each method k, $k = \{1, \ldots, K\}$, one obtains an estimated error $e(k \parallel L_b, S \setminus L_b)$, where the notation "$L_b, S \setminus L_b$" means that method k is fitted to the learning set L_b and evaluated on the test set $S \setminus L_b$. Note that we use the notation e for estimators and ε for true errors.

In contrast to the conditional true error $\varepsilon(k \parallel S)$, the estimated error $e(k \parallel L_b, S \setminus L_b)$ is conditional on the considered sample not only with regard to the estimation of the decision function but in addition with regard to the estimation of the error. For each method k, the iteration-wise test errors are eventually combined into an error rate estimate by averaging over the iterations $b = 1, \ldots, B$, yielding

$$e(k \parallel S) = \frac{1}{B} \sum_{b=1}^{B} e(k \parallel L_b, S \setminus L_b), \tag{2}$$

which obviously may depend on the random choices of the partitions $\{L_b, T_b\}$, $b = 1, \ldots, B$, a fact that is however omitted in the notation.

2.3. *The Unconditional Error Rate of Wrapper Algorithms*

Let us further denote the method yielding the smallest estimated error rate based on S as $k^*(S)$, that is, $k^*(S) = \arg\min_k e(k \parallel S)$. For a sample s_0, the error estimate $e(k^*(s_0) \parallel s_0)$ obtained by repeated resampling incorporates a source of a downward bias because $k^*(s_0)$ is chosen such that $e(k^*(s_0) \parallel s_0)$ is minimal. If one simply chooses the method yielding the minimal error rate $e(k^*(s_0) \parallel s_0)$, this minimal error rate underestimates the true conditional error rate $\varepsilon(k^*(s_0) \parallel s_0)$ of the chosen method. This problem is due to the fact that the same sample s_0 is used both for error estimation and for the choice of the optimal classification method $k^*(s_0)$. The corresponding prediction rule $\hat{f}^{s_0}_{k^*(s_0)}$ is expected to perform worse on an independent sample which was not used for choosing the method. This bias is related to the problem of multiple comparisons. The minimal error rate out of K methods decreases with increasing K.

In the context of ICV, the current gold standard for avoiding this bias, Varma and Simon (2006) reformulate the estimation task by defining wrapper algortihms.

A wrapper algorithm consists of two steps. The first step is a tuning process k^* on S which chooses a parameter or method. It can be described as a function:

$$k^* : \mathcal{S} \mapsto \mathcal{K}, \quad S \mapsto k^*(S) = \arg\min_k e(k \parallel S),$$

where $\mathcal{K}$ is the space of tuning parameters or candidate methods (here: $\mathcal{K} = \{1, \ldots, K\}$). Note that $k^*(S)$ is actually a random variable even for a fixed sample S because it depends on the randomly drawn learning sets L_b, $b = 1, \ldots, B$. For simplicity, however, we ignore this dependence on the specific learning sets in our notation.

The second step consists in learning a prediction rule by applying the chosen tuning parameter or candidate method $k^*(S)$ on S. For a sample S, this learning process can be described using the function $\psi : \mathcal{K} \times \mathcal{S} \mapsto \mathcal{F}_{\mathcal{X}}$, $k \mapsto \psi(k, S) = \hat{f}^S_k$. Using these functions, we can define the wrapper algorithm ϕ:

$$\phi : \mathcal{S} \mapsto \mathcal{F}_{\mathcal{X}}, \quad S \mapsto \phi(S) = \psi(k^*(S), S) = \hat{f}^S_{k^*(S)}.$$

Please note that this definition of ϕ is parallel to the definition of the methods k. However, ϕ incorporates an additonal source of randomness—the tuning process—whereas the methods k are deterministic for a fixed sample S.

Now, the main idea for bias correction in ICV and our new method consists in estimating the unconditional error rate of such wrapper algorithms ϕ:

$$Err = \mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)) = \varepsilon^{n_L}(\phi). \tag{3}$$

The error $\varepsilon(k^*(s_0) \parallel s_0)$ of the s_0-best method fitted on s_0 is a realization of the random variable $\varepsilon(k^*(S) \parallel S)$ whose mean over P^{n_L} is Err. We show in the next section and in Web Appendix G that the well-known ICV estimator actually targets at Err, which is also the reason why we use the simulated counterpart of Err for evaluation in our simulation studies later on.

Before turning our focus to ICV, we still have to treat another problem. Note that $e(k \parallel S)$ is a good estimator for $\varepsilon^{n_L}(k)$ but an upwardly biased estimator of $\varepsilon(k \parallel S)$ and $\varepsilon^n(k)$, because the decision functions are estimated based on n_L observations instead of n, with $n_L < n$. This bias affects any resampling based approach for estimating the generalization error. Since the mixture of this bias and the tuning bias would seriously distort the evaluation of the correction methods we stick to Err from Equation (3) as our estimation target throughout the article.

2.4. *Revisiting Internal Cross-Validation*

ICV is often performed within a nested CV procedure, that is, with test sets T_b forming a partition of the sample S. In the following, however, we formulate ICV in a general way without specifying how the learning and test sets (L_b, T_b) are chosen. In this article they are chosen by repeated subsampling.

In our notation, the ICV error estimate can be written as

$$\widehat{Err}_{ICV} = \frac{1}{B} \sum_{b=1}^{B} e(k^*(L_b) \parallel L_b, S \setminus L_b). \tag{4}$$

This formula, which is substantially different from formula (2) due to referring to $k^*(L_b)$ instead of a fixed method k, can be interpreted as follows. For each iteration b ($b = 1, \ldots, B$), the following procedure is repeated. Firstly, the "L_b-best method" $k^*(L_b)$ is determined by ICV within L_b. Secondly, the classification rule fitted on L_b using the best method $k^*(L_b)$ is evaluated on $S \setminus L_b$, yielding $e(k^*(L_b) \parallel L_b, S \setminus L_b)$. When introducing ICV Varma and Simon (2006) already explain that their method actually estimates the error rate of wrapper algorithms. In addition to this explanation, we present an asymptotic consideration in Web Appendix G, which further clarifies that the ICV-estimator is interpreted best as a natural estimator of Err (Equation 3).

The difference between Equations (4) and (2) is that ICV builds the average error of the best methods $k^*(L_b)$

(as assessed in ICV) instead of averaging the error rates of a specific method k. Note that these L_b-best methods again vary with the choice of the internal learning sets, which means that one may not obtain the same final results when repeating the same procedure twice—even if the outer learning sets L_b are fixed. Roughly speaking, in ICV the quantity *Err* (Eq. 3) is estimated through averaging over B subsets of s_0. Each term $e(k^*(L_b) \parallel L_b, S \setminus L_b)$ can be seen as an estimator of $\varepsilon(k^*(L_b) \parallel L_b)$, which roughly plays the role of a realization of $\varepsilon(k^*(S) \parallel S)$. The determination of $k^*(L_b)$ within each iteration is computationally expensive, which makes ICV very difficult to apply in practice when the prediction methods are time consuming, especially when they involve a tuning step that itself has to be performed through ICV. As a consequence, researchers often run ICV with a small number of folds (e.g., threefold-CV), yielding even more variable results. In extreme cases, this high variability may lead to estimates $\widehat{Err}_{ICV}$ larger than $\max_k e(k \parallel s_0)$ or lower than $\min_k e(k \parallel s_0)$, which is very unintuitive. Motivated by these disadvantages, we suggest an alternative computationally effective estimator in the following section.

3. A Smooth Analytical Alternative to ICV

3.1. *Basic Idea*

The rationale behind ICV is that the construction of the decision function *and* the tuning/method selection process, which are normally applied to the whole sample $S = s_0$, are mimicked on each external learning set L_b. In this way the tuning/method selection process is empirically incorporated into the estimation procedure. In practice, the best method $k^*(L_b)$ is typically not the same for all iterations $b = 1, \ldots, B$. ICV builds a so-to-say hard-weighted average of error estimates obtained with different methods or tuning parameters. The term hard-weighted is used here to emphasize that for each resampling iteration b only one of the $e(k \parallel L_b, S \setminus L_b)$ $(k = 1, \ldots, K)$ is chosen by ICV to be included in the average. The weight of $e(k^*(L_b) \parallel L_b, S \setminus L_b)$ is 1, while the weight of all other $e(k \parallel L_b, S \setminus L_b)$ (for $k \neq k^*(L_b)$) is 0. This way, results from different tuning parameters are eventually combined, which basically imitates the wrapper algorithm ϕ whose unconditional error rate is estimated.

Our new method is also based on a combination of error estimates of different parameter values/methods, albeit in a completely different and more direct way. While ICV combines errors of different parameter values/methods $e(k \parallel L(b), S \setminus L_b)$ computed for different test sets, the new procedure combines the global error estimates $e(k \parallel s_0)$ of the different parameter values/methods k. Furthermore, the way these average errors are combined does not depend on an empirical experiment as performed in ICV. Our main idea is to decompose the unconditional error rate $\mathbf{E}_{P^{n_L}}[\varepsilon(k^*(S) \parallel S)]$ with regard to the random variable $k^*(S)$, that is, the index of the best method, as follows:

$$\mathbf{E}_{P^{n_L}}[\varepsilon(k^*(S) \parallel S] = \sum_{k=1}^{K} \mathrm{P}(k^*(S) = k) \mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S) | k^*(S) = k]. \quad (5)$$

As argued below, in most cases, it is reasonable to assume that, for a fixed method k,

$$\varepsilon(k \parallel S) \perp k^*(S), \quad (6)$$

that is, the conditional error rate of method k constructed on S is independent from $k^*(S)$. It follows from Equation (6) that the conditional expectations in Equation (5), which cannot be estimated easily, can be replaced by the respective unconditional expectations,

$$\mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S) \approx \sum_{k=1}^{K} \mathrm{P}(k^*(S) = k) \mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S)] \quad (7)$$

and thus, to estimate $\mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)$, we have to estimate the quantities in Equation (7), as discussed in Sections 3.2 and 3.3.

Before that, let us come back to assumption (6). It means that the true error rate $\varepsilon(k \parallel s_0)$ of method k fitted on s_0 does not depend on which method performed best in repeated subsampling based on s_0—the unconditional error rates $\varepsilon^{n_L}(1), \ldots, \varepsilon^{n_L}(K)$ being fixed. Note that this assumption, of course, should not be misinterpreted in the sense that parameter tuning with CV is useless. Even if assumption (6) holds, tuning is useful to identify which method may have the smallest unconditional error rate $\varepsilon^{n_L}(k)$. A counterexample for which assumption (6) does not completely hold is SVM—denoted as $k = 1$ here—in the case of a sample with a mislabeled observation. The error rate $\varepsilon(1 \parallel s_0)$ of SVM is likely to be large, because SVM classifiers are strongly affected by mislabeled observations that often take the role of support vectors. Hence, $k^*(s_0) = 1$ is not likely. Thus, in this case, we obviously do not have $\varepsilon(k \parallel S) \perp k^*(S)$. However, especially in the presence of variable selection, assumption (6) holds in most cases, as illustrated by Hanczar, Hua, and Dougherty (2007) based on an extensive empirical study. Note that, if assumption (6) does not hold, it is more likely that $\mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S) | k^*(S) = k] < \mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S)]$, that is, that a classifier performs better if it is chosen by the model selection procedure. Consequently, this assumption could be a potential source for a pessimistic bias in our approach. We will revisit this aspect after analyzing our correction method on simulated data.

3.2. *A Weighted Mean Approach*

The terms $\mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S)]$ (for $k = 1, \ldots, K$) in Equation (7) can be naively estimated by $e(k \parallel s_0)$ (for $k = 1, \ldots, K$), suggesting to estimate the quantity of interest $\mathbf{E}_{P^{n_L}}(\varepsilon(k^*(S) \parallel S)$ by a weighted mean of the average errors $e(k \parallel s_0)$. We thus suggest the following estimator, denoted from now on as "weighted mean correction" (WMC):

$$\widehat{Err}_{WMC} = \sum_{k=1}^{K} \hat{\mathrm{P}}(k^*(S) = k) e(k \parallel s_0), \quad (8)$$

where $\hat{\mathrm{P}}(k^*(S) = k)$ stands for an adequate estimator of $\mathrm{P}(k^*(S) = k)$. Two such estimation procedures are presented in Section 3.3, where we also introduce a variant of $\widehat{Err}_{WMC}$

based on improved estimates of $\mathbf{E}_{P^{n_L}}[\varepsilon(k \parallel S)]$, which is called WMCS.

An illustrative way to explain the WMC estimator is to parallel it to the raw mean and to $\widehat{Err}_{ICV}$. The raw mean is obtained by giving equal weights to all parameters/methods, that is, by replacing $\hat{\mathrm{P}}(k^*(S)=k)$ by $1/K$ in Equation (8). This equal weight approach can be considered as a sensible upper bound for the corrected error because it corresponds to a random choice of the parameter/method. By definition, a random choice cannot lead to a tuning or method selection bias. That is why we do not expect any corrected error to be higher than

$$\widehat{Err}_{RawMean} = \sum_{k=1}^{K} \frac{1}{K} e(k \parallel s_0). \tag{9}$$

Regardless of the choice of the weights, there is a connection between our approach and ICV. The WMC estimator can be paralleled to the ICV estimator through a reformulation as

$$\widehat{Err}_{WMC} = \frac{1}{B} \sum_{k=1}^{K} \sum_{b=1}^{B} \hat{\mathrm{P}}\left(k^*(S)=k\right) e(k \parallel L_b, S \setminus L_b). \tag{10}$$

Similarly, the estimator $\widehat{Err}_{ICV}$ can also be reformulated as

$$\widehat{Err}_{ICV} = \frac{1}{B} \sum_{k=1}^{K} \sum_{b=1}^{B} I(k^*(L_b)=k) e(k \parallel L_b, S \setminus L_b). \tag{11}$$

Bringing these two estimators to a similar form highlights their crucial difference. $\widehat{Err}_{WMC}$ smoothly weights the errors $e(k \parallel L_b, S \setminus L_b)$ with the probabilities $\hat{\mathrm{P}}(k^*(S)=k)$ estimated from an analytical parametric model (whose parameters are estimated from the quantities $e(k \parallel L_b, S \setminus L_b)$ only). On the contrary, in $\widehat{Err}_{ICV}$ the weights are empirical, discrete and depend on the results of a computationally intensive internal CV. Figure 1 illustrates the similarities and differences of WMC and ICV graphically.

3.3. *Algorithmic Description of WMC and WMCS*

Our new WMC estimator is obtained through the following steps:

(1) *Inputs of the WMC algorithm:* estimated fold errors $\{e(k \parallel L_b, s_0 \setminus L_b)\}_{b=1,\ldots,B}^{k=1,\ldots,K}$.

(2) *Estimating the weights* $\mathrm{P}(k^*(S)=k)$ based on the assumption of a multivariate normal distribution $MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ of the vector $(e(1 \| S), \ldots, e(K \parallel S))^\top$:
   (a) *Estimating the parameters* $\boldsymbol{\mu}$ *and* $\boldsymbol{\Sigma}$ of the multivariate normal distribution
      - *The vector of means* is estimated by $\hat{\boldsymbol{\mu}} = (e(1 \| s_0), \ldots, e(K \parallel s_0))^\top$.
      - *The covariance matrix* is estimated by the nearest positive definite $\hat{\boldsymbol{\Sigma}}$ of $\hat{\boldsymbol{V}}^{\frac{1}{2}} \tilde{\boldsymbol{C}} \hat{\boldsymbol{V}}^{\frac{1}{2}}$ as computed by the algorithm in Higham (1988), where $\hat{\boldsymbol{V}}$ is the diagonal matrix with diagonal elements $\hat{v}_1, \ldots, \hat{v}_K$ computed using the procedure by Nadeau and Bengio (2003) as $\hat{v}_k = \widehat{var}(e(k \parallel S)) = \frac{1}{B-1} \sum_{b=1}^{B} \left(e(k \parallel L_b, s_0 \setminus L_b) - e(k \parallel s_0)\right)^2$ and the element $\hat{\rho}_{k_1,k_2}$ (for $k_1 \neq k_2$) of the correlation matrix $\tilde{\boldsymbol{C}} = (\hat{\rho}_{k_1,k_2})_{k_1=1,\ldots,K}^{k_2=1,\ldots,K}$ is obtained as the empirical correlation between the corresponding fold errors: $\hat{\rho}_{k_1,k_2} = \widehat{cor}(e(k_1 \parallel L_b, S \setminus L_b), e(k_2 \parallel L_b, S \setminus L_b))$.
   (b) *Estimating the weights* $\mathrm{P}(k^*(S)=k)$ (for $k = 1, \ldots, K$) based on the assumption of a multivariate normal distribution of $(e(1 \| S), \ldots, e(K \parallel S))^\top$ by plugging $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ into Equation (13) given in Section 3.4.

(3) *Computing the weighted average* of the original average resampling errors $e(k \parallel s_0)$, yielding the WMC estimator $\widehat{Err}_{WMC} = \sum_{k=1}^{K} \hat{\mathrm{P}}\left(k^*(S)=k\right) e(k \parallel s_0)$.

As supported by our results in Section 4, the WMC procedure yields an optimistic estimate. To address this problem, we suggest a second algorithm using a data-driven shrinkage procedure. The idea of the resulting WMCS procedure (standing for WMC-Shrinkage) is to replace $e(k \parallel s_0)$ by a shrunken estimate both in the weighted average and in the estimation of the weights $\mathrm{P}(k^*(S)=k)$ (further details can be found in Web Appendix A):

(1) *Inputs of the algorithm:* estimated fold errors $\{e(k \parallel L_b, s_0 \setminus L_b)\}_{b=1,\ldots,B}^{k=1,\ldots,K}$.

(2) *Estimating the shrinkage factor* ξ based on a preliminary multivariate normal distribution $MVN_{pre}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ for $(e(1 \| S), \ldots, e(K \parallel S))^\top$:
   (a) *Estimating the parameters* $\boldsymbol{\mu}$ *and* $\boldsymbol{\Sigma}$ *of* MVN_{pre} in the same way as in step (2)(a) of the original WMC algorithm.
   (b) *Computing the expected bias* $\hat{\zeta}$ as $\hat{\zeta} = e(k^*(s_0) \parallel s_0) - \mathbf{E}_{MVN_{pre}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})}(e(k^*(s_0) \parallel S) | k^*(S) = k^*(s_0))$, whereby the latter term is determined by Monte-Carlo simulation.
   (c) *Determining the shrinkage factor* ξ as

$$\hat{\xi} = \begin{cases} \hat{\zeta}[\widehat{Err}_{RawMean} - e(k^*(s_0)|s_0)]^{-1} & \text{if } \hat{\zeta} < \widehat{Err}_{RawMean} - e(k^*(s_0)|s_0) \\ 1 & \text{otherwise.} \end{cases}$$

(3) *Shrinking the average resampling errors* $e(k \parallel s_0)$ *with shrinkage factor* ξ:

$$e^{\xi}(k \| s_0) = (1-\xi) e(k \| s_0) + \hat{\xi} \widehat{Err}_{RawMean} \text{ for } k \text{ in } 1, \ldots, K.$$

(4) *Estimating the weights* by $\hat{\mathrm{P}}^{\xi}(k^*(S)=k)$ similarly to the WMC procedure (step (2)(b)), except that $\boldsymbol{\mu}$ is now estimated by $(e^{\xi}(1 \parallel S), \ldots, e^{\xi}(K \parallel S))^\top$ instead of $(e(1 \parallel S), \ldots, e(K \parallel S))^\top$.

(5) *Computing the weighted average* yields the WMCS estimator

$$\widehat{Err}_{WMCS} = \sum_{k=1}^{K} \hat{\mathrm{P}}^{\xi}(k^*(S)=k) e^{\xi}(k \parallel s_0). \tag{12}$$

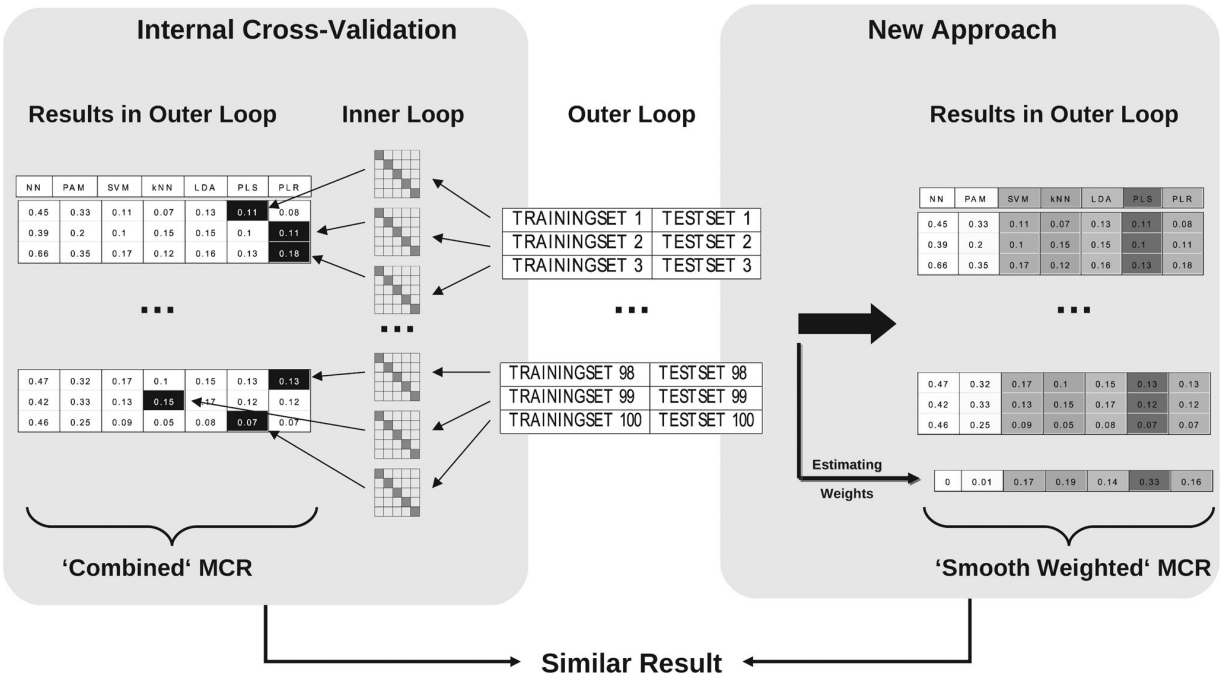


Figure 1. Schematic illustration contrasting the weighted mean correction (WMC) and internal cross-validation (ICV). The misclassification rate (MCR) obtained by WMC (compare to Eq. 10) can be interpreted as a smoothed variant of the one obtained by internal cross-validation (compare to Eq. 11).

3.4. *More Details on Estimating $P(k^*(S) = k)$*

Under the assumption that the vector $(e(1 \parallel S), \ldots, e(K \parallel S))^\top$ follows a multivariate normal distribution with mean vector $\hat{\boldsymbol{\mu}}$ and covariance matrix $\hat{\boldsymbol{\Sigma}}$, the probability $\hat{\mathrm{P}}(k^*(S) = k)$ is obtained as $\widehat{\mathrm{P}}(k^*(S) = k) = \mathrm{P}_{MVN(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})}(e(k \parallel S) \leqslant e(k' \parallel S), \forall k' : k' \neq k)$. This quantity can be derived analytically by considering the $K - 1$ differences $\delta_{k'} = e(k \parallel S) - e(k' \parallel S)$ for $k' \neq k$, which are simple linear combinations of the original random vector $(e(1 \parallel S), \ldots, e(K \parallel S))^\top$. The probability $\mathrm{P}(k^*(S) = k) = \mathrm{P}(e(k \parallel S) - e(k' \parallel S) \leqslant 0, \forall k' : k' \neq k)$ is then obtained from the density of the multivariate normal distribution of the random vector $\boldsymbol{\delta}$ as

$$\int_{-\infty}^{0} \cdots \int_{-\infty}^{0} \frac{1}{(2\pi)^{\frac{K-1}{2}} \sqrt{\boldsymbol{T}\boldsymbol{\Sigma}\boldsymbol{T}^\top}} \times \exp\left((\boldsymbol{\delta} - \boldsymbol{T}\hat{\boldsymbol{\mu}})^\top \left(\boldsymbol{T}\hat{\boldsymbol{\Sigma}}\boldsymbol{T}^\top \right)^{-1} (\boldsymbol{\delta} - \boldsymbol{T}\hat{\boldsymbol{\mu}}) \right) \Pi_{k'} \, \mathrm{d}\delta_{k'}, \quad (13)$$

where the $(K - 1) \times K$ matrix $\boldsymbol{T}$ contains the linear combinations yielding the corresponding differences, that is, such that $\boldsymbol{\delta} = \boldsymbol{T}\boldsymbol{e}$. These integrals can be approximated very precisely by common statistical software like the function *pmvnorm* from the R package **mvtnorm** (Genz et al., 2011). Computation times of this function are marginally small in comparison with computation times of other steps of the analysis. Of course, the normality assumption can not hold exactly since the considered errors are averages of binary variables. In order to assess the deviation from the normal distribution we provide a selection of representative normal quantile plots for the distribution of the average errors in simulation settings in Web Appendix C. In many cases the assumption seems to hold whereas in some cases the distributions tend to more extreme values than expected under normality assumptions.

3.5. *Implementation*

The weighted mean correction method is implemented in the R function *weighted.mcr* included in a new version of the R/Bioconductor package **CMA** (Slawski et al., 2009) that can be downloaded from the companion website (URL:`http://www.ibe.med.uni-muenchen.de/organisation/mitarbeiter/070_drittmittel/bernau/cvbias/index.html`). The codes implementing our analyses are also provided there. For larger data sets and computationally intensive methods like SVMs, ICV runtimes can be drastically higher in comparison to the two WMC variants. Detailed information on the runtimes of the different methods are provided in Web Appendix E.

4. Empirical Results and Comparison of the Three Estimators

4.1. *Study Design*

We evaluate our new estimator $\widehat{Err}_{WMC}$ (Eq. 10) and its shrunken version $\widehat{Err}_{WMCS}$ (Eq. 12) on both real and simulated data sets and compare them to the widely established estimator $\widehat{Err}_{ICV}$ (Eq. 4) and to the naive raw mean estimator $\widehat{Err}_{RawMean}$ (Eq. 9). The real data study is based on four

microarray data sets that are described in more details in Web Appendix D. We also consider modified versions of these four data sets obtained by replacing the response Y by a randomly generated Bernoulli distributed variable $Y' \sim \mathcal{B}(1, 0.5)$. These modified data sets are denoted as "non-informative" setup, in contrast to the original version of the data sets including "informative" predictors. In the simulation study we use five different data generating processes (DGPs), including DGPs with correlated and uncorrelated predictors as well as different signal strengths and sample sizes. $T = 200$ data sets are randomly drawn from each setting. For each data set the true error rate of the wrapper algorithm is approximated on a large validation set. A more detailed description of the simulation study can be found in Web Appendix B.

In the whole study, error rate estimation is performed through repeated subsampling into learning and test sets with $B = 100$ subsampling iterations. The proportion of observations included in the learning sets is set to 80% and 63.2% successively. In contrast to the two WMC variants, $\widehat{Err}_{ICV}$ involves another parameter, the number of folds in internal CV. There are no commonly accepted guidelines to choose the number of folds in internal CV, which can be seen as a further inconvenience of ICV. In this study, this number is chosen such that each internal test set contains approximately five observations. In each setup, the whole procedure is repeated $T = 50$ times in order to analyze the variability of the results. By "repeated," we mean that $T = 50$ different *sets* of partitions $(L_b, T_b)_{b=1,\ldots,B}$ are considered successively for the original data sets, and that $T = 50$ different randomly generated responses Y' are considered successively for the modified data sets. In the whole analysis, we thus consider a total of $50 \times 100 = 5000$ splittings into learning data L_b and test data T_b.

As outlined in the introduction and in Section 2, our methodology can both be applied to the correction of the tuning bias or to the correction of the method selection bias. To illustrate these two features, we successively consider two setups. In the first setup (illustrating the correction of the tuning bias and denoted as "tuning setup" (labels in tables and figures according to the methods used: "pls" and "knn")), methods $1, \ldots, K$ stand for different parameter values of a unique classification method. Two classifiers are considered successively. The first classifier is kNN, where methods $1, \ldots, K$ correspond to different values $(1, \ldots, 15)$ of the parameter "number of neighbors". The second classifier is Partial Least Squares dimension reduction followed by Linear Discriminant Analysis (PLS-LDA) where methods $1, \ldots, K$ correspond to different numbers $(1, \ldots, 10)$ of PLS components. In both cases, a preliminary variable selection is performed by selecting the variables yielding the lowest p values with the two-sample t-test (50 variables for kNN, 250 variables for PLS-LDA). Note that, in all resampling iterations, variable selection is performed using the learning set only. For ICV, this holds for the outer as well as for the inner loop.

In the second setup (illustrating the correction of method selection bias and denoted as "selection setup" (label in tables and figures: "sel")), methods $1, \ldots, K$ correspond to different combinations of classification methods and parameter values. The parameters are fixed, because tuning them with internal CV would imply three embedded CVs for $\widehat{Err}_{ICV}$, which is computationally intractable. The following classification methods are considered: nearest shrunken centroids with $\Delta = 0.5$, linear SVM with $cost = 50$, kNN with 1 neighbor based on the 20 top-variables, kNN with 18 neighbors based on the 50 top-variables, diagonal linear discriminant analysis (DLDA) based on the 20 top-variables, PLS-LDA with 3 PLS components based on the 100 top-variables and L_2-penalized logistic regression with penalty $\lambda = 0.01$. The simulation study is performed for the selection setup only, since it is generally more challenging due to the larger bias.

Note that these three different classifier pools in combination with the resampling approach for tuning/selection define wrapper algorithms ϕ, whose unconditional error rates are the actual estimation target of ICV and WMC.

Table 1
Average corrected errors (over 50 replications) for informative pls and selection (sel) setups with 80% of the observations in the training data sets

Setup	ICV	WMCS	WMC	Raw	Min	Max
pls-alon	0.176	0.183	0.168	0.186	0.149	0.210
pls-singh	0.087	0.092	0.080	0.091	0.075	0.180
pls-golub	0.048	0.034	0.030	0.035	0.024	0.041
pls-chiaretti	0.431	0.434	0.417	0.441	0.398	0.459
sel-alon	0.164	0.179	0.163	0.190	0.142	0.257
sel-singh	0.097	0.104	0.092	0.133	0.083	0.319
sel-golub	0.026	0.021	0.018	0.061	0.004	0.226
sel-chiaretti	0.398	0.403	0.383	0.420	0.365	0.452

4.2. *Study Results*

An overview of the results on real data sets is given in Table 1 (informative setup) and Table 2 (non-informative setup) displaying the averages over the $T = 50$ replications of the corrected estimates $\widehat{Err}_{WMC}$, $\widehat{Err}_{WMCs}$, $\widehat{Err}_{ICV}$ and of the raw mean $\widehat{Err}_{RawMean}$ as well as the minimal and the maximal error rates $\min_k e(k \parallel s_0)$ and $\max e(k \parallel s_0)$. Figure 2 displays the boxplots of these error estimates for a representative real data set in the informative setting (top) and non-informative setting (bottom), while Figure 3 shows the boxplots obtained in a representative simulation setting for different signal strengths (top: strong, middle: weak, bottom:

Table 2
Average corrected errors (over 50 replications) for non-informative pls and selection (sel) setups with 80% of the observations in the training data sets

Setup	ICV	WMCS	WMC	Raw	Min	Max
pls-alon	0.502	0.497	0.483	0.504	0.465	0.538
pls-singh	0.494	0.490	0.482	0.492	0.468	0.524
pls-golub	0.500	0.489	0.479	0.495	0.463	0.533
pls-chiaretti	0.495	0.492	0.482	0.498	0.469	0.523
sel-alon	0.492	0.482	0.462	0.488	0.440	0.531
sel-singh	0.504	0.496	0.479	0.503	0.463	0.541
sel-golub	0.498	0.487	0.466	0.493	0.443	0.536
sel-chiaretti	0.505	0.498	0.484	0.501	0.468	0.532

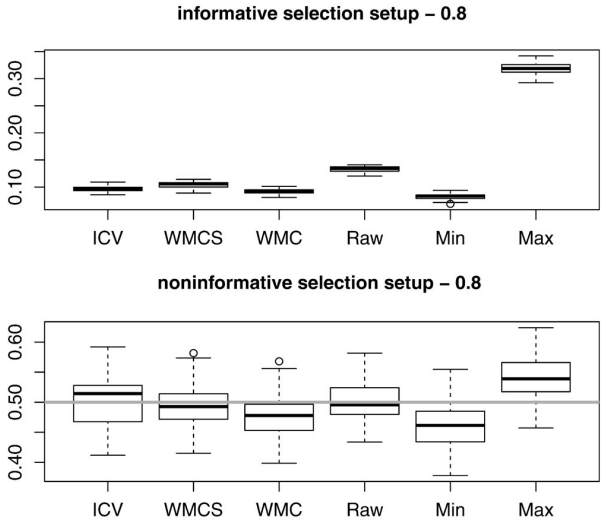


Figure 2. Comparison of internal cross-validation (ICV), weighted mean correction (WMC), and weighted mean correction with shrinkage (WMCS) for the selection setup on the prostate cancer data set by Singh with 80% of the observations in the training data sets (top: informative setting, bottom: non-informative setting with an horizontal line at 50%).

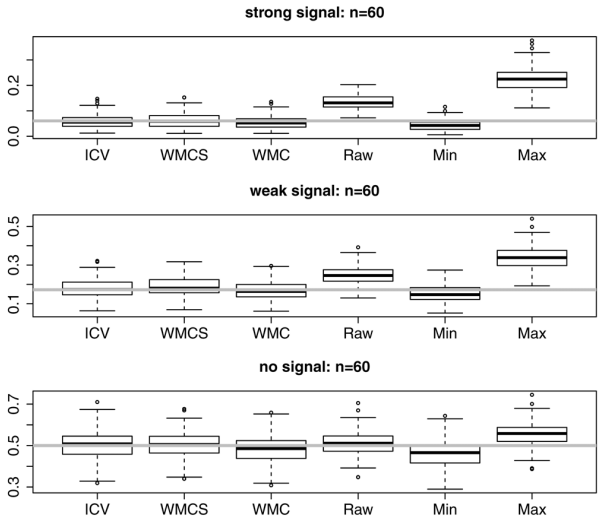


Figure 3. Comparison of internal cross-validation (ICV), weighted mean correction (WMC), and weighted mean correction with shrinkage (WMCS) for the selection setup on simulated data with correlated predictors and different signal strengths (top: strong signal, middle: weak signal, bottom: no signal) with 80% of the observations in the training data sets and $n = 60$. The true error rate of the wrapper algorithm *Err* is represented as an horizontal line in each setting.

no signal). We refer to Web Appendices B (simulated data) and D (real data) for a more exact and exhaustive analysis of the results, whereas we summarize the most important aspects in the following paragraphs.

In most simulation and real data settings the ICV and WMCS estimates are similar and range between the WMC errors and the raw mean errors. In the tuning setups, WMCS often yields results close to the raw mean indicating that its shrinkage mechanism considers differences among classifiers only marginally relevant. As pointed out before, the raw mean error is a sensible upper bound for the corrected error. Our new WMCS method yields the raw mean if ξ equals 1. As mentioned in Section 3, the raw mean corresponds to a random choice of the parameter/method, which obviously cannot lead to a tuning or method selection bias. We do not expect a good correction method to produce estimates higher than the raw mean approach. Corrected errors estimated by ICV, however, fall beyond this upper bound in some of the investigated setups, which makes poor sense in most situations and may be considered as an important disadvantage. Such a failure may also exceptionally occur with the WMC and WMCS methods, but to a much lesser extent.

Most WMC estimates are slightly more optimistic than ICV in the informative setups with the real data sets. In simulations, this tendency to under-correction is even more pronounced.

In contrast, WMCS slightly over-corrects, that is, tends to over-estimate the error, but this tendency decreases with increasing sample size in simulations. The analysis of the shrinkage parameter ξ suggests that this slight over-correction especially occurs in intermediate cases where ξ takes values around 0.5 or 0.6. In general, however, the shrinkage parameter behaves as expected: larger values are selected more often for non-informative setups ($\xi \to 1$) than for informative setups ($\xi \to 0$). See Web Appendices B and D for more details on the shrinkage parameter.

On the Golub data set, which is characterized by very small errors, we rarely observe ICV-corrected error rates higher than the average maximal error rate, $\max_k e(k \parallel S)$. Corrected error rates exceeding the error rate of the worst classifier can be considered as obvious failures of the correction method. In contrast to ICV, both variants of the new estimator are upper-bounded by $\max_k e(k \parallel S)$.

To conclude, WMCS yields more convincing results than WMC in most settings, in the sense that it leads to estimates i) near 0.5 in non-informative settings, ii) near the true error rate of the wrapper algorithms *Err* in simulated informative settings, and iii) near the ICV estimates in real data informative settings. The strongest deviations from ICV occur on the 0.63-setups. In the tuning setups, WMCS mostly yields results close to the raw mean. Further analyses are needed to show whether WMCS is substantially superior to the raw mean in this case. It is worth mentioning, however, that the shrinkage factor ξ always provides important evidence for the applicability of the raw mean, and WMCS is thus more informative.

5. Discussion and Concluding Remarks

In the context of error estimation through repeated subsampling, we suggest two variants of a new weighting-based method for correcting the tuning bias and the method

selection bias by estimating the unconditional error rate of the corresponding wrapper algorithms as with ICV. Both of our methods avoid the additional computational costs of ICV while producing comparable results. The shrinkage-based variant, WMCS, addresses the optimistic tendency of WMC. WMCS yields the most accurate estimates in simulations and the best approximation to ICV on real data despite a slight tendency to over-correction. Our correction method cannot only be applied in the well-known context of parameter tuning but also to address the method selection bias. To our knowledge, correction of the latter bias has never been addressed explicitly in the literature, neither with ICV nor with any other approach. We also suggest to extend ICV to the method selection setup, based on the idea that a "method" can in a broad sense be considered as a categorically scaled tuning parameter. There are however some differences between the tuning setup and the method selection setup. Quite generally, the bias is larger in the method selection setup than in the tuning setup. That is probably because in the tuning setup the error is expected to depend smoothly on the parameter value. Large differences between errors obtained with similar parameter values are unlikely. In contrast, in the selection setup methods do not have a natural ordering and may yield more contrasted errors.

Besides the lower computational effort, an important advantage of our method over ICV is that the obtained corrected error remains within reasonable bounds defined by the minimal and maximal errors. As shown in Section 4, ICV may produce estimates outside this interval. Regardless of whether the ICV-estimates fall above the highest error or below the lowest error, such a "correction" makes poor sense. Our correction method is clearly superior in such cases. Another extreme situation where our method yields more plausible results is when all tuning parameter values/methods lead to very similar resampling error rates ($e(1 \parallel s) \approx \cdots \approx e(K \parallel s)$). In this case our correction methods do not perform any correction, which is intuitively reasonable. On the contrary, ICV may produce a different corrected error. Whereas the results of the new weighted mean correction are deterministic (once the outer learning sets are fixed), ICV depends on the specific choice of the internal learning sets when selecting the L_b-best method $k^*(L_b)$. This aspect of ICV is consistent with its main idea of mimicking the selection or tuning process on each learning set of the resampling approach. However, by this dependence, ICV suffers from another source of variability which is difficult to correct within a reasonable time.

In contrast to ICV, WMC, and WMCS directly use the information on the correlation between the errors of different parameter values/methods, which allows an assessment of the "effective cardinality" of the pool of parameter values/methods. Obviously, the potential for tuning or method selection bias increases with the number K of parameter values/methods. However, if they are all very similar the bias is not expected to increase dramatically. Our method automatically takes into account correlation between errors including such highly correlated "blocks" of similar parameter values/methods. Another practical advantage over ICV is that our approach can be applied "a posteriori" as long as one has used the same training sets for all classifiers and saved all fold errors ($e(k||L_b, S \setminus L_b), \forall b, k$). With ICV the whole procedure has to be performed again if the classifier pool or the tuning grid is changed or enlarged.

Next, we have to discuss the small optimistic bias of WMC and the even smaller pessimistic bias of WMCS. The WMC procedure is based on a number of assumptions and possibly biased estimation steps. On the one hand, if assumption (6) is violated we expect our method to be conservative, that is, to over-correct the error, because a method chosen by internal CV based on a specific data set is expected to perform better rather than worse when applied to this data set, yielding $\mathbf{E}_{P_{n_L}}[\varepsilon(k \parallel S)|k^*(S) = k] \leqslant \mathbf{E}_{P_{n_L}}[\varepsilon(k \parallel S)]$. On the other hand we have the optimistic bias which is induced by the biased mean estimate $\hat{\boldsymbol{\mu}}$. This bias is corrected by the shrinkage approach in WMCS. However, WMCS sometimes over-corrects the optimistic tendency of WMC especially in setups where ξ is close to 0.5 or 0.6. In these particular cases, WMC often performs better than WMCS. In the present context of bias correction, however, the optimistic bias of WMC epitomizes a clear disadvantage and we do not advice to apply it in practice. The pessimistic bias of WMCS is definitely more acceptable, the more so as it is substantially smaller on average. In the tuning setup, the WMCS method often produces results very similar to the raw mean. In future work our methods should be further assessed and refined in this context. Moreover, the method could also be extended to the case where the tuning parameter is the number of variables used by a specific method. This setup can be considered an intermediate case between selection setup and tuning setup.

Finally, let us recapitulate the shift of focus from the conditional error rate of a classifier to the unconditional error rate of wrapper algorithms, which have been introduced in Varma and Simon (2006) and used as a basis by our approach. On the one hand, one can certainly argue that the ultimate quantity of interest for a physician is the conditional error rate of the eventually constructed prediction rule. In this context, ICV and the WMC variants can only provide a "surrogate estimate" based on the assumption that *Err* is usually close to this conditional error rate. Especially in the case of method selection setups, this assumption may be violated because the models constructed in the respective ICV iterations can be substantially different from the model constructed on the whole data set. However, even in this case, *Err* can be an informative quantity if one is more interested in the general utility of the data at hand. In pilot studies meant as proof of concept, *Err* can provide a more realistic picture of the signal in the data because it considers the high variability of the model construction procedure of wrapper algorithms. Considering the conditional error rate of a single model only may lead to highly variable conclusions.

On the other hand, from a statistician's point of view, the unconditional error rate of the wrapper algorithm, *Err*, can be an interesting quantity for its own sake. If several wrapper algorithms have to be compared, it is recommendable to compare them on the basis of *Err* and not of the conditional error rate. For this purpose, *Err* has clear advantages over the conditional error rate because it indeed reports the performance of the underlying wrapper algorithm whereas the conditional error rate is the performance of a single prediction rule only. At least on simulated data (where the corresponding quantities can be obtained via Monte Carlo

simulations), we would also consider as a next step to include the variance of the conditional error rate of wrapper algorithms ($\mathbf{Var}_{P^{n_L}}\left[\varepsilon(\phi(S))\right] = \mathbf{Var}_{P^{n_L}}\left[\varepsilon(k^*(S) \parallel S)\right] = \mathbf{E}_{P^{n_L}}\left[\varepsilon(k^*(S) \parallel S) - Err\right]^2$) into the comparison in order to provide additional insight into the characteristics of the competing wrapper algorithms.

6. Supplementary Materials

Web Figures and Tables referenced in Sections 3 and 4 are available with this paper at the Biometrics website on Wiley Online Library.

Acknowledgements

CB and ALB are supported by the LMU-innovativ Project BioMed-S. CB is supported by grant BO3139/2-1 and BO3139/2-2 from the German Research Foundation (DFG) to ALB. We thank Vincent Guillemot and Robert Hable for helpful discussions and the anonymous referees and the associated editor for their constructive remarks which lead to a substantial improvement of our manuscript.

References

Boulesteix, A.-L. and Strobl, C. (2009). Optimal classifier selection and negative bias in error rate estimation: An empirical study on high-dimensional prediction. *BMC Medical Research Methodology* **9**, 85.

Dupuy, A. and Simon, R. M. (2007). Critical review of published microarray studies for cancer outcome and guidelines on statistical analysis and reporting. *Journal of the National Cancer Institute* **99**, 147–157.

Genz, A., Bretz, F., Miwa, T., Mi, X., Leisch, F., Scheipl, F., and Hothorn, T. (2011). *mvtnorm: Multivariate Normal and t Distributions*. R package version 0.9-96.

Hanczar, B., Hua, J., and Dougherty, E. R. (2007). Decorrelation of the true and estimated classifier errors in high-dimensional settings. *EURASIP Journal of Bioinformatics and Systems Biology* **2007**, 38473.

Higham, N. J. (1988). Computing a nearest symmetric positive semidefinite matrix. *Linear Algebra and its Applications* **103**, 103–118.

Jelizarow, M., Guillemot, V., Tenenhaus, A., Strimmer, K., and Boulesteix, A.-L. (2010). Over-optimism in bioinformatics: an illustration. *Bioinformatics* **26**, 1990–1998.

Nadeau, C. and Bengio, Y. (2003). Inference for the generalization error. *Machine Learning* **52**, 239–281.

Slawski, M., Boulesteix, A.-L., and Bernau, C. (2009). *CMA: Synthesis of microarray-based classification*. R package version 1.5.5.

Tibshirani, R. J. and Tibshirani, R. (2009). A bias correction for the minimum error rate in cross-validation. *Annals of Applied Statistics* **3**, 822–829.

Varma, S. and Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics* **7**, 91.

Received April 2011. Revised February 2013. Accepted February 2013.

D.2 Cross-study validation for assessment of prediction models and algorithms (conditionally accepted at Bioinformatics)

This paper evolved in the context of the work presented in Chapter 2. Levi Waldron (Harvard Medical School and Dana-Faber Cancer Institute) has primarily written the Introduction (*Section 1*) and the description of the datasets (*Section 2*) and the corresponding sections in Chapter 2. Lorenzo Trippa (Harvard Medical School and Dana-Faber Cancer Institute) contributed the main parts of Section 3.1. Moreover both coauthors contributed many helpful suggestions during numerous meetings as far as the simulation design and the layout of the manuscript are concerned. The remaining coauthors helped to improve the manuscript mainly by providing valuable feedback and comments on the manuscript.

BIOINFORMATICS

Vol. 00 no. 00 2014
Pages 1–9

Cross-study validation for assessment of prediction models and algorithms

Christoph Bernau[1,2], Markus Riester[3,4], Anne-Laure Boulesteix[2], Giovanni Parmigiani[3], Curtis Huttenhower[4], Levi Waldron[5,‡,*], Lorenzo Trippa[3,‡]

[1]Leibniz Supercomputing Center, Garching, Germany
[2]Department for Medical Informatics, Biometry and Epidemiology, Munich, Germany
[3]Dana-Farber Cancer Institute, Boston, U.S.A.
[4]Harvard School of Public Health, Boston, U.S.A.
[5]City University of New York School of Public Health, Hunter College, New York, U.S.A.
[‡] joint senior authors

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: Numerous competing algorithms for prediction modeling in high-dimensional settings have been developed in the statistical and machine learning literature. Learning algorithms and the prediction models they generate are typically evaluated on the basis of cross-validation error estimates in a few examplary datasets. However, in most applications, the ultimate goal of prediction modeling is to provide accurate predictions for independent samples processed in different laboratories, and cross-validation within examplary datasets may not adequately reflect performance in this context.

Methods: Systematic cross-study validation is performed in simulations and in a collection of eight estrogen-receptor positive breast cancer microarray gene expression datasets, with the objective of predicting distant metastasis-free survival (DMFS). An evaluation statistic, in this paper the C-index, is computed for all pairwise combinations of training and validation datasets. We evaluate several alternatives for summarizing the pairwise validation statistics, and compare these to conventional cross-validation.

Results: We develop a systematic approach to "cross-study validation" to replace or supplement conventional cross-validation for evaluation of high-dimensional prediction models when independent datasets are available. In data-driven simulations and in our application to survival prediction with eight breast cancer microarray datasets, standard cross-validation suggests inflated discrimination accuracy for all competing algorithms when compared to cross-study validation. Furthermore, the ranking of learning algorithms differs, suggesting that algorithms performing best in cross-validation may be suboptimal when evaluated through independent validation.

Availability: The *survHD: Survival in High Dimensions* package (http://www.bitbucket.org/lwaldron/survhd) will be made available through Bioconductor.

Contact: levi.waldron@hunter.cuny.edu

*to whom correspondence should be addressed

1 INTRODUCTION

Cross-validation and related resampling methods are *de facto* standard for ranking supervised learning algorithms. They allow estimation of prediction accuracy using subsets of data that have not been used to train the algorithms. This avoids over-optimistic accuracy estimates caused by "re-substitution," a property that has been carefully considered (Molinaro *et al.*, 2005; Baek *et al.*, 2009; Simon *et al.*, 2011). It is common to evaluate algorithms and prediction models by estimating accuracy via cross-validation based on several datasets, with results summarized across datasets to rank algorithms (Demšar, 2006; Boulesteix, 2013). This approach recognizes possible variations in the performance of learning algorithms across studies. However, it is not fully consistent with the ultimate goal of providing accurate predictions for fully independent samples originating from different institutions and processed by different laboratories.

It has been observed that accuracy estimates of genomic prediction models based on independent validation data are often substantially inferior to cross-validation estimates (Castaldi *et al.*, 2011). In some cases this has been attributed to incorrect application of cross-validation; however even strictly performed cross-validation may not avoid over-optimism resulting from potentially unknown sources of heterogeneity across datasets. These include differences in design, acquisition, and ascertainment strategies (Simon *et al.*, 2009), hidden biases, measured variables, technologies used for measurements, and populations studied. In addition, many genomics studies are affected by experimental batch effects (Baggerly *et al.*, 2008; Leek *et al.*, 2010). Quantifying these heterogeneities and predicting their impact on the performance of prediction algorithms is critical in the practical implementation of personalized medicine procedures that use genomic information.

There are potentially conflicting, but valid, perspectives on what constitutes a good learning algorithm. The first perspective is that a good learning algorithm should perform well when trained and applied to a single population and experimental setting, but it is not expected to perform well when the resulting model is applied to different populations and settings. We call such an algorithm

"*specialist*", in the sense that it can adapt and specialize to the population at hand. This is the mainstream perspective for assessing prediction algorithms and is consistent with validation procedures performed within studies (Molinaro *et al.*, 2005; Baek *et al.*, 2009; Simon *et al.*, 2011). However, we feel it does not reflect the reality that "samples of convenience" and uncontrolled specimen collection are the norm in genomic biomarker studies (Simon *et al.*, 2009).

We promote another perspective: a good learning algorithm should be "*generalist*", in the sense that it yields models that may not be optimal for the training population, that are likely not fully representative of the dataset at hand, but that perform reasonably well across populations or laboratories employing comparable but not identical methods. *Generalist* algorithms may be preferable in important settings, for instance when a researcher develops a model using samples from a highly controlled environment, but hopes the model to be applicable to other hospitals, labs, or more heterogeneous populations. Although concern has been expressed about the lack of independent validation of genomic prediction *models* (Subramanian and Simon, 2010; Micheel *et al.*, 2012), computational scientists have not systematically adopted independent validation in the comparison of learning *algorithms*. We thus propose what we term "leave-one-dataset-in" cross-study validation to formalize the use of independent validation in the evaluation of learning algorithms. Through data-driven simulations and an example involving eight publicly available estrogen receptor-positive breast cancer microarray datasets, we assess several established survival prediction algorithms using the proposed approach and compare it to conventional cross-validation.

2 METHODS

2.1 Notations and settings

We consider multiple datasets $i = 1, \ldots, I$ with sample sizes $N_1, \ldots, N_I$. Each observation s appears only in one dataset i (datasets do not overlap), and the corresponding record includes a primary outcome Y_i^s and a vector of predictor variables $\mathbf{X}_i^s$; throughout this paper $\mathbf{X}_i^s$ will be gene expression measurements. Our goal is to compare the performance of different learning algorithms $k = 1, \ldots, K$ that generate prediction models for Y_i^s using $\mathbf{X}_i^s$. Throughout this paper, the primary outcome Y_i^s is a possibly censored survival time. We are interested in evaluating and ranking competing prediction methods $k = 1, \ldots, K$. Since the ranking may depend on the application field, the first step is to define the prediction task of interest. We focus on the prediction of survival time in breast cancer patients based on high-throughput gene expression measurements. Our approach and the concept of cross-study validation, however, can be applied to any other type of response variable.

2.2 Algorithms considered

We assess six learning algorithms ($k = 1, \ldots, 6$) appropriate for high-dimensional continuous predictors and possibly censored survival time outcome: LASSO regression (Goeman, 2010), CoxBoost (Binder and Schumacher, 2008), SuperPC (Blair and Tibshirani, 2004), Unicox (Tibshirani, 2009), and PlusMinus (Zhao *et al.*, 2013). Our focus is not to provide a comprehensive array of algorithms, but simply to use a few popular, representative algorithms to study the properties of cross-study validation.

2.3 Ranking algorithms by cross-study validation: the CSV matrix

We refer in this paper to k-fold cross-validation and related resampling methods collectively as cross-validation (CV).

Our ranking procedure for learning algorithms is based on a squared matrix $\mathbf{Z}^k$ of scores ($k = 1, \ldots, K$), with the element in the i-th row and j-th column measuring how well the model produced by algorithm k trained on dataset i performs when validated on dataset j. Since we consider K methods we end up with K method-specific squared matrices $\mathbf{Z}^1, \ldots, \mathbf{Z}^K$. The diagonal entries of the matrices are set equal to the performance estimates obtained with 4-fold CV in each dataset. We also refer to $\mathbf{Z}^k$ as the cross-study validation matrix, or *CSV matrix*.

Possible definitions for the non-diagonal $\mathbf{Z}_{i,j}^k$ scores include the concordance index in survival analysis (Harrell *et al.*, 1996; Gnen and Heller, 2005), which we use in this paper, the area under the operating characteristic curve in binary classification problems, or the mean squared distance between predicted and observed values in regression problems. As an illustration, Figure 1a displays the *CSV* matrix of C-statistics obtained through validation of ridge regression models for the eight studies of Table 1.

2.4 Summarization of the CSV matrix

In order to rank learning algorithms $k = 1, \ldots, K$, each matrix $\mathbf{Z}^k$ must be summarized by a single score. We consider two candidate approaches:

1) **Simple Average** $\frac{\sum_i \sum_{i \neq j} \mathbf{Z}_{i,j}^k}{I(I-1)}$ of all non-diagonal elements of the $\mathbf{Z}^k$-matrix.

2) **Median** or more generally $q-$**quantile** of the non-diagonal entries of $\mathbf{Z}^k$. Quantiles offer robustness to outlier values, and the possibility to reduce the influence of uninformative studies where all algorithms perform poorly by selection of an appropriate quantile.

2.5 True global ranking

From a statistical perspective the score $\mathbf{Z}_{i,j}^k$ is a random variable. First, studies i and j can be seen as randomly drawn from a population of studies. Second, observations within each study can be considered as randomly drawn from the unknown and possibly different distributions F_i and F_j underlying studies i and j.

With this view of $\mathbf{Z}_{i,j}^k$ as random variable, we consider the theoretical counterparts of the empirical aggregating scores (simple average and quantiles) described in Section 2.4 to summarize $\mathbf{Z}^k$. The theoretical counterparts are the expected value or quantiles of each $\mathbf{Z}_{i,j}^k$ score, $i \neq j$, obtained by integrating the two levels of randomness that we described. The *true global* ranking of the learning algorithms $k = 1, \ldots, K$ is then defined by these expected values (or quantiles), one for each algorithm. We will call the ranking *global* because it is not specific to the available studies.

The true global ranking can be considered as the estimation target of evaluation procedures such as CV or CSV. In section 2.7 we present the design of a data-driven simulation study in which the true ranking is obtained through Monte Carlo integration. This allows us to evaluate and compare the ability of CV and CSV to recover the true global ranking.

2.6 Description of datasets

We used a compendium of breast cancer microarray studies curated for the meta-analysis of Haibe-Kains *et al.* (2012) and available as a supplement to that article. We selected all datasets for which metastasis-free survival (DMFS), the most commonly available survival endpoint, as well as Estrogen Receptor (ER) status, were available, and which were generated with Affymetrix HGU GeneChips HG-U133A, HG-U133B and HG-U133PLUS2. We considered exclusively ER-positive tumors. Of the remaining 8 datasets (Table 1), only one originated from a population-based cohort (Schmidt *et al.*, 2008). Four studies considered only patients who did not receive hormone therapy or chemotherapy adjuvant treatment. Only four provided date ranges of patient recruitment Foekens *et al.* (2006); Desmedt *et al.* (2007); Schmidt *et al.* (2008); Chin *et al.* (2006). This variability in design strategies and reporting, and cohort differences in survival that are not easily explicable (Table 1, column 3Q survival) highlight the prevalence of "samples of convenience" in biomarker studies discussed by Simon *et al.* (2009).

Samples from dataset SP1 duplicated in dataset VDX were removed. Expression of each gene was summarized using the probeset with maximum mean (Miller *et al.*, 2011). The 50% of genes with lowest variance were removed. Subsequently, gene expression values were scaled by linear scaling of the 2.5% and 97.5% quantiles as described by Haibe-Kains *et al.* (2012).

2.7 Simulation design

We simulate heterogeneous datasets from a joint probability model with survival outcomes. The probability model is defined by a resampling procedure that we apply to the eight breast cancer datasets in Table 1. The resampling scheme is a combination of parametric and nonparametric bootstrap (Efron and Tibshirani, 1993; Bender *et al.*, 2005). The goal of our simulation study is to compare CV and CSV when used for evaluation and ranking of competing learning algorithms, in synthetic data that realistically simulate multiple independent datasets, where the true relationship between the independent and dependent variables is known. CV and CSV are then assessed with respect to their ability to recover the true global ranking, which we compute through Monte-Carlo integration. We assess the ability to recover the ranking by Kendall correlation between the true global ranking and the CV or CSV estimates.

For $b = 1, \ldots, B = 1000$ iterations, a collection of $I = 8$ datasets is generated as follows. First, 8 studies are sampled with replacement from the list of breast cancer studies. In other words, we resample the collection of studies to mimic the fact that studies are not considered as fixed but rather drawn from a population. This step only involves simulations from a multinomial Mult$(8, [1/8, \ldots, 1/8])$ distribution. Second, for each of the generated studies, $N = 150$ patients are sampled from the corresponding original dataset with replacement. Each of the 150 predictor vectors is directly generated from a study-specific empirical distribution (non-parametric bootstrap). Finally, the corresponding survival times are simulated from a proportional hazards model (parametric bootstrap) fitted to one of the available studies:

$$M^i_{true} : \lambda^i(t|x) = \lambda^i_0(t) \times exp(x^T \boldsymbol{\beta}_i), \qquad (1)$$

$i = 1, \ldots, I$, where $\lambda^i(t|x)$ is the individual hazard function when the vector of predictors is equal to x and $\boldsymbol{\beta}_i$ denotes a vector of regression coefficients. We combine the truncated inversion method in Bender *et al.* (2005) and the Nelson-Aalen estimator for cumulative hazard functions to simulate survival times that reflect survival distributions and follow-up of the real studies. The vector $\boldsymbol{\beta}_i$ is set identical to the coefficients fitted in study $i = 1, \ldots, I$ using the *CoxBoost* method (Binder and Schumacher, 2008). Note that a different regression method could have been used at this stage.

The collections of simulated datasets are then used both (i) to compute by Monte Carlo method the true global ranking defined in Section 2.5, and (ii) to compute estimated ranks by CV and CSV.

Figure 1a displays, for each pair of studies (i, j) in Table 1, C-index obtained when training a model by ridge regression on dataset i (rows), and validating that model on dataset j (columns). Diagonal elements $(i = j)$ are obtained by 4-fold CV. Figure 1b displays mean C-indices for each (i, j) combination across simulations, when the training and validation studies are generated resampling the i-th and j-th study. The diagonal elements are computed by averaging C-indices with the training and validation datasets independently generated by resampling from the same study.

The similarity between the two panels is striking, in particular with respect to the clear separation of the eight studies into two groups. The first group includes the studies MNZ, ST1, ST2, TRP, UNT and VDX, and seems to produce more accurate prediction models than the remaining studies. The datasets in this group seem also associated with higher values of the concordance index when used for validation. This difference between the two groups is also illustrated in Figure 1c. It displays the non-diagonal entries of the matrices represented in the left and middle panels, i.e. average C-indices from simulated datasets vs. C-indices from real data. This scatterplot shows a clear two-cluster structure: the yellow dots display the 30 training and validation combinations within the larger group of studies, i.e MNZ, ST1, ST2, TRP, UNT and VDX.

2.8 Evaluation criteria in simulations

In simulation studies, we can assess and rank learning algorithms based on their ability to recover the true underlying models M^i_{true}. In this section, we introduce a criterion that reflects the similarity between the true regression coefficients $\boldsymbol{\beta}_i$ that were used to simulate the i-th dataset and the coefficients $\hat{\boldsymbol{\beta}}^{(k)}_j$ fitted on dataset j. We consider separately pairs of studies with $i \neq j$ and pairs with $i = j$. The standard way to assess similarity between vectors is to compute the euclidean distance between them. However, since our focus is on prediction we consider the alternative criterion $\widehat{\text{cor}}(\mathbf{X}_i\boldsymbol{\beta}_i, \mathbf{X}_i\hat{\boldsymbol{\beta}}^{(k)}_j)$ to measure the similarity between the true $\boldsymbol{\beta}_i$ and fitted regression coefficients $\hat{\boldsymbol{\beta}}^{(k)}_j$. Here $\mathbf{X}_i$ is the matrix of predictors of dataset i and $\widehat{\text{cor}}$ denotes the empirical Pearson's correlation. The average

$$S^k_{self} = (1/I) \cdot \sum_i \widehat{\text{cor}} \left(\mathbf{X}_i\boldsymbol{\beta}_i, \mathbf{X}_i\widehat{\boldsymbol{\beta}}^{(k)}_i \right), \qquad (2)$$

over the I studies, provides a measure of the ability of learning algorithm k to recover the model that has generated the training dataset, hence the index *self*.

Another criterion of interest is the ability of a learning algorithm to recover the vector of regression coefficients $\boldsymbol{\beta}_i$ when we relax

No.	Name	Adjuvant therapy	# patients	# ER+	3Q survival $[mo.]$	Median follow-up $[mo.]$	Original identifiers ‡	Reference
1	CAL	chemo, hormonal	118	75	42	82	CAL	Chin *et al.* (2006)
2	MNZ	none	200	162	120	94	MAINZ	Schmidt *et al.* (2008)
3	MSK	combination	99	57	76	82	MSK	Minn *et al.* (2005)
4	ST1	hormonal	512*	507*	114	106	MDA5, TAM, VDX3	Foekens *et al.* (2006)
5	ST2	hormonal	517	325	126	121	EXPO, TAM	Symmans *et al.* (2010)
6	TRB	none	198	134	143	171	TRANSBIG	Desmedt *et al.* (2007)
7	UNT	none	133	86	151	105	UNT	Sotiriou *et al.* (2006)
8	VDX	none	344	209	44	107	VDX	Minn *et al.* (2007)

Table 1. Public microarray datasets of breast cancer patients as curated and summarized by Haibe-Kains *et al.* (2012). Datasets are referred to using the following acronyms: CAL = University of California, San Francisco and the California Pacific Medical Center (United States), MNZ = Mainz hospital (Germany). MSK = Memorial Sloan-Kettering (United States), ST1, ST2 are meta-datasets as provided by Haibe-Kains *et al.* (2012), therein named SUPERTAM1 and SUPERTAM2. TRB = dataset collected by the TransBIG consortium (Europe), UNT = cohort of untreated patients from the Oxford Radcliffe (United Kingdom), VDX = Veridex (the Netherlands). **# ER+** refers to the number of patients with the Estrogen Receptor positive subtype. **3Q survival** provides the Kaplan-Meier estimate of 75% survival probability for each cohort. **Median follow-up** is calculated from the reverse Kaplan-Meier estimate. *Numbers shown are after removal of samples duplicated in the dataset VDX. ‡ Dataset identifier(s) as specified in Haibe-Kains *et al.* (2012).

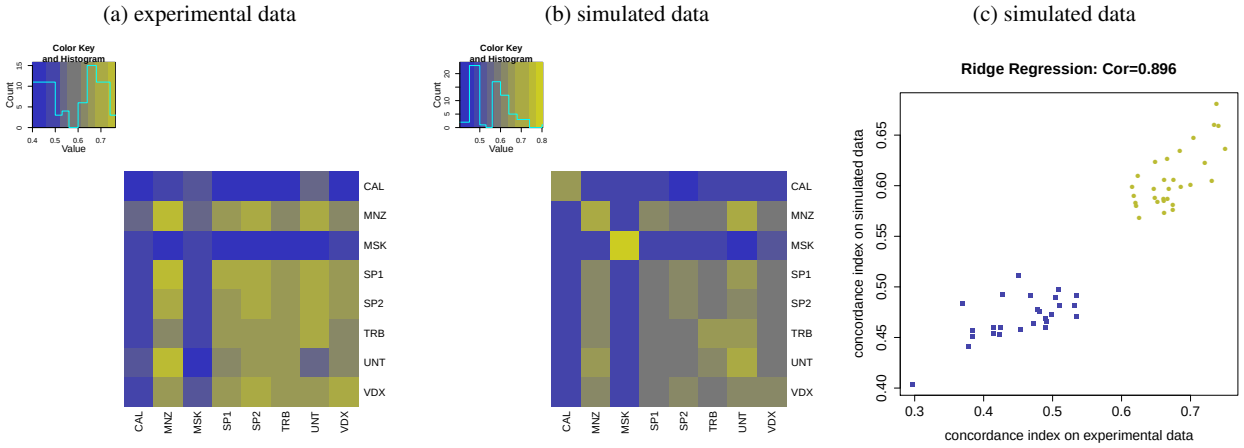


Fig. 1: Cross-study validation matrices $\mathbf{Z}^k$ in simulated and experimental data for Ridge Regression. Panel (a) displays C-indices for training and validation on each pair of actual datasets in Table 1. The diagonal of this matrix shows estimates obtained through 4-fold CV. The heatmap in panel (b) displays, for each pair of studies (i, j), the average C-index obtained when ridge regression is fit on simulated datasets generated by resampling gene expression data from the i-th study in Table 1 by non-parametric bootstrap and simulating censored survival outcome by parametric bootstrap; the resulting model is validated on a simulated dataset generated by resampling study j. Two independent datasets from the same study are sampled for the diagonal elements. Both heatmaps strongly resemble each other, indicating a realistic simulation scenario. CAL and MSK are outlier studies: cross-study C-index is approximately 0.5 when they are used either for training or validation. All values of the Z-matrix corresponding to these two studies build the blue "bad performance" cluster in panel (c) which compares the C-indices obtained for study pairs (i, j), $i \neq j$, on simulated data (y-axis) and experimental data (x-axis). Pearson correlation is ≈ 0.9. The three plots illustrate high similarity between simulated and real data in our application.

the assumption that the unknown models underlying training an validation datasets coincide. This can be quantified with

$$S^k_{across} = (1/(I(I-1))) \cdot \sum_i \sum_{j \neq i} \widehat{\text{cor}}\left(\mathbf{X}_i \boldsymbol{\beta}_i, \mathbf{X}_i \widehat{\boldsymbol{\beta}}^{(k)}_j\right), \quad (3)$$

where the index *across* emphasizes the focus on cross-study similarity, i.e. on the ability of algorithm k to recover the coefficients $\boldsymbol{\beta}_i$ when fitted on dataset j, with $j \neq i$. Instead of taking simple averages in Eqs. (2-3), one could also use different summaries, e.g. median or quantiles, as presented in Section 2.4.

Both S^k_{self} and S^k_{across} are criteria to assess and compare learning algorithms. The ranking obtained by ordering the algorithms according to their value of S_{self} (S_{across}) are denoted by R_{self} (R_{across}). Note, however, that S^k_{self} and S^k_{across} are by definition specific to the considered collection of studies and datasets: they involve the vectors $\boldsymbol{\beta}_i$ and the matrices $\boldsymbol{X}_i$ ($i = 1, \ldots, I$). We will

call the corresponding rankings *local* because they are specific to the collection of datasets at hand.

3 RESULTS

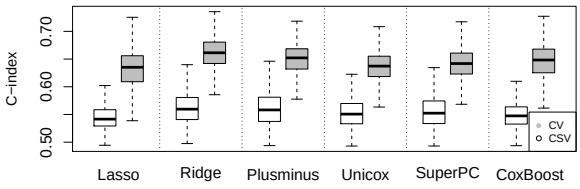
3.1 Simulated Data

Our focus in the simulation study is on differences between the rankings and performance estimates obtained by CV and CSV. Figure 2a shows the distributions of $\overline{CSV}$ and $\overline{CV}$, and Figure 2b shows the distribution of the rankings, across 1000 simulated 8-dataset compendia. Table 2 shows the median of these rank distributions, along with true global median ranks. The rank of method k is 1 if it yields the largest score of the K training algorithms. We observe large differences in the distributions of $\overline{CSV}$ and $\overline{CV}$ across simulations (Figure 1a): the average of the $\overline{CV}$ scores is around 0.65, while $\overline{CSV}$ scores are centered around 0.55. The variability of $\overline{CV}$ and $\overline{CSV}$ across simulations, however, is comparable.

Performance differences across algorithms, whether estimated by CV or CSV, are relatively small compared to the overall difference between CV and CSV performance estimates. We also observe differences between the rank distributions produced by CV and CSV. By both CV and CSV, *Glmnetlasso* is ranked as one of the worst performing algorithms, while *Glmnetridge* and *Plusminus* are ranked first or second. However the consistent advantage of *Glmnetridge* over *Plusminus* in $\overline{CV}$ vanishes under $\overline{CSV}$. The median rank of CoxBoost across simulations is two positions better as estimated by $\overline{CV}$ than by $\overline{CSV}$; in this case CSV is more consistent with the global true rankings (Table 2). Note that the exchange in top global true ranking between *Glmnetridge* over *Plusminus* when we consider average and median summaries (see section 2.4 and criterion in Table 2) occurs because the global true performance of these two algorithms is nearly indistinguishable.

The true *local rankings* of the $K = 6$ algorithms, defined by S^k_{across} or S^k_{self} in Section 2.8 vary over the 1000 simulated collections of studies. Furthermore, the median Kendall's correlation between R^k_{across} and R^k_{self} amounts to approximately 0.5, i.e. the local performance measures S^k_{across} and S^k_{self} defines distinctly different rankings (see Supplementary Figure 1). A natural question is how well $\overline{CV}_k$ and $\overline{CSV}_k$ can recover the unknown rankings R^k_{across} and R^k_{self}. The boxplots in Figure 3 display the Kendall's correlation between local rankings R_{across} (a) or R_{self} (b) and the ranking found by $\overline{CV}$ (gray boxes) or $\overline{CSV}$ (white boxes) across simulations. Figure 3(c) shows the same for true global ranking. Both local rankings (R_{across} and R_{self}) can be recovered by $\overline{CSV}$ with a Kendall correlation around 0.5. $\overline{CV}$ tends to be less correlated with R_{across}. This pattern is reversed for the local ranking R_{self}. We recall that, similarly to $\overline{CV}$, R_{self} is defined in terms of within-study validation. However, the difference between the medians in the second panel is less pronounced than in the first one. Finally, $\overline{CSV}$ features a considerably higher correlation to the true global ranking. This suggests that $\overline{CSV}$ is more suitable for recovering the global ranking. If the two outlier studies (CAL and MSK) are removed, the advantage of $\overline{CSV}$ over $\overline{CV}$ in recovering true global ranking is further increased (median Kendall correlation: 0.8 vs. 0.6, see Supplementary Figures S2-4, and also surpasses $\overline{CV}$ for recovering true local self ranking R_{self}. Overall, as displayed by the Supplementary Figure S3, it appears that, after outlier studies are removed, CSV outperforms substantially CV when used for ranking algorithms.

(a) Distribution of C-index



(b) Distribution of ranks

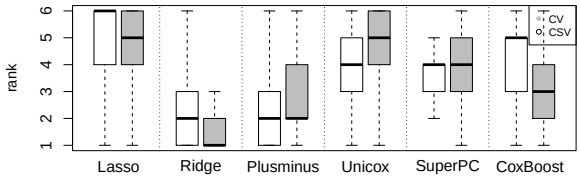


Fig. 2: Comparison of cross-study validation ($\overline{CSV}_k$) and cross-validation ($\overline{CV}_k$) on simulated data. Each boxplot represents evaluations of $K = 6$ algorithms in 1000 simulations of a compendium of $I = 8$ datasets. For each simulation the diagonal or off-diagonal elements of the Z^k matrix of validation C-statistics is summarized by (a) simple mean and (b) rank of the simple mean across algorithms. CV estimates of the C-index are much higher (approximately increased by 0.1) than CSV estimates. Lasso is ranked worst by both CV and CSV and Ridge / Plusminus are ranked best, however some differences are apparent. CoxBoost ranks two positions worse in CSV, and the distinct advantage seen in CV for Ridge over Plusminus vanishes in CSV.

Algorithm	global true ranking		CSV (med. ranks)		CV (med. ranks)	
Glmnetridge	1	2	2	2	1	2
Plusminus	2	1	2	2	2	2
Superpc	3	3	4	3	4	4
Unicox	4	4	4	4	5	4
CoxBoost	5	5	5	5	3	4
Glmnetlasso	6	6	6	6	5	6
criterion	Av.	Med.	Av.	Med.	Av.	Med.

Table 2. True global ranking and median rank estimate of CV and CSV on simulated data. Ranks shown for CV and CSV are the median across 1000 simulations; individual columns refer to summarization of Z^k_{ij} by Average or Median value. Third quartile ranks are the same as the median ranks, these are not shown. All methods rank GLMnetridge and Plusminus well, and GLMnetlasso poorly; however CSV ranks with Z^k_{ij} summarized using the median are the closest to the true global ranking. Variability of CV and CSV rank estimates across simulations is shown in Figure 2b.

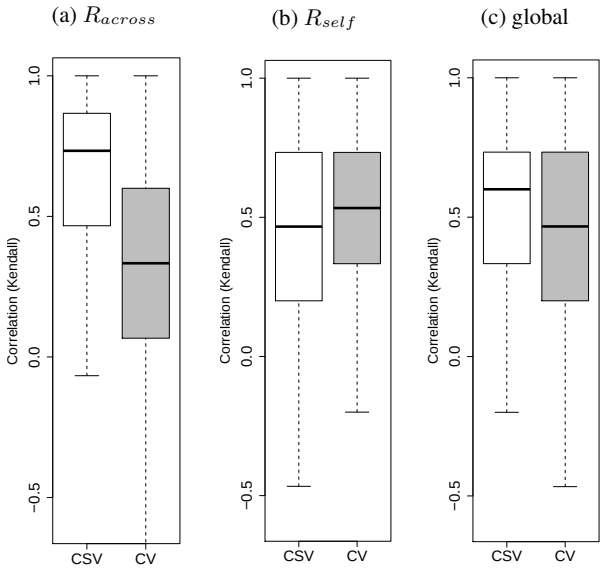


Fig. 3: Distribution of Kendall's correlation between true algorithm rankings and rankings estimated by $\overline{CSV}$ (cross-study validation, white) and $\overline{CV}$ (cross-validation, gray) on simulated data. Panels (a) and (b) compare CV and CSV in terms of their correlation to the *local rankings* (true rankings for each simulation iteration); (c) compares these to the global ranking based on all iterations. R_{across} and R_{self} are the ranks calculated from S_{across} and S_{self}, respectively. Each box represents correlations computed in each of the 1000 iterations of the simulation study. $\overline{CSV}$ achieves higher correlations, and lower variance, to the global ranking and this improvement is more pronounced for R_{across}. For R_{self} (within-study validation), this pattern is reversed as expected, although the difference between medians is smaller. Thus in simulated independent datasets, $\overline{CSV}$ recovers true across-study rankings more accurately and with less variability than $\overline{CV}$.

3.2 Application to breast cancer prognostic modelling

In this section we apply CV and CSV to the $I = 8$ breast cancer studies described in Section 2. Generally, the results resemble those obtained on simulated data. The top panel in Figure 4 illustrates the distributions of $\overline{CSV}$ and $\overline{CV}$ for each of the $K = 6$ algorithms. Except for the distinctly larger interquartile ranges of the boxes, the same patterns are observed as in Figure 2. Note that in these boxplots an observation represents a single entry of the $\mathbf{Z}^k$-matrix, whereas in Figure 2 each box represents the distribution across iterations of the $\mathbf{Z}^k$-matrix mean. This explains the higher variance observed in Figure 4. We observe in Figure 4 that:

- $\overline{CV}$ estimates are approximately 0.06 higher than $\overline{CSV}$ estimates on the C-index scale. To illustrate the magnitude of this improvement on the C-index scale consider a population with two groups of patients, high and low risk patients, covering identical proportions 0.5 of the population. A *perfect discrimination model* that correctly recognizes the subpopulation of each individual, when the hazard ratio between high versus low risk patients is 2.7, achieves on average a C-index of 0.62. For the average C-index of the perfect discrimination model to be increased to 0.68, a doubling of the true hazard ratio to 5.4 is necessary. Thus it is fair to characterize the validation results seen here by CV as much more optimistic than as seen by CSV.
- The presence of outlier studies (CAL and MSK, see Section 4 for a brief discussion on their specific characteristics) has a strong effect on the ranking estimates when we use the mean to summarize $\mathbf{Z}^k$ matrices. Using mean summarization, both $\overline{CSV}$ and $\overline{CV}$ rank *Superpc* first. This is mainly caused by high variability around the $C = 0.5$ of the $\mathbf{Z}^k_{i,j}$ validation scores obtained by models trained by outlier studies. In particular, *Superpc* and *Unicox* are the only algorithms that produce models with good discrimination when trained on the MSK study. With median summarization, ranking estimates are less influenced by the presence or absence of outlier studies. We therefore recommend the use of the median to summarize $\mathbf{Z}^k$ matrices.
- Using median aggregation of the $\mathbf{Z}^k_{ij}$ scores, the ranking defined by $\overline{CSV}$ is identical to the ranking found in our simulation example (cf. Supplementary Table ST1 and Table 2), reflecting the realistic simulation scenario and the robustness of median aggregation. For both median and third quartile aggregation the rankings defined by $\overline{CV}$ and $\overline{CSV}$ differ substantially (Kendall's correlations 0.6 and 0.07). This is consistent with results of the simulation study, where median correlation of the rankings defined by $\overline{CSV}$ and $\overline{CV}$ was approximately 0.5.
- Figure 4b illustrates that CSV performance estimates do not necessarily reflect CV results. This panel shows $\overline{CV}$ and $\overline{CSV}$ estimates for *Glmnetridge* averaged over all datasets combinations, with a fixed study used for training (black) or validation (gray). Cross-validation statistics are only moderately correlated with CSV statistics for models trained from the same dataset ($\rho = 0.2$), and negatively correlated with CSV statistics when that dataset is used for validation ($\rho = -0.33$).

3.3 $\overline{CV}$ performance is not necessarily indicative of cross-study performance

A more detailed analysis of the correlation between $\overline{CSV}$ and $\overline{CV}$ shows that cross-study prediction and within-study prediction are indeed less related than one might expect. Figure 4b, study specific values for $\overline{CSV}$ are plotted against $\overline{CV}$ for the learning algorithm *Glmnetridge* as an example (outlier studies removed). For each study we have a single value for CV but two values for CSV depending whether we average the $\mathbf{Z}$-matrix column-wise (identical validation study) or row-wise (identical training study). The correlations vary between algorithms but their values are usually around 0.5 and in the latter case (identical validation study) mostly negative. This phenomenon could be caused by 'study-specific' signals. Such study-specific signals may support within study prediction as tested in CV but will impede cross-study prediction. From this point of view, cross-study and within-study

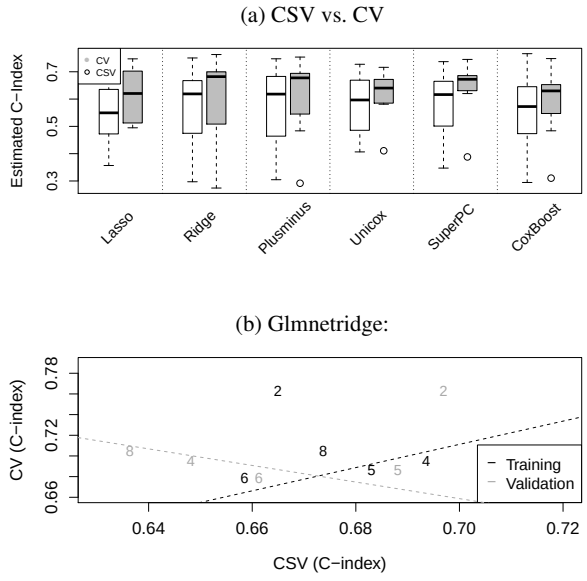


Fig. 4: Comparison of $\overline{CSV}$ and $\overline{CV}$ on Panel (a) describes the distributions of $\overline{CSV}$ and $\overline{CV}$ estimates separately for each of the six considered algorithms. In contrast to Figure 2a, each boxplot represents the distribution of the values of a single matrix $\mathbf{Z}^k$ as depicted in Figure 1b. The pattern is similar to the one obtained on the simulated data. $\overline{CV}$ leads to substantially higher estimates of discrimination performance compared to $\overline{CSV}$. Panel (b) illustrates the relation between $\overline{CV}$ and $\overline{CSV}$ estimates by averaging over all datasets combinations with a fixed study used for training (black) or validation (gray). The illustration shows results for the learning algorithm *Glmnetridge* and the numbers refer to the study number given in Table 1 (outliers CAL and MSK have been removed). Cross-validation statistics on the y-axis are moderately correlated to CSV (row wise means [fixed training study] or column-wise means [fixed validation study] of the non-diagonal elements in the corresponding $\mathbf{Z}^k$-matrix) on the x-axis. For *Glmnetridge* the correlation is 0.2 when we plot CV versus the row wise means and -0.33 when we consider the column wise means. These correlations vary over the K algorithms, but the same pattern is predominant.

prediction can be considered as two different types of problems with different characteristics.

Finally, we note that $\overline{CV}$ is a less suitable method for the detection of outlier studies, since it can estimate relatively good prediction performances even on studies where all algorithms fail in cross-study validation. In our example, this occurred for the MSK study with the algorithms *Superpc* and *Unicox*.

3.4 Specialist and generalist algorithms

This leads to the question of whether some algorithms might be considered as specialist algorithms according to the definition given in the introduction. It is obvious that our examples here are not exhaustive and additional examples will be required in order to determine 'specialist' or 'generalist' tendencies of these algorithms. However in this example, the fact that *Glmnetridge*, *Glmnetlasso* and *CoxBoost* rank distinctly better for $\overline{CV}$ than for $\overline{CSV}$ in our simulation example suggests that these two algorithms may adapt more strongly to the specific properties of an individual data set than other algorithms. This problem can be analyzed in greater detail using the local performance criteria S_{self} and S_{across} which directly measure the correlation of the true and fitted linear predictors on the training study it*self* as well as *across* studies.

CoxBoost and *Glmnetridge* indeed achieve better ranks in R_{self} than in R_{across}. CoxBoost improves its position by 1 or 2 ranks, which is also the difference which could be found between CoxBoost's $\overline{CSV}$ and $\overline{CV}$ rankings. Thus, one may conclude that these two algorithms have the most pronounced tendency to specialize to the dataset at hand.

Nonetheless, it must be mentioned that all the algorithms share this tendency at some extent since S_{self} is consistently substantially higher than S_{across} (median: 0.6 vs. 0.2). In this sense, the higher $\overline{CV}$ values are in part justified by the fact that all algorithms perform better in within-study prediction than in cross-study prediction. To address this issue more deeply, we also compare cross-validation to independent within-study validation for our simulated data. For the independent validation, we simulate two separate datasets using the true coefficients and gene expressions of a single study. Subsequently, a model is trained on the first dataset and evaluated on the second dataset. As can be seen in Supplementary Figure S5, $\overline{CV}$ values are slightly smaller than for independent validation. This matches the expectation because models are trained on less data in $\overline{CV}$. From this point of view, $\overline{CV}$ is not optimistic in our simulation study if it is used for estimating within-study performance, but it is optimistic for estimating performance in new studies of different patient cohorts.

4 DISCUSSION & CONCLUSION

In applying genomic approaches to clinical problems, it is rarely safe to assume that the studies used in a research environment faithfully represent what will be encountered in clinical application, across a variety of populations and medical environments. From this standpoint, study heterogeneity can be a strength, as it allows to quantify the degree of generalizability of results, and to investigate the sources of the heterogeneity. This aspect has long been recognized in meta-analysis of clinical trials (Moher and Olkin, 1995). Therefore, we expect that an increased focus on quantifying cross-study performance of prediction algorithms will contribute to the successful implementation of the personalized medicine paradigm.

In this paper we provide a conceptual framework, statistical approaches, and software tools for this quantification. As an illustrating example, we demonstrate cross-study validation on eight independent microarray studies of ER-positive breast cancer, with overall survival as the endpoint of interest. We also develop a simulation procedure involving two levels of non-parametric bootstrap (sampling of studies and sampling of observations within studies) in combination with parametric bootstrap, to simulate a compendium of independent datasets with characteristics of predictor variables, censoring, baseline hazards, prediction

accuracy, and between-dataset heterogeneity realistically based on available experimental datasets.

Cross-validation is the dominant paradigm for assessment of future prediction performance and comparison of prediction algorithms. The perils of inflated prediction accuracy estimations by incorrectly or incompletely performed cross-validation are well known (Molinaro *et al.*, 2005; Varma and Simon, 2006; Subramanian and Simon, 2010; Simon *et al.*, 2011). However, we show that even strictly performed cross-validation can provide optimistic estimates relative to cross-study validation performance. All algorithms, in both our simulation and example, showed distinctly decreased performance in cross-study validation compared to cross-validation. We believe this reflects the reality of clinical genomic study, and likely other applications, where it is impossible to control all sources of between-study heterogeneity or to ensure consistent application of new technologies in diverse laboratory settings.

In simulations, the ranking of algorithms by cross-study validation was closer to true rankings defined by cross-study prediction both within each sample of studies (R_{across}) and by global true ranking determined through Monte Carlo simulation. Surprisingly, cross-study validation was also competitive to cross-validation for recovering true rankings based on within-study prediction, even out-performing cross-validation in this application after the removal of outlier studies. Considering that the ultimate goal of prediction modeling is to provide predictions for independent observations, we claim that assessing algorithms by cross-study validation may provide a more useful indicator of the performance than the current standard of cross-validation.

Systematic cross-study validation also provides a means to prioritize relevant sources of heterogeneity within the context of the prediction problem of interest. By simple inspection of the CSV matrix we identified two outlier studies that yielded prediction models no better than random guessing in new studies. This may be related to known differences in these studies: smaller numbers of observations, higher proportions of node positive patients, or larger tumors. However, differences in other important clinical variables which are related to outcome, such as type of adjuvant therapy, had no obvious impact on cross-study prediction accuracy. Although we cannot verify indicators of poor cross-study prediction accuracy within the scope of this study, we can confidently conclude that these two datasets are outliers within the context of the compendium of datasets studied here.

We suppose that in practice it is neither possible nor even desirable to eliminate all sources of heterogeneity between study cohorts of complex disease. The adoption of 'leave-one-in' cross-study validation, in settings where at least two comparable independent datasets are available, can provide more realistic expectations of future prediction model performance, identify outlying studies or clusters of studies, and help to develop "generalist" prediction algorithms with less tendency to fit to dataset-specific biases. Further work is needed to formalize the identification of clusters of comparable studies, to develop databases for larger-scale cross-study assessment of prediction algorithms, and to develop better "generalist" prediction algorithms. With the development of software and data resources, cross-study validation is in practice no more difficult or CPU-consuming than cross-validation, and should become an equally standard tool for assessment of prediction models and algorithms.

ACKNOWLEDGEMENT

We wish to thank Benjamin Haibe-Kains for contributions to reproducible research in making the curated breast cancer datasets used in this study publicly available.

Funding: This work was supported by the National Science Foundation [grant number CAREER DBI-1053486 to CH].

REFERENCES

Baek, S., Tsai, C.-A., and Chen, J. J. (2009). Development of biomarker classifiers from high-dimensional data. *Brief. Bioinform.*, **10**(5), 537–546.

Baggerly, K. A., Coombes, K. R., and Neeley, E. S. (2008). Run batch effects potentially compromise the usefulness of genomic signatures for ovarian cancer. *J. Clin. Oncol.*, **26**(7), 1186–7; author reply 1187–8.

Bender, R., Augustin, T., and Blettner, M. (2005). Generating survival times to simulate Cox proportional hazards models. *Statistics in Medicine*, **24**(11), 1713–1723.

Binder, H. and Schumacher, M. (2008). Allowing for mandatory covariates in boosting estimation of sparse high-dimensional survival models. *BMC Bioinformatics*, **9**, 14.

Blair, E. and Tibshirani, R. (2004). Semi-supervised methods to predict patient survival from gene expression data. *PLoS Biology*, **2**, 511–522.

Boulesteix, A. L. (2013). On representative and illustrative comparisons with real data in bioinformatics: response to the letter to the editor by smith et al. *Bioinformatics*, **29**, 2664–2666.

Castaldi, P. J., Dahabreh, I. J., and Ioannidis, J. P. A. (2011). An empirical assessment of validation practices for molecular classifiers. *Briefings in Bioinformatics*, **12**(3), 189–202.

Chin, K., DeVries, S., Fridlyand, J., Spellman, P. T., Roydasgupta, R., Kuo, W. L., Lapuk, A., Neve, R. M., Qian, Z., Ryder, T., Chen, F., Feiler, H., Tokuyasu, T., Kingsley, C., Dairkee, S., Meng, Z., Chew, K., Pinkel, D., Jain, A., Ljung, B. M., Esserman, L., Albertson, D. G., Waldman, F. M., and Gray, J. W. (2006). Genomic and transcriptional aberrations linked to breast cancer pathophysiologies. *Cancer Cell*, **10**(6), 529–541.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, **7**, 1–30.

Desmedt, C., Piette, F., Loi, S., Wang, Y., Lallemand, F., Haibe-Kains, B., Viale, G., Delorenzi, M., Zhang, Y., d'Assignies, M. S., Bergh, J., Lidereau, R., Ellis, P., Harris, A. L., Klijn, J. G., Foekens, J. A., Cardoso, F., Piccart, M. J., Buyse, M., Sotiriou, C., and TRANSBIG Consortium (2007). Strong time dependence of the 76-gene prognostic signature for node-negative breast cancer patients in the transbig multicenter independent validation series. *Clin Cancer Res*, **13**(11), 3207–3214.

Efron, B. and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman and Hall, New York.

Foekens, J. A., Atkins, D., Zhang, Y., Sweep, F. C., Harbeck, N., Paradiso, A., Cufer, T., Sieuwerts, A. M., Talantov, D., Span, P. N., Tjan-Heijnen, V. C., Zito, A. F., Specht, K., Hoefler, H., Golouh, R., Schittulli, F., Schmitt, M., Beex, L. V., Klijn, J. G., and Wang, Y. (2006). Multicenter validation of a gene ExpressionBased prognostic signature in lymph NodeNegative primary breast cancer. *J. Clin. Oncol.*, **24**(11), 1665–1671.

Goeman, J. (2010). l_1 penalized estimation in the cox proportional hazards model. *Biometrical Journal*, **52**, 70–84.

Gnen, M. and Heller, G. (2005). Concordance probability and discriminatory power in proportional hazards regression. *Biometrika*, **92**, 965–970.

Haibe-Kains, B., Desmedt, C., Loi, S., Culhane, A. C., Bontempi, G., Quackenbush, J., and Sotiriou, C. (2012). A three-gene model to robustly identify breast cancer molecular subtypes. *J. Natl. Cancer Inst.*, **104**(4), 311–325.

Harrell, F. E. *et al.* (1996). Multivariate prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in Medicine*, **15**, 361–387.

Leek, J. T., Scharpf, R. B., Bravo, H. C., Simcha, D., Langmead, B., Evan Johnson, W., Geman, D., Baggerly, K., and Irizarry, R. A. (2010). Tackling the widespread and critical impact of batch effects in high-throughput data. *Nat. Rev. Genet.*, **11**(10), 733–739.

Micheel, C., Nass, S., Omenn, G., Trials, C., Services, B., Policy, B., and Medicine, I. (2012). *Evolution of Translational Omics: Lessons Learned and the Path Forward*. National Academies Press.

Miller, J. A., Cai, C., Langfelder, P., Geschwind, D. H., Kurian, S. M., Salomon, D. R., and Horvath, S. (2011). Strategies for aggregating gene expression data: the collapserows R function. *BMC Bioinformatics*, **12**, 322.

Minn, A. J., Gupta, G. P., Siegel, P. M., Bos, P. D., Shu, W., Giri, D. D., Viale, A., Olshen, A. B., Gerald, W. L., and Massagué, J. (2005). Genes that mediate breast cancer metastasis to lung. *Nature*, **436**(7050), 518–524.

Minn, A. J., Gupta, G. P., Padua, D., Bos, P., Nguyen, D. X., Nuyten, D., Kreike, B., Zhang, Y., Wang, Y., Ishwaran, H., Foekens, J. A., van de Vijver, M., and Massagué, J. (2007). Lung metastasis genes couple breast tumor size and metastatic spread. *Proc Natl Acad Sci U S A*, **104**(16), 6740–6745.

Moher, D. and Olkin, I. (1995). Meta-analysis of randomized controlled trials: A concern for standards. *JAMA*, **274**(24), 1962–1964.

Molinaro, A. M., Simon, R., and Pfeiffer, R. M. (2005). Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, **21**(15), 3301–3307.

Schmidt, M., Böhm, D., von Törne, C., Steiner, E., Puhl, A., Pilch, H., Lehr, H. A., Hengstler, J. G., Kölbl, H., and Gehrmann, M. (2008). The humoral immune system has a key prognostic impact in node-negative breast cancer. *Cancer Res*, **68**(13), 5405–5413.

Simon, R. M., Paik, S., and Hayes, D. F. (2009). Use of archived specimens in evaluation of prognostic and predictive biomarkers. *J. Natl. Cancer Inst.*, **101**(21), 1446–1452.

Simon, R. M., Subramanian, J., Li, M.-C., and Menezes, S. (2011). Using cross-validation to evaluate predictive accuracy of survival risk classifiers based on high-dimensional data. *Briefings in Bioinformatics*, **12**, 203–217.

Sotiriou, C., Wirapati, P., Loi, S., Harris, A., Fox, S., Smeds, J., Nordgren, H., Farmer, P., Praz, V., Haibe-Kains, B., Desmedt, C., Larsimont, D., Cardoso, F., Peterse, H., Nuyten, D., Buyse, M., Van de Vijver, M. J., Bergh, J., Piccart, M., and Delorenzi, M. (2006). Gene expression profiling in breast cancer: understanding the molecular basis of histologic grade to improve prognosis. *J Natl Cancer Inst*, **98**(4), 262–272.

Subramanian, J. and Simon, R. (2010). Gene expression-based prognostic signatures in lung cancer: ready for clinical use? *J. Natl. Cancer Inst.*, **102**(7), 464–474.

Symmans, W. F., Hatzis, C., Sotiriou, C., Andre, F., Peintinger, F., Regitnig, P., Daxenbichler, G., Desmedt, C., Domont, J., Marth, C., Delaloge, S., Bauernhofer, T., Valero, V., Booser, D. J., Hortobagyi, G. N., and Pusztai, L. (2010). Genomic index of sensitivity to endocrine therapy for breast cancer. *J Clin Oncol*, **28**(27), 4111–4119.

Tibshirani, R. (2009). *uniCox: Univarate shrinkage prediction in the Cox model*. R package version 1.0.

Varma, S. and Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics*, **7**, 91.

Zhao, S., Huttenhower, G. P. C., and Waldron, L. (2013). Mas-o-menos: a simple sign averaging method for discrimination in genomic data analysis. http://biostats.bepress.com/harvardbiostat/paper158/. Accessed: 2013-10-24.

D.3 Application of Microarray Analysis on Computer Cluster and Cloud Platforms (Bernau et al., 2013)

The following paper has been written in cooperation with Jochen Knaus (Institute of Medical Biometry and Medical Informatics, Albert-Ludwigs-University Freiburg) and Anne-Laure Boulesteix (Department for Medical Informatics, Biometry and Epidemiology). Jochen Knaus has substantially contributed to *Section 2.1* and the discussion (*Section 3*) and to the corresponding sections in Chapter 3. Moreover, he has revised some of the remaining parts as well and provided helpful feedback and comments. The script for starting the several cloud instances and setting up a MPI environment has also been created by him. Anne-Laure Boulesteix has mainly contributed to the final manuscript by providing feedback and comments.

Application of Microarray Analysis on Computer Cluster and Cloud Platforms

C. Bernau[1]; A.-L. Boulesteix[1]; J. Knaus[2]
[1]Department for Medical Informatics, Biometry and Epidemiology (IBE), Ludwig-Maximilians-University Munich, Munich, Germany;
[2]Institute of Medical Biometry and Medical Informatics, Albert-Ludwigs-University Freiburg, Freiburg, Germany

Keywords
Biostatistics, cloud computing, computing methodologies, microarray analysis, statistical computing

Summary
Background: Analysis of recent high-dimensional biological data tends to be computationally intensive as many common approaches such as resampling or permutation tests require the basic statistical analysis to be repeated many times. A crucial advantage of these methods is that they can be easily parallelized due to the computational independence of the resampling or permutation iterations, which has induced many statistics departments to establish their own computer clusters. An alternative is to rent computing resources in the cloud, e.g. at Amazon Web Services.
Objectives: In this article we analyze whether a selection of statistical projects, recently implemented at our department, can be efficiently realized on these cloud resources. Moreover, we illustrate an opportunity to combine computer cluster and cloud resources.
Methods: In order to compare the efficiency of computer cluster and cloud implementations and their respective parallelizations we use microarray analysis procedures and compare their runtimes on the different platforms.
Results: Amazon Web Services provide various instance types which meet the particular needs of the different statistical projects we analyzed in this paper. Moreover, the network capacity is sufficient and the parallelization is comparable in efficiency to standard computer cluster implementations.
Conclusion: Our results suggest that many statistical projects can be efficiently realized on cloud resources. It is important to mention, however, that workflows can change substantially as a result of a shift from computer cluster to cloud computing.

Correspondence to:
Christoph Bernau
Ludwig-Maximilians-University Munich
Department for Medical Informatics, Biometry and Epidemiology (IBE)
Marchioninistr. 15
81377 Munich
Germany
E-mail: bernau@ibe.med.uni-muenchen.de

Methods Inf Med 2012; 51: ■–■
doi: 10.3414/ME11-02-0043
received: November 7, 2011
accepted: March 5, 2012
prepublished: ■■■

1. Introduction

With the arrival of new biological methods like gene expression microarrays and high throughput technology, statisticians are faced with huge amounts of data and computationally intensive methods for analyzing them. Moreover, the growing popularity of resampling approaches and permutation tests requires iterating a basic analysis many times which is demanding even more computational power. An important advantage of these statistical tools is that they can be easily parallelized which drastically reduces computation times provided that enough computational resources are available. Therefore, many statistics departments established their own computer clusters in order to apply these computationally intensive methods. These computer clusters are significant investments and especially for smaller departments it is questionable whether they are really necessary. A possible alternative to buying and maintaining one's own computer cluster could be renting computational resources 'in the cloud', e. g. at Amazon Web Services [1]. Meanwhile, several statistical open source projects like Bioconductor, which provides many routines for the statistical analysis of modern biological data, have developed Amazon Machine Images for their respective users [2]. In this paper we want to assess whether such statistical methods, which are implemented using the R language and environment [3], can sensibly be realized at Amazon Web Sercvices Elastic Compute Cloud (AWS EC2) and which problems might occur. Please note that AWS EC2 possibly does not match one's personal definition of the term cloud computing. The term "utility computing" [4] may be more adequate since one actually rents a computer cluster in a specific datacenter. We do not integrate machines from all over the world which might be a different interpretation of cloud computing.

In this article we will primarily focus on programming and parallelization aspects whereas we will mostly disregard financial, security, privacy and general technical aspects which will be subject of other articles in this special topic. The closely related topic of costs for cloud computing is treated in more detail in [5]. In this perspective we arrange our paper in the following way: the first section will be devoted to basic benchmarks of the different EC2 instance types and their network capacity. In the following

section, we will present two statistical projects whose realization in the cloud will be compared to the common computer cluster implementation. Subsequently, we will introduce an alternative approach which tries to combine machines from both sources. At the end, we will discuss the results and experiences we obtained while implementing our projects in the cloud.

2. Methods and Results

Our test of the usability and efficiency of the individually assembled computer clusters at AWS EC2 consists of four stages. At first we analyze the basic computation and network capacity of the different instances and compare them to other computer clusters used by our statistical department, namely the computer cluster at the IBE (ibec) and the mpp1-cluster (mpp) at the Leibniz Computing Center in Munich. Subsequently, we will implement in the cloud some important parts of two statistical projects which have already been realized on common computer clusters. This is principally the crucial part of our article since it will answer our main question, namely whether the statistical analysis of high-dimensional biological data can be reasonably realized in the cloud. In our last experiment we will go one step further by combining machines from several computer clusters at different departments and EC2. This experiment will present an opportunity to let machines from very heterogeneous sources jointly compute different sub-tasks of a statistical analysis.

2.1 Simple Trend Benchmarks

Synthetically benchmarking computer systems is basically a science on its own, which we do not want to treat here. But for an overview and estimation of runtime behavior of our real applications on the different EC2 instances, two simplistic synthetic benchmarks were used. Both are not benchmarking the complete (virtual) machine, but speed and memory aspects of the destination platform R.

Overall system benchmarks like the Spec suite [6] are far more elaborated and precise, but lack the integration of the R interpreter in the results. Moreover, the Spec test is rather difficult to configure and time consuming to run (Spec results for some instance types can be found in [7]). Our small benchmarks are using the arithmetic system of R – as desired for the biostatistical software used in the bigger tests afterward.

The performance of single cores of the CPU is tested by calculation of a specific subset of the Mandelbrot set [8], which is done using a recursive loop calculating whether a complex polynomial is bound or not. The inner loop is very small, so depending on the specific R environment it can be held and executed inside the processor's cache.

Our memory test is even simpler. It copies memory blocks inside a matrix guaranteed bigger than the processors cache. Overall runtime of the benchmarks is short, between one and three minutes depending on the system. Therefore the benchmark can be repeated many times (at least 20 iterations), giving a measurement of the variance in execution time, which is only taken inside the R program and therefore no startup time is included.

These short running benchmarks are giving a basic trend of the systems processor's speed using R and they provide an opportunity to quickly and therefore cheaply test the basic system performance, which is important due to the sheer amount of different EC2 instance types.

▶ Table 1 provides an overview over the instances and servers we used for our analysis. We focused on the instance types m1.small, m1.large and c1.medium, which are basically the cheapest instances. Nonetheless, they provide enough memory for the statistical projects which we will analyze in the subsequent sections. The first impression is that all three instance types are slower than our computer cluster machines which were bought about four years ago. Especially the m1.small instance, which is EC2's standard instance, needs about two to three times more time for our first benchmark. For compatibility issues, this instance type is representing and running at the speed of a 1 Ghz CPU from 2006 when EC2 was introduced. Via virtualization, it only gets time shares of about 40% of a single physical core [9]. The m1.large and c1.medium instances are remarkably faster whereby both instances provide two virtual cores. In our appendix, we also provide the results of these two benchmarks for the larger, faster instances.

For computations with low memory demands c1.medium is basically the instance of choice having almost the same price per core as the slower m1.small instance. The m1.large instance provides 3.75GB per core. However, its costs are four-fold. This price structure clearly indicates that writing memory efficient code can save money. Thus, it is quite probable that a statistical analysis in the cloud will produce some avoidable costs. For example, we usually had to perform our computations using only one core per c1.medium instance. It is also worth mentioning that the performance of the AWS instances is subject to higher variability.

Depending on the structure of the underlying Xen virtualization [10], cores and memory are separated quite well on the instances. But network and storage I/O

Table 1 Technical information and simple trend benchmark results for computer cluster and cloud instances. * Prices from the Amazon US east data center in North Virginia (September 2011). Prices vary between the individual data centers. ** Although running on a 2.7GHz CPU this instance only features the performance of a 1.7GHz Xeon processor [9].

Identifier/ API-Name	Provider	CPU [Ghz]	Cores/ machine	RAM per core	Costs/ h*	Benchmark 1 Mean (sd) [s]	Benchmark 2 Mean (sd) [s]
ibec	IBE	2.7	4	1.5 GB	–	79.1 (1.03)	71.8 (0.7)
mpp	LRZ	2.0	16	2 GB	–	98.6 (2.5)	91.9 (3.6)
gvs1/3	LRZ	2.3	16	8 GB	–	100.6 (1.8)	97.8 (0.23)
m1.small	AWS	2.7	1	1.7 GB	0.085$	309.1 (6.9)	317.6 (6.6)
c1.medium	AWS	2.3**	2	0.88 GB	0.17$	132.6 (4.9)	135.1 (5.3)

cannot be separated this way and therefore all instances on a host can affect each other. In order to check the network capacity we used Rmpi's broadcast command [11] for sending a 2 gigabytes AffyBatch object [12], which is the commonly used R-object for microarray batches, to a varying number of worker cores.

As shown in ▶Figure 1, we can observe that this broadcast takes slightly longer on the EC2 instances and that their runtimes are growing faster if the number of worker cores is increased. In the case of the mpp-cluster, which is equipped with an infiniband network connection, the broadcasting time basically does not increase after 10 cores. However, network performance seems to be sufficient for most statistical methods where larger amounts of data are only sent once. Further analysis of the network connection and description of technical details can be found in the appendix. The result of these first tests are that computations might take longer on the chosen EC2 instances but the network capacity seems to be sufficient for the planned projects

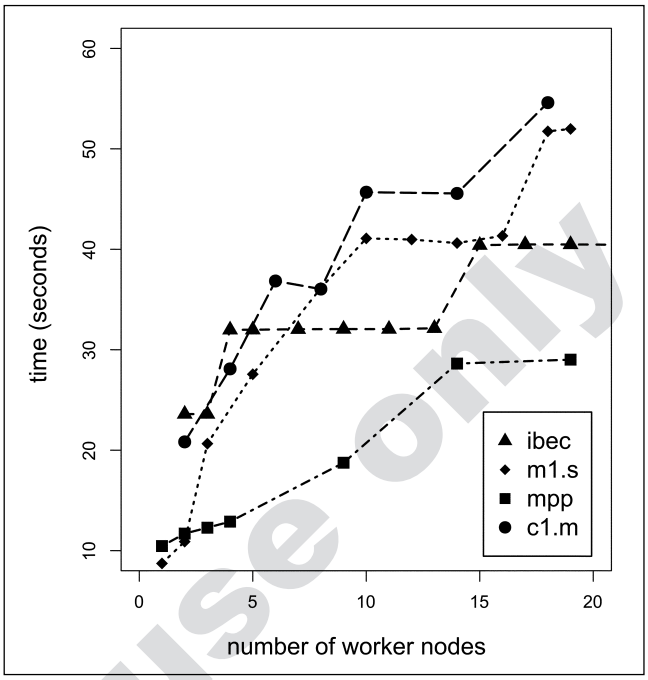


Fig. 1
Time needed to broadcast an AffyBatch object of size 2GB to an increasing number of worker cores.

2.2 Project 1: Optimization Bias

The first project to be conducted in the cloud deals with nested resampling procedures in the context of supervised classification with high-dimensional microarray data as predictors. Supervised classification algorithms, like Support Vector Machines (SVM) [13], are commonly assessed through a resampling procedure such as cross-validation. Such a resampling procedure whose aim is to estimate the misclassification rate is here denoted as external. Many supervised classification algorithms for high-dimensional data involve a tuning parameter, for instance the cost parameter for SVM that has to be adequately chosen. Simply trying several values of the tuning parameter and reporting the best performance only yields an optimistic bias [14]. A possible approach to avoid this optimization bias consists in performing nested resampling which means that, in each external resampling iteration, the optimal value of the tuning parameter is chosen by performing an additional internal resampling loop within the corresponding external training set.

The goal of this project is to estimate the bias induced by simply trying several predefined cost parameter values and reporting the minimal resampling misclassification rate only. In order to analyze this bias, one has to reiterate many times the whole resampling approach with SVM as a classification method. Additionally, the whole procedure has to be repeated for several microarray data sets because the optimization bias, just as the cost parameter itself, strongly depends on the classification

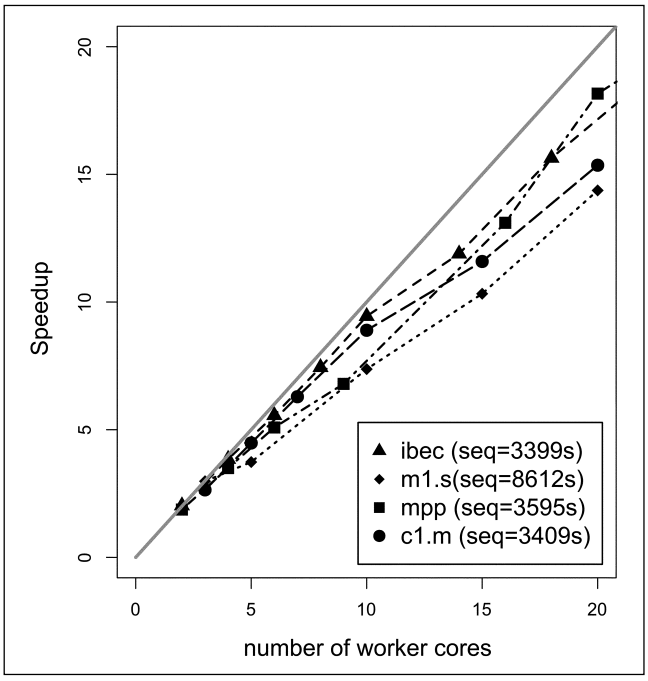


Fig. 2
Speedup and sequential runtimes in seconds for the 100 resampling iterations in Project 1 on the various platforms.

task at hand [15]. For financial reasons, we do not implement the whole project on the EC2 instances but demonstrate the efficiency of the parallelization for the most important part of this analysis. This central routine consists in the resampling approach for a specific cost parameter value. If this part scales sufficiently we can expect the whole project to be efficiently scaling as well. ▶Figure 2 shows the speedup for computer cluster and cloud instances, which consists of the sequential time divided by the parallel time for an increasing amount of workers. The gray line represents a perfect linear speedup on adding additional workers. Technically, this is based on a parallelization in R using MPI. We provide the code used for our analyses on a publically available Amazon Machine Image (see appendix). The consumed time is only measured inside the R program, so the overhead of instance startup is not included. Starting up 20 EC2 instances can take several minutes whereby this start up time is quite variable. Including this overhead into our analysis would severely distort our results since we analyze a small part of our project. Moreover, AWS charges fees per started hour and instance which also has to be considered appropriately. Both of these issues are negligible if the whole project is calculated. We can see that scaling is almost identical for cloud and computer cluster instances. Of course, the absolute computation times are remarkably higher for the m1.small instance. However, since the implementation scales well, the higher sequential runtime can be easily reduced by renting more instances. The main result of this experiment is that the main computation part can also be efficiently parallelized on EC2 instances.

2.3 Project 2: Normalization of Microarray Data

The characteristics of the second project are quite different since it combines a resampling approach with a computationally intensive task which can be parallelized itself. Gene expression microarray data have to be normalized before they can be used for high-level statistical analysis. Since in resampling procedures training and test data sets have to be strictly separated and classifiers should be trained without "seeing" the test data, microarrays should in principle be normalized either individually or, for multi-array approaches, in each resampling iteration anew. In the latter case, which is considered here, two variants are conceivable: the data can be normalized either separately for training and test set or an add-on normalization procedure can be used to normalize the test set subsequently using normalization parameters estimated from the training set [16]. In practice, however, researchers often normalize the whole data set only once before splitting it into training and test sets for computational reasons. Doing so, they indirectly violate the separation between training and test sets. The goal of project 2 is to analyze the effect of this imperfect separation on the classification results by comparing this approach to the more computationally intensive variants mentioned above performing normalization in each iteration anew and requiring a strong computational effort.

The core part to be analyzed consists in 6 repetitions of 5-fold cross validation procedure for a relatively small microarray data set of size n = 47 [17]. In this case we have two possible options for parallelization. The first option consists in parallelizing the normalization step itself based on a distributed data approach. This approach will also decrease the amount of memory necessary to perform the task which therefore also might decrease EC2 costs. The corresponding implementation can be found in the package affyPara [18]. This package was actually designed with larger data sets in mind (▶Appendix). Nonetheless we can compare the efficiency of the parallelization in the cloud to its efficiency on computer cluster machines. As can be seen from ▶Figure 3, the parallelization is scalable up to about five cores and saturates afterwards, whereby the speedup curves are comparable for computer cluster and cloud instances.

The actual implementation of the project is based on the parallelization of the resampling iterations. On our local computer cluster, the serial implementation requires about two hours even for this small core part which clearly demonstrates the need of parallelization. The second version scales well up to at least 20 cores which is the upper limit for EC2 instances in our analysis. We know from an analysis on different computer clusters that this implementation can scale up to approximately

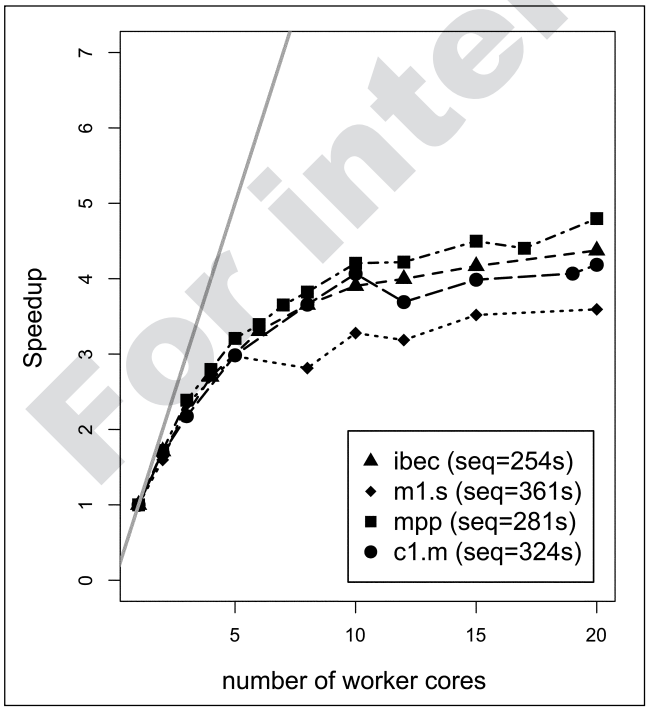


Fig. 3 Speedup and sequential runtime in seconds for an AffyPara-Preprocessing routine on the different platforms.

125 cores depending on the size of the data set at hand. Again there is not much difference between the efficiency of the parallelization on computer cluster and cloud instances. Speedups are almost perfectly linear on all architectures. For the m1.small instances computation times are more than doubled in comparison to our computer cluster machines and to the other EC2 instances. However, this can be alleviated by simply taking more instances or the more expensive, faster instances. Of course, one advantage of slower instances is that parallelization yields even more profit as far as the reduction of the absolute computation time is concerned. Please note that the required memory of our computation forces the second c1.medium core to remain idle in this implementation.

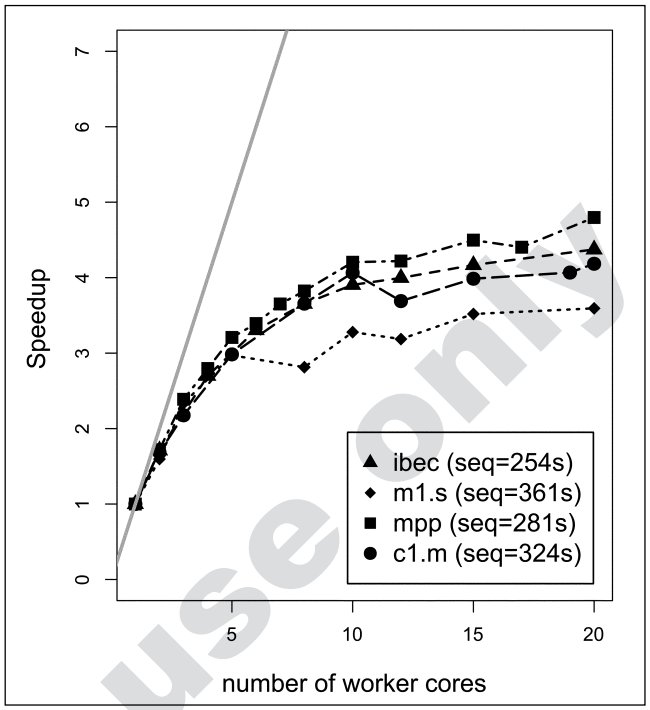


Fig. 4 Speedup and sequential runtime in seconds for Project 2 on computer cluster and cloud instances.

2.4 A Hybrid Cloud Solution: Combining Computer Cluster & Cloud Resources

An advantage of the cloud is its scalability, i.e. the possibility to get more resources if more resources are needed. If a given infrastructure is fully utilized in peak times, the possibility to add machines from the cloud to existing computer clusters would be helpful. With some efforts, EC2 instances can be directly integrated into local computer clusters manually. Far more convenient is the usage of a system which can handle this automatically. In the following, we show how to use the "data structure server" Redis [19] together with its R connector doRedis [20] to combine local computer clusters and cloud instances. As networking is partly routed over the public Internet, it is clearly a bottleneck. But for embarrassingly parallel [21] programs like the examples shown above, networking speed is not essential.

The concept of the Redis approach is that the server manages all the subtasks (e. g. a single resampling iteration) of a job and sends them to whatever worker node we connect to it. To illustrate this approach we use an enlarged version of the core part of Project 1. We compute 3000 subsampling steps instead of one hundred. Once we have a function for a single subsampling iteration (here: clrun) the whole approach can be implemented in less than ten lines (see appendix). An important optional feature is the opportunity to specify the number of tasks a certain connected worker machine shall perform. If we do not want some worker machines to participate until the very end of the whole computation we can set this parameter to a small number and the corresponding worker machine will stop as soon as it has performed the respective number of tasks. A really big advantage of this separation of job administration and computation is that we do not encounter any problems if we temporarily do not have any workers at our disposal, i.e. our job can be paused. Results and the description of the individual subtasks of the job are stored independently from the workers. For example, if one wants to use the staff's computers for larger computation tasks at night, one can use a small script connecting the respective computer to the Redis server as soon as the staff leaves the office. In our example we combined resources from the LRZ (gvs1, gvs3), IBE and EC2.

Since this example was intended to demonstrate the sheer possibility of this approach our pool of worker machines primarily consisted of free computing resources. ▶Figure 5 shows how many tasks have been performed by the different machines. Of course, the main part of the tasks has been processed by the 30 IBEC cores. However, we can see that the c1.medium instances processed more tasks per core than the IBEC machines although their connection to the Redis server is slower. In practice, the actual number of integrated cloud instances can be chosen depending on the availability of free computing resources and the urgency of the job.

In this context, one has to consider some of the problematic aspects of cloud computing. For example, some of the machines in the previous illustration used R 2.12 whereas other ones used R 2.13. The individual worker cores actually do not know about each other. These consistency problems might be overcome by the solutions already available at AWS, namely virtual machines and images.

3. Discussion

As illustrated in this paper, computationally intensive statistical projects can be real-

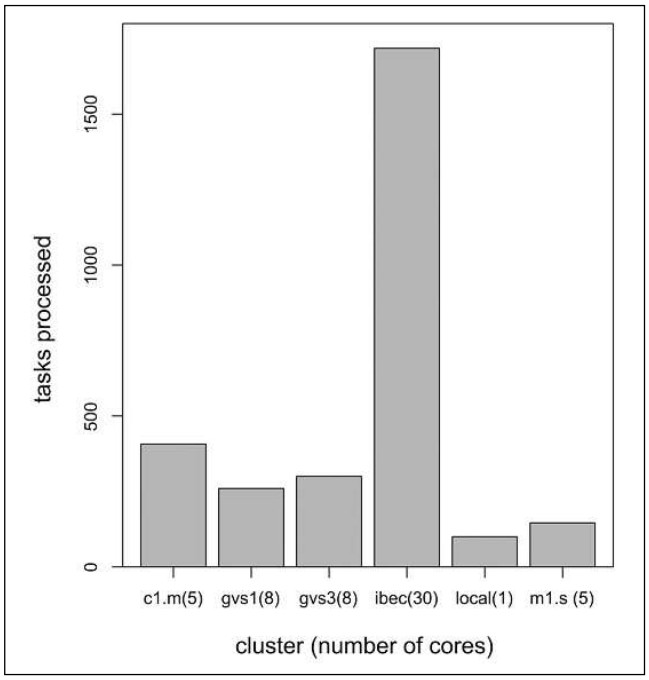


Fig. 5
Distribution of the 3000 tasks among the different resources for the redis server based combination of computer cluster and cloud instances.

ized in the cloud. Basically, EC2 provides different instance types for almost all computational requirements a statistical analysis of high-dimensional biological data may need. The default instance, m1.small, is quite slow. However, the c1.medium instance, which costs only slightly more per CPU, can keep up with usual computer cluster machiness as far as speed is concerned. Since our projects scaled sufficiently well on the EC2 instances it is also possible to reduce computation time by simply increasing the number of instances. Especially if one needs more than 850MB RAM which means that the second c1.medium core remains idle, one can rent a larger number of m1.small instances instead. If projects are even more memory-intensive they can be realized on the m1.large instance with 7.5GB RAM and sufficiently fast processors. For financial reasons we run our projects on smaller data sets. As shown in ▶Table 1, EC2 provides several instance types which offer enough memory for performing the same analysis on remarkably larger data sets for remarkably higher prices. From our experience we do not expect the implementations of our projects to scale worse if applied to larger data sets.

We would also like to emphasize here that we performed our analysis on the small EC2 instances in order to reduce costs. Of course, it is possible to rent faster, more expensive instances at EC2 which will further reduce the computation time in the cloud such that runtimes might be even smaller than on the computer clusters we used.

Overall, we can state that the efficiency of the parallelization in the cloud is comparable to computer cluster implementations for the presented applications and therefore most likely also for a wide range of other biostatistical applications.

Finally, we would like to discuss some experiences concerning the workflow in the cloud and on computer clusters which might also have an impact on one's individual choice. A very convenient feature of utility computing is its high flexibility. You can build your computer cluster at EC2 almost whenever you want and you can customize it accordingly to your individual needs. You also have the opportunity to rent single instances for sequential code, e.g. if the memory requirements of your computations might exceed the capacity of your local machines. Another advantage is that you have full control over your resources and you do not have to resort to job scheduling systems coordinating the needs of the different computer cluster users. On the contrary, the workflow on cloud resources is remarkably changed by the fact that one has to pay for each computation. One always reflects twice whether a specific computation is really necessary which limits the opportunity to test and experiment. Of course, this will induce researches to plan their computations more elaborately and avoid unnecessary tests which might also be considered a positive side-effect. Cloud computing, however, also has some hurdles for inexperienced users. Simple mistakes like forgetting to shut down instances or moving results to the own computer can easily produce unnecessary costs. Moreover, one must always estimate time and memory needs in order to decide which instances should be preferred for the task at hand. Consequently, currently AWS and similar services are probably not the best place for the first steps in parallel computing. Another problem concerns experienced users as well. After finding an error or recognizing that a certain part of one's algorithm has to be changed, all the computations have to be repeated. Also, discovered problems in used packages can force a recalculation. In an advanced stage of a project this means that the computation costs will effectively multiply.

After all, utility computing seems to be a tool which has to be applied in a considered way. Instead of switching entirely to cloud computing, a possible way to get started consists of supporting one's own computer cluster resources by renting cloud instances in times of workload peaks. An option to flexibly combine computer cluster and cloud resources even within the scope of a single computation task consists in the redis-server based implementation we applied in this article. After using both resources for a while, one will probably be able to assess which solution better matches one's individual needs.

Acknowledgments

We would like to thank Ferdinand Jamitzky for helping us to implement our analysis methods in parallel and run them at the Leibniz Computing Center and Monika Jelizarow for providing details about the

computation hours of her project on global tests for ordinal data. Moreover, we want to thank the anonymous referees for their helpful remarks and suggestions for improvements.

References

1. Amazon Web Services (2011): Amazon Elastic Compute Cloud (EC2). Available: http://aws.amazon.com/ec2 (accessed: Jan 20, 2012).
2. Bioconductor (2011): Bioconductor – Cloud AMI. Available: http://www.bioconductor.org/help/bioconductor-cloud-ami/ (accessed: Oct 28, 2011).
3. R development Core Team (2011): R: A Language and Environment for Statistical Computing. Available: http://www.R-project.org/ (accessed: 2012 Jan 14)
4. Wikimedia Foundation. Wikipedia (2012): Utility Computing. Available: http://en.wikipedia.org/wiki/Utility_computing (accessed: Jan 29, 2012)
5. Knaus J., Hieke S., Binder H., Schwarzer G. (2012): Costs and Benefits of Cloud Computing for a Biometry Department. Submitted to: Grid and cloud computing methods in biomedical research [Special topic]. Methods Inf Med.
6. Standard Performance Evaluation Corporation (2011): Spec CPU [online]. Available: http://www.spec.org/benchmarks.html (accessed: 2011 Oct 28).
7. Coffey P., Beliveau J., Mogre N., Harner A. (2011): Benchmarking the Amazon Elastic Compute Cloud (EC2) [online]. Available: http://www.wpi.edu.Pubs/E-project/Available/E-project-030811-115350/unrestricted/AmaznEC2_MQP_Final.pdf (accessed: Oct 23, 2011).
8. Mandelbrot BB. The fractal geometry of nature. New York: W.H. Freeman and Company; 2003.
9. Evangelinos C, Hill CN. Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. Proceedings of CCA-08.2008.
10. Citrix Systems (2012): Xen. Available: http://xen.org/ (accessed Jan 25, 2012).
11. Yu H. (2010): Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface). R package version 0.5–9 (online). Available: http://CRAN.R-project.org/package=Rmpi (accessed: Oct 28, 2011).
12. Gautier L, Cope L, Bolstad B M, Irizarry RA. affy-analysis of Affymetrix GeneChip data at the probe level. Bioinformatics 2004; 20 (3): 307–315.
13. Hastie T, Tibshirani R, Friedman .H. The Elements of Statistical Learning. New York: Springer-Verlag; 2001.
14. Varma S, Simon, R. Bias in error estimation when using cross-validation for model selection. BMC Bioinformatics 2006; 7: 91.
15. Bernau C, Augustin T, Boulesteix A-L. Correcting the optimally selected resampling-based error rate: A smooth analytical alternative to nested cross-validation. Department of Statistics: Technical Reports, Nr. 105, 2011. Available: http://epub.ub.uni-muenchen.de/12231/ (accessed: Jan 29, 2012).
16. Kostka D, Spang R. Microarray Based Diagnosis Pro?ts from Better Documentation of Gene Expression Signatures. PLoS Computational Biology 2008, 4, e22.
17. Ancona N, Maglietta R, Piepoli A, D'Addabbo A, Cotugno ., Savino M, Liuni S, Carella M, Pesole G, Perri F. On the statistical assessment of classifiers using DNA microarray data. BMC Bioinformatics 2006; 19 (7): 387.
18. Schmidberger M, Vicedo E, Mansmann U. affyPara: Parallelized preprocessing methods for Affymetrix Oligonucleotide Arrays. Rpackage version 1.13.0, 2011 (online). Available: http://bioconductor.org/packages/2.9/bioc/html/affyPara.html (accessed: Oct 28, 2011).
19. Sanfilippo S, Noordhuis P. Redis. Version 2.4.2, 2011 (online). Available: http://redis.io/download (accessed: Oct 28, 2011).
20. Lewis BW. (): doRedis: Foreach parallel adapter for the redis package. R package version 1.0.4, 2011 (online). Available: http://CRAN.R-project.org/package=doRedis (accessed: Oct 28, 2011).
21. Wikimedia Foundation.Wikipedia. Embarrassingly parallel, 2012. Available: http://en.wikipedia.org/wiki/Embarrassingly parallel (accessed: Jan 28, 2012).

Bibliography

Alon, U., Barkai, N., Notterman, D., Gish, K., Ybarra, S., Mack, D., and Levine, A. (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissue probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences of the United States of America* **15,** 6745–6750.

Ambroise, C. and McLachlan, G. (2002). Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Accademy of Siences of the United States of America* **99,** 6562–6566.

Ancona, N., Maglietta, R., Piepoli, A., D'Addabbo, A., Cotugno, R., Savino, M., Liuni, S., Carella, M., Pesole, G., and Perri, F. (2006). On the statistical assessment of classifiers using dna microarray data. *BMC Bioinformatics* **7,** 387:400.

Armitage, P. (1989). Inference and decision in clinical trials. *Journal of Clinical Epidemiology* **42,** 293–299.

Baek, S., Tsai, C.-A., and Chen, J. J. (2009). Development of biomarker classifiers from high-dimensional data. *Brief. Bioinform.* **10,** 537–546.

Baggerly, K. A., Coombes, K. R., and Neeley, E. S. (2008). Run batch effects potentially compromise the usefulness of genomic signatures for ovarian cancer. *J. Clin. Oncol.* **26,** 1186–7; author reply 1187–8.

Bender, R., Augustin, T., and Blettner, M. (2005). Generating survival times to simulate Cox proportional hazards models. *Statistics in Medicine* **24,** 1713–1723.

Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society (Series B)* **57,** 289–300.

Berger, J. (1980). *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, NewYork.

Bernau, C., Augustin, T., and Boulesteix, A.-L. (2013). Correcting the optimal resampling-based error rate by estimating the error rate of wrapper algorithms. *Biometrics* **69,** 693–702.

Bernau, C., Boulesteix, A.-L., and Knaus, J. (2013). Application of microarray analysis on computer cluster and cloud platforms. *Methods of Information in Medicine* **52,** 65–71.

Binder, H., Allignol, A., Schumacher, M., and Beyersmann, J. (2009). Boosting for high-dimensional time-to-event data with competing risks. *Bionformatics* **25,** 890–896.

Binder, H. and Schumacher, M. (2008). Allowing for mandatory covariates in boosting estimation of sparse high-dimensional survival models. *BMC Bioinformatics* **9,** 14.

Blair, E. and Tibshirani, R. (2004). Semi-supervised methods to predict patient survival from gene expression data. *PLoS Biology* **2,** 511–522.

Boulesteix, A. L. (2013). On representative and illustrative comparisons with real data in bioinformatics: response to the letter to the editor by smith et al. *Bioinformatics* **29,** 2664–2666.

Boulesteix, A.-L. and Strobl, C. (2009). Optimal classifier selection and negative bias in error rate estimation: An empirical study on high-dimensional prediction. *BMC Medical Research Methodology* **9,** 85.

Castaldi, P. J., Dahabreh, I. J., and Ioannidis, J. P. A. (2011). An empirical assessment of validation practices for molecular classifiers. *Briefings in Bioinformatics* **12,** 189–202.

Chen, W.-C., Ostrouchov, G., Schmidt, D., Patel, P., and Yu, H. (2013). pbdmpi: Programming with big data – interface to mpi, r package. `http://cran.r-project.org/package=pbdMPI`. Accessed: 2013-10-25.

Chiaretti, S., Li, X., Gentleman, R., Vitale, A., Vignetti, M., Mandelli, F., Ritz, J., and Foa, R. (2004). Gene expression profile of adult t-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival. *Blood* **103,** 2771–2778.

Chin, K., DeVries, S., Fridlyand, J., Spellman, P. T., Roydasgupta, R., Kuo, W. L., Lapuk, A., Neve, R. M., Qian, Z., Ryder, T., Chen, F., Feiler, H., Tokuyasu, T., Kingsley, C., Dairkee, S., Meng, Z., Chew, K., Pinkel, D., Jain, A., Ljung, B. M., Esserman, L., Albertson, D. G., Waldman, F. M., and Gray, J. W. (2006). Genomic and transcriptional aberrations linked to breast cancer pathophysiologies. *Cancer Cell* **10,** 529–541.

Coffrey, P., Beliveau, J., Mogre, N., and Harner, A. (2011). Benchmarking the amazon elastic compute cloud(ec2). `http://www.wpi.edu.Pubs/E-project/Available/E-project-030811-115350/unrestricted/AmaznEC2_MQP_Final.pdf`. Accessed: 2011-10-23.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research* **7,** 1–30.

Desmedt, C., Piette, F., Loi, S., Wang, Y., Lallemand, F., Haibe-Kains, B., Viale, G., Delorenzi, M., Zhang, Y., d'Assignies, M. S., Bergh, J., Lidereau, R., Ellis, P., Harris, A. L., Klijn, J. G., Foekens, J. A., Cardoso, F., Piccart, M. J., Buyse, M., Sotiriou, C., and TRANSBIG Consortium (2007). Strong time dependence of the 76-gene prognostic signature for node-negative breast cancer patients in the transbig multicenter independent validation series. *Clin Cancer Res* **13,** 3207–3214.

Dupuy, A. and Simon, R. M. (2007). Critical review of published microarray studies for cancer outcome and guidelines on statistical analysis and reporting. *Journal of the National Cancer Institute* **99,** 147–157.

Efron, B. and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman and Hall, New York.

Evaluation, S. P. (2011). Spec cpu. `http://www.spec.org/benchmarks.html`. Accessed: 2011-10-28.

Evangelinos, C. and Hill, C. (2008). Cloud computing for par- allel scientific hpc applications: Feasibility of run- ning coupled atmosphere-ocean climate models on amazon's ec2. *Proceedings of CCA-08* .

Foekens, J. A., Atkins, D., Zhang, Y., Sweep, F. C., Harbeck, N., Paradiso, A., Cufer, T., Sieuwerts, A. M., Talantov, D., Span, P. N., Tjan-Heijnen, V. C., Zito, A. F., Specht, K., Hoefler, H., Golouh, R., Schittulli, F., Schmitt, M., Beex, L. V., Klijn, J. G., and Wang, Y. (2006). Multicenter validation of a gene Expression–Based prognostic signature in lymph Node–Negative primary breast cancer. *J. Clin. Oncol.* **24,** 1665–1671.

Gautier, L., Cope, L., Bolstad, B., and Irizarry, R. (2004). affy - analysis of affymetrix genechip data at the probe level. *Bioinformatics* **20,** 307–315.

Genz, A., Bretz, F., Miwa, T., Mi, X., Leisch, F., Scheipl, F., and Hothorn, T. (2011). *mvtnorm: Multivariate Normal and t Distributions*. R package version 0.9-96.

Goeman, J. (2010). l_1 penalized estimation in the cox proportional hazards model. *Biometrical Journal* **52,** 70–84.

Golub, T., Slonim, G., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J., Coller, H., Loh, M., Downing, J.R. Caligiuri, M., Bloomfield, C., and Lander, E. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science* **15,** 531–537.

Gönen, M. and Heller, G. (2005). Concordance probability and discriminatory power in proportional hazards regression. *Biometrika* **92,** 965–970.

Haibe-Kains, B., Desmedt, C., Loi, S., Culhane, A. C., Bontempi, G., Quackenbush, J., and Sotiriou, C. (2012). A three-gene model to robustly identify breast cancer molecular subtypes. *J. Natl. Cancer Inst.* **104,** 311–325.

Hanczar, B., Hua, J., and Dougherty, E. R. (2007). Decorrelation of the true and estimated classifier errors in high-dimensional settings. *EURASIP Journal of Bioinformatics and Systems Biology* **2007,** 38473.

Harrell, F. E. et al. (1996). Multivariate prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in Medicine* **15,** 361–387.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning.* Springer-Verlag, New York.

Higham, N. J. (1988). Computing a nearest symmetric positive semidefinite matrix. *Linear Algebra and its Applications* **103,** 103–118.

Huber, W., von Heydebreck, A., Sültmann, H., Poustka, A., and Vingron, M. (2002). Variance stabilization applied to microarray calibration and to the quantification of differential expression. *Bioinformatics* **18,** 96–104.

Irizary, R., Hobbs, B., Collin, F., Beazer-Barclay, Y. D., Antonellis, K. J., Scherf, U., and Speed, T. P. (2003). Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics* **4,** 249–264.

Ishwaran, H., Kogular, U., Blackstone, E., and Lauer, M. S. (2008). Random survival forests. *Annals of Applied Statistics* **2,** 841–860.

Jelizarow, M., Guillemot, V., Tenenhaus, A., Strimmer, K., and Boulesteix, A.-L. (2010). Over-optimism in bioinformatics: an illustration. *Bioinformatics* **26,** 1990–1998.

Kane, M. J. and Emerson, J. W. (2012). *bigmemory: Manage massive matrices with shared memory and memory-mapped files.* R package version 4.3.0.

Kostka, D. and Spang, R. (2008). Microarray based diagnosis profits from better documentation of gene expression signatures. *PLoS Computational Biology* **4,** e22.

Leek, J. T., Scharpf, R. B., Bravo, H. C., Simcha, D., Langmead, B., Evan Johnson, W., Geman, D., Baggerly, K., and Irizarry, R. A. (2010). Tackling the widespread and critical impact of batch effects in high-throughput data. *Nat. Rev. Genet.* **11,** 733–739.

Lewis, B. (2011). doredis: Foreach parallel adapter for the redis package. `http://CRAN.R-project.org/package=doRedis`. Accessed: 2011-10-28.

Mandelbrot, B. (2003). *The fractal geometry of nature.* W.H. Freeman and Company, New York.

McCall, M., Bolstad, B., and Irizarry, R. (2009). Frozen robust multi-array analysis (frma). *Biostatistics* **11,** 242–253.

McCall, M. N., Irizarry, R. A., and with contributions from Terry Therneau (2012). *frma: Frozen RMA and Barcode*. R package version 1.8.0.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2012). *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.6-1.

Micheel, C., Nass, S., Omenn, G., Trials, C., Services, B., Policy, B., and Medicine, I. (2012). *Evolution of Translational Omics: Lessons Learned and the Path Forward*. National Academies Press.

Miller, J. A., Cai, C., Langfelder, P., Geschwind, D. H., Kurian, S. M., Salomon, D. R., and Horvath, S. (2011). Strategies for aggregating gene expression data: the collapserows R function. *BMC Bioinformatics* **12,** 322.

Minn, A. J., Gupta, G. P., Padua, D., Bos, P., Nguyen, D. X., Nuyten, D., Kreike, B., Zhang, Y., Wang, Y., Ishwaran, H., Foekens, J. A., van de Vijver, M., and Massagué, J. (2007). Lung metastasis genes couple breast tumor size and metastatic spread. *Proc Natl Acad Sci U S A* **104,** 6740–6745.

Minn, A. J., Gupta, G. P., Siegel, P. M., Bos, P. D., Shu, W., Giri, D. D., Viale, A., Olshen, A. B., Gerald, W. L., and Massagué, J. (2005). Genes that mediate breast cancer metastasis to lung. *Nature* **436,** 518–524.

Moher, D. and Olkin, I. (1995). Meta-analysis of randomized controlled trials: A concern for standards. *JAMA* **274,** 1962–1964.

Molinaro, A. M., Simon, R., and Pfeiffer, R. M. (2005). Prediction error estimation: a comparison of resampling methods. *Bioinformatics* **21,** 3301–3307.

MPI-Forum (2012). *MPI: A Message-Passing Interface Standard*. High-Performance Computing Center Stuttgart, Stuttgart, version 3.0 edition.

Nadeau, C. and Bengio, Y. (2003). Inference for the generalization error. *Machine Learning* **52,** 239–281.

Porzelius, C., Schumacher, M., and Binder, H. (2011). The benefit of data-based model complexity selection via prediction error curves in time-to-event data. *Computational Statistics* **26,** 293–302.

Revolution-Analytics (2013). *doSNOW: Foreach parallel adaptor for the snow package*. R package version 1.0.7.

Sanfilippo, S. and Noordhuis, P. (2012). Redis. version 2.4.2. `http://redis.io/download`. Accessed: 2012-1-25.

Schmidberger, M., Vicedo, E., and Mansmann, U. (2011). *affyPara: Parallelized preprocessing methods for Affymetrix Oligonucleotide Arrays*. R package version 1.16.0.

Schmidt, M., Böhm, D., von Törne, C., Steiner, E., Puhl, A., Pilch, H., Lehr, H. A., Hengstler, J. G., Kölbl, H., and Gehrmann, M. (2008). The humoral immune system has a key prognostic impact in node-negative breast cancer. *Cancer Res* **68,** 5405–5413.

Simon, N., Friedman, J., Hastie, T., and Tibshirani, R. (2011). Regularization paths for cox's proportional hazards model via coordinate descent. *Journal of Statistical Software* **39,** 1–13.

Simon, R. M., Paik, S., and Hayes, D. F. (2009). Use of archived specimens in evaluation of prognostic and predictive biomarkers. *J. Natl. Cancer Inst.* **101,** 1446–1452.

Simon, R. M., Subramanian, J., Li, M.-C., and Menezes, S. (2011). Using cross-validation to evaluate predictive accuracy of survival risk classifiers based on high-dimensional data. *Briefings in Bioinformatics* **12,** 203–217.

Singh, D., Febbo, P., Ross, K., Jackson, D., Manola, J., Ladd, C., Tamayo, P., Renshaw, A., D'Amico, A., Richie, J., Lander, E., Loda, M., Kantoff, P., Golub, T., and Sellers, W. (2002). Gene expression correlates of clinical prostate cancer behavior. *Cancer Cell* **1,** 203–209.

Slawski, M., Boulesteix, A.-L., and Bernau., C. (2009). *CMA: Synthesis of microarray-based classification*. R package version 1.5.5.

Sotiriou, C., Wirapati, P., Loi, S., Harris, A., Fox, S., Smeds, J., Nordgren, H., Farmer, P., Praz, V., Haibe-Kains, B., Desmedt, C., Larsimont, D., Cardoso, F., Peterse, H., Nuyten, D., Buyse, M., Van de Vijver, M. J., Bergh, J., Piccart, M., and Delorenzi, M. (2006). Gene expression profiling in breast cancer: understanding the molecular basis of histologic grade to improve prognosis. *J Natl Cancer Inst* **98,** 262–272.

Subramanian, J. and Simon, R. (2010). Gene expression-based prognostic signatures in lung cancer: ready for clinical use? *J. Natl. Cancer Inst.* **102,** 464–474.

Symmans, W. F., Hatzis, C., Sotiriou, C., Andre, F., Peintinger, F., Regitnig, P., Daxenbichler, G., Desmedt, C., Domont, J., Marth, C., Delaloge, S., Bauernhofer, T., Valero, V., Booser, D. J., Hortobagyi, G. N., and Pusztai, L. (2010). Genomic index of sensitivity to endocrine therapy for breast cancer. *J Clin Oncol* **28,** 4111–4119.

Systems, C. (2012). Xen. Accessed: 2012-1-25.

Tibshirani, R. (2009). *uniCox: Univarate shrinkage prediction in the Cox model*. R package version 1.0.

Tibshirani, R. J. and Tibshirani, R. (2009). A bias correction for the minimum error rate in cross-validation. *Annals of Applied Statistics* **3,** 822–829.

Uno, H. and Cai, T. (2012). *survIDINRI: IDI and NRI for comparing competing risk prediction models with censored survival data*. R package version 1.0-1.

Uno, H., Cai, T., Pencina, M., D'Agostino, R., and Wei, L. (2011). On the c-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. *Statistics in Medicine* **30,** 1105–1017.

van Wieringen, W., Kun, D., Hampel, R., and Boulesteix, A.-L. (2009). Survival prediction using gene expression data: a review and comparison. *Computational Statistics & Data Analysis* **53,** 1590–1603.

Varma, S. and Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics* **7,** 91.

Verweij, P. and Houweilingen, H. (1993). Cross-validation in survival analysis. *Statistics in Medicine* **12,** 2305–2314.

Waldron, L., Haibe-Kains, B., Culhane, A., Riester, M., Ding, J., Wang, V., Tyekucheva, S., Bernau, C., Risch, T., Ganzfried, B., Huttenhower, C., Birrer, M., and Parmigiani, G. (2013). A comparative meta-analysis of prognostic gene signatures for late-stage ovarian cancer. *Journal of the National Cancer Institute – accepted* .

Yu, H. (2011). Rmpi: Interface (wrapper) to mpi (message-passing interface). `http://CRAN.R-project.org/package=Rmpi`. Accessed: 2011-10-28.

Zhao, S., Huttenhower, G. P. C., and Waldron, L. (2013). Mas-o-menos: a simple sign averaging method for discrimination in genomic data analysis. http://biostats.bepress.com/harvardbiostat/paper158/. Accessed: 2013-10-24.

Acknowledgments

I owe thanks to many people who supported me during this thesis. First of all, I would like to thank my supervisor Anne-Laure Boulesteix for giving me the opportunity to work in her group at the IBE and providing helpful suggestions and ideas in numerous discussions. Furthermore, I am thankful for her contributions to the different papers this thesis is based upon. In particular, I also want to thank her for letting me the freedom to pursue own projects in cooperation with colleagues from other institutes and for initiating the KONWIHR project with the Leibniz Supercomputing Center.

In the context of the first part of the thesis, I am grateful to Thomas Augustin who helped to formulate the decision theoretic framework of the WMC estimator and provided valuable contributions to the corresponding paper.

The second part about the ranking of wrapper algorithms in the framework of cross-study validation would not have been possible without the contributions by Levi Waldron and Lorenzo Trippa from the Harvard Medical School in Boston. Our weekly online-meetings and their various suggestions for improvements were of great value for the resulting paper. Moreover, their edits to the manuscript helped to improve the paper a lot. Finally, I would like to thank Markus Riester, Curtis Huttenhower, Anne-Laure Boulesteix and Giovanni Parmigiani for their helpful suggestions and comments on the manuscript.

Also for the last part of the thesis, I owe thanks to several colleagues. I would like to thank Jochen Knaus for his scripts for creating parallel environments on Amazon EC2 as well as his contributions to the paper resulting from the work in this cloud environment. With regard to the sections on parallel computing and the processing of large datasets with **R**, I am grateful to Ferdinand Jamitzky who had many suggestions for the implementation and supervised my work at the Leibniz Supercomputing Center including the two KONWIHR projects. I would also like to give thanks to Markus Schmidberger who introduced me to parallel approaches in **R**. In the context of the development of the **R**-package *survHD*, my thank goes to Levi Waldron and Markus Riester from the Harvard Medical School and Dana Faber Cancer Institute. They contributed several functions and helped with the design of the package and its structure. Moreover, I would like to thank Levi Waldron for integrating me into some of his projects which also helped to grasp the bigger picture of the analysis of microarrays and other molecular data.

Additionally, I would like to express my gratitude to the proofreaders of my thesis and the manuscripts of the papers it is based upon: Ferdinand Jamitzky, Krisztian Borbely, Jochen Knaus, Thomas Augustin, Markus Riester, Levi Waldron, Giovanni Parmigiani,

Curtis Huttenhower, Doris Lindörfer and Roman Hornung.

I would also like to thank my colleagues at the IBE and the computational molecular medicine group as well as my colleagues at the LRZ, who supported me during the progress of this thesis.

Finally, I would like to give thanks to the reviewers of my thesis Anne-Laure Boulesteix and Harald Binder, as well as the remaining members of the reviewer committee Thomas Augustin and Christian Heumann.

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Name, Vorname

Ort, Datum

Unterschrift Doktorand/in

Formular 3.2