Martin Slawski, Martin Daumer, Anne-Laure Boulesteix

CMA - A comprehensive Bioconductor package for supervised classification with high dimensional data

Technical Report Number 029, 2008
Department of Statistics
University of Munich

http://www.stat.uni-muenchen.de

CMA - A comprehensive Bioconductor package for supervised classification with high dimensional data

M. Slawski[*], M. Daumer[*], A.-L. Boulesteix[*†]

May 30, 2008

Abstract

For the last eight years, microarray-based class prediction has been a major topic in statistics, bioinformatics and biomedicine research. Traditional methods often yield unsatisfactory results or may even be inapplicable in the $p \gg n$ setting where the number of predictors by far exceeds the number of observations, hence the term "ill-posed-problem". Careful model selection and evaluation satisfying accepted good-practice standards is a very complex task for inexperienced users with limited statistical background or for statisticians without experience in this area. The multiplicity of available methods for class prediction based on high-dimensional data is an additional practical challenge for inexperienced researchers.

In this article, we introduce a new Bioconductor package called CMA (standing for "**C**lassification for **M**icro**A**rrays") for automatically performing variable selection, parameter tuning, classifier construction, and unbiased evaluation of the constructed classifiers using a large number of usual methods. Without much time and effort, users are provided with an overview of the unbiased accuracy of most top-performing classifiers. Furthermore, the standardized evaluation framework underlying CMA can also

[*]Sylvia Lawry Centre for MS Research, Hohenlindenerstr. 1, D-81677 Munich, Germany
[†]Department of Statistics, University of Munich, Ludwigstr. 33, D-80539 Munich, Germany

be beneficial in statistical research for comparison purposes, for instance if a new classifier has to be compared to existing approaches.

CMA is a user-friendly comprehensive package for classifier construction and evaluation implementing most usual approaches. It is freely available from the Bioconductor website at `http://bioconductor.org/packages/2.3/bioc/html/CMA.html`.

1 Background

For the last eight years, microarray-based class prediction has been a major topic in statistics, bioinformatics and biomedicine research. Traditional methods often yield unsatisfactory results or may even be inapplicable in the $p \gg n$ setting where the number of predictors by far exceeds the number of observations, hence the term "ill-posed-problem". Microarray studies have thus stimulated the development of new approaches and motivated the adaptation of known traditional methods to the high-dimensional setting. Most of them are implemented in the R language (1) and are freely available at `cran.r-project.org` or from the bioinformatics platform `www.bioconductor.org`. Meanwhile, the latter has established itself as a standard tool for analyzing various types of high-throughput genomic data including microarray data (2). Throughout this article, the focus is on microarray data, but the presented package can be applied to any supervised classification problem involving a large number of continuous predictors such as, e.g. proteomic or metabolomic data.

Model selection and evaluation of prediction rules turn out to be highly difficult in the $p \gg n$ setting for several reasons: i) the hazard of overfitting, which is common to all prediction problems, is considerably increased by high dimensionality, ii) the usual evaluation scheme based on the splitting into learning and test data sets often applies only partially in the case of small samples, iii) modern classification techniques rely on the proper choice of hyperparameters whose optimization is highly computer-intensive, especially with high-dimensional data.

The multiplicity of available methods for class prediction based on high-dimensional data is an additional practical challenge for inexperienced statisticians. Whereas logistic regression is well-established as the standard method to be used when analyzing classical data sets with much more observations than variables ($n > p$), it is still unclear which one of the many available approaches for high-dimensional should be used as a reference standard method in the $n \ll p$ case. Moreover, the programs implementing well-known popular methods such as penalized logistic regression, nearest shrunken centroids (3), random forests (4), or partial least squares (5) are characterized by a high heterogeneity as far as input format, output format, and tuning procedures are concerned. Inexperienced users have thus to spend much effort understanding each of the programs and modifying the data formats, while potentially introducing severe errors which may considerably affect the final results. Furthermore, the users may overlook important tuning parameters or detail settings that sometimes noticeably contribute to the success of the classifier. Note that circumventing the problem of the multiplicity of methods by always using a single "favorite method" (usually the method in the user's expertise area or a method which has been identified as top-performing method in a seminal comparison study) potentially leads to poor results, especially when the considered method involves strong assumptions on the data structure.

From the difficulties outlined above, we conclude that careful model selection and evaluation satisfying accepted good-practice standards (6) is a very complex task for inexperienced users with limited statistical background. In this article, we introduce a new Bioconductor package called CMA (standing for "**C**lassification for **M**icro**A**rrays") for automatically performing variable selection, parameter tuning, classifier construction, and unbiased evaluation of the constructed classifiers. The primary goal of CMA is to enable statisticians with limited experience on high-dimensional class prediction or biologists and bioinformaticians with statistical background to achieve such a demanding task on their own. Without much time and effort, users are provided with an overview of the unbiased accuracy of most top-performing classifiers. Furthermore, the standardized evaluation framework underlying CMA involving variable selection and hyperparameter tuning can also be beneficial for comparison purposes, for instance if a new classifier has to be compared to existing approaches.

In a nutshell, CMA offers an interface to a total of more than twenty different classifiers, seven univariate and multivariate variable selection methods, different evaluation schemes (such as, e.g. cross-validation or bootstrap), and different measures of classification accuracy. A particular attention is devoted to preliminary variable selection and hyperparameter tuning, issues that are often neglected in current literature and software. More specifically, variable selection is always performed using the training data only, i.e. for each iteration successively in the case of cross-validation, following well-established good-practice guidelines (7; 8; 9; 6). Hyperparameter tuning is performed through an inner cross-validation loop, as usually recommended (10). This feature is intended to prevent users from trying several hyperparameter values on their own and selecting the best results a posteriori, a strategy which would obviously lead to severe bias (11).

The CMA package is freely available from the Bioconductor website at `http://bioconductor.org/packages/2.3/bioc/html/CMA.html`

Overview of existing packages

The idea of an R interface for the integration of microarray-based classification methods is not new. The CMA package shows similarities to the Bioconductor package 'MLInterfaces' standing for "An interface to various machine learning methods" (12), see also the Bioconductor textbook (13) for a presentation of an older version. The MLInterfaces package includes numerous facilities such as the unified `MLearn` interface, the flexible `learnerSchema` design enabling the introduction of new procedures on the fly, and the `xvalSpec` interface that allows arbitrary types of resampling and cross-validation to be employed. `MLearn` also returns the native R object from the learner for further interrogation. The package architecture of MLInterfaces is similar the CMA structure in the sense that wrapper functions are used to call classification methods from other packages.

However, CMA includes additional predefined features as far as variable selection, hyperparameter tuning, classifier evaluation and comparison are concerned. While the method `xval` is flexible for experienced users, it provides only cross-validation (including leave-one-out) as predefined option. As the CMA package also addresses inexperienced users, it includes the most common validation schemes in a standardized manner. In the current version of MLInterfaces, variable selection can also be carried out separately for each different learning set, but it does not seem to be a standard procedure. In the examples presented in the book mentioned above, variable selection is only performed once using the complete sample. In contrast, CMA performs variable selection separately for each learning set by default. Further, CMA includes additional features for hyperparameter tuning, thus allowing an objective comparison of different class prediction methods. If tuning is ignored, simpler methods without (or with few) tuning parameters tend to perform seemingly better than more complex algorithms. CMA also implements additional measures of prediction accuracy and user-friendly visualization tools.

The package 'MCRestimate' (14) emphasizes very similar aspects as CMA, focussing on the estimation of misclassification rates and cross-validation for model selection and evaluation. It is (to our knowledge) the only Biconductor package supporting hyperparameter tuning, but obviously referring to the function `e1071:::tune`. The CMA package includes additional variable selection features and is implemented in the S4 class structure.

Overview of class prediction with high-dimensional data and notations

Settings and Notation

The classification problem can be briefly outlined as follows:

- we have a predictor space $\mathcal{X}$, here $\mathcal{X} \subseteq \mathbb{R}^p$ (for instance, the predictors may be gene expresssion levels, but the scope of CMA is not limited to this case),
- we have a finite set of class labels $\mathcal{Y} = \{0, \dots, K-1\}$, with K denoting the total number of classes,
- $P(\mathbf{x}, y)$ denotes the joint probability distribution on $\mathcal{X} \times \mathcal{Y}$,
- we are given a finite sample $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ of n predictor-class pairs.

The considered task is to construct a decision function

$$\begin{array}{rccl} \widehat{f}: & \mathcal{X} & \rightarrow & \mathcal{Y} \\ & \mathbf{x} & \mapsto & \widehat{f}(\mathbf{x}) \end{array}$$

such that the *generalization error*

$$R[f] = \mathbf{E}_P[L(\widehat{f}(\mathbf{x}), y)] = \int_{\mathcal{X}\times\mathcal{Y}} L(y, \widehat{f}(\mathbf{x}))\, dP(\mathbf{x}, y) \qquad (1)$$

is minimized, where $L(\cdot,\cdot)$ is a suitable loss function, usually taken to be the indicator loss ($L(u, v) = 1$ if $u \neq v$, $L(u, v) = 0$ otherwise). Other loss functions and performances measures are discussed extensively in section 3.1.4. The symbol $\widehat{}$ indicates that the function is estimated from the given sample S.

Estimation of the generalization error

As we are only equipped with a finite sample S and the underlying distribution is unknown, approximations to Eq. (1) have to be found. The empirical counterpart to $R[f]$

$$R_{\text{emp}}[f] = n^{-1} \sum_{i=1}^{n} L(y_i, \widehat{f}(\mathbf{x}_i)) \qquad (2)$$

has a (usually large) negative bias, i.e. prediction error is underestimated. Moreover, model selection based on Eq. (2) leads to overfitting the sample S. More details can be found in recent overview articles (15; 16; 17). A better strategy consists of splitting S into distinct subsets $\mathcal{L}$ (learning sample) and $\mathcal{T}$ (test sample) with the intention to separate model selection and model evaluation. The classifier $\widehat{f}(\cdot)$ is constructed using $\mathcal{L}$ only and evaluated using $\mathcal{T}$ only, as depicted in Figure 1.

In microarray data, the sample size n is usually very small, leading to serious problems for both the construction of the classifier and the estimation of its prediction accuracy. Increasing the size of the learning set ($n_{\mathcal{L}} \rightarrow n$) typically improves the constructed prediction rule $\widehat{f}(\cdot)$, but decreases the reliability of its evaluation. Conversely, inccreasing the size of the test set ($n_{\mathcal{T}} \rightarrow n$) improves the accuracy estimation but leads to poor classifiers, since these are based on fewer observations. While a compromise can be found for a reasonable sample size, alternative designs are needed for the case of small sizes. The CMA package implements several approaches which are all based on the following scheme.

1. Generate B learning sets $\mathcal{L}_b$ $(b = 1, \ldots, B)$ from S and define the corresponding test set as $\mathcal{T}_b = S \setminus \mathcal{L}_b$.

2. Obtain $\widehat{f}_b(\cdot)$ from $\mathcal{L}_b$, for $b = 1, \ldots, B$.

3. The quantity

$$\widehat{\epsilon} = \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathcal{T}_b|} \sum_{i \in \mathcal{T}_b} L(y_i, \widehat{f}_b(\mathbf{x}_i)) \tag{3}$$

   is then used as an estimator of the error rate, where $|.|$ stands for the cardinality of the considered set.

The underlying idea is to reduce the variance of the error estimator by averaging, in the spirit of the bagging principle introduced by Breiman (18). The function `GenerateLearningsets` from the package CMA implements several methods for generating $\mathcal{L}_b$ and $\mathcal{T}_b$ in step 1, which are described below.

`LOOCV` **Leaving-one-out cross-validation**

For the b-th iteration, $\mathcal{T}_b$ consists of the b-th observation only. This is repeated for each observation in S, so that $B = n$.

`CV` ***k*-fold cross-validation** (`method = "CV", fold, niter`)

S is split into `fold` non-overlapping subsets of approximately equal size. For each iteration b, the b-th subset is used as $\mathcal{T}_b$ and the union of the remaining subsets as $\mathcal{L}_b$, such that $B =$`fold`. Setting `fold = n` is equivalent to `method = "LOOCV"`. For `fold` $< n$, the splitting is not uniquely determined. It is thus recommended to repeat the whole procedure `niter` times (15) (for instance `niter`=5 or `niter`=10) to partly average out random variations.

`MCCV` **Monte-Carlo-cross-validation** (`method = "MCCV", fold, ntrain, niter`)

Each of the B=`niter` learning sets of cardinality `ntrain` is drawn randomly from S without replacement.

`boot` **Bootstrap** (`method = "bootstrap", ntrain, niter`)

B=`niter` bootstrap samples (drawn with replacement) (19) of cardinality `ntrain` are used as learning sets. In practice, `ntrain` is usually set to the total sample size n.

See Figure 2 for a schematic representation of CV, MCCV and bootstrap sampling. *stratified sampling* is possible by setting `strat = TRUE`. This implies that, in each learning set $\mathcal{L}_b$, the proportion of the classes $\{0, \dots, K-1\}$ is approximately the same as in S. This option is very useful (and sometimes even necessary) in order to guarantee that each class is sufficiently represented in each $\mathcal{L}_b$, in particular if there are classes of small size. A schematic display of above splitting rules is given below. For more details on the evaluation of classifiers, readers may refer to recent overview articles discussing the respective drawbacks and advantages of these methods in (15; 16).

In CMA, cross-validation is also used for hyperparameter tuning. The optimal value(s) of the method parameter(s) is(are) determined within an inner cross-validation, as commonly recommended (10; 11). If cross-validation is used for both tuning parameters and evaluating a classifiers, the whole procedure is denoted as nested cross-validation. See Figure 3 for a schematic representation.

2 Implementation

The Bioconductor package CMA is user-friendly in the sense that

- the methods automatically adapt to the data format provided by the user,
- convenience functions take over frequent tasks such as automatic visualization of results,
- reasonable default settings for hyperparameter tuning and other parameters requiring expert knowledge of particular classifiers are provided,
- it works with uniform data structures.

To do so, CMA exploits the rich possibilities of object-oriented programming as realized by S4 classes of the methods package (20) which make it easy to incorporate new features into an existing framework. For instance, with some basic knowledge of the S4 class system (which is standard for bioconductor packages), users can easily embed new classification methods in addition to the 21 currently available in CMA. Moreover, the process of classifier building described in more detail in section 3.1.2 can either be partitioned into several transparent small steps (variable selection, hyperparameter tuning, etc) or executed by only one compact function call. The last feature is beneficial for users who are not very familiar with R commands.

3 Results

3.1 CMA features

3.1.1 Overview

In a nutshell, the package has the following features.

- It offers a uniform, user-friendly interface to a total of more than twenty classification methods (see Table 1) comprising classical approaches as well as more sophisticated methods. User-friendliness means that the input formats are uniform among different methods, that the user may choose between three different input formats and that the output is self-explicable and informative.
- It automatically generates learning samples as explained in section 1, including the generation of stratified samples. Different schemes for generating learning sets and test sets are displayed schematically in Figure 2.
- The method `GeneSelection` provides optional variable selection preceding classification for each iteration $b = 1, \ldots, B$ separately, based on various ranking procedures.
- The method `tune` carries out hyperparameter tuning for a *fixed* (sub-)set of variables. It can be performed in a fully automatic manner using pre-defined grids, see Figure 3 for a schematic representation. Alternatively, it can be completely customized by the experienced user. The principle of the tuning procedure based on nested cross-validation is schematically represented in Figure 3.
- The method `classification` enables the user to combine gene selection, hyperparameter tuning and class prediction into one single step.

- Performance can be assessed using the method `evaluation` for several performance measures commonly used in practice.
- Comparison of the performance of several classifiers can be quickly tabulated and visualized using the method `comparison`.
- Estimations of conditional class probabilities for predicted observations are provided by most of the classifiers, with only a few exceptions. This is more informative than only returning class labels and allows a more precise comparison of different classifiers.
- Most results can conveniently be summarized and visualized using predefined convenience methods as demonstrated in section 3.2, for example:
  - `plot,cloutput-method` produces probability plots, also known as “voting plot”,
  - `plot,genesel-method` visualizes variable importance derived from one of the ranking procedures via a barplot,
  - `roc,cloutput-method` draws empirical ROC curves,
  - `toplist,genesel-method` lists the most relevant variables,
  - `summary,evaloutput-method` makes a summary out of iteration- or observationwise performance measures.
- The implementation is fully organized in S4 classes, thus making the extension of CMA very easy. In particular, own classification methods can easily be integrated if they return a proper object of class `cloutput`.
- In addition to the packages listed in Table 1, CMA only requires the package 'limma' for full functionality. For all other features, no code of foreign packages is used.
- Like most R packages, CMA is more flexible than, e.g., web-based tools. Experienced used can easily modify the programs or add new methods.

3.1.2 Classification methods

This subsection gives a brief summarizing overview of the classifiers implemented in CMA. We have tried to compose a balanced mixture of methods from several fields although we do not claim our selection to be representative, taking into account the large amount of literature on that subject. For more detailed information on the single methods, readers are referred to the references given in Table 1 and the references therein. All classifiers can be constructed using the CMA method `classification`, where the argument `classifier` specifies the classification method to be used.

Discriminant Analysis

Discriminant analysis is the (Bayes-)optimal classifier if the conditional distributions of the predictors given the classes are Gaussian. Diagonal, linear and quadratic discriminant analysis differ only by their assumptions for the (conditional) covariance matrices $\mathbf{\Sigma}_k = \mathrm{Cov}(\mathbf{x}|y=k),\ k = 0, \dots, K-1$.

(a) Diagonal discriminant analysis (`classifier="dldaCMA"`) assumes that the within-class covariance matrices $\mathbf{\Sigma}_k$ are diagonal and equal for all classes, i.e. $\mathbf{\Sigma}_k = \mathbf{\Sigma} = \mathrm{diag}(\sigma_1^2, \dots, \sigma_p^2),\ k = 1, \dots, K-1$, thus requiring the estimation of p covariance parameters.

(b) Linear discriminant analysis (`classifier="ldaCMA"`) assumes $\mathbf{\Sigma}_k = \mathbf{\Sigma},\ k = 1, \dots, K-1$ without further restrictions for $\mathbf{\Sigma}$ so that $p(p+1)/2$ parameters have to be estimated.

(c) Quadratic discriminant analysis (`classifier="qdaCMA"`) does not impose any restriction on $\mathbf{\Sigma}_k, k = 1, \dots, K-1$.

While (a) turns out to be still practicable for microarray data, linear and quadratic discriminant analysis are not competitive in this setting, at least not without dimension reduction or excessive variable selection (see below).

The so-called PAM method (standing for "Prediction Analysis for Microarrays"), which is also commonly denoted as "shrunken centroids discriminant analysis" can be viewed as a modification of diagonal discriminant analysis (also referred to as "naive Bayes" classifier) using univariate soft thresholding (21) to perform variable selection and yield stabilized estimates of the variance parameters (`classifier="scdaCMA"`).

Fisher's discriminant analysis (FDA) (`classifier="fdaCMA"`) has a different motivation, but can be shown to be equivalent to linear discriminant analysis under certain assumptions. It looks for projections $\mathbf{a}^{\mathbf{T}}\mathbf{x}$ such that the ratio of between-class and within-class variance is maximized, leading to a linear decision function in a lower dimensional space. Flexible discriminant analysis (`classifier="flexdaCMA"`) can be interpreted as FDA in a higher-dimensional space generated by basis functions, also allowing non-linear decision functions (22). In CMA, the basis functions are given by penalized splines as implemented in the R package 'mgcv' (23).
Shrinkage discriminant analysis (24; 25) (`classifier="shrinkldaCMA"`) tries to stabilize covariance estimation by shrinking the unrestricted covariance matrix from linear discriminant analysis to a more simply structured target covariance matrix, e.g. a diagonal matrix.

Partial Least Squares
Partial Least Squares is a dimension reduction method that looks for directions $\{\mathbf{w}_r\}_{r=1}^{R}$ maximizing $|\mathrm{Cov}(y, \mathbf{w}_r^{\mathbf{T}}\mathbf{x})|$ $(r = 1, \ldots, R)$ subject to the constraints $\mathbf{w}_r^{\mathbf{T}}\mathbf{w}_r = 1$ and $\mathbf{w}_r^{\mathbf{T}}\mathbf{w}_s = 0$ for $r \neq s$, where $R \ll p$. Instead of working with the original predictors, one then plugs the projections living in a lower dimensional space into other classification methods, for example linear discriminant analysis (`classifier="pls_ldaCMA"`), logistic regression (`classifier="pls_lrCMA"`) or random forest (`classifier="pls_rfCMA"`). See Boulesteix and Strimmer (9) for an overview of partial least squares applications to genomic data analysis.

Regularization and shrinkage methods
In both penalized logistic regression and support vector machines, $\widehat{f}(\cdot)$ is constructed such that it minimizes an expression of the form

$$\sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + \lambda J[f], \tag{4}$$

where $L(\cdot, \cdot)$ is a loss function as outlined above and $J[f]$ is a regularizer preventing overfitting. The trade-off between the two terms is known as bias-variance trade-off and governed via the tuning parameter λ. For ℓ^2 penalized logistic regression (`classifier="plrCMA"`), $f(\mathbf{x}) = \mathbf{x}^{\mathbf{T}}\beta$ is linear and depends only on the vector β of regression coefficients, $J[f]$ is the ℓ_2 norm $J[f] = \beta^{\mathbf{T}}\beta$ and $L(\cdot,\cdot)$ is the negative log-likelihood of a binomial distribution. Setting $J[f] = |\beta| = \sum_{j=1}^{p} |\beta_j|$ yields the Lasso (26) (`classifier="LassoCMA"`), while combining both regularizers into $J[f] = \beta^{\mathbf{T}}\beta + |\beta|$ yields the elastic net (27) (`classifier="ElasticNetCMA"`). CMA also implements a multi-class version of ℓ^2 penalized logistic regression, replacing the binomial negative likelihood by its multinomial counterpart.

For Support Vector Machines (`classifier="svmCMA"`), we have

$$f(\mathbf{x}) = \sum_{i \in \mathcal{V}} \alpha_i k(\mathbf{x}, \mathbf{x}_i),$$

where $\mathcal{V} \subset \{1, \ldots, n\}$ is the set of the so-called "support vectors", α_i are coefficients and $k(\cdot,\cdot)$ is a positive definite kernel. Frequently used kernels are the linear kernel $\langle \cdot,\cdot \rangle$, the polynomial kernel $\langle \cdot,\cdot \rangle^d$ or the Gaussian kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp((\mathbf{x}_i - \mathbf{x}_j)^{\mathbf{T}}(\mathbf{x}_i - \mathbf{x}_j)/\sigma^2)$. The function $J[f]$ is given as $J[f] = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$ and $L(\cdot,\cdot)$ is the so-called hinge loss (28).

Random Forests

The random forest method (4) aggregates an ensemble of binary decision-tree classifiers (29) constructed based on bootstrap samples drawn from the learning set (`classifier="rfCMA"`). The "**b**ootstrap **agg**regat**ing**" strategy (abbreviated as "bagging") turns out to be particularly successful in combination with unstable classifiers such as decision trees. In order to make the obtained trees even more different and thus increase their stability and to reduce the computation time, random forests have an additional feature. At each split, a subset of candidate predictors is selected out of the available predictors. Random forest also performs implicit variable selection and can be used to assess variable importance (see section 3.1.3).

Boosting

Similarly to random forests, boosting is based on a weighted ensemble of "weak learners" for classification, i.e. $f(\cdot) = \sum \alpha_m f_{\text{weak}}(\cdot)$, where

$\alpha_m > 0$ $(m = 1, \ldots, M)$ are adequately chosen coefficients. The term weak learner which stems from the machine learning community (30), denotes a method with poor performance (but still significantly better performance than random guessing) and low complexity. Famous examples for weak learners are binary decision trees with few (one or two) splits or linear functions in one predictor which is termed componentwise boosting. Friedman (31) reformulates boosting as a functional gradient descent combined with appropriate loss functions. The CMA package implements decision tree-based (`classifier="gbmCMA"`) and componentwise (`classifier="compBoostCMA"`) boosting with exponential, binomial and squared loss in the two-class case, and multinomial loss in the multi-class case.

Feed-Forward Neural Networks

CMA implements one-hidden-layer feed-forward neural networks (`classifier="nnetCMA"`). Starting with a vector of covariates $\mathbf{x}$, one forms projections $\mathbf{a}_r^{\mathbf{T}}\mathbf{x}$, $r = 1, \ldots, R$, that are then transformed using an activation function $h(\cdot)$, usually sigmoidal, in order to obtain a hidden layer consisting of units $\{z_r = h(\mathbf{a}_r^{\mathbf{T}}\mathbf{x})\}_{r=1}^{R}$ that are subsequently used for prediction. Training of neural networks tends to be rather complicated and unstable. For large p, CMA works in the space of "eigengenes", following the suggestion of (32) by applying the singular value decomposition (33) to the predictor matrix.

Probabilistic Neural Networks

Although termed "Neural Networks", probabilistic neural networks (`classifier="pnnCMA"`) are actually a Parzen-Windows type classifier (34) related to the nearest neighbors approach. For $\mathbf{x} \in \mathcal{T}$ from the test set and each class $k = 0, \ldots, K-1$, one computes

$$w_k = n_k^{-1} \sum_{\mathbf{x}_i \in \mathcal{L}} I(y_i = k) \cdot \exp((\mathbf{x}_i - \mathbf{x})^{\mathbf{T}}(\mathbf{x}_i - \mathbf{x})/\sigma^2), \; k = 0, \ldots, K-1$$

where n_k denotes the number of observations from class k in the learning set and $\sigma^2 > 0$ is a method parameter. The quotient $\{w_k / \sum_{k=0}^{K-1} w_k\}_{k=0}^{K-1}$ is then considered as an estimate of the class probability, for $k = 0, \ldots, K-1$.

Nearest Neighbors and Probabilistic Nearest Neighbors

CMA implements one of the variants of the ordinary nearest neighbors approach using the euclidean distance as distance measure (`classifier="knnCMA"`) and another variant called "probabilistic" that additionally provides estimates for class probabilities by using distances as weights, however without a genuine underlying probability model (`classifier="pknnCMA"`). Given a learning set $\mathcal{L}$ and test set $\mathcal{T}$, respectively, the probabilistic nearest neighbors method determines for each element in $\mathcal{L}$ the $k > 1$ nearest neighbors $\mathcal{N} \subset \mathcal{L}$ and then estimates class probabilities as

$$P(y = k|\mathbf{x}) = \frac{\exp\left(\beta \sum_{\mathbf{x}_i \in \mathcal{N}} -d(\mathbf{x}, \mathbf{x}_i) I(y_i = k)\right)}{\exp\left(1 + \beta \sum_{x_i \in \mathcal{N}} -d(\mathbf{x}, \mathbf{x}_i) I(y_i = k)\right)}, \; k = 0, \dots, K-1, \; \mathbf{x} \in \mathcal{T}$$

where $\beta > 0$ is a method parameter and $d(\cdot, \cdot)$ a distance measure.

3.1.3 Variable selection methods

This section addresses the variable ranking- and selection procedures available in CMA. We distinguish three types of methods: pure filter methods (f) based on parametric or nonparametric statistical tests not directly related to the prediction task, methods which rank variables according to their discriminatory power (r), and classification methods selecting sparse sets of variables that can be used for other classification methods in a hybrid way (s). The multi-class case is fully supported by all the methods. Methods that are defined for binary responses only are applied within a "one-vs-all" or "pairwise" scheme. The former means that for each class $k = 0, \dots, K-1$, one recodes the class label y into $K-1$ pseudo class labels $\widetilde{y}_k = I(y = k)$ for $k = 1, \dots, K-1$, while the latter considers all $\binom{K}{2}$ possible pairs of classes successively. The variable selection procedure is run $K-1$ times or $\binom{K}{2}$ times, respectively, and the same number of genes are selected for each run. The final subset of selected genes consists of the union of the subsets obtained in the different runs.

In the CMA package, variable selection can be performed (for each learning set separately) using the method `geneselection`, with the argument `method` specifying the procedure and the argument `scheme` indicating which scheme (one-vs-all or pairwise) should be used in the $K > 2$ case. The implemented methods are:

(f) ordinary two-sample t.test (`method = "t.test"`)

(f) Welch modification of the t.test (`method = "welch.test"`)

(f) Wilcoxon rank sum test (`method = "wilcox.test"`)

(f) F test (`method = "f.test"`)

(f) Kruskal-Wallis test (`method = "kruskal.test"`)

(f) "moderated" t and F test, respectively, using the package 'limma' (35) (`method = "limma"`)

(r) one-step Recursive Feature Elimination (RFE) in combination with the linear SVM (36) (`method = "rfe"`)

(r) random forest variable importance measure (4) (`method = "rf"`)

(s) Lasso (26) (`method = "lasso"`)

(s) elastic net (27) (`method = "elasticnet"`)

(s) componentwise boosting (`method = "boosting"`) (37)

(f) ad-hoc "Golub" criterion (38)

Each method can be interpreted as a function $\mathcal{I}(\cdot)$ on the set of predictor indices: $\mathcal{I} : \{1, \ldots, p\} \rightarrow \mathbb{R}^+$ where $\mathcal{I}(\cdot)$ increases with discriminating power. $\mathcal{I}(\cdot)$ is the absolute value of the test statistic for the (f) methods and the absolute value of the corresponding regression coefficient for the (s)-methods, while the (r)-methods are already variable importance measures per definition. Predictor j is said to be more important than predictor l if $\mathcal{I}(l) < \mathcal{I}(j)$. It should be noted that the variable ordering is not necessarily determined uniquely, especially for the (s)-methods where variable importances are different from zero for few predictors only and for the (f) methods based on ranks. Variable selection is then completed by choosing a suitable number of variables (as defined by the user) that should be used by the classifier. For the multi-class case with one-vs-all or pairwise, one obtains K and $\binom{K}{2}$ separate rankings, respectively, and the union of them forms the set of predictor variables. We again emphasize that the variable importance assignment is based on learning data only, which means that the procedure is repeated for each learning/test set splitting successively.

3.1.4 Performance measures

Once the classification step has been performed for all B iterations using the method `classification`, the method `evaluation` offers a variety of possibilities for evaluation of the results. As accuracy measures, the user may choose among the following criteria.

- **Misclassification rate**
  This is the simplest and most commonly used performance measure, corresponding to the indicator loss function in Eq. (1). From B iterations, one obtains a total of $\sum_b |\mathcal{T}_b|$ predictions. It implies that, with most procedures, the class label of each predictor-class pair in the sample S is predicted several times. The method `evaluation` can be applied in two directions: one can compute the misclassification rate either iterationwise, i.e. for each iteration separately (`scheme="iterationwise"`), yielding $\widehat{\boldsymbol{\epsilon}}^{\text{iter}} = (\widehat{\epsilon}_b)_{b=1}^B$ or observationwise, i.e. for each observation separately (`scheme="observationwise"`), yielding $\widehat{\boldsymbol{\epsilon}}^{\text{obs}} = (\widehat{\epsilon}_i)_{i=1}^n$. The latter can be aggregated by classes which is useful in the frequent case where some classes can be discriminated better than the other. Furthermore, observationwise evaluation can help identifying outliers which are often characterized by high misclassification error rates. Although $\widehat{\boldsymbol{\epsilon}}^{\text{iter}}$ or $\widehat{\boldsymbol{\epsilon}}^{\text{obs}}$ can be further averaged, the whole vectors are preferred to their less informative average, in order to reflect uncertainty more appropriately. A second advantage is that graphical summaries in the form of boxplots can be obtained.

- **Cost-based evaluation**
  Cost-based evaluation is a generalization of the misclassification error rate. The loss function is defined on the discrete set

$\{0,\dots,K-1\} \times \{0,\dots,K-1\}$, associating a specific cost to each possible combination of predicted and true classes. It can be represented as a matrix $\boldsymbol{L} = (l_{rs}),\ r,s = 1,\dots,(K-1)$ where l_{rs} is the cost or loss caused by assigning an observation of class r to class s. A usual convention is $l_{rr} = 0$ and $l_{rs} > 0$ for $r \neq s$. As for the misclassification rate, both iteration- and observationwise evaluation are possible.

- **Sensitivity, specificity and area under the curve (AUC)**
  These three performance measures are standard measures in medical diagnosis, see (17) for an overview. They are computed for binary classification only.

- **Brier Score and average probability of correct classification**
  In classification settings, the Brier Score is defined as

  $$n^{-1}\sum_{i=1}^{n}\sum_{k=0}^{K-1}(I(y_i = k) - \widehat{P}(y_i = k|\boldsymbol{x}_i))^2,$$

  where $\widehat{P}(y = k|\boldsymbol{x})$ stands for the estimated probability for class k, conditional on $\boldsymbol{x}$. Zero is the optimal value of the Brier Score.

  A similar measure is the average probability of correct classification which is defined as

  $$n^{-1}\sum_{i=1}^{n}\sum_{k=0}^{K-1} I(y_i = k)\widehat{P}(y_i = k|\boldsymbol{x}),$$

  and equals 1 in the optimal case. Both measures have the advantage that they are based on the continuous scale of probabilities, thus yielding more precise results. As a drawback, however, they cannot be applied to all classifiers but only to those associated with a probabilistic background (e.g. penalized regression). For other methods, they can either not be computed at all (e.g. nearest neighbors) or their application is questionable (e.g. support vector machines).

- **0.632 and 0.632+ estimators**
  The ordinary misclassification error rate estimates resulting from working

with learning sets of size $< n$ tend to overestimate the true prediction error. A simple correction proposed for learning sets generated from bootstrapping (argument `method="bootstrap"` in the function `GenerateLearningsets`) uses a convex combination of the resubstitution error -which has a bias in the other direction (weight: 0.368) and the bootstrap error estimation (weight: 0.632). A further refinement of this idea is the 0.632+ estimator (39) which is approximately unbiased and seems to be particularly appropriate in the case of overfitting classifiers.

The method `compare` can be used as a shortcut if several measures have to be computed for several classifiers. The function `obsinfo` can be used for outlier detection: given a vector of observationwise performance measures, it filters out observations for which the classifier fits poorly on average (i.e. high misclassification rate or low Brier Score, for example).

3.2 A real-life data example

This section gives a demonstration of the CMA package through an application to real world microarray data. It illustrates the typical workflow comprising learning set generation, variable selection, hyperparameter tuning, classifier training, and evaluation. The small blue round cell tumor data set was first analyzed by Khan et al (40) and is available from the R package 'pamr' (41). It comprises $n = 65$ samples from four tumor classes and expression levels from $p = 2309$ genes. In general, good classification results can be obtained with this data set, even with relatively simple methods (42). The main difficulty arises from the two classes with small size (8 and 12 observations, respectively).

CMA implements a complete bundle of discriminant analysis methods and related approaches. In this demonstrating example, we compare the performance of five of them: diagonal-, linear- and quadratic discrimininant analysis, shrunken centroids discriminant analysis and Partial Least Squares followed by linear discriminant analysis. From a theoretical point of view, linear- and quadratic analysis are a priori inferior due to the fact that they do not work in the $p \gg n$ setting without variable selection. Shrunken centroids discriminant analysis is assumed to work better than simple diagonal discriminant analysis because it is able to "shrink-out" noise variables. Partial Least Squares is also expected to work well.

For a fair comparison, all (global) settings that may influence the results should be fixed in advance to common values for all methods. In particular, the type of learning sets and the number of variables to be used (though this could also be interpreted as tuning parameter in a future version of the package) have to be fixed. We choose to work with stratified five-fold cross-validation, repeated ten times in order to achieve more stable results (15). For linear discriminant analysis we decide to work with ten and for quadratic discriminant analysis only with two variables. The numbers are chosen arbitrarily without any deeper motivation, which we consider legitimate for the purpose of illustration. In practice, this choice should be given more attention. For the remaining three classifiers, no variable selection is performed. We start by preparing the data and generating learning sets:

```
> data(khan)
> khanY <- khan[, 1]
> khanX <- as.matrix(khan[, -1])
> set.seed(27611)
> fiveCV10iter <- GenerateLearningsets(y = khanY, method = "CV",
+     fold = 5, niter = 10, strat = TRUE)
```

`khanY` is an n-vector of class labels coded as 1,2,3,4, and `khanX` an $n \times p$-matrix, where $p = 2309$ stands for the number of transcripts. `fiveCV10iter` is an object of class `learningsets`. For each of the `niter`= 10 iterations, `fiveCV10iter` stores which observations belong to the learning sets that are generated by the method specified through the arguments `method`, `fold` and `strat`. For reproducibility purposes, it is crucial to set the random seed. As a second step, we perform variable selection for those methods requiring it:

```
> genesel_da <- GeneSelection(X = khanX, y = khanY, learningsets = fiveCV10iter,
+     method = "f.test")
```

Here, the choice of the method is motivated by the multi-class setting. For visualization one can now use the `toplist` method on the object `genesel_da` to display the genes with highest F ratio in a listwise manner or create a barplot of the F ratios using `plot()`.

We now turn to hyperparameter tuning, which is performed via nested cross-validation. For Partial Least Squares, we optimize the number of latent components R over the grid $\{1, \dots, 5\}$. For the nearest shrunken centroids approach, the shrinkage intensity is optimized over the grid $\{0.1, 0.25, 0.5, 1, 2, 5\}$ (which is the predefined setting in CMA). We point out that the second grid is rather coarse and could be finer in pratice.

```
> tune_pls <- tune(X = khanX, y = khanY, learningsets = fiveCV10iter,
+     classifier = pls_ldaCMA, grids = list(comp = 1:5))
> tune_scda <- tune(X = khanX, y = khanY, learningsets = fiveCV10iter,
+     classifier = scdaCMA, grids = list())
```

In the second function call to `tune`, the argument `grids()` is an empty list: the default settings are used. The objects created in the steps described above are now passed to the function `classification`. The object `genesel_da` is passed for the classifiers `ldaCMA,qdaCMA`. The argument `nbgene` indicates that only the best `nbgene` genes are used, where best is understood in terms of the F ratio.

```
> class_dlda <- classification(X = khanX, y = khanY, learningsets = fiveCV10iter,
+     classifier = dldaCMA)
> class_lda <- classification(X = khanX, y = khanY, learningsets = fiveCV10iter,
+     classifier = ldaCMA, genesel = genesel_da, nbgene = 10)
> class_qda <- classification(X = khanX, y = khanY, learningsets = fiveCV10iter,
+     classifier = qdaCMA, genesel = genesel_da, nbgene = 2)
> class_plsda <- classification(X = khanX, y = khanY, learningsets = fiveCV10iter,
+     classifier = pls_ldaCMA, tuneres = tune_pls)
> class_scda <- classification(X = khanX, y = khanY, learningsets = fiveCV10iter,
+     classifier = scdaCMA, tuneres = tune_scda)
```

Note that the function `classification` can also be used directly, i.e. without calling `tuning` and `GeneSelection` separately, and perform hyperparameter tuning and/or variable selection automatically.

The classification results can now be visualized using the function `comparison`, which takes a list of classifier outputs as input. For instance, the results may be tabulated and visualized in the form of boxplots. The following commands yield the boxplots included in Figure 4.

```
> dalike <- list(class_dlda, class_lda, class_qda, class_scda,
+     class_plsda)
> par(mfrow = c(3, 1))
> comparison <- compare(dalike, plot = TRUE, measure = c("misclassification",
+     "brier score", "average probability"))
> print(comparison)
```

```
        misclassification brier.score average.probability
DLDA    0.06807692        0.13420913  0.9310332
LDA     0.04269231        0.07254283  0.9556106
QDA     0.24000000        0.34247861  0.7362778
scDA    0.01769231        0.02523875  0.9781925
pls_lda 0.01435897        0.02127534  0.9856975
```

4 Conclusions

CMA is a new user-friendly Bioconductor package for constructing and evaluating classifiers based on a high number of predictors in a unified framework. It was originally motivated by microarray-based classification, but can also be used for prediction based on other types of high-dimensional data such as, e.g. proteomic, metabolomic data, or signal data. CMA combines user-friendliness (simple and intuitive syntax, visualization tools) and methodological strength (especially in respect to variable selection and tuning procedures). We plan to further develop CMA and include additional features. Some potential extensions are outlined below.

- In the hyperparameter tuning procedure, fixing the set of selected variables for the inner cross-validation step might induce bias. At the inner level, each learning set (on which variable selection is based) is again split into learning- and test set. Hence, within the inner cross-validation loop, variable selection is performed based on both learning and test data. The set of selected variables may thus tend to overfit the test set, which potentially leads to the selection of different hyperparameter values. To address this problem, one could perform variable selection for each inner-loop iteration separately. In our opinion, this issue needs further investigations.

- In the context of clinical bioinformatics, researchers often focus their attention on the additional predictive value of high-dimensional molecular data given that good clinical predictors are already available. In this context, combined classifiers using both clinical and high-dimensional molecular data have been recently developed (43; 17). Such methods could be integrated into the CMA framework by defining an additional argument corresponding to (mandatory) clinical variables.

- Another potential extension is the development of procedures for measuring the stability of classifiers, following the scheme of our Bioconductor package 'GeneSelector' (44) which implements resampling methods in the context of univariate ranking for the detection of differential expression. In our opinion, it is important to check the stability of predictive rules with respect to perturbations of the original data. This last aspect refers to the issue of 'noise discovery' and 'random findings' from microarray data (45; 46).

- In future research, one could also work on the inclusion of additional information about predictor variables in the form of gene ontologies or pathway maps as available from KEGG (47) or cMAP (`http://pid.nci.nih.gov/`) with the intention to stabilize variable selection and to simultaneously select groups of predictors, in the vein of the so-called "gene set enrichment analysis" (48).

- CMA deals only with classification. The framework could be extended to other forms of high-dimensional regression, for instance high-dimensional survival analysis (49; 50; 51; 52).

In conclusion, we would like to outline in which situations CMA may help and warn against potential wrong use. CMA provides a unified interface to a large number of classifiers and allows a fair evaluation and comparison of the considered methods. Hence, CMA is a step towards reproducibility and standardization of research in the field of microarray-based outcome prediction. In particular, CMA users do not favor a given method or overestimate prediction accuracy due to wrong variable selection/tuning schemes. However, they should be cautious while interpreting and presenting their results. Trying all available classifiers successively and reporting only the best results would be a wrong approach (6) potentially leading to severe "optimistic bias". In this spirit, Ioannidis (45) points out that most results obtained with microarray data are nothing but "noise discovery" and Daumer et al (53) recommend to try to validate findings in an independent data set, whenever possible and feasible. In summary, instead of fishing for low prediction errors using all available methods, one should rather report all the obtained results or validate the best classifier using independent fresh validation data. Note that both procedures can be performed using CMA.

Acknowledgements

This work was partially supported by the Porticus Foundation in the context of the International School for Technical Medicine and Clinical Bioinformatics.

References

[1] R. Ihaka, R. Gentleman, R: A language for data analysis and graphics, Journal of Computational and Graphical Statistics 5 (1996) 299–314.

[2] R. Gentleman, J. Carey, D. Bates, et al., Bioconductor: Open software development for computational biology and bioinformatics, Genome Biology 5 (2004) R80.

[3] R. Tibshirani, T. Hastie, B. Narasimhan, G. Chu, Diagnosis of multiple cancer types by shrunken centroids of gene expression, Proceedings of the National Academy of Sciences 99 (2002) 6567–6572.

[4] L. Breiman, Random forests, Machine Learning 45 (2001) 5–32.

[5] A. L. Boulesteix, K. Strimmer, Partial least squares: A versatile tool for the analysis of high-dimensional genomic data, Briefings in Bioinformatics 8 (2007) 32–44.

[6] A. Dupuy, R. Simon, Critical review of published microarray studies for cancer outcome and guidelines on statistical analysis and reporting, Journal of the National Cancer Institute 99 (2007) 147–157.

[7] C. Ambroise, G. J. McLachlan, Selection bias in gene extraction in tumour classification on basis of microarray gene expression data, Proceedings of the National Academy of Science 99 (2002) 6562–6566.

[8] D. Berrar, I. Bradbury, W. Dubitzky, Avoiding model selection bias in small-sample genomic datasets, BMC Bioinformatics 22 (2006) 2245–2250.

[9] A.-L. Boulesteix, Wilcoxcv: An r package for fast variable selection in cross-validation, Bioinformatics 23 (2007) 1702–1704.

[10] A. Statnikov, C. F. Aliferis, I. Tsamardinos, D. Hardin, S. Levy, A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis, Bioinformatics 21 (2005) 631–643.

[11] S. Varma, R. Simon, Bias in error estimation when using cross-validation for model selection, BMC Bioinformatics 7 (2006) 91.

[12] J. Mar, R. Gentleman, V. Carey, MLInterfaces: Uniform interfaces to R machine learning procedures for data in Bioconductor containers, r package version 1.10.2 (2007).

[13] R. Gentleman, V. Carey, W. Huber, R. Irizarry, S. Dudoit, Bioinformatics and Computational Biology Solutions Using R and Bioconductor, Springer, New York, 2005.

[14] M. Ruschhaupt, U. Mansmann, P. Warnat, W. Huber, A. Benner, MCRestimate: Misclassification error estimation with cross-validation, r package version 1.10.2 (2007).

[15] U. Braga-Neto, E. R. Dougherty, Is cross-validation valid for small-sample microarray classification?, Bioinformatics 20 (2004) 374–380.

[16] A. Molinaro, R. Simon, R. M. Pfeiffer, Prediction error estimation: a comparison of resampling methods, Bioinformatics 21 (2005) 3301–3307.

[17] A. Boulesteix, C. Porzelius, M. Daumer, Microarray-based classification and clinical predictors: On combined classifiers and additional predictive value, In revision.

[18] L. Breiman, Bagging predictors, Machine Learning 24 (1996) 123–140.

[19] B. Efron, R. Tibshirani, An introduction to the bootstrap, Chapman and Hall, 1993.

[20] J. Chambers, Programming with Data, Springer, N.Y., 1998.

[21] D. Donoho, I. Johnstone, Ideal spatial adaption by wavelet shrinkage, Biometrika 81 (1994) 425–455.

[22] B. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, 1996.

[23] S. Wood, Generalized Additive Models: An Introduction with R, Chapman and Hall/CRC, 2006.

[24] J. Friedman, Regularized discriminant analysis, Journal of the American Statistical Association 84 (405) (1989) 165–175.

[25] Y. Guo, T. Hastie, R. Tibshirani, Regularized discriminant analysis and its application in microarrays, Biostatistics 8 (2007) 86–100.

[26] R. Tibshirani, Regression shrinkage and selection via the LASSO, Journal of the Royal Statistical Society B 58 (1996) 267–288.

[27] H. Zhou, T. Hastie, Regularization and variable selection via the elastic net., Journal of the Royal Statistical Society B 67 (2004) 301–320.

[28] T. Hastie, R. Tibshirani, J. H. Friedman, The elements of statistical learning, Springer-Verlag, New York, 2001.

[29] L. Breiman, J. H. Friedman, R. A. Olshen, J. C. Stone, Classification and Regression Trees, Wadsworth, Monterey, CA, 1984.

[30] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences 55 (1997) 119–139.

[31] J. Friedman, Greedy function approximation: A gradient boosting machine., Annals of Statistics 29 (2001) 1189–1232.

[32] T. Hastie, R. Tibshirani, Efficient quadratic regularization for expression arrays, Biostatistics 5 (2004) 329–340.

[33] G. Golub, C. V. Loan, Matrix Computations, Johns Hopkins University Press, 1983.

[34] E. Parzen, On estimation of a probability density function and mode., Annals of Mathematical Statistics 33 (1962) 1065–1076.

[35] G. Smyth, Limma: linear models for microarray data, Springer, New York, 2005, pp. 397–420.

[36] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene selection for cancer classification using support vector machines., Journal of Machine Learning Research 46 (2002) 389–422.

[37] P. Bühlmann, B. Yu, Boosting with the l2 loss: Regression and classification, Journal of the American Statistical Association 98 (2003) 324–339.

[38] T. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. Downing, M. A. Caligiuri, C. D. Bloomfield, E. S. Lander, Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, Science 286 (1999) 531–537.

[39] B. Efron, R. Tibshirani, Improvements on cross-validation: The .632+ bootstrap method, Journal of the American Statistical Association 92 (1997) 548–560.

[40] J. Khan, J. Wei, M. Ringner, L. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. Antonescu, C. Peterson, P. Meltzer, Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks, Nature Medicine 7 (2001) 673–679.

[41] R. Tibshirani, T. Hastie, B. Narasimhan, G. Chu, Class prediction by nearest shrunken centroids, with applications to dna microarrays, Statistical Science 18 (2002) 104–117.

[42] A. L. Boulesteix, PLS dimension reduction for classification with microarray data, Statistical Applications in Genetics and Molecular Biology 3 (2004) 33.

[43] H. Binder, M. Schumacher, Allowing for mandatory covariates in boosting estimation of sparse high-dimensional survival models, BMC Bioinformatics 9 (2008) 14.

[44] M. Slawski, A.-L. Boulesteix, Geneselector, BioconductorR package version 1.1.0: `http://www.bioconductor.org/packages/devel/bioc/html/GeneSelector.html`.

[45] J. P. Ioannidis, Microarrays and molecular research: noise discovery, The Lancet 365 (2005) 488–492.

[46] C. Davis, F. Gerick, V. Hintermair, C. Friedel, K. Fundel, R. Kueffner, R. Zimmer, Reliable gene signatures for microarray classification: assessment of stability and performance, Bioinformatics 22 (2006) 2356–2363.

[47] M. Kanehisa, S. Goto, Kegg: Kyoto encyclopedia of genes and genomes, Nucleic Acids Research 28 (2000) 27–30.

[48] A. Subramanian, P. Tamayo, V. Mootha, S. Mukherjee, B. Ebert, M. Gilette, A. Paulovich, S. Pomeroy, T. Golub, E. Lander, J. Mesirov, Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles., Proceedings of the National Acadeny of Science 102 (2005) 15545–15550.

[49] H. M. Bovelstad, S. Nygard, H. L. Storvold, M. Aldrin, O. Borgan, A. Frigessi, O. C. Lingjaerde, Predicting survival from microarray data a comparative study, Bioinformatics 23 (2007) 2080–2087.

[50] M. Schumacher, H. Binder, T. Gerds, Assessment of survival prediction models based on microarray data, Bioinformatics 23 (2007) 1768–1774.

[51] R. Diaz-Uriarte, Signs: a parallelized, open-source, freely available, web-based tool for gene selection and molecular signatures for survival and censored data, BMC Bioinformatics 9 (2008) 30.

[52] W. van Wieringen, D. Kun, R. Hampel, A.-L. Boulesteix, Survival prediction using gene expression data: a review and comparison, Computational Statistics Data Analysis (accepted).

[53] M. Daumer, U. Held, K. Ickstadt, M. Heinz, S. Schach, G. Ebers, Reducing the probability of false positive research findings by pre-publication validation: Experience with a large multiple sclerosis database, BMC Medical Research Methodology 8 (2008) 18.

[54] G. McLachlan, Discriminant Analysis and Statistical Pattern Recognition, Wiley, New York, 1992.

[55] M. Young-Park, T. Hastie, L1-regularization path algorithm for generalized linear models., Journal of the Royal Statistical Society B 69 (2007) 659–677.

[56] J. Zhu, T. Hastie, Classification of gene microarrays by penalized logistic regression, Biostatistics 5 (2003) 427–443.

[57] D. Specht, Probabilistic neural networks., Neural Networks 3 (1990) 109–118.

[58] B. Schölkopf, A. Smola, Learning with Kernels, MIT Press, Cambridge, MA, USA, 2002.

Figures

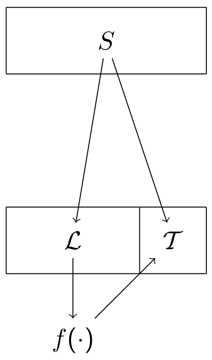


Figure 1: **Splitting into learning and test data sets.** The whole sample S is split into a learning set $\mathcal{L}$ and a test set $\mathcal{T}$. The classifier $f(.)$ is constructed using the learning set $\mathcal{L}$ and subsequently applied to the test set $\mathcal{T}$.

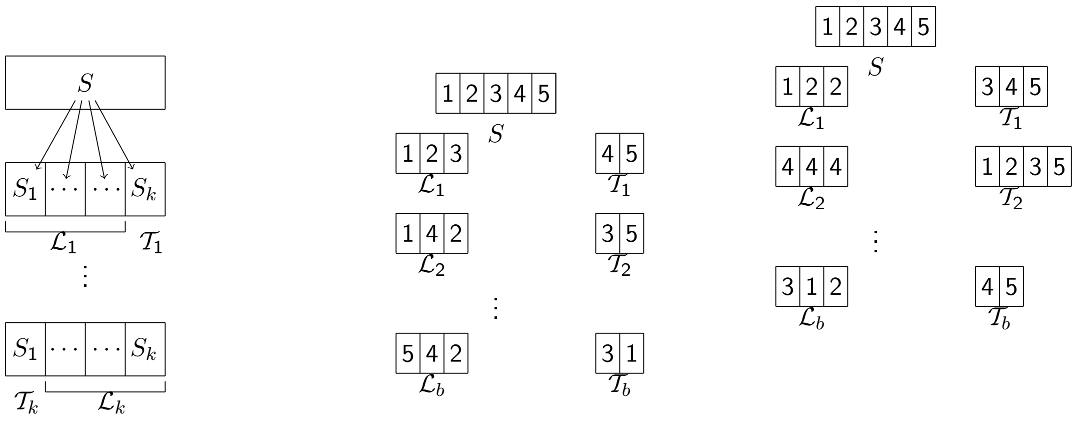
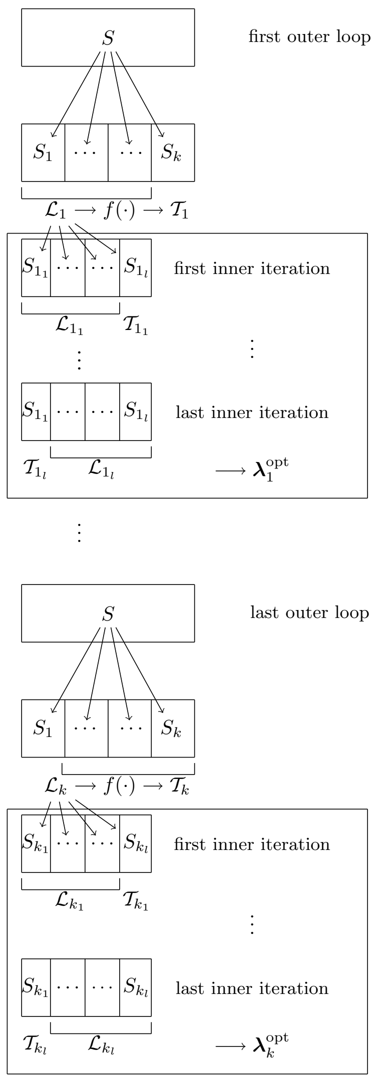


Figure 2: **Evaluation schemes.** Schematic display of k-fold cross-validation (left), Monte-Carlo cross-validation with $n = 5$ and `ntrain`=3 (middle), and bootstrap sampling (with replacement) with $n = 5$ and `ntrain`=3 (right).



S
first outer loop
S_1 $\cdots$ $\cdots$ S_k
$\mathcal{L}_1 \longrightarrow f(\cdot) \longrightarrow \mathcal{T}_1$
S_{1_1} $\cdots$ $\cdots$ S_{1_l}
first inner iteration
$\mathcal{L}_{1_1}$ $\mathcal{T}_{1_1}$
last inner iteration
$\mathcal{T}_{1_l}$ $\mathcal{L}_{1_l}$
$\longrightarrow \boldsymbol{\lambda}_1^{\text{opt}}$
last outer loop
$\mathcal{L}_k \longrightarrow f(\cdot) \longrightarrow \mathcal{T}_k$
S_{k_1} $\cdots$ $\cdots$ S_{k_l}
$\mathcal{L}_{k_1}$ $\mathcal{T}_{k_1}$
$\mathcal{T}_{k_l}$ $\mathcal{L}_{k_l}$
$\longrightarrow \boldsymbol{\lambda}_k^{\text{opt}}$

Figure 3: **Hyperparameter tuning:** Schematic display of nested cross-validation. In the procedure displayed below, k-fold cross-validation is used for evaluation purposes, whereas tuning is performed within each iteration using inner (l-fold) cross-validation.

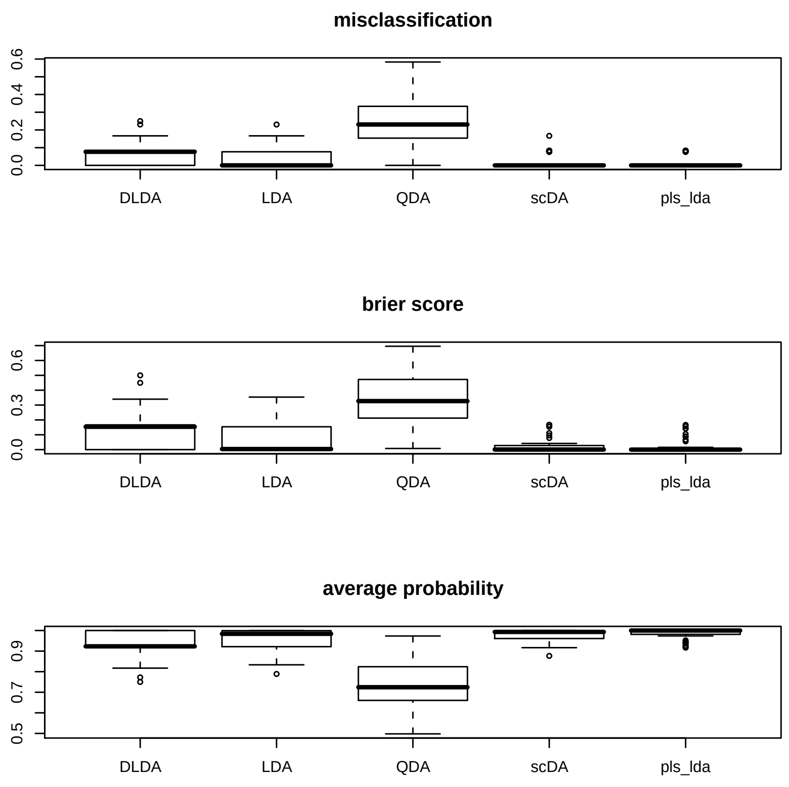


Figure 4: Boxplots representing the misclassification rate (top), the Brier score (middle), and the average probability of correct classification (bottom) for Khan's SRBCT data, using five classifiers: diagonal linear discriminant analysis, linear discriminant analysis, quadratic discriminant analysis, shrunken centroids discriminant analysis (PAM), and PLS followed by linear discriminant analysis.

Tables

Method name	CMA **function name**	**Package**	**Reference**
Componentwise boosting	`compBoostCMA`	CMA	(37)
Diagonal discriminant analysis	`dldaCMA`	CMA	(54)
Elastic net	`ElasticNetCMA`	'glmpath'	(27)
Fisher's discriminant analysis	`fdaCMA`	CMA	(22)
Flexible discriminant analysis	`flexdaCMA`	'mgcv'	(22)
Tree-based boosting	`gbmCMA`	'gbm'	(31)
k-nearest neighbors	`knnCMA`	'class'	(22)
Linear discriminant analysis $*$	`ldaCMA`	'MASS'	(54)
Lasso	`LassoCMA`	'glmpath'	(55)
Feed-forward neural networks	`nnetCMA`	'nnet'	(22)
Probalistic nearest neighbors	`pknnCMA`	CMA	–
Penalized logistic regression	`plrCMA`	CMA	(56)
Partial Least Squares $\star + *$	`pls_ldaCMA`	'plsgenomics'	(5)
$\star$ + logistic regression	`pls_lrCMA`	'plsgenomics'	(5)
$\star$ + random forest	`pls_rfCMA`	'plsgenomics'	(5)
Probabilistic neural networks	`pnnCMA`	CMA	(57)
Quadratic discriminant analysis $*$	`qdaCMA`	'MASS'	(54)
Random forest	`rfCMA`	'randomForest'	(4)
PAM	`scdaCMA`	CMA	(41)
Shrinkage discriminant analysis	`shrinkldaCMA`	CMA	–
Support vector machine	`svmCMA`	'e1071'	(58)

Table 1: **Overview of the classification methods in CMA.** The first column gives the method name, whereas the name of the classifier in the CMA package is given in the second column. For each classifier, CMA uses either own code or code borrowed from another package, as specified in the third column.