c060: Extended Inference with Lasso and Elastic-Net Regularized Cox and Generalized Linear Models

Martin Sill **Thomas Hielscher** **Natalia Becker** **Manuela Zucknick**
Division of Biostatistics (C060)
German Cancer Research Center, Heidelberg

Abstract

We have developed the R package **c060** (**?**) with the aim of improving R software functionality for high-dimensional risk prediction modelling, e.g., for prognostic modelling of survival data using high-throughput genomic data. Penalized regression models provide a statistically appealing way of building risk prediction models from high-dimensional data. The popular CRAN package **glmnet** package (**?**) implements an efficient algorithm for fitting penalized Cox and generalized linear models. However, in practical applications the data analysis will typically not stop at the point where the model has been fitted. One is for example often interested in the stability of selected features and in assessing the prediction performance of a model and we provide functions to deal with both of these tasks. Our R functions are computationally efficient and offer the possibility of speeding up computing time through parallel computing. Another feature which can drastically reduce computing time is an efficient interval-search algorithm, which we have implemented for selecting the optimal parameter combination for elastic net penalties. These functions have been useful in our daily work at the Biostatistics department (C060) of the German Cancer Research Center where prognostic modelling of patient survival data is of particular interest. Although we focus on a survival data application of penalized Cox models in this article, the functions in our R package are in general applicable to all types of regression models implemented in the **glmnet** package, with the exception of prediction error curves, which are specific to time-to-event data.

Keywords: **glmnet**, penalized log-likelihood method, stability selection, interval search, prediction error.

1. Introduction

Penalized regression models provide a statistically appealing method to build prediction models from high-dimensional data sources, where it is the aim to simultaneously select features and to fit the model (**??**). Since the introduction of the lasso for linear regression models (**?**), the methodology has been extended to generalized linear regression models and time-to-event endpoints (**?**) among others. In addition to the well-known L_1-norm (lasso) and L_2-norm (ridge) penalty functions, various other penalties have been proposed in recent years to select features and/or estimate their effects. In particular, we will use the elastic net penalty function (**?**), which is a linear combination of the L_1- and L_2-norms.

With ever increasing data, the properties of the algorithm used for fitting the model have become almost as important as the statistical model itself. **?** proposed a coordinate descent algorithm for

generalized linear regression models, which has since then been extended to penalized Cox proportional hazards (PH) regression models (**?**). Due to its efficiency this algorithm is considered one of the state-of-the-art approaches to estimate penalized regression models with lasso, ridge or elastic net penalty terms, especially in high-dimensional data scenarios. First references about coordinate descent algorithms date back to **?**.

This algorithm has been implemented in R (**?**) in the **glmnet** package (**?**). The package provides functions to tune and fit regression models, plot the results, and make predictions. However, in practical applications, where often an independent validation data set is lacking, some additional features and routines are desirable as part of a complete data analysis. We have assembled some functions that enhance the existing functionality of the **glmnet** package or allow to use it within the framework of other existing R packages. These functions have been useful in our daily work at the Biostatistics department (C060) of the German Cancer Research Center where prognostic modelling of patient survival data is of particular interest. Therefore, for illustration purposes we focus on penalized Cox PH regression models in this article. But the R functions are generally applicable to all types of regression models implemented in the **glmnet** package.

Computational efficiency is an important requirement of the software to make applications feasible for real-life data analysis tasks in fields such as molecular oncology, where one aims to develop sparse risk prediction models based on very large numbers of molecular features measured with high-throughput technologies such as microarrays or next-generation sequencing. Therefore, we provide functionality to speed up computations, in particular through parallel computing.

We provide R functions to perform stability selection (**?**) in a computationally efficient way which allows to select the most stable features at a given Type I error level. We have also implemented an approach to select the optimal parameter combination (α, λ) for elastic net penalties using an interval-search algorithm (**?**) which is often faster and more accurate than a standard grid search (**?**). Another very useful addition for real-life applications of **glmnet** for building risk-prediction models is the provision of wrapper functions to allow the computation of resampling-based prediction errors within the framework of the R package **peperr** (**?**). The **peperr** package makes it computationally feasible to assess the predictive accuracy of a penalized regression model via resampling methods even for very large-scale applications by employing parallel computing. We also provide the possibility to speed up stability selection by parallel computing using the functionalities of the R base package **parallel** (**?**).

Stability selection and interval search are currently defined for Gaussian, binomial, Poisson, multinomial and Cox models, prediction error curves for Cox models and classification errors for binomial models. Since all functions are basically wrapped around the **glmnet** fitting or cross-validation function calls, the type of response is always specified by the `family` argument.

The software is available as R package **c060** on CRAN (URL `http://cran.r-project.org/web/packages/c060/`, version 0.2-3 at time of manuscript publication) and on R-forge (URL `https://r-forge.r-project.org/projects/c060/`, version 0.2-3 at the time of publication).

2. Methods and algorithms

2.1. Penalized generalized linear models and Cox models

An efficient implementation for fitting generalized linear models and Cox proportional hazards models with regularization by the lasso or elastic net penalty terms is provided by the R package **glmnet**. This implementation uses a coordinate descent algorithm for fitting the models for specified values of

penalty parameters $\lambda > 0$ and $\alpha \in (0,1]$. The computation of an entire regularization path across a range of values $\Lambda = \{\lambda_1, \lambda_2 ..., \lambda_K\}$ at fixed α with **glmnet** is generally very fast, because previously computed solutions for $\{\lambda_1, ..., \lambda_{k-1}\}$ are used as 'hot' starting values for the computation of λ_k. This implies that it is often more efficient to compute the models for the entire regularization path of Λ rather than just individual models. We use this feature in all of our implemented algorithms to make most use of the computational speed of **glmnet**.

Models are fitted by maximizing the penalized log-likelihood function for generalized linear models and the penalized partial log-likelihood for Cox models. The penalized (partial) log-likelihood function is given by

$$l_n(\beta) - \sum_{j=1}^{p} p_{\alpha,\lambda}(|\beta_j|) \tag{1}$$

where $l_n(\beta)$ denotes the (partial) log-likelihood given n observations. The dimension of the parameter vector β is p and $p_{\alpha,\lambda}(|\cdot|)$ is the penalty function with tuning parameters λ and α.

Cross-validation can be performed to decide which model, i.e., which penalty parameter values, to choose by using the negative cross-validated penalized (partial) log-likelihood as the loss function. Actually, within the **glmnet** package, the penalized (partial) log-likelihood deviance is used as the loss function rather than the log-likelihood function itself. The deviance is equal to -2 times the log-likelihood ratio of the model of interest compared to the saturated model, which has one free parameter per observation. Obviously, both versions will result in the same optimization result.

2.2. L_2-penalized Cox regression

Penalized maximum likelihood estimation in Cox regression with the ridge penalty

$$p_\lambda(|\beta_j|) = \lambda \beta_j^2 \tag{2}$$

was introduced by **?**. The ridge penalty results in parameter estimates that are biased towards zero, but does not set values exactly to zero, and hence does not perform feature selection. On the other hand, it has been found to produce models with good prediction performance in high-dimensional genomic applications (e.g., **?**), in particular if predictors are highly correlated.

2.3. L_1-penalized Cox regression

? proposed to use an L_1-penalized Cox model with

$$p_\lambda(|\beta_j|) = \lambda |\beta_j| \tag{3}$$

and described a technique, called the lasso for "least absolute shrinkage and selection operator", for parameter estimation. The L_1-penalty has the advantage over the L_2-penalty of shrinking some of the coefficients to zero, i.e., it performs automatic feature selection.

2.4. The elastic net

? introduced the elastic net, which employs a combination of the L_1- and L_2-penalty. Like lasso the elastic net performs automatic feature selection by setting some coefficient estimates to zero. But the additional L_2-penalty term distributes the weight to more features, so that the elastic net tends to select more features than the lasso. This is especially the case in situations with high correlation, since the

lasso would select only one feature of a set of perfectly correlated features, while the ridge penalty would give them equal weight.

Throughout this manuscript we use the same parametrization of the elastic net penalty function as the formulation used in the **glmnet** package:

$$p_{\alpha,\lambda}(|\beta_j|) = \lambda \times (\alpha|\beta_j| + (1-\alpha)\frac{1}{2}\beta_j^2). \tag{4}$$

Here, $\alpha \in (0,1]$ determines the influence of the L_1 penalty relative to the L_2 penalty. Small α values will result in models with many features, getting closer to the non-sparse ridge solution as α tends to zero.

The interval-search algorithm to select the optimal elastic net parameter combination

The elastic net penalty function contains two tuning parameters which are data-dependent and hence cannot be set to some *a priori* values. The challenge is to find a set of tuning parameters (α, λ), for which the k-fold cross-validated loss function of the model is minimal.

The commonly used fixed grid search has its major disadvantage in the need to systematically compute the penalized log likelihood deviance at each point of the grid, which implies that the grid density affects the accuracy and the time complexity of the algorithm. Furthermore, the choice of the grid is highly arbitrary and the solution depends on the choice of grid as well as on grid density.

? proposed an efficient algorithm for finding a global optimum on the tuning parameter space called Efficient Parameter Selection via Global Optimization (EPSGO). The main idea of the algorithm is to treat the task of finding the optimal tuning parameter values as a global optimization problem. For that purpose one learns a Gaussian process model of the loss function surface in parameter space and samples systematically at points where the so-called expected improvement criterion reaches the maximum.

The interval search can be divided into two phases. In the initial phase, a set of uniformly distributed points is randomly selected throughout the parameter space. Then, in the iteration phase, the algorithm learns the Gaussian process model from the points which have already been visited. By adding new points one updates the Gaussian process model. New points in the parameter space are sampled by using the expected improvement criterion as described by **?**. The EPSGO algorithm stops when one of the stopping criteria is met, i.e., if either convergence of the algorithm has been reached or if there was no change in the solution during the last ten iterations.

? showed that the algorithm is robust against local minima. One can observe an immense improvement in the training time for the Gaussian process model compared to more commonly used fixed grid search methods (**?**). This is because the number of training points for the Gaussian process (and hence the number of evaluations of the loss function surface of the regression model) mainly depends on the dimensionality of the tuning parameter space, which is very small compared to the number of training points on the grid.

Summing up, the EPSGO algorithm provides two main advantages when compared to grid search methods:

- Robustness against starting values: EPSGO solutions are not dependent on an arbitrary choice of a grid.
- Scalability of accuracy improvements: The accuracy of EPSGO solutions can be easily improved without the need for a massive increase in computing time implied by an increase in the

grid density.

2.5. Stability selection

The penalized regression models that we have described above are typically used to find sparse models with good predictive performance. In contrast, the stability selection proposed by **?** aims to find stable features which show strong association with the outcome. The stability selection is a general approach that combines feature selection methods such as L_1 penalized models with resampling. By applying the corresponding feature selection method to subsamples that were drawn without replacement, selection probabilities for each feature can be estimated as the proportion of subsamples where the feature is included in the fitted model. These selection probabilities are used to define a set of stable features. **?** provide a theoretical framework for controlling Type I error rates of falsely assigning features to the estimated set of stable features. The selection probability of each feature along the regularization path, e.g., along the range of possible penalization parameters $\Lambda = \{\lambda_1, \lambda_2 ..., \lambda_K\}$, is called stability path. Given an arbitrary threshold $\pi_{thr} \in (0.5, 1)$ and the set of penalization parameters Λ, the set of stable features estimated with stability selection is:

$$\hat{S}_{\beta}^{stable} = \left\{ j : \max_{\lambda_k \in \Lambda} \hat{\Pi}_j^{\lambda_k} \geq \pi_{thr} \right\}, \tag{5}$$

where $\hat{\Pi}_j^{\lambda_k}$ denotes the estimated selection probability of the jth feature at λ_k. Then according to Theorem 1 in **?**, the expected number of falsely selected features $E(V)$ will be bounded by:

$$E(V) \leq \frac{1}{(2\pi_{thr} - 1)} \frac{q_{\Lambda}^2}{p}, \tag{6}$$

where q_{Λ} is the average of the number of non-zero coefficients with respect to the drawn subsamples. Equation 6 shows that the bound on expected number of falsely selected features can be decreased by either reducing the average number of selected features q_{Λ} or by increasing the threshold π_{thr}. Suppose that π_{thr} is fixed, then $E(V)$ can be controlled by limiting q_{Λ} by the length of the regularization path Λ. In multiple testing the expected number of falsely selected features is also known as the per-family error rate (PFER) and if divided by the total number of features p will become the per-comparison error rate (PCER) (**?**). The stability selection allows to control these Type I error rates. For instance, suppose the threshold $\pi_{thr} = 0.8$ is fixed, then choosing Λ such that $q_{\Lambda} \leq \sqrt{0.6p}$ will control $E(V) \leq 1$. Moreover, choosing Λ so that $q_{\Lambda} \leq \sqrt{0.6p\alpha}$ will control the family-wise error rate (FWER) at level α, $P(V > 0) \leq \alpha$. As mentioned before, according to **?** the coordinate descent algorithm implemented in the **glmnet** package is most efficient regarding the computational time, when used to calculate the full regularization path. To utilize this property our algorithm calculates the stability path by first generating subsets by subsampling and then calculating for each subsample the regularization path using the coordinate descent algorithm. The resulting regularization paths are then averaged to form the stability path. Furthermore, since the calculations of the regularization paths for each subset are independent of each other, the algorithm can easily be parallelized using the package **parallel**.

2.6. Prediction error curves for survival models

The time-dependent Brier score (**?**) can be used to assess and compare the prediction accuracy of prognostic models for time-to-event endpoints. The Brier score at time point t is a weighted mean squared error between predicted survival probability and observed survival status. Weighting depends

on the estimated censoring distribution to account for the observations under risk (**?**). Computing the error for each time point over the entire follow-up horizon yields a prediction error curve. As a reference we use prediction errors based on the Kaplan-Meier curves estimated without any covariate information.

The empirical time-dependent Brier score $BS(t)$ is defined as a function of time $t > 0$ by

$$BS(t) = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{\hat{S}(t|x_i)^2 I(t_i \leq t \wedge \delta_i = 1)}{\hat{G}(t_i)} + \frac{(1-\hat{S}(t|x_i))^2 I(t_i > t)}{\hat{G}(t)}\right],$$

with individual survival time t_i, censoring indicator δ_i and estimated survival probability $\hat{S}(t|x_i)$ at time t based on the prognostic model given covariate values x_i for subject i out of n patients (**?**). $\hat{G}(t)$ denotes the Kaplan-Meier estimate of the censoring distribution at time t, which is based on the observations $(t_i; 1-\delta_i)$, $i = 1, ..., n$. I signifies the indicator function.

In case no independent validation data are available, resampling-based prediction error curves are used to adequately assess the model's prediction accuracy. The .632+ bootstrap estimator (**?**) is commonly used for these applications, which is a weighted mean of the apparent error and the average out-of-bag bootstrap error. For the apparent error the same data is used to develop the prognostic model and assess its performance. Due to overfitting, this error is far too optimistic, particularly with high-dimensional data. The average out-of-bag bootstrap error is too conservative since only a proportion of the entire data is used to develop the prognostic model in each bootstrap run. The .632+ estimator balances both estimators, and additionally accounts for the relative overfitting based on the no-information error rate. Further, in our application the .632+ bootstrap estimator is calculated based on subsampling (without replacement) rather than classical bootstrap sampling with replacement, as that has been demonstrated to lead to more accurate estimates in a high-dimensional context (**?**).

3. Application and demonstration of software

3.1. Data set

In the following we will demonstrate the use of the functions provided in the **c060** package in an application to a gene expression data set and corresponding clinical data of cytogenetically normal acute myeloid leukemia (AML) patients (**?**). The data can be accessed from the Gene Expression Omnibus (GEO) data repository (`http://www.ncbi.nlm.nih.gov/geo`) by the National Center for Biotechnology Information (NCBI). We find the data set under GEO accession number GSE12417. To simulate the typical situation that only one data set is available for model training and evaluation, we only use the data set that was used as validation data in the original publication. This data set contains gene expression data for 79 patient samples measured with Affymetrix HG-U133 Plus 2.0 microarrays. The median survival time of these patients was 17.6 months with a censoring rate of 40%.

For the sake of convenience we reduce the total number of 54675 gene expression features that have been measured with the Affymetrix HG-U133 Plus 2.0 microarray technology to the top 10000 features with largest variance across all 79 samples. For all computations the data set is stored as an `ExpressionSet` (from Bioconductor package **Biobase** (**?**)) called `eset`. The gene expression data matrix can be accessed through the call `exprs(eset)` and overall survival data and other patient-specific data (e.g., patient age) are stored within the `phenoData` object `pData(eset)`. Over-

all survival times are stored in the variable `os`, the corresponding survival status variable is called `os_status` and the patient age variable is `age`.

3.2. Starting off: Fitting the lasso-penalized Cox model

Our goal is to develop a prognostic model for patient overall survival based on the gene expression data. The purpose of this modelling exercise is not just to fit a prognostic model that is capable of predicting overall survival rates, but we also want to find out which gene expression features are most relevant for this task. Traditionally, this problem is solved by feature selection methods and we start our data analysis exercise by fitting the lasso-penalized Cox model, which provides automatic feature selection.

We can apply the `glmnet` function to fit a lasso-penalized Cox model to the AML data set. The function call with default penalty parameter settings will fit the lasso model for 100 λ values within a data-derived range of values:

```
R> fit <- glmnet(y=Surv(pData(eset)$os, pData(eset)$os_status),
+                x=t(exprs(eset)), family="cox")
```

In order to determine the optimal lasso penalty parameter value, we perform 10-fold cross-validation using the `cv.glmnet` function.

```
R> set.seed(1234)
R> cvres <- cv.glmnet(y=Surv(pData(eset)$os, pData(eset)$os_status),
+                     x=t(exprs(eset)), family="cox", nfolds=10)
R> res <- cvres$glmnet.fit
R> plot(cvres)
```

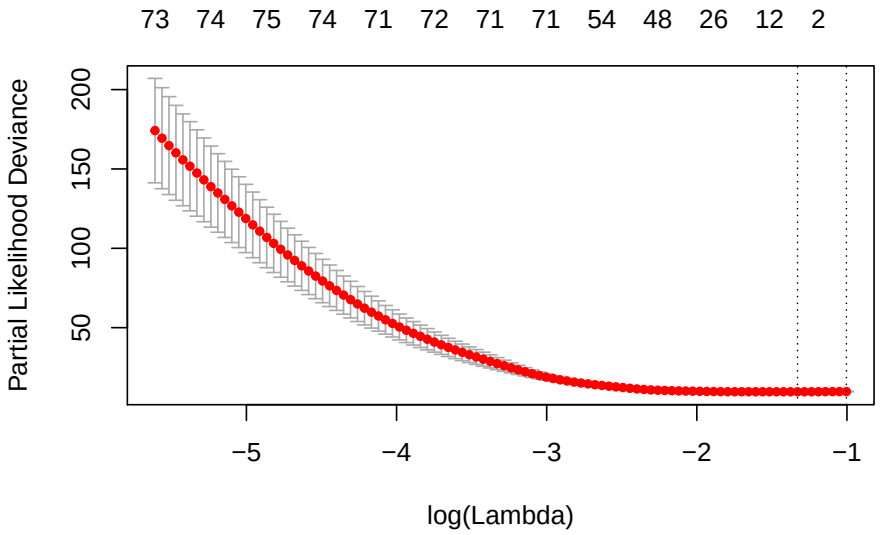

Figure 1: Cross-validated partial log-likelihood deviance, including upper and lower standard deviations, as a function of $\log\lambda$ for the AML data set. The dotted vertical lines indicate the λ values with minimal deviance (left) and with the largest λ value within one standard deviation of the minimal deviance (right).

The loss function, i.e., the cross-validated partial log-likelihood deviance, is shown in Figure 1 including upper and lower standard deviations as a function of $\log\lambda$ for the AML data set. The penalty parameter value minimizing the loss function is λ =0.265 ($\log\lambda$ =-1.329) and corresponds to a final lasso model with the following 5 selected features and corresponding lasso regression coefficient estimates:

```
  203640_at 204419_x_at 222462_s_at   226169_at   233371_at
    -0.1134     -0.0166       0.2742      0.0430     -0.0122
```

The selected features are highlighted as red lines in the coefficient paths shown in Figure 2, which illustrate the development of the regression coefficient estimates with increasing regularization. While the selected 5 features are the only features selected at the optimal λ value, they do not remain among the features with largest effect sizes when the penalty is reduced and thus more and more coefficients start to enter the model. In fact, for 4 out of the 5 features the coefficient estimates go back down to zero for small values of $\log\lambda$, indicating that these features get replaced by other gene expression features in very large models.

```
R> cof <- coef(res, s=cvres$lambda.min)
R> Plot.coef.glmnet(cvfit=cvres, betas=rownames(cof)[which(cof!=0)])
```

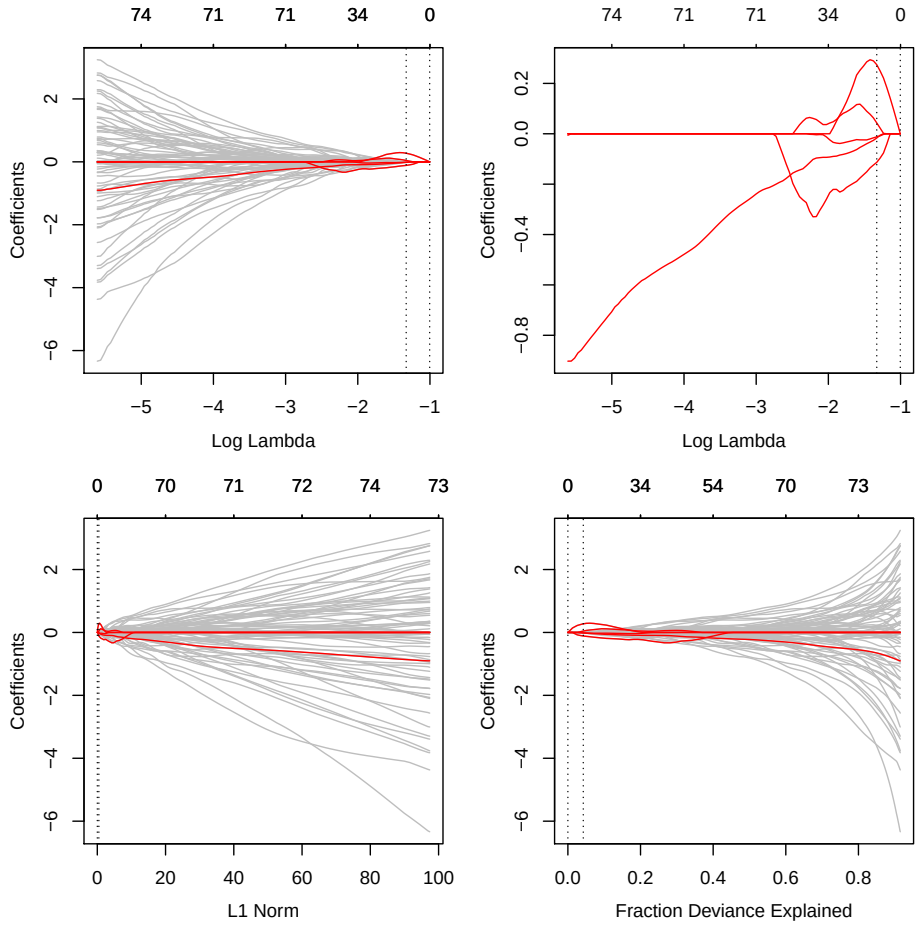

Figure 2: Coefficient paths for lasso-penalized Cox PH regression models applied to the AML data set. The features with highlighted paths have non-zero coefficients in the model with the optimal λ value as determined by ten-fold cross-validation. The top plots show the coefficient path scaled to reflect $\log(\lambda)$ on the x-axis (top left: full path, top right: zoomed in to only show the selected features). The bottom plots show the coefficient paths relative to the L_1-norms of the estimated coefficient vector (left) and to the fraction of the null partial log-likelihood deviance explained (right). The dotted vertical lines indicate the λ values with minimal deviance and with the largest λ value within one standard deviation of the minimal deviance.

3.3. Assessment of prediction performance with resampling-based prediction errors

Once the final prognostic model is selected, the next task is to assess its prediction accuracy for future patients, where one is often particularly interested in a comparison with established clinico-pathological prognostic markers. In many applications no independent validation data set is available, and thus the same data set needs to be used to both develop and assess the prognostic model.

This is especially problematic for high-dimensional data, where the risk of overfitting is very high. Resampling-based methods can be used to unbiasedly estimate the predictive accuracy of the prognostic model in this situation. This is also called internal validation.

For this purpose the R package **peperr** (**??**) provides a modular framework for survival and binary endpoints. Wrapper functions for new or customized prediction model algorithms can be defined and passed to the generic call function `peperr`. In case of prognostic models for survival endpoints, algorithm-specific wrapper functions are required for model fitting, tuning and prediction. Wrapper functions for selected machine learning approaches are already implemented, but not yet for the **glmnet** package.

With the **peperr** package prediction accuracy of survival models is by default assessed with prediction error curves based on the time-dependent Brier score, but it is also possible to define and use customized accuracy measures. We have implemented additional wrapper functions for the **glmnet** algorithm for fitting (`fit.glmnet`) and tuning (`complexity.glmnet`) the model, and predicting survival probabilities (`predictProb.glmnet`) based on the fitted model and the estimated baseline hazard from the training data using the Breslow estimator. We here want to assess the prognostic value of the L_1-penalized Cox PH regression model fitted in the previous section. The .632+ subsampling-based bootstrap estimator is calculated using 1000 bootstrap samples. The **peperr** package is designed for high-dimensional covariates data and allows for various parallel computation setups. Also, additional arguments can be passed directly to the **glmnet** call by specifying additional arguments for the corresponding fitting and/or tuning procedure. Here, we include patient age as a mandatory feature, i.e., age is not subject to penalization, and run the calculation on 3 CPUs in parallel using a socket cluster setup.

```
R> obj  <- peperr(response=Surv(pData(eset)$os, pData(eset)$os_status),
+                 x=data.frame(eset$age,t(exprs(eset))),
+                 fit.fun=fit.glmnet, args.fit=list(standardize=FALSE, family="cox",
+                 penalty.factor=rep(0:1, times=c(1,dim(eset)[1]))),
+                 complexity=complexity.glmnet,
+                 args.complexity=list(standardize=FALSE, nfolds=10, family="cox",
+                 penalty.factor=rep(0:1, times=c(1,dim(eset)[1]))),
+                 RNG="fixed", seed=0815, cpus=3, parallel=TRUE, clustertype="SOCK",
+                 load.list=list(packages=c("c060")),
+                 indices=resample.indices(n=dim(eset)[2],
+                 sample.n=1000, method="sub632"))
```

Bootstrap results can be visualized with the `plot.peperr` function from the **peperr** package showing the selected complexity parameters, out-of-bag prediction error curves as well as the prediction error integrated over time, and the predictive partial log-likelihood (PLL) values. In order to calculate the predictive PLL values again, an algorithm-specific wrapper (here `PLL.coxnet`) needs to be provided. In addition, we provide a slightly modified version of the prediction error curves plot function from the **peperr** package, which allows the display of the numbers of samples still at risk and pointwise bootstrap quantiles (`Plot.peperr.curves`) as shown in Figure 3. By default, the .632+ bootstrap estimate is calculated and displayed. Optionally, one can additionally display the .632 estimator, the no-information error rate and the average out-of-bag bootstrap error in `Plot.peperr.curves` by setting the option `allErrors=TRUE`.

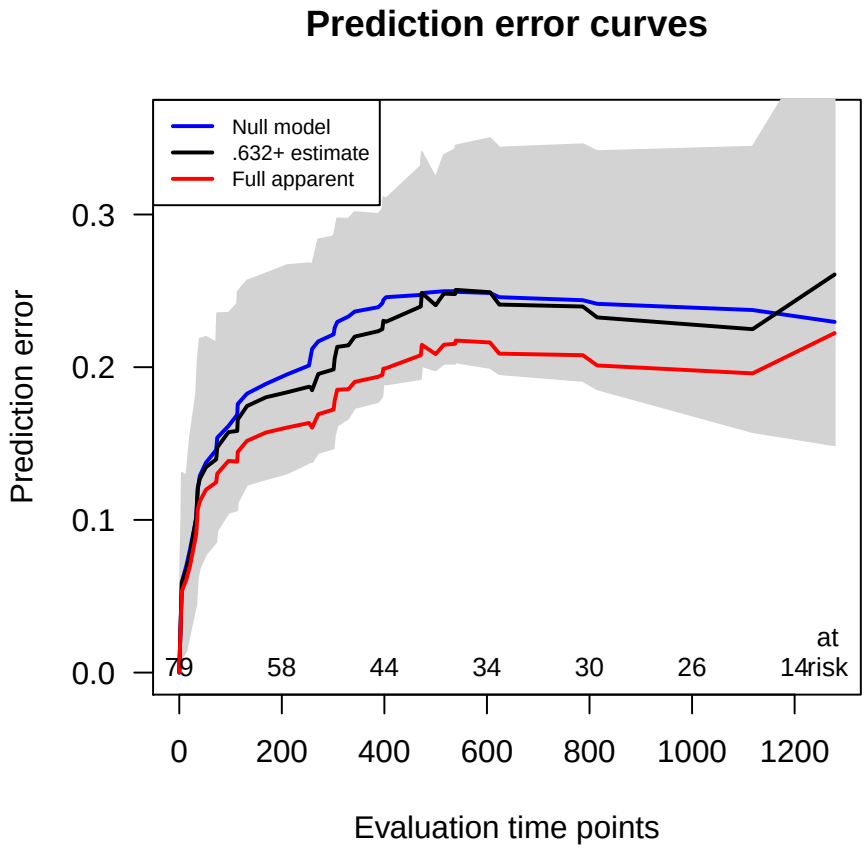

Figure 3: Prediction error curves based on time-dependent Brier score for the lasso-penalized Cox PH regression model applied to the AML data set (evaluation time points reflect days). The gray area indicates the pointwise 2.5% and 97.5% quantiles of the 1000 out-of-bag bootstrap samples. The other lines show the prediction error curves of the null model (estimated by the Kaplan-Meier estimator without covariate information), the full apparent error estimates (i.e., the errors as estimated when applying the model to the entire training data set), and the .632+ bootstrap error estimates.

```
R> Plot.peperr.curves(obj, at.risk=TRUE, allErrors=FALSE, bootRuns=FALSE,
+                     bootQuants=TRUE, bootQuants.level=0.95, leg.cex=0.7)
```

For classification models for binary endpoints, both tuning and fitting of the model are done with the same wrapper functions as used for the Cox regression. Model performance measures for classification tasks that are shipped with the **peperr** package are the misclassification rate and the Brier score. The predicted class probability is calculated within each generic performance function by calling the algorithm-specific predict function. Whenever a new algorithm/method is applied, the generic performance/aggregation function needs to be adapted accordingly. Therefore for binary responses, the **peperr** package does not provide quite the same modular flexibility as for time-to-event endpoints where prediction and performance assessment are done in separate functions. We have extended the functionality of the **peperr** functions for calculating the Brier score (`aggregation.brier`) and the

misclassification rate (`aggregation.misclass`) to allow their use with the **glmnet** algorithm. In addition, we have implemented the area under the receiver operating characteristic (ROC) curve (AUC) (`aggregation.auc`) as an alternative performance measure for binary response classifications. An example is included in the corresponding help file. For classification models, there are numerous alternative R and Bioconductor packages available to assess models fitted with **glmnet**. A good starting point is the package **caret** (**?**).

An alternative implementation of the time-dependent Brier score for assessing the prognostic performance of prognostic models for time-to-event endpoints can be found in the package **pec** (**?**). The basic approach is similar to **peperr**, i.e. one has to define a wrapper for each fitting procedure in order to determine the estimated survival probabilities. Just as for **peperr**, no wrapper for **glmnet** is available yet with the **pec** package. The main difference of **peperr** compared to **pec** is that the tuning of the hyper-parameter and the fitting procedure are done in two separate steps, which reflects the usual workflow for high-dimensional data analysis better. While **pec** provides additional prediction accuracy measures for survival models, such as the time-dependent c-index, it cannot be used to assess classification models for binary endpoints. One strong point of the implementations of both **peperr** and **pec** is their easy-to-use setup for using resampling methods for internal validation. In addition to the use of prediction error curves and time-dependent Brier scores, another popular approach for assessing the prediction accuracy of survival models is the use of time-dependent ROC and AUC curves, for which many implementations exist, for example in R packages **timeROC** (**?**), **survivalROC** (**?**) and **risksetROC** (**?**).

3.4. Stability selection

So far implementations of stability selection can be found in the packages **s4vd**, (**??**), **mboost** (**?**), **lol** (**?**) and **BioMark** (**?**). While in the package **s4vd** the stability selection is applied to sparse singular value decomposition in the context of biclustering, the **mboost** package offers stability selection for model based boosting. The implementation in the **lol** package is based on penalized generalized linear models and penalized Cox models available through the package **penalized** (**?**). As both the **penalized** and the **glmnet** packages offer penalized models for survival, Poisson, binary and Gaussian response variables, the stability selection in **lol** is comparable to our implementation. Due to the computational efficiency of the coordinate descent algorithm in **glmnet** it is more appropriate for complex resampling methods like stability selection. Moreover, the code for the stability selection in the **lol** package is not yet parallelized and does not offer the possibility to compute the whole stability path. The **BioMark** package offers an implementation of stability selection for partial least squares (PLS), principal component regression (PCR), variable importance of projection (VIP) and logistic and Gaussian **glmnet** models. Until now the **BioMark** package allows only to calculate selection frequencies and does not provide type I error control nor the 'randomized lasso' as described in **?**.

Here we use stability selection to identify prognostic features, which have a relevant influence on the survival times of the patients in the AML data set, while controlling Type I errors to ensure that the features identified are truly associated with the survival times. To calculate the stability path for the L_1-penalized Cox regression we use the function `stabpath` from our R package. Via the `weakness` argument of the function `stabpath` it is possible to induce additional randomization by reweighting the penalization of each feature. In brief, in each subsampling step the individual penalization of each feature is randomized such that it lies in the range of $[\lambda, \lambda/\kappa]$, where κ is represented by the `weakness` parameter which indicates the amount of this additional randomization. The weights $w_i, \cdots, w_p$ to

replace each λ_i by λ_i / w_i are generated by sampling from a uniform distribution, i.e. $w_i \sim \mathscr{U}(\kappa, 1)$. **?** call this additional randomization 'randomized lasso' and showed that it greatly improves the variable selection performance of the stability selection. The function `stabpath` draws subsets and calculates in parallel the stability path, e.g., the selection probabilities of each feature along the range of possible penalization parameter values. For parallelization we use the package **parallel**, which has been a base package since R version 2.14.0. On Unix-like systems the parallelization is done by forking via the function `mclapply` whereas under Windows systems socket clusters are used.

```
R> y <- cbind(time=pData(eset)$os, status=pData(eset)$os_status)
R> set.seed(1234)
R> spath <- stabpath(y=y, x=t(exprs(eset)), mc.cores=2,
+                          family="cox", weakness=.8)
```

After calculating the stability path, the function `stabsel` can be called to estimate the stable set of features. Controlling a per-family error rate (PFER) of 1, e.g., expecting one falsely selected feature, the estimated set of stable features comprises a single feature (with $\hat{\Pi} > 0.6$).

```
R> stabsel(spath, error=1, type="pfer", pi_thr=0.6)$stable

206932_at
     2823
```

Alternatively, the `stabsel` function allows to control the per-comparison wise error rate (PCER) and family-wise error rate (FWER). In addition, we provide a `plot` function to visualize the stability path. This function calls `stabsel` to estimate stable features and indicates them in the plot (Figure 4).

3.5. Parameter tuning for the elastic net Cox model

In the previous sections we have seen that the lasso-penalized Cox model does not seem to perform very well in terms of predicting overall survival for the AML data set. The lasso model identified as the optimal model by 10-fold cross-validation is very sparse and contains only 5 features. Furthermore, we have observed that these features are not very stable and 4 of them do not even remain in the set of selected features when the amount of regularization is decreased and more features start to enter the model.

In this section we fit an elastic net model instead of lasso to the same data set. As outlined above, fitting an elastic net model requires the simultaneous tuning of two parameters α and λ. For this computationally challenging task, we use the interval search algorithm in an efficient implementation in R function `EPSGO`. The EPSGO algorithm was originally implemented for support vector machines in the R package **penalizedSVM** (**??**). Here we provide a version for `glmnet` and in addition `summary` and `plot` functions to illustrate the interval search results.

The following code specifies the required objects and parameter values for optimizing the tuning parameters of the elastic net Cox model. The `balancedFolds` function splits the data into balanced folds for 10-fold cross-validation.

```
R>   x <- t(exprs(eset))
R>   y <- cbind(time=pData(eset)$os,status=pData(eset)$os_status)
```

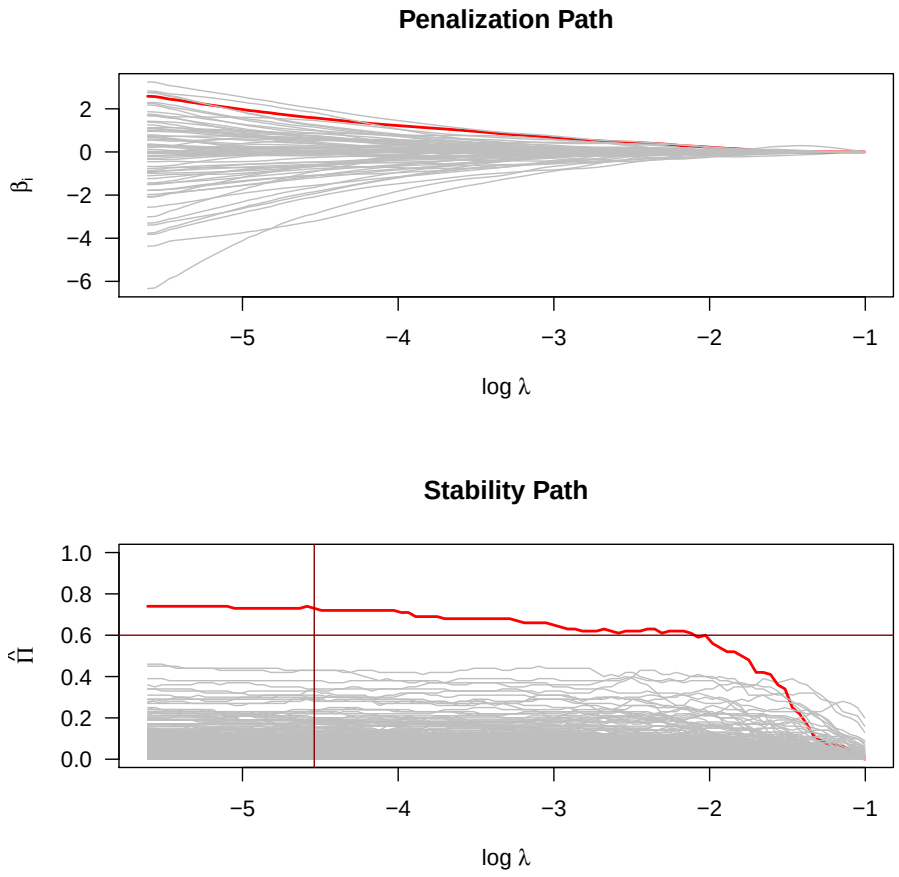

Figure 4: Coefficient and stability paths for lasso penalized Cox PH regression model applied to the AML data set. The feature with highlighted path is the only stable feature found by stability selection with PFER=1 and $\hat{\Pi} > 0.6$.

```
R>   bounds <- t(data.frame(alpha=c(0, 1)))
R>   colnames(bounds)<-c("lower","upper")
R>   nfolds <- 10
R>   set.seed(1234)
R>   foldid <- balancedFolds(class.column.factor=y[,2], cross.outer=nfolds)
```

Usually, the task is to find a setting of tuning parameter values (α, λ), for which the 10-fold cross-validated penalized (partial) log likelihood deviance of the model is minimal. Here, however, the optimal λ is chosen as the largest value of λ such that the loss function is within one standard error of the minimum, which will result in a smaller model (a strategy suggested by the authors of the **glmnet** package). That is, for each given α an optimal λ is found via the computation of the entire regularization path with the `glmnet` function with option `type.min ='lambda.1se'`.

The wrapper function `tune.glmnet.interval` calculates the (partial) log likelihood deviance of a model with given tuning parameter setting (α, λ).

```
R>   fit <- epsgo(Q.func="tune.glmnet.interval",
+                bounds=bounds,
+                parms.coding="none",
+                seed = 1234,
+                fminlower = -100,
+                x = x, y = y, family = "cox",
+                foldid = foldid,
+                type.min = "lambda.1se",
+                type.measure = "deviance")
```

Summary information can be extracted from the `fit` object using the `summary` function.

```
R> sumint <- summary(fit, verbose=TRUE)

Summary interval search

show the first 5 out of 37 entries
    alpha    lambda deviance n.features
1 0.67605 0.4939544 9.711522          1
2 0.17194 1.5391420 9.662976         19
3 0.81895 0.4077635 9.708716          1
4 0.31188 0.9756053 9.699170          4
5 0.61671 0.5168716 9.700599          2

.............................

 Optimal parameters found are:

alpha =  0.013          lambda =  14.722 deviance =  9.6329
```

At the initial step we sample 21 points in the parameter space for α as suggested for the original algorithm by the authors of (**?**). Those points are randomly distributed and uniformly cover the whole interval (0,1]. A Gaussian process model is trained based on these initial points. Then, iteratively, new points are added to the Gaussian process model in order to find an optimal combination of tuning parameter values. In total, 37 iterations were needed to reach the optimum.

The final elastic net model contains 220 selected features, which obviously reflects much less sparsity than the final lasso model. The results are consistent in the sense, that the features contained in the final lasso model are also contained in the elastic net model. Also, the individual feature selected by the stability algorithm is in the set of selected elastic net features.

Figure 5 illustrates the relationship between both tuning parameters α and λ for the 'visited' points in the parameter space. The partial log likelihood deviance is color-coded with black for small values and gray for large values. The number of features selected in the corresponding model is written near each point. To distinguish between initial and iteration points, the initial points are plotted as squares and iteration points as circles. One can observe that the iteration points were chosen in the regions with lower deviance values.

The distribution of initial points (iteration=0) and visited points (iteration>0) in the parameter space is plotted in Figure 6. This plot shows nicely that the interval search algorithm does not sequentially

```
R> plot(sumint)
```

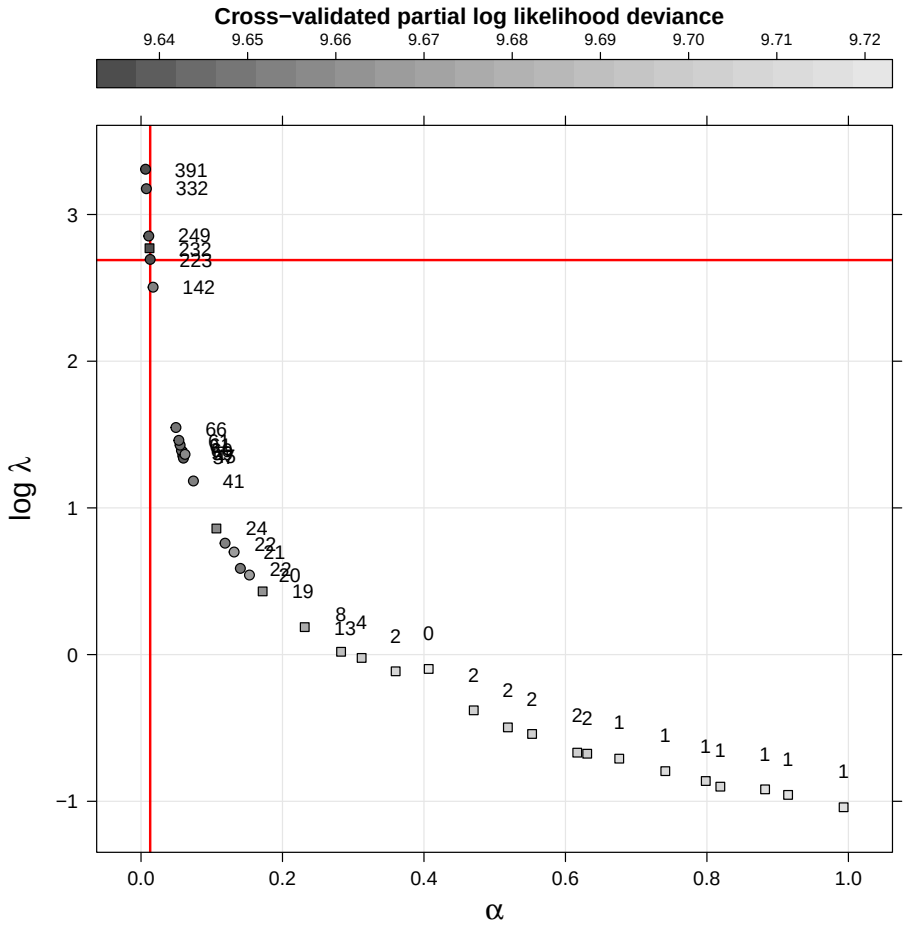

Figure 5: Cross-validated partial log likelihood deviance as a function of both tuning parameters α and $\log\lambda$ when fitting the elastic net Cox model for the AML data set. For each evaluated point in the parameter space the number of selected features in the corresponding model is printed next to the data point symbol. Rectangles correspond to initially selected α values. The solid red lines highlight the final solution where the loss function is within one standard error of the minimum.

cover the entire parameter space, but rather quickly finds promising regions and draws new samples there. The optimal model contains with minimal log-likelihood deviance is found for $\alpha = 0.013$ $\log\lambda = 2.689$ and highlighted as a vertical line.

To our knowledge, the interval search algorithm has not yet been applied for the purpose of optimizing the tuning parameters of elastic net models fitted with the **glmnet** package. The only previous R implementation of this approach was in the R package **penalizedSVM**, which implements classification and simultaneous feature selection with support vector machines.

```
R> plot(sumint,type="points")
```

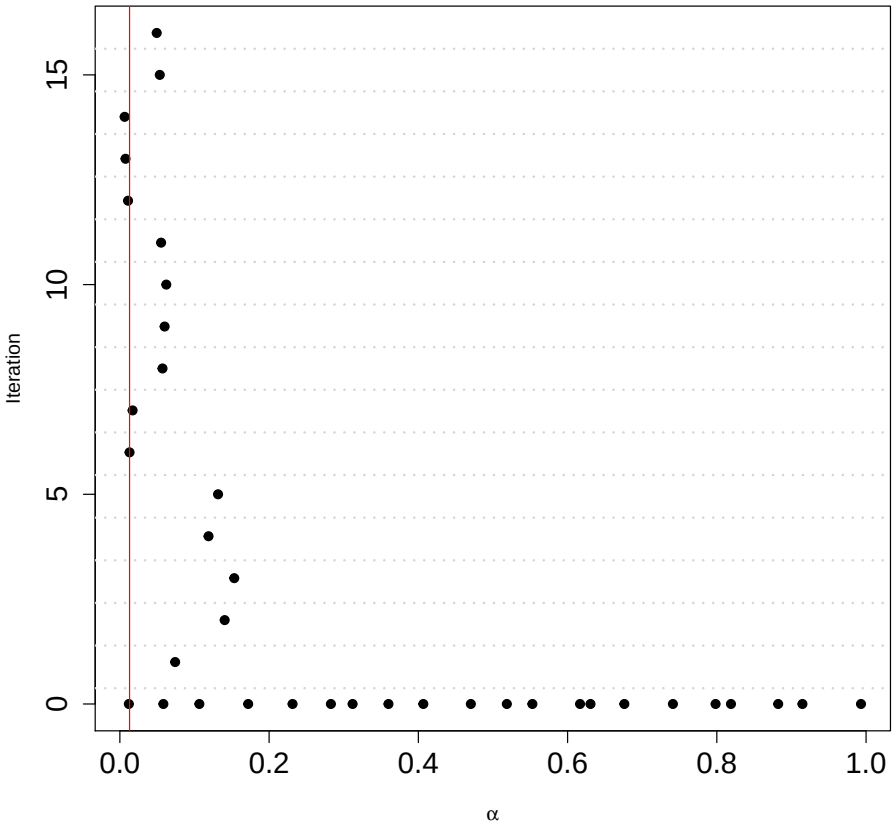

Figure 6: The distribution of initial and visited points of the interval search plotted in chronological order. The interval search is employed to identify the optimal parameter value combination (α,λ) for the elastic net Cox model fitted to the AML data set.

4. Conclusions and outlook

The programming language and statistical computing environment R provides a highly useful framework for statistical data analysis and modelling. It is the dominating statistical software in many areas, for example in molecular biology and molecular medicine, which is largely due to the highly successful Bioconductor project (**?**), which provides tools for the analysis and interpretation of high-throughput genomic data. Due to the open-source nature of R and Bioconductor, many useful software packages have been developed by R users and made available for the R community. One example is the **glmnet** package, which implements an efficient state-of-the-art algorithm for fitting penalized Cox and generalized linear models with lasso, ridge or elastic net penalties.

We have presented our R package **c060**, which provides extensions to **glmnet** and additional features, which are essential for a complete data analysis in real-life applications, including stability selection, estimation of prediction error (curves) and an efficient interval search algorithm for finding the optimal

Function	Description
`Plot.coef.glmnet`	Plot the `glmnet` coefficient path and highlight the path of a pre-specified set of variables
`PLL.coxnet`	Predictive partial log-likelihood for `glmnet` Cox PH model fit
`aggregation.auc`	Determine the area under the ROC curve for a fitted model
`complexity.glmnet`	Interface for determination of penalty `lambda` in `glmnet` models via cross-validation
`fit.glmnet`	Interface function for fitting a penalized regression model with `glmnet`
`Plot.peperr.curves`	Plot method for prediction error curves of a `peperr` object
`predictProb.coxnet`	Extract predicted survival probabilities from a `coxnet` fit
`predictProb.glmnet`	Extract predicted survival probabilities from a `glmnet` fit
`stabpath`	Calculate the stability path for Gaussian, binomial, Poisson, multinomial and Cox `glmnet` models
`stabsel`	Estimate a stable set of variables and allows to control the PFER, PCER or FWER
`plot.stabpath`	Display stability path and indicates estimated stable features
`epsgo`	Efficient Parameter Selection via Global Optimization
`summary.intsearch`	Summary method for interval search models
`tune.glmnet.interval`	Wrapper function to apply `epsgo` to `glmnet` objects
`plot.sum.intsearch`	Plot `sum.intsearch` objects generated by `summary.intsearch`

Table 1: Overview of available functions in the **c060** package.

elastic net tuning parameter combination. These extensions have proved useful in our daily work, in particular for the task of performing prognostic modelling of patient survival data based on high-dimensional molecular biology data. Table 1 lists all functions that are available as part of the **c060** package.

The **c060** package will be kept updated in the future to keep up with new developments in the field of penalized regression methodology for feature selection and risk prediction modelling with high-dimensional input data. One example are developments for the estimation of standard errors, confidence intervals and the determination of p-values in high-dimensional regularized regression models, e.g., through subsampling methods similar to the approach taken by **?** and **?**.

Acknowledgments

This work was partially funded by the Virtual Helmholtz Institute VH-VI-404.

Affiliation:

Martin Sill
Division of Biostatistics
DKFZ
German Cancer Research Center
69120 Heidelberg, Germany
E-mail: m.sill@dkfz.de
URL: http://www.dkfz.de/en/biostatistics