

# 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  
German Cancer Research Center

---

## 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** 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 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 applicable to all types of regression models implemented in the **glmnet** package.

*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 (Fan and Lv 2010; Benner, Zucknick, Hielscher, Ittrich, and Mansmann 2010). Since the introduction of the lasso for linear regression models (Tibshirani 1996), the methodology has been extended to generalized linear regression models and time-to-event endpoints (Tibshirani 1997) among others. In addition to the well-known  $L_1$ - (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 (Zou and Hastie 2005), 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. In 2010, Friedman et al. proposed a coordinate descent algorithm (Friedman, Hastie, and Tibshirani 2010) for generalized linear regression models,

which has since then been extended to penalized Cox proportional hazards (PH) regression models (Simon, Friedman, Hastie, and Tibshirani 2011). 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.

This algorithm has been implemented in R (R Development Core Team 2011) 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 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 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 (Meinshausen and Bühlmann 2010) 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 ( $\alpha$ ,  $\lambda$ ) for elastic net penalties using an interval-search algorithm (Froehlich and Zell 2005) which is often faster and more accurate than a standard grid search (Jones, Schonlau, and Welch 1998). 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** (Porzelius, Binder, and Schumacher 2009). 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**.

The software is available as R package **c060** on R-forge (URL <http://c060.r-forge.r-project.org>).

## 2. Data application

Throughout this article we use the gene expression data set of cytogenetically normal acute myeloid leukemia (AML) patients by Metzeler, Hummel, Bloomfield, Spiekermann, Braess, Sauerland, Heinicke, Radmacher, Marcucci, Whitman, Maharry, Paschka, Larson, Berdel, Buchner, Wormann, Mansmann, Hiddemann, Bohlander, and Buske (2008) and corresponding clinical data in order to illustrate a typical application of penalized Cox PH regression models, where the aim is to develop a prognostic model for patient survival while at the same time identifying the most influential gene expression features. 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 by Metzeler *et al.* (2008). 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 Metzeler *et al.* data set under GEO accession number GSE12417.

The 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%.

### 3. Methods and algorithms

#### 3.1. 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** (Friedman *et al.* 2010; Simon *et al.* 2011). 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  $\Lambda = \{\lambda_1, \lambda_2, \dots, \lambda_K\}$  at fixed  $\alpha$  with **glmnet** is generally very fast, because previously computed solutions for  $\{\lambda_1, \dots, \lambda_{k-1}\}$  are used as 'hot' starting values for the computation of  $\lambda_k$ . This implies that it is often more efficient to compute the models for the entire regularization path of  $\Lambda$  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

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

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

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.

#### 3.2. $L_2$ -penalized Cox regression

Penalized maximum likelihood estimation in Cox regression with the ridge penalty

$$p_\lambda(|\beta_j|) = \lambda \beta_j^2 \quad (2)$$

was introduced by Verweij and van Houwelingen (1994). The ridge penalty results in parameter estimates that are biased towards zero, but does not set values to exactly 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. Bøvelstad, Nygård, Størvold, Aldrin, Borgan, Frigessi, and Lingjærde 2007), in particular if predictors are highly correlated.

### 3.3. $L_1$ -penalized Cox regression

Tibshirani (1997) proposed to use an  $L_1$ -penalized Cox model with

$$p_\lambda(|\beta_j|) = \lambda|\beta_j| \quad (3)$$

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.

### 3.4. The elastic net

Zou and Hastie (2005) introduced the elastic net, which employs a combination of the  $L_1$ - and  $L_2$ -penalty

$$p_{\lambda^1, \lambda^2}(|\beta_j|) = \lambda^1|\beta_j| + \lambda^2\beta_j^2. \quad (4)$$

Zou and Hastie (2005) rescale the initial solutions from the optimization of the doubly-penalized log-likelihood function by the factor  $1 + \lambda^2$ , in order to reduce the effect of the double shrinkage. 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, such that the elastic net tends to select more features than the lasso. This is especially the case in situations with high correlation, where the lasso would select only one feature of a set of highly correlated features, while the ridge penalty would give them equal weight.

Throughout this manuscript we use an alternative parametrization of the elastic net penalty function equivalently to the formulation used in the **glmnet** package:

$$p_{\alpha, \lambda}(|\beta_j|) = \lambda \times (\alpha|\beta_j| + (1 - \alpha)\frac{1}{2}\beta_j^2). \quad (5)$$

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

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

The elastic net penalty function constrains 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  $(\alpha, \lambda)$ , for which the k-fold cross-validated loss function of the model is minimal.

**Fixed grid** Tuning parameters are usually determined by a grid search. The grid search method calculates a target value, e.g. the penalized log likelihood deviance, at each point over a fixed grid of parameter values. This method may offer some protection against local minima but it is not very efficient. The density of the grid plays a critical role in finding global optima. By increasing the density of the grid, the computation cost increases rapidly with no guarantee of finding global optima. Furthermore, the choice of the fixed grid is highly arbitrary and does not guarantee that the algorithm will not get stuck in a local optimum.

**Interval search** Froehlich and Zell (2005) 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 Jones *et al.* (1998), which avoids getting stuck in local minima - How? Is this guaranteed? If not rephrase. The EPSGO algorithm stops when the global optimum is achieved.

One can observe an immense improvement in the training time for the Gaussian process model compared to that of the underlying classification and regression tasks - Should we not compare with grid search here? The underlying task is finding the optimal tuning parameter values, not the classification task itself (Froehlich and Zell 2005). 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 parameter space, which is very small compared to the number of training points for the regression model - Again: should we not compare with the number of training points = evaluated tuning parameter combinations used in the grid search?.

### 3.5. 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 Meinshausen and Bühlmann (2010) 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. Meinshausen and Bühlmann (2010) 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  $\Lambda = \{\lambda_1, \lambda_2, \dots, \lambda_K\}$ , is called stability path. Given an arbitrary threshold  $\pi_{thr} \in (0.5, 1)$  and the set of penalization parameters  $\Lambda$ , the set of stable features estimated with stability selection is:

$$\hat{S}_\beta^{stable} = \left\{ j : \max_{\lambda_k \in \Lambda} \hat{\Pi}_j^{\lambda_k} \geq \pi_{thr} \right\}, \quad (6)$$

where  $\hat{\Pi}_j^{\lambda_k}$  denotes the estimated selection probability of the  $j$ th feature at  $\lambda_k$ . Then according to Theorem 1 in Meinshausen and Bühlmann (2010), the expected number of falsely selected features  $E(V)$  will be bounded by:

$$E(V) \leq \frac{1}{(2\pi_{thr} - 1)} \frac{q_\Lambda^2}{p}, \quad (7)$$

where  $q_\Lambda$  is the average of the number of non-zero coefficients w.r.t. to the drawn subsamples. Interpreting Equation 7 the expected number of falsely selected features decreases by either reducing the average number of selected features  $q_\Lambda$  or by increasing the threshold  $\pi_{thr}$ . Suppose that  $\pi_{thr}$  is fixed,

then the stability selection controls  $E(V)$  as long as the average number of selected features is less than the upper bound

$$e_\Lambda = \sqrt{E(V)p(2\pi_{thr} - 1)}, \quad (8)$$

which can be controlled by reducing the length of the regularization path  $\Lambda$ . The equation for  $e_\Lambda$  contains  $E(V)$ . How is this useful? Could we just remove this sentence? 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) (Dudoit, Shaffer, and Boldrick 2003). The stability selection allows to control these Type I error rates. For instance, suppose the threshold  $\pi_{thr} = 0.8$  is fixed, then choosing  $\Lambda$  such that  $q_\Lambda \leq \sqrt{0.6p}$  will control  $E(V) \leq 1$ . Moreover, by choosing  $\Lambda$  so that  $q_\Lambda \leq \sqrt{0.6p\alpha}$  will control the family wise error rate (FWER) at level  $\alpha$ ,  $P(|V| > 0) \leq \alpha$ .

As mentioned before, according to Friedman *et al.* (2010) the coordinate descent algorithm implemented in the **glmnet** package is most efficient regarding the computational time, when used to calculate a whole 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 merged - How? Could we say “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**.

### 3.6. Prediction error curves for survival models

The time-dependent Brier score (Graf, Schmoor, Sauerbrei, and Schumacher 1999) can be used to assess and compare the prediction accuracy of prognostic models. 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 or more general estimates thereof - More general estimates of what? Remove this phrase? (Gerds and Schumacher 2006). 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  $\delta_i$  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 (Graf *et al.* 1999).  $\hat{G}(t)$  denotes the Kaplan-Meier estimate of the censoring distribution at time  $t$ , which is based on the observations  $(t_i; 1 - \delta_i)$ ,  $i = 1, \dots, n$ .  $I$  stands for 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 (Efron and Tibshirani 1997), which is a weighted sum of the apparent error and the average out-of-bag bootstrap error - This does not really help without further explanation of “apparent” and “out-of-bag”. Could we just remove this?, can be used in this context, balancing a too optimistic and a too conservative error estimation. The .632+ bootstrap estimator is calculated based on subsampling (with replacement) rather than classical sampling without replacement, as that has been demonstrated to lead to more accurate estimates in a high-dimensional context (Binder and Schumacher 2008).



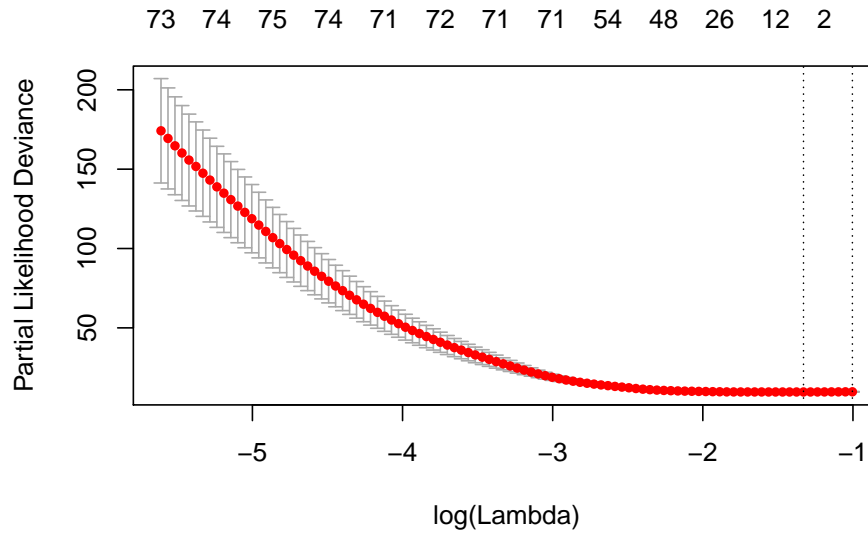


Figure 1: Cross-validated partial log-likelihood deviance, including upper and lower standard deviations, as a function of  $\log \lambda$  for the AML data set. The dotted vertical lines indicate the  $\lambda$  values with minimal deviance (left) and with the largest  $\lambda$  value within one standard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

## 4. Application and demonstration of software

In the following we will demonstrate the use of the functions provided with the **c060** package in an application to the acute myeloid leukemia (AML) data set by [Metzeler \*et al.\* \(2008\)](#). 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** ([Gentleman, Carey, Bates, and others 2004](#))) 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)`.

### 4.1. 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  $\lambda$  values with 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. 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  $\lambda = 0.265$  ( $\log \lambda = -1.329$ ) and corresponds to a final lasso model with the following 5 selected features and 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 illustrating the development of the regression coefficient estimates with increasing regularization. While the selected 5 features are the only features selected at the optimal  $\lambda$  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.

## 4.2. Assessment of prediction performance with resampling-based prediction errors

Once the final prognostic model is selected, we need to assess its prediction accuracy for future patients, frequently also in comparison with established clinico-pathological prognostic markers. In many applications no independent validation data set is available, thus the same data set needs to be used to both develop and assess the prognostic model. This is even more problematic for high-dimensional data, where the risk of overfitting is especially 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** (Porzelius *et al.* 2009) provides a modular framework for survival and binary endpoints, i.e. prognostic and classification models. 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 unfortunately 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.

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 100 bootstrap samples for illustration purposes (usually one would choose larger numbers of bootstrap samples, e.g. 500 or 1000). - This makes sense for the vignette, but not for the JSS paper, where running time is no



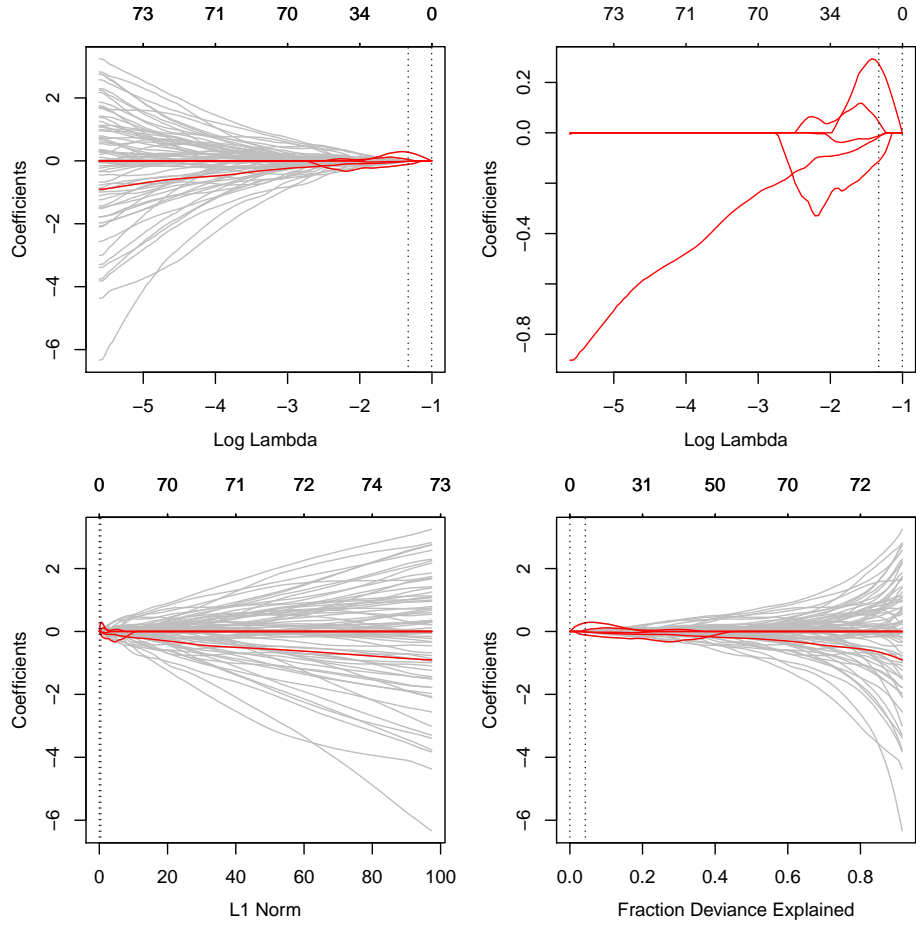


Figure 2: Coefficient paths for lasso-penalized Cox PH regression models applied to the AML data set. The features with highlighted paths have non-zero coefficients in the model with the optimal  $\lambda$  value as determined by ten-fold cross-validation. The dotted vertical have the same meaning as in Figure 1. 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).

problem. Should we rerun this with more bootstrap samples? We could also include a comparison of computing times with and without parallelisation to demonstrate how useful parallelisation is.

The **peperr** package is designed for high-dimensional covariates data and allows for various set-ups of parallel computations. Also, additional arguments can be passed directly to the **glmnet** call by specifying additional arguments for the fitting and/or tuning procedure. Here, we include patient's age as mandatory model feature into the prognostic model, i.e. age is not subject to penalization, and run the calculation on 3 CPUs in parallel using a socket cluster set-up. Where in the function call do we actually specify that we use .632+ bootstrap?

```
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=F, family="cox",
+                             penalty.factor=rep(0:1, times=c(1,dim(eset)[1]))),
+               complexity=complexity.glmnet,
+               args.complexity=list(standardize=F, nfolds=10,
+                                    family="cox",
+                                    penalty.factor=rep(0:1, times=c(1,dim(eset)[1]))),
+               trace=F, RNG="fixed", seed=0815,
+               cpus=3, parallel=T, clustertype="SOCK",
+               load.list=list(packages=c("c060")),
+               indices=resample.indices(n=dim(eset)[2], sample.n=100,
+                                       method="sub632"))
```

Individual 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 defined.

In addition, we provide a slightly modified version of the prediction error curves plot function from the **peperr** package which allows to display the number still at risk (`plot.peperr.curves`) as shown in Figure 3.

Note, that for classification models, the same wrapper functions for fitting and tuning the model are called. Model performance measures for classification tasks shipped with the **peperr** package are misclassification rate and Brier score. We have extended the functionality of the Brier score (`aggregation.brier`) and misclassification rate (`aggregation.misclass`) calculation for the **glmnet** algorithm, and defined AUC under the ROC curve (`aggregation.auc`) as an additional performance measure. For binary responses, the **peperr** package does not quite provide the same modular flexibility as for time-to-event endpoints. The predicted class probability is calculated within the performance/aggregation function by calling the algorithm-specific predict function. Whenever a new algorithm is incorporated, the aggregation function has to be modified and overwritten accordingly.

```
R> plot.peperr.curves(obj, at.risk=T)
```

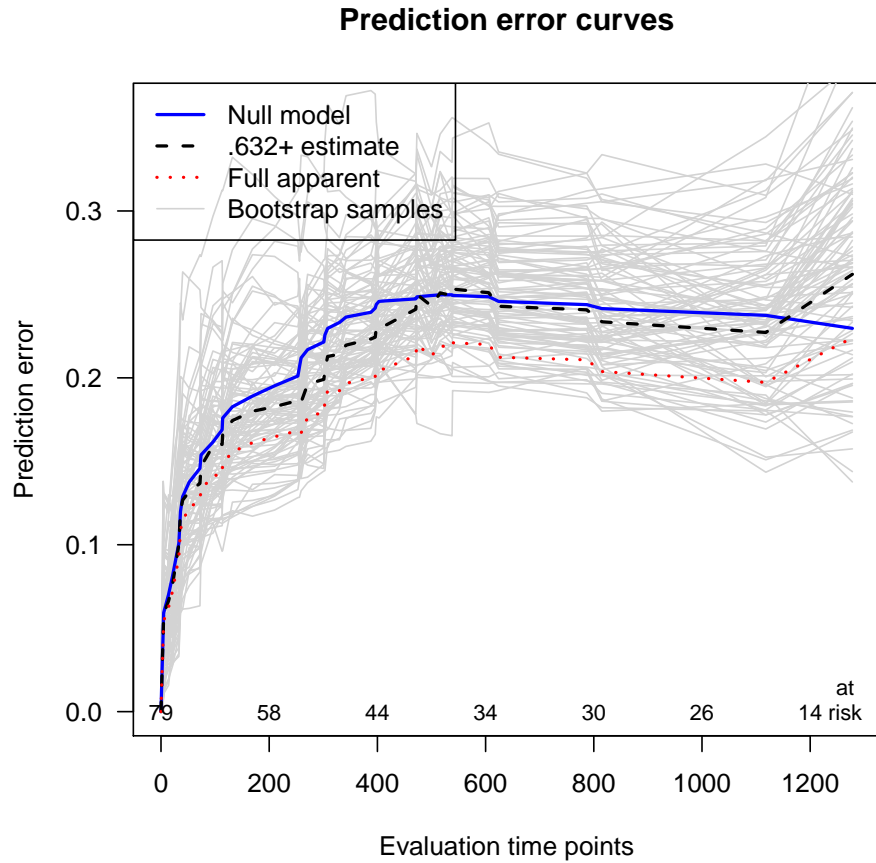


Figure 3: Prediction error curves based on time-dependent Brier score for the lasso-penalized Cox PH regression model applied to the AML data set (evaluation time points reflect days). The gray lines show the prediction error curves of 100 bootstrap samples. The other lines show the prediction error 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.

### 4.3. Stability selection

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 and to control 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 `stability.path` from our R package. The function 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> set.seed(1234)
R> y <- cbind(time=pData(eset)$os, status=pData(eset)$os_status)
R> spath <- stability.path(y=y, x=t(exprs(eset)), mc.cores=2, family="cox")
```

The function `stability.selection` can be called to estimate the stable set of features. Controlling a family-wise error rate (FWER) of 0.5 the estimated set of stable features comprises a single feature (with  $\hat{\Pi} > 0.6$ ).

```
R> stability.selection(spath,fwer=0.5)$stable

206932_at
      2823
```

Note that an FWER threshold of 0.5 is unusually high. For this data set the use of smaller, more conventional thresholds, e.g. 0.05 or 0.1, will result in an empty set of stable features. The stability path can be visualized by the function `plot.stabpath`, which will call `stability.selection` to identify stable features and highlight them in the plot.

Note that an FWER threshold of 0.5 is unusually high. For this data set the use of smaller, more conventional thresholds, e.g. 0.05 or 0.1, will result in an empty set of stable features. The stability path can be visualized by the function `plot.stabpath`, which will call `stability.selection` to identify stable features and highlight them in the plot.

```
R> par(mar=c(4,4,2,1), mgp=c(2.5,1,0))
R> plotstabpath(spath, fwer=0.5, pi_thr=0.6, xvar="lambda", col.all="gray")
```

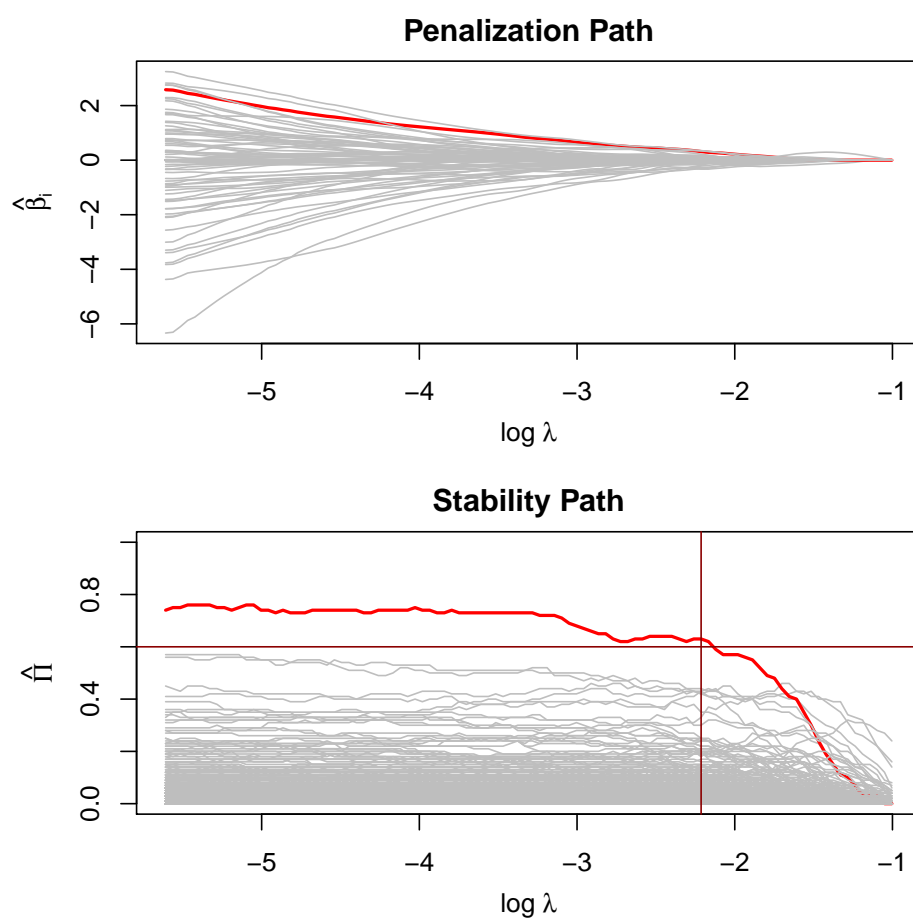


Figure 4: Coefficient and stability paths for lasso penalized Cox PH regression model applied to the AML data set. The feature with highlighted path is the only stable feature found by stability selection with FWER=0.5 and  $\hat{\pi} > 0.6$ .

```
R> par(mar=c(4,4,2,1), mgp=c(2.5,1,0))
R> plotstabpath(spath, fwer=0.5, pi_thr=0.6, xvar="lambda", col.all="gray")
```

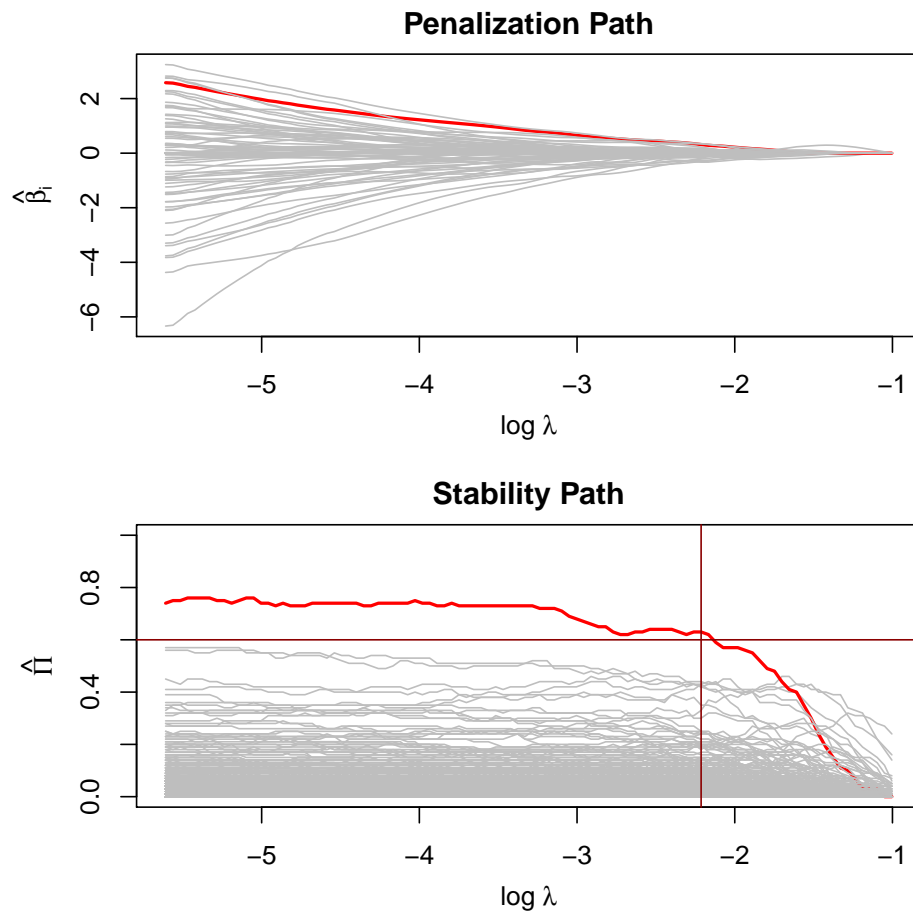


Figure 5: Coefficient and stability paths for lasso penalized Cox PH regression model applied to the AML data set. The feature with highlighted path is the only stable feature found by stability selection with FWER=0.5 and  $\hat{\pi} > 0.6$ .



#### 4.4. 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. While the lasso model identified as the optimal model by 10-fold cross-validation is very sparse and contains only 5 features, we have seen that these features are not very stable and 4 out of these features 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. Also, the .632+-bootstrap estimated prediction error curve indicates that the prognostic value of the lasso model does not seem to be very good.

In this section we try to fit an elastic net model instead of lasso to the same data set. As outlined above, fitting an elastic net model require the simultaneous tuning of two parameters  $\alpha$  and  $\lambda$ . For this computationally challenging task, we provide an efficient implementation in R function EPSGO using the interval search algorithm.

```
R> x <- t(exprs(eset))
R> y <- cbind(time=pData(geo_exprs_data)$os,status=pData(geo_exprs_data)$os_status)
R> Q.func <- "tune.glmnet.interval"
R> # bounds for alpha [0,1]
R> bounds <- t(data.frame(alpha=c(0, 1)))
R> colnames(bounds) <- c("lower", "upper")
R> parms.coding <- "none"
R> seed <- 1234
R> type.measure <- 'deviance'
R> family <- "cox"
R> nfolds <- 10
R> show <- "none"
R> fminlower <- -100
R> type.min <- "lambda.1se"
R> verbose <- TRUE
```

The task is to find a setting of tuning parameter values  $(\alpha, \lambda)$ , for which the 10-fold cross-validated penalized (partial) log likelihood function of the model is minimal. The parameter space of tuning parameters is defined as follows:

	lower	upper
alpha	0	1

The second tuning parameter  $\lambda > 0$  will be found for each given  $\alpha$  via the computation of the entire regularization path with the **glmnet** function. The two-dimensional parameter space has the form  $(0, 1) \times \mathbb{R}_+$ . For each given  $\alpha$  an optimal  $\lambda$  is defined as the largest value of  $\lambda$  such that the loss function value is within one standard error of the minimum.

```
R> # fix folds for each model
R> set.seed(seed)
R> foldid <- my.balanced.folds(class.column.factor=y[,2], cross.outer=nfolds)
R> print("start interval search")
R> fit <- EPSGO(Q.func,
```

```

+         bounds=bounds,
+         parms.coding=parms.coding,
+         show=show, N=NULL,
+         seed=seed,
+         fminlower=fminlower,
+         # Q.func arguments
+         x=x, y=y, family=family,
+         #nfolds=nfolds,
+         foldid=foldid,
+         type.min=type.min,
+         type.measure=type.measure,
+         verbose=verbose)
R> names(fit)
R> print("chose the model with min num of FS ")
R> print("# FS: ")
R> sel.models <- sapply(fit$model.list, "[", "model") [fit$Ytrain == fit$fmin ]
R> sel.alpha <- fit$xmin
R> sel.error <- fit$fmin
R> out <- list(model=sel.models, alpha=sel.alpha, error=sel.error)
R> save(fit, out, file="Rdata/fit_interval_search.RData")

R> #
R> alphas <- fit$Xtrain[,1]
R> lambdas <- unlist(sapply(sapply(fit$model, "[", "model"), "[", "lambda"))
R> deviances <- fit$Ytrain
R> # round problems!!!! take first from the fit
R> # number of selected features in the models; dfs
R> tmp.models<-sapply(sapply(sapply(fit$model, "[", "model"), "[", "cvreg"), "[", "glmnet.
R> n.features<-mapply( function(List, lam) List$df[which(List$lambda %in% lam)], tmp.model
R> opt.alpha <- out$alpha
R> opt.lambda <- out$model[[1]]$lambda
R> summary_int_search <- as.data.frame(cbind(alpha=alphas,lambda=lambdas,deviance=deviance
R> rownames(summary_int_search) <- c(1:nrow(summary_int_search))

```

At the initial step we sample 21 points in the parameter space, Those points are randomly distributed and uniformly cover the whole interval. 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 223 selected features, which obviously reflects much less sparsity than the final lasso model.

```
R> summary(cofn)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
	-0.0097260	-0.0009488	0.0002751	0.0004659	0.0018040	0.0137500

```
R> head(sort(cofn))
```

```
      203640_at      235556_at      217147_s_at      202478_at      213539_at      220030_at
-0.009726168 -0.008236617 -0.005984097 -0.005580783 -0.005292165 -0.005179050
```

```
R> tail(sort(cofn))
```

```
      201951_at      235911_at      220637_at      226169_at      203560_at      222462_s_at
0.005978883 0.006124166 0.007558411 0.007911658 0.008278894 0.013750606
```

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.

```
R> names(cofn.lasso) %in% names(cof)
```

```
[1] TRUE TRUE TRUE TRUE TRUE
```

```
R> '206932_at' %in% names(cof)
```

```
[1] TRUE
```

Figure 6 illustrates the relationship between both tuning parameters  $\alpha$  and  $\lambda$  for the 'visited' points. The partial log likelihood deviance is color-coded with black for small values and gray for large values. To distinguish between initial and iteration points, the initial points are plotted as squares and iteration points as circles. The iteration points were chosen in the regions with lower deviance values. The number of features selected in the corresponding model is written near each point.

The distribution of initial points (iteration=0) and visited points (iteration>0) in the parameter space is plotted in Figure 7). This plot shows nicely that the interval search algorithm does not sequentially cover the entire parameter space, but rather quickly finds promising regions and draws new samples in those promising regions. The optimal model is found at a small value for  $\alpha = 0.01298$  highlighted as a vertical line.

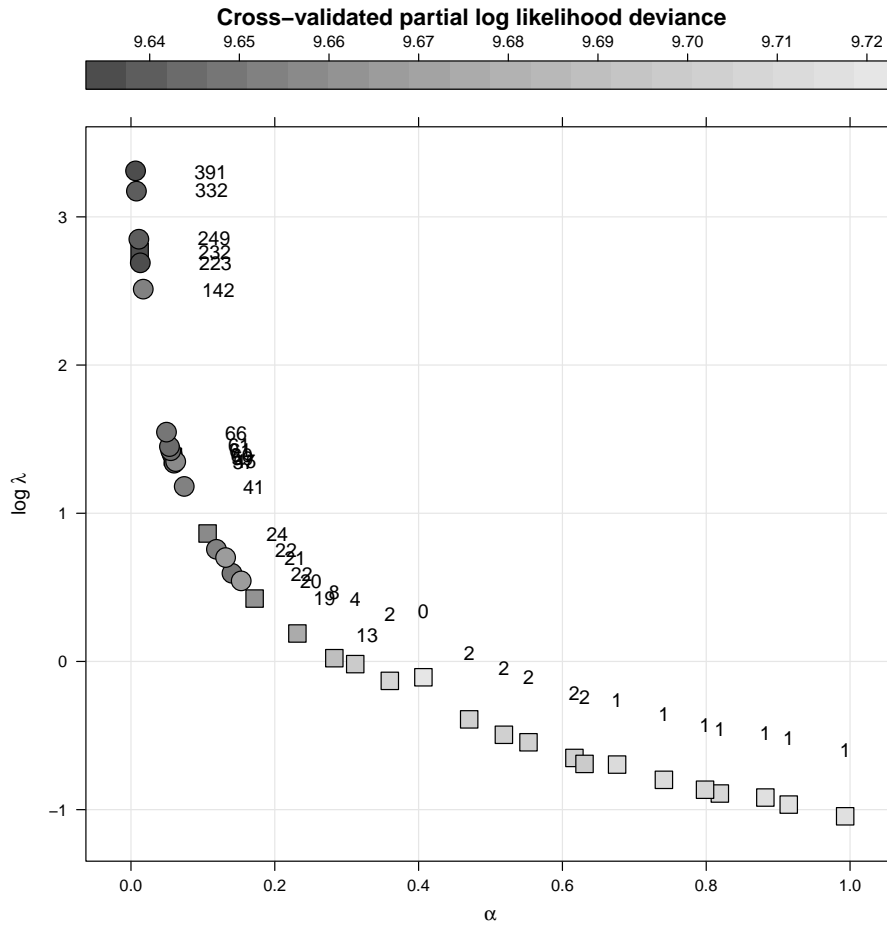


Figure 6: Partial log likelihood deviance as a function of both tuning parameters  $\alpha$  and  $\log \lambda$  when fitting the elastic net Cox model for the AML data set. For each evaluated point in the parameter space the number of selected features in the corresponding model is printed next to the data point symbol. Rectangles correspond to initially selected  $\alpha$  values.

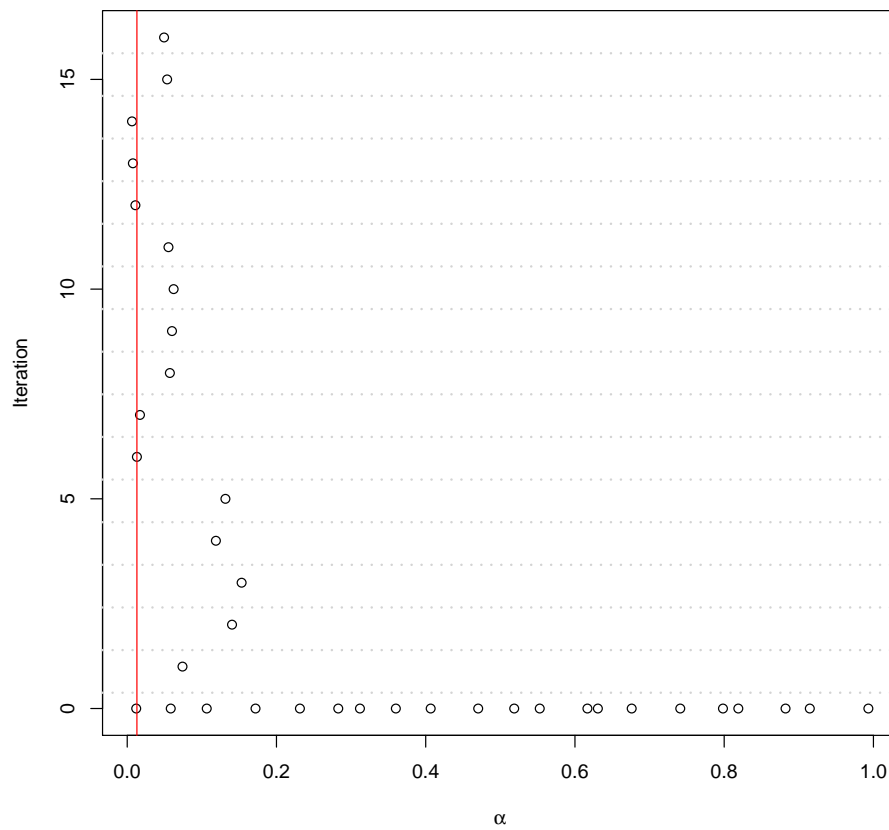


Figure 7: The distribution of initial and visited points of the interval search plotted in chronological order. The interval search is employed to identify the optimal parameter value combination  $(\alpha, \lambda)$  for the elastic net Cox model fitted to the AML data set.

## 5. 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 ([Gentleman \*et al.\* 2004](#)), which provides tools for the analysis and comprehension of high-throughput genomic data. Due to the open-source nature of R and Bioconductor, many very useful software packages have been developed by R users and made available for the entire 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 ([Friedman \*et al.\* 2010](#); [Simon \*et al.\* 2011](#)).

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

The **c060** package will be kept updated in the future to keep up with the fast-developing 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 [Wasserman and Roeder \(2009\)](#) and [Meinshausen and Bühlmann \(2010\)](#).



## References

- Benner A, Zucknick M, Hielscher T, Ittrich C, Mansmann U (2010). “High-dimensional Cox models: The choice of penalty as part of the model building process.” *Biometrical Journal*, **52**(10), 50–69.
- Binder H, Schumacher M (2008). “Adapting prediction error estimates for biased complexity selection in high-dimensional bootstrap samples.” *Statistical Applications in Genetics and Molecular Biology*, **7**(1).
- Bøvelstad HMM, Nygård S, Størvold HLL, Aldrin M, Borgan O, Frigessi A, Lingjærde OCC (2007). “Predicting survival from microarray data - a comparative study.” *Bioinformatics*, **23**, 2080–2087.
- Dudoit S, Shaffer JP, Boldrick JC (2003). “Multiple Hypothesis Testing in Microarray Experiments.” *Statistical Science*, **18**(1), 71–103. ISSN 08834237. doi:10.2307/3182872. URL <http://dx.doi.org/10.2307/3182872>.
- Efron B, Tibshirani R (1997). “Improvements on cross-validation: The. 632+ bootstrap method.” *Journal of the American Statistical Association*, pp. 548–560.
- Fan J, Lv J (2010). “A selective overview of variable selection in high dimensional feature space.” *Statistica Sinica*, **20**, 101–148.
- Friedman J, Hastie T, Tibshirani R (2010). “Regularization Paths for Generalized Linear Models via Coordinate Descent.” *Journal of Statistical Software*, **33**(1), 1–22. URL <http://www.jstatsoft.org/v33/i01/>.
- Froehlich H, Zell A (2005). “Efficient parameter selection for support vector machines in classification and regression via model-based global optimization.” In *Proceedings of the International Joint Conference of Neural Networks*, pp. 1431–1438.
- Gentleman RC, Carey VJ, Bates DM, others (2004). “Bioconductor: Open software development for computational biology and bioinformatics.” *Genome Biology*, **5**, R80. URL <http://genomebiology.com/2004/5/10/R80>.
- Gerds T, Schumacher M (2006). “Consistent Estimation of the Expected Brier Score in General Survival Models with Right-Censored Event Times.” *Biometrical Journal*, **48**(6), 1029–1040.
- Graf E, Schmoor C, Sauerbrei W, Schumacher M (1999). “Assessment and comparison of prognostic classification schemes for survival data.” *Statistics in Medicine*, **18**(17–18), 2529–2545.
- Jones D, Schonlau M, Welch W (1998). “Efficient global optimization of expensive black-box functions.” *Journal of Global Optimization*, **13**, 455–492.
- Meinshausen N, Bühlmann P (2010). “Stability selection.” *Journal of the Royal Statistical Society - Series B: Statistical Methodology*, **72**(4), 417–473.
- Metzeler K, Hummel M, Bloomfield C, Spiekermann K, Braess J, Sauerland MC, Heinecke A, Radmacher M, Marcucci G, Whitman S, Maharry K, Paschka P, Larson R, Berdel W, Buchner T, Worman B, Mansmann U, Hiddemann W, Bohlander S, Buske C (2008). “An 86 probe set gene expression signature predicts survival in cytogenetically normal acute myeloid leukemia.” *Blood*, **112**(10), 4193–4201.

- Porzelius C, Binder H, Schumacher M (2009). “Parallelized prediction error estimation for evaluation of high-dimensional models.” *Bioinformatics*, **25**(6), 827–829.
- R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org/>.
- Simon N, Friedman J, Hastie T, Tibshirani R (2011). “Regularization Paths for Cox’s Proportional Hazards Model via Coordinate Descent.” *Journal of Statistical Software*, **39**(5), 1–13. URL <http://www.jstatsoft.org/v39/i05/>.
- Tibshirani R (1996). “Regression Shrinkage and Selection Via the Lasso.” *Journal of the Royal Statistical Society, Series B: Methodological*, **58**, 267–288.
- Tibshirani R (1997). “The lasso method for variable selection in the Cox model.” *Statistics in Medicine*, **16**, 385–395.
- Verweij PJM, van Houwelingen HC (1994). “Penalized Likelihood in Cox Regression.” *Statistics in Medicine*, **13**, 2427–2436.
- Wasserman L, Roeder K (2009). “High dimensional variable selection.” *The Annals of Statistics*, **37**(5A), 2178–2201.
- Zou H, Hastie T (2005). “Regularization and variable selection via the elastic net.” *Journal of the Royal Statistical Society, Series B*, **67**(2), 301–320.

**Affiliation:**

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