To appear in the *Journal of Statistical Computation and Simulation*
Vol. 00, No. 00, Month 20XX, 1–15

Automatic model selection for high-dimensional survival analysis [SPECIAL ISSUE: StatConf13]

M. Lang[a*], H. Kotthaus[b], P. Marwedel[b], C. Weihs[a], J. Rahnenführer[a], B. Bischl[a*]
[a]*Department of Statistics, TU Dortmund University, 44227 Dortmund, Germany*
[b]*Department of Computer Science 12, TU Dortmund University, 44227 Dortmund, Germany*

(*Received 00 Month 20XX; final version received 00 Month 20XX*)

Many different models for the analysis of high-dimensional survival data have been developed over the past years. While some of the models and implementations come with an internal parameter tuning automatism, others require the user to accurately adjust defaults, which often feels like a guessing game. Exhaustively trying out all model and parameter combinations will quickly become tedious or infeasible in computationally intensive settings, even if parallelization is employed. Therefore, we propose to use modern algorithm configuration techniques, e. g., iterated F-racing, to efficiently move through the model hypothesis space and to simultaneously configure algorithm classes and their respective hyperparameters. In our application we study four lung cancer microarray data sets. For these we configure a predictor based on five survival analysis algorithms in combination with eight feature selection filters. We parallelize the optimization and all comparison experiments with the `BatchJobs` and `BatchExperiments` R packages.

Keywords: machine learning; survival analysis; feature selection; high-dimensional data; model selection; algorithm configuration; parameter tuning; racing

1. Introduction

Survival analysis deals with the analysis of time to events. Failure times are observed in many application fields, from demography over econometrics and technometrics to epidemiology and medical statistics. The main interest is in quantifying the impact of other variables on the survival time. For example, in oncology, Cox proportional hazards models are a standard method for analyzing the prognostic impact of clinical variables. Also fitting suitable more complex multivariate survival models and comparing their ability to generalize with respect to unseen data is an established research topic.

However, in recent years due to the rapid development in the fields of both molecular biology and computer science new data analysis challenges appeared. Especially the simultaneous reliable measurement of thousands of genetic variables, e. g., in the form of DNA data, gene expression measurements, or proteomic measurements, has initiated the development and application of new statistical survival analysis methods for 'small n, large p' problems, where the number of variables greatly exceeds the number of observations, in this case patients.

Modern approaches include: linear methods that penalize extreme parameter estimates with some form of shrinkage, e. g., based on a Ridge penalty [1], a Lasso penalty [2] or on componentwise boosting [3]. The latter two have the advantage regarding subsequent interpretation that they automatically perform variable selection. However, in comparisons on large gene expression data sets, Ridge regression – that retains all parameters

*Corresponding authors. Email: {lang,bischl}@statistik.tu-dortmund.de in its final coefficient vector – has been demonstrated to often perform better in terms of predictive power [4, 5].

Whereas for regression and classification tasks in the area of machine learning many benchmark data sets are established and comprehensive comparisons between statistical methods have been performed, in survival analysis it is still common practice to compare only a small number of methods on a small number of data sets. There is extreme need for objective and reproducible comparison studies in this field. Next to the mentioned penalized and boosting regression approaches, survival trees and survival forests are promising algorithms that have not been stringently compared in the high-dimensional survival setting.

Obviously, the quality of the fitted models depends in many cases on the tuning of algorithm parameters. The default settings of internal parameters are often not optimal and must be tuned in tedious trials that even experts might struggle with. In addition, prefiltering of variables has often proved to be effective. For example, in a typical gene expression study, most genes are not related to survival and can be regarded as irrelevant or even detrimental for its prediction. Again, a multitude of different filter methods exist, which also require the experimenter to set respective control parameters. The exhaustive evaluation of all algorithm and parameter combinations is infeasible, even if parallelization is employed, especially for the high-dimensional case, as single model fits can become computationally expensive.

A modern approach to solve this dilemma is to adaptively make all choices for the current data set at hand through an efficient black-box optimization approach that considers the desired performance measure. While in the past generic evolutionary algorithms have been used, modern methods are specifically tailored for the characteristics of this optimization problem, i. e., an expensive-to-evaluate objective function, noisy objective values and mixed numerical and categorical as well as hierarchical parameter spaces. The emerged research field has become known under the label 'algorithm configuration'. Two prominent methodologies are iterated racing [6] and model-based optimization [7]. The latter has already been applied in a supervised machine learning setting in the Auto-WEKA project [8] or for tuning kernel methods [9]. In this article we will use the former to optimize over the survival model space.

2. Methods

This section is divided into three parts: We start by giving a brief introduction to survival analysis and summarize the most important concepts. Subsequently, filters for variable pre-selection are introduced. Finally, we describe the methodology of algorithm configuration and the iterated F-racing method, that we use in the following experiments.

2.1 *Survival analysis*

In survival analysis we observe the time to a certain event (e. g., death or tumor remission) for a group of patients. Because the predefined event is in practice often not observable for all patients, e. g., because a patient moved towns or the event never occurred before the end of the study, a censoring problem naturally emerges. This type of censoring is called 'right-censoring'. Nevertheless, a censored observation still holds important information because we can at least infer that the event did non occur before a certain point in time. Thus one major goal and special characteristic of the following methods is to utilize non-censored as well as censored observations.

We assume to have n patients and for each of them a survival time t_i. These survival times are assumed to be drawn from a non-negative random variable T with density $f(t)$ and cumulative distribution function $F(t)$. The probability to survive longer than t is given by the survival function

$$S(t) = P(\{T > t\}) = \int_t^\infty f(u)\,\mathrm{d}u = 1 - F(t).$$

With n_i defined as the number of observations still under risk at t_i and d_i defined as the number of events at t_i, the survival function can be estimated by the Kaplan-Meier estimator

$$\hat{S}(t) = \begin{cases} 1, & t < t_1 \\ \prod_{t_i \le t} \left(1 - \frac{d_i}{n_i}\right), & t \ge t_1 \end{cases}.$$

Closely related to $S(t)$ are the hazard function $\lambda(t)$, expressing the risk for an event at a certain point in time t conditioned on surviving until t, and the cumulative hazard function $\Lambda(t)$:

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{S(t) - S(t + \Delta t)}{\Delta t S(t)} = \frac{f(t)}{S(t)}, \quad \Lambda(t) = \int_0^t \lambda(u)\,\mathrm{d}u = -\log S(t).$$

Furthermore, let us define a non-censoring indicator variable δ_i, which is 1 if an event occurred at the associated survival time t_i and 0 otherwise. We assume the censoring to be non-informative, i. e., that the censoring distribution contains no information about the survival distribution.

In addition to the survival time and censoring information we observe for each patient a covariate vector $\boldsymbol{x}_i \in \mathbb{R}^p$. The $(n \times p)$-matrix formed by all covariate vectors is denoted by $\boldsymbol{X}$. The covariates can consist of a low number of clinical features, e. g. age or sex, but also high-dimensional gene expression data. They are used to predict the risk of an event expressed by the hazard function $\lambda(t)$ or the cumulative hazard function $\Lambda(t)$.

A common performance measure to evaluate survival models is the concordance index [10]. It can be interpreted as the number of observation pairs correctly ordered according to the risk predictor, normalized by the maximal number of such pairs, accounting for censoring. Define ϵ as the set of all pairs (t_i, t_j) with $i, j = 1, \ldots, n$ where we can, despite censoring, conclude $t_i < t_j$, $t_i = t_j$ or $t_i > t_j$. Furthermore, let $f(\boldsymbol{x})$ denote the predicted risk of an event given covariate vector $\boldsymbol{x}$. Now the C-index, including a correction for ties, can be defined by

$$C = 1 - \frac{1}{|\epsilon|} \sum_{\{i:\, \delta_i = 1\}} \sum_{t_i < t_j} \left(\mathbb{1}_{f(\boldsymbol{x}_i) < f(\boldsymbol{x}_j)} + \frac{1}{2} \mathbb{1}_{f(\boldsymbol{x}_i) = f(\boldsymbol{x}_j)} \right), \quad C \in [0, 1]. \tag{1}$$

Values close to 0 indicate near perfect prediction, values close to 0.5 random prediction and values close to 1 very bad predictions. The last case should not occur in practice as simply inverting the predictions would result in well performing models. Note that some authors define the C-Index as $1 - C$, which is then to be maximized, and values greater than 0.5 indicate better performance than random prediction.

2.1.1 Cox proportional hazards model

A standard regression technique for censored data is the Cox proportional hazards model [11] which uses the hazard function as response and a linear combination of the covariates:

$$\lambda\left(t \mid \boldsymbol{x}\right) = \lambda_0(t) \exp\left(\boldsymbol{\beta}^T \boldsymbol{x}\right).$$

Here, $\lambda_0(\cdot)$ denotes an arbitrary baseline hazard, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^T$ is the vector of unknown regression coefficients. Because $\lambda_0(\cdot)$ is independent of the covariates, the regression coefficients can be estimated by maximizing the partial log-likelihood

$$LL(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[\boldsymbol{\beta}^T \boldsymbol{x}_i - \log \left(\sum_{j \in R(t_i)} \exp\left(\boldsymbol{\beta}^T \boldsymbol{x}_j\right) \right) \right], \quad R\left(t^*\right) = \{i \mid t_i \geq t^*\}. \tag{2}$$

The baseline hazard is determined in a second step through an estimator of the cumulative hazard function $\Lambda(t)$.

2.1.2 Penalized regression

The previously described Cox model is infeasible in high-dimensional settings ($p \gg n$), as the maximization of $LL(\boldsymbol{\beta})$ will become ill-posed due to collinearity. A well-known solution from standard regression is to include a regularization term in the objective criterion [1, 2]. Maximizing the partial log-likelihood (2) then yields

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left(-LL\left(\boldsymbol{\beta}\right) + \lambda_{\text{reg}} \left((1-\alpha) \frac{1}{2} \left\|\boldsymbol{\beta}\right\|_2^2 + \alpha \left\|\boldsymbol{\beta}\right\|_1 \right) \right),$$

where $\lambda_{\text{reg}} \geq 0$ controls the amount and $\alpha \in [0, 1]$ the type of regularization. For $\alpha = 0$ the above formula reduces to ridge regression, penalizing large values in the covariate vector quadratically through the L_2 norm, and setting $\alpha = 1$ results in lasso regression using the L_1 norm. The former is known to usually yield better results, while the latter enforces feature selection and sparsity. The mixture of both penalty types ($0 < \alpha < 1$) is called elastic net regression [12]. A good value for λ_{reg} is typically identified using cross-validation.

2.1.3 Boosted regression

Gradient boosting is a recent reinterpretation of the well known AdaBoost algorithm which constructs a highly competitive ensemble predictor out of weak learners by iteratively re-weighting the training data towards the 'harder-to-learn' observations. This algorithm can be reformulated as a gradient descent method in a function space of an additive expansion of basis functions [13]. The resulting algorithm is generic and flexible in the sense that regression, classification and other supervised tasks can simply be covered by combining the algorithm with an appropriate loss function, e. g., for survival analysis the negative partial log-likelihood for the Cox model can be used. The `mboost` version [14, 15] we consider in our following experiments boosts a linear survival model where linear basis functions defined on single covariates are added in each boosting iteration (componentwise boosting). This approach is specifically well suited for the high-dimensional setting, as stopping the boosting early results in a sparse model. We also include a further likelihood-based boosting variant [16], implemented in the package `CoxBoost` [17].

2.1.4 Survival trees and forests

Survival trees use binary decisions to recursively split observations into groups of similar hazard rates. A popular decision rule is to search over all features x_i and their respective possible cut points c to minimize the p-value of the logrank test [18] for the problem

$$H_0\colon \lambda_1(t) = \lambda_2(t)\ \forall t \quad \text{vs.} \quad H_1\colon \exists t : \lambda_1(t) \neq \lambda_2(t).$$

For prediction, new observations are dropped down the tree and a cumulative hazard function $\Lambda(t)$ is constructed for the respective terminal node. This $\Lambda(t)$ can be used to calculate the so-called mortality measure.

Random survival forests are constructed through the usual mechanism of fitting survival trees on bootstrapped data samples and randomly sampling candidate feature sets for each node. We refer the reader to [19, 20] and the respective implementation in the package `randomForestSRC` for more details.

2.2 *Filter*

Pre-selecting features is a two-edged sword: On the one hand, one explicitly distrusts the model by holding back potentially useful information. Furthermore, basic filter methods are rather naive in the way they handle correlations between features, while the more elaborate variants demand for substantial computational resources. On the other hand, restricting the features to a reasonable subset can significantly increase the model's performance. Moreover, preliminary dimension reduction allows to employ models otherwise not applicable and drastically decreases computational requirements in the model fitting process. In the following we present some popular feature selection methods we tried upstream in the model building process. For a more complete overview we refer the reader to [21].

2.2.1 Simple filters

For our mixture of clinical and genetic covariates, we considered three basic filter methods which completely ignore the observed feature values and survival times:

All features: Lets all features, both clinical and genetic, pass.
Only clinical: Lets only clinical variables pass and discards all genetic variables.
Kratz: Lets all clinical variables pass and adds a predefined set of genes known to be related to survival from literature. For the lung cancer data this set is derived from [22] and consists of 19 genes.

2.2.2 Univariate scoring filters

A scoring filter is defined as a rule to assign each feature a numerical value which expresses its 'relevance'. This way the features can be ordered to select a smaller percentage of useful candidates. We have chosen three variants which are popular in the high-dimensional survival analysis setting:

Var: The underlying idea here is that features with low variance cannot explain the differences in survival, thus a low variance should imply low importance or noise. Note that this scoring rule is independent of the response.
Uni: A univariate Cox model (see 2.1.1) is separately applied for each feature and p-values are obtained by the score test [11]. These p-values are used as importance scores.

C-index: The C-index calculation is performed for each feature without fitting a survival model. In the C-index formula (1) the observations are ordered by only considering the respective feature values. The resulting C-index is used as a score for that feature. In contrast to p-value scoring, this filter targets the performance measure more directly.

2.2.3 Minimum redundancy, maximum relevance

The mRMR filter [23] tries to strike a compromise between non-redundant and very relevant feature sets. One option to measure redundancy is to calculate the mean correlation between a considered feature and all other $(p-1)$ features: $\mathrm{red}_i = \sum_{i \neq j} \mathrm{cor}(x_i, x_j)$. Instead of the correlation the mutual information criterion can be used. To assess the relevance rel_i, in principle any performance measure can be used. We opted for the C-index here. Both measures can now be combined through either a quotient $\frac{\mathrm{rel_i}}{\mathrm{red_i}}$ or difference $\mathrm{rel}_i - \mathrm{red}_i$ to form a filter score.

Note that we struggled with the existing implementations for the mRMR filter in terms of memory consumption in parallel environments. Therefore, we wrote our own more efficient version, which can be found at `http://github.com/mllg/fmrmr`.

2.2.4 Nearest shrunken centroids

The nearest shrunken centroid method [24] starts with assigning the observations to a preliminary defined number of groups k using quantiles of the survival time. Standardized features form class centroids which are then shrunken towards the overall centroid using a shrinkage parameter Δ. In this process the features with the lowest distance to the overall centroid are systematically sorted out as they provide the least information. This allows to use nearest shrunken centroids as a filter, although the resulting number of features is not directly controllable. The implementation can be found in the R package `pamr` [25]. The amount of shrinkage is typically tuned using cross-validation.

2.3 *Algorithm configuration and iterated F-racing*

Algorithm configuration has recently become a very active research area that was mainly motivated by applications from discrete optimization. In these scenarios, computationally challenging – often NP-hard – problems can be solved by a large variety of either exact or heuristic methods. The algorithms not only vary w. r. t. their basic methodology, but also offer an extensive number of algorithmic ‘nuts and bolts’ to influence their behavior. Depending on the selected algorithm and its configured settings, measured performance on problem instances often varies considerably. Therefore, an algorithm is configured for a certain application domain by ‘tuning’ its performance, so it becomes optimal in the mean when evaluated on a set of representative benchmark instances. This optimization task can be a nontrivial endeavor, because the parameter space is usually not small, one run of an algorithm might require a large amount of runtime, and the performance evaluation is often stochastic.

2.3.1 Algorithm configuration

Let us assume we have a parametrized algorithm $\mathcal{A}(\omega, \boldsymbol{\theta})$ at our disposal. $\omega \in \Omega$ is a so-called instance, which is the object the algorithm is run on and supposed to solve in a certain way. $\boldsymbol{\theta} \in \Theta$ is the configuration of the algorithm for this run. We also assume a cost function $c : \Omega \times \Theta \to \mathbb{R}$ measuring the performance of the run $\mathcal{A}(\omega, \boldsymbol{\theta})$. The usual convention is to minimize that measure. Our task now is to obtain a configuration $\boldsymbol{\theta}^*$

with minimal costs when these costs are averaged over all instances from Ω. As the instance space might be infinitely large (and is in practice often accessed by sampling), we furthermore assume a probability distribution $\mathcal{P}$ over Ω and a random variable $I \sim \mathcal{P}$. Also note that the algorithm $\mathcal{A}$ might be stochastic in nature, which introduces a further potential source of random variation. Therefore, the property of interest is a random variable $c(I, \boldsymbol{\theta})$, and we have to solve the following optimization problem

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}}\, E[c(I, \boldsymbol{\theta})].$$

A further complicating factor is the structure of our parameter space. Usually, if our algorithm $\mathcal{A}$ has l parameters, Θ will be a cross-product of l one-dimensional spaces. But in many applications, these parameters will include categorical and integer types. Moreover, there often is a hierarchical structure in our parameter space so that setting a certain element is only feasible if a specific condition for the remaining values holds. Such parameters are often called ‘dependent’ or ‘subordinate’. Often the condition is of the form that a superordinate categorical parameter is set to a certain level. An example of such a conditional parameter is given in Section 2.3.4.

2.3.2 F-Racing

Before we describe iterated racing, let us consider the older, simpler racing technique [26]. Here we assume $\Theta = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_m\}$ to be finite, not too large, and without ‘further structure’, i. e., we disregard that Θ is probably a cross-product of one-dimensional parameter spaces. The task again is to find $\boldsymbol{\theta}^* \in \Theta$. Note that without structure on Θ and with stochastic $c(\omega, \boldsymbol{\theta})$, we have to ‘touch’ each $\boldsymbol{\theta}_j$ a few times. In the racing algorithm we try to reach that goal by performing as few evaluations $c(\omega, \boldsymbol{\theta}_j)$ as possible. In each iteration we sample a new instance ω from $\mathcal{P}$ and evaluate all configurations $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_m\}$ on it. This results in a growing collection of cost values. After each iteration we search for configurations $\boldsymbol{\theta}_j$ which are significantly outperformed by at least one other configuration in the current candidate set w. r. t. to the so far obtained performance values. This can be accomplished through a multi-sample paired location test, and F-racing usually employs the Friedman test combined with post-hoc analysis [27] to discover significant differences between all pairs (hence the ‘F’ in F-racing). In this fashion the costly function evaluations are focused on the more promising candidate configurations instead of the poorer ones.

After we have spent a predefined budget B of function evaluations (or reduced Θ to a small enough set within the current budget), we return all remaining configurations. We will abbreviate this simple procedure as race(Θ, B). Note that we employ the statistical test somewhat heuristically to reduce the number of candidate configurations in a reasonable fashion and all issues of sequential testing are usually disregarded.

2.3.3 Iterated racing

Iterated racing [6] extends the basic racing algorithm by exploiting the specific structure of the parameter space mentioned in Section 2.3.1 and assumes a probability distribution Q over Θ. In an iterative procedure it now samples a set of new candidate configurations S from Q, races the current candidates to a low number of so-called elite configurations and adapts the distribution Q by centering it around the elites as well as reducing its spread. The latter results in exploration in the beginning and exploitation in the later stages of the optimization. The general procedure is outlined in Algorithm 1.

The parameter distribution Q is constructed and adapted in the following way: For each parameter an individual independent univariate distribution is used. For numerical

Algorithm 1 Outline of iterated F-racing.

```
Q ← uniform distribution on $\Theta$
$S_{\text{elite}} \leftarrow \emptyset$
while budget B not exhausted do
    $S_{\text{new}} \leftarrow \text{sample}(\Theta, Q, N_{\texttt{new}})$        ▷ Sample $N_{\text{new}}$ new configurations from $Q$
    $S_j \leftarrow S_{\text{elite}} \cup S_{\text{new}}$
    $S_{\text{elite}} \leftarrow \text{race}(S_j, B_j)$        ▷ See Section 2.3.2
    $Q \leftarrow \text{adapt}(Q, S_{\text{elite}})$        ▷ See end of Section 2.3.3
    $j \leftarrow j + 1$
end while
return $S_{\text{elite}}$
```

and integer parameters a truncated normal distribution is assumed, whose support corresponds to the box constraints of the respective parameter. The mean of the distribution is taken from the respective parameter value of a randomly sampled elite configuration of the current iteration. The standard deviation is reduced in later stages to enforce increasing exploitation in the search process. Values for integer parameters in S_{new} are rounded after sampling. For categorical parameters a discrete probability distribution is used where the mass of the observed values in the elite configurations are increased after each iteration. Dependent parameters are handled by sampling the values of non-dependent parameters first, then by sampling the values of those subordinate parameters whose requirements are satisfied and so on.

For further details, e. g., exact formulas for Q, how the number of new candidates N_{new} as well as the budget B_j of the race are set or some algorithmic extensions like a restart mechanism, we refer the reader to [6], which also documents the `irace` R package that we have used.

2.3.4 Further remarks

In our case an algorithm is a survival analysis method, possibly combined with a feature filter, although in general one is certainly not restricted to feature filtering but can optimize any general machine learning operation chain with this approach. $\boldsymbol{\theta}$ specifies all choices and respective hyperparameters. Note that we encode the basic survival model class and filter technique as categorical parameters and make all other parameters subordinate to these choices, so from that perspective we only configure one 'super' survival analysis algorithm. Hence, that algorithm has a parameter vector $\boldsymbol{\theta}$ which looks like (filter=[filter-name], filter-parameters, model=[model-name], model-parameters)^T, where the first and third entry are non-dependent parameters and the second and fourth are dependent ones. For example, the parameter selecting the redundancy measure of our `fmrmr` filter only makes sense if filter = `fmrmr` holds. The reader is already referred to Table 2 which defines our complete parameter space.

As we want to determine the best model for one data set individually, instances are defined as subsampled training-test splits for one data set. $c(\omega, \boldsymbol{\theta})$ is then defined as the resulting C-index when the model is fit on the training set and evaluated on the test set. Note that we will embed the model selection with iterated F-racing into an outer cross-validation. Therefore an instance is more precisely a training-test split of the training set of the current cross-validation fold.

One could also consider the possibility to configure for a whole domain of similar data sets. We will not follow this approach in this work, although it is certainly a valid goal. Usually, one obtains better results in machine learning when one does model selection for individual data sets and performing model selection across different data sets (at

Table 1. Characteristics of lung cancer data sets.

GEO ID	observations	events	clinical covariates	genetic covariates
GSE31210	204	30	age, gender, smoker, stage	54 675
GSE4573	129	66	age, gender, stage	22 283
Uppsala	196	145	gender, smoker	54 675
Jacob	440	235	age, gender, stage	22 283

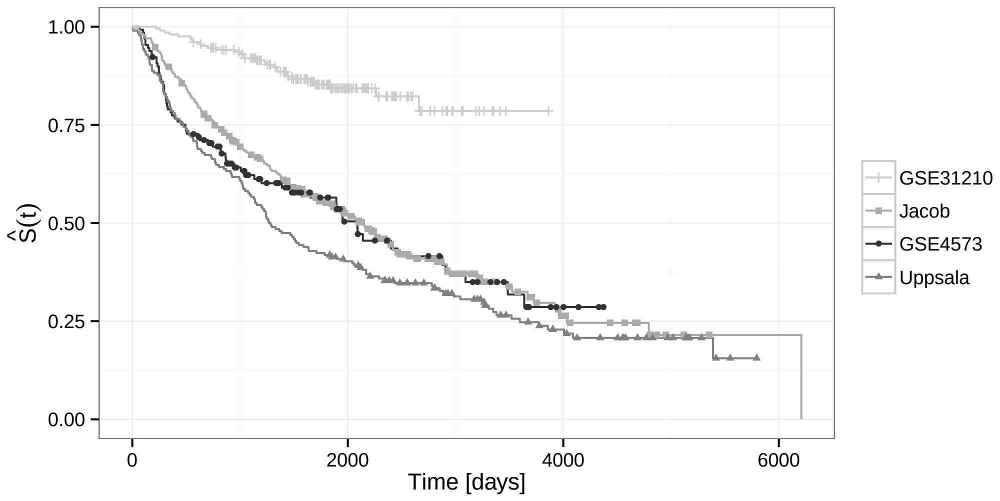

Figure 1. Kaplan-Meier estimators of survival times.

least as an optimization problem) is much less frequently considered. Especially, in our application, it is uncertain whether, e. g., the same percentage of features is optimal across different data sets.

3. Data sets

We picked four publicly available lung cancer data sets with genome wide gene expression measurements and additional clinical information. Data sets 'GSE31210', 'GSE4573' and 'GSE37745' (in the following named 'Uppsala') were extracted from the Gene Expression Omnibus [28] database. The fourth data set, which we name 'Jacob', is taken from a website referenced in [29]. In addition to survival times and gene expressions (measured on the Affymetrix HGU1333a or HGU133 Plus 2.0 microarray chip and normalized using RMA [30]), we selected some important clinical covariates for each data set. The main criterion for inclusion was the proportion of missing values, as we eliminated all incomplete observations before applying any method. Table 1 gives an overview of the major characteristics of all four data sets, Figure 1 displays their estimated survival functions. As we can clearly see, patients in the data sets 'GSE4573', 'Uppsala' and 'Jacob' have a similar probability to survive, while patients in the 'GSE31210' data set have a considerably better prognosis.

4. Experimental setup

On all previously described data sets we evaluate through a three-fold cross-validation (CV). This means that the data sets are randomly partitioned into three blocks which act as the respective test set per CV iteration. The remaining two blocks are combined and used as a training set to fit the model. Models are always compared based on the same cross-validation splits to reduce variance in their performance differences. All our models are evaluated by the previously defined concordance index using either the linear predictor $\boldsymbol{X}_{\text{test}}\hat{\boldsymbol{\beta}}_{\text{train}}$ (regression based models) or the mortality (random survival forest) as risk predictions.

Cross-validation is a standard resampling procedure for model evaluation and comparison in general supervised machine learning and also survival analysis [31, 32]. The reason we opt for a rather low number of folds is explained by the fact that we need a large enough number of samples in the test set to evaluate the concordance index in a meaningful way. Splitting the data in a 'two thirds for training, one third for testing' fashion is also standard in that regard.

We run the following experiments: First, we cross-validate four reference (or baseline) models that a reasonable experimenter might try on our considered data sets. Note that all of the proposed models come with an internal cross-validation based auto-tuning of the arguably most important parameters. These baseline models are:

CoxPH: A Cox proportional hazards model on the clinical covariates from package `survival` [33, 34].

Ridge: A ridge regression, where the penalty parameter is determined by an inner cross-validation on the training set, from package `glmnet` [35].

mboost: Gradient boosting from package `mboost` [14, 15], where the 'family' parameter was set to 'CoxPH', the number of boosting steps was determined by an inner CV on the training set and all other settings were left at their defaults.

CoxBoost: Likelihood-based boosting from package `coxboost` [17], where the number of boosting iterations and the penalty parameter are determined by an inner CV on the training set and all other settings are left at their defaults.

We now compare these baseline results against models found by configuring over a comprehensive parameter space, including different survival analysis models, filter techniques and their respective control parameters. All parameters and their constraints are displayed in Table 2. A budget of 10 000 function evaluatios (model fits) is granted to `irace`. In the high-dimensional setup this demands for considerable computing power. Therefore we patched `irace` to roll out the computations on a high-performance cluster using the `BatchJobs` and `BatchExperiments` packages [36]. Nevertheless, runtime requirements are very heterogeneous: The time to evaluate a single point can range from a few seconds to several hours, largely influenced by the amount of filtering. The benefit of parallelization is therefore limited as we periodically have to wait for the slowest evaluation in each racing iteration. Excessive memory requirements justified one more constrained not listed in Table 2: The random survival forest is only applied whenever less than 1000 covariates remain after filtering and the C-index is set to the worst possible value otherwise.

It should be noted that we mix continuous, integer and categorical parameters and also have a hierarchical structure where almost every parameter either depends on the setting of the model or filter parameter.

For each CV iteration we run the irace configurator on the training set of the outer cross validation. Irace then again subsamples this training set in its internal racing procedure an adaptive number of times. After our budget is spent, the optimizer recommends an optimized parameter vector $\boldsymbol{\theta}^*$, we fit the model on the complete CV training set and evaluate the predictions on the CV test set. Note that this results in three different model configurations $(\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \boldsymbol{\theta}_3^*)$ per data set. The outer CV splits used for tuning are exactly the same as the ones used for our baseline models.

Table 2. Overview of parameter space for configuration.

Name	Description	Type	Box Constraints / Values
filter	Type of filter	categorical	all_features, only_clinical, kratz var, uni, cindex, fmrmr, pamr
model	Type of model	categorical	coxboost, glmnet rfsrc, penalized, mboost
use.clinical	filter = uni	logical	TRUE, FALSE
filter.perc	Percentage of features For filter ∈ {var, uni, fmrmr}	continuous	0 – 0.3333
fmrmr			
redundance	Measure of redundancy	categorical	mi, pearson
combine	Combination of rel and red	categorical	difference, quotient
pamr			
ngroup.survival	Number of groups	integer	2 – 3
coxboost			
stepsize.factor	Sets the step-size modification factor	continuous	0.1 – 2
penalty	Update penalty for single coeff per step	continuous	0.1 – 1000
penalized			
penalty	Type of penalty	categorical	L1, L2
glmnet			
alpha	Penalty control	continuous	0 – 1
rfsrc			
splitrule	Rule used for splits in tree	categorical	logrank, logrankscore
nodesize	Min. size of terminal node	integer	1 – 10
ntree	Number of trees in forest	integer	1 – 2000
bootstrap	Bootstrap protocol	categorical	by.root, by.node, none
mboost			
m	Boosting iterations	integer	1 – 500
nu	Step size / shrinkage	continuous	0 – 1
family	Loss function	categorical	CoxPH, Lognormal Weibull, Loglog

Table 3. Mean C-indices for baseline models and configuration approach, cross-validated with three folds.

Data set	CoxBoost	CoxPH	mboost	Ridge	Configurator
GSE31210	0.24	0.35	0.24	0.23	0.18
GSE4573	0.47	0.45	0.46	0.47	0.44
Uppsala	0.49	0.54	0.46	0.49	0.42
Jacob	0.33	0.34	0.33	0.33	0.31

5. Results

Table 3 displays the obtained results for all four baseline methods and our tuning approach. We can draw two important conclusions: First, although the data sets are from the same domain, fitting a predictive model on these data sets is of unequal difficulty. This is clear just by comparing the performance measures of the Cox proportional hazards models across data sets. Second, tuning boosts performance on all data sets, but to a different extent. Compared to the best reference model, the performance gain on data sets ‘GSE31210’ and ‘Uppsala’ is very visible. On data sets ‘GSE4573’ and ‘Jacob’ on the other hand, the tuning seems to have less impact. Yet the differences are significant which is demonstrated later in this section.

Table 4 lists the configurations $\boldsymbol{\theta}^*$ found by `irace` for each outer cross-validation fold and its estimated C-index from the iterated racing procedure on the CV training set. The combinations of selected filters and models reveal some information about the data sets: For the ‘Jacob’ and ‘GSE4573’ data sets genetic covariates seem less important. The filter completely discards them or leaves only a very small fraction for the model. Then the choice of model apparently has a lower impact on performance, or, in other words, many

Table 4. Results of configuration per data set and outer cross-validation fold. The displayed $\hat{C}$ refers to the estimated C-index of the configurator, i. e., not on the outer test set, but from the repeated splits of the outer training set.

Data Set	Fold	Filter	Model	$\hat{C}$
GSE31210	1	uni perc=0.317, use.clinical=TRUE	glmnet alpha=0.003	0.14
GSE31210	2	all_features	glmnet alpha=0.01	0.14

GSE31210	3	fmrmr
		perc=0.324, relevance=cindex
		redundance=mi, combine=quotient
			glmnet
			alpha=0.017
				0.24

GSE4573	1	fmrmr
		perc=0.023, relevance=cindex
		redundance=pearson, combine=quotient
			penalized
			penalty=L2
				0.41

GSE4573	2	var
		perc=0.044
			rfsrc
			splitrule=logrank, nodesize=3
			ntree=1842, bootstrap=none
				0.37

GSE4573	3	var
		perc=0.029
			rfsrc
			splitrule=logrankscore, nodesize=7
			ntree=1562, bootstrap=by.root
				0.32

Uppsala	1	fmrmr
		perc=0.009, relevance=cindex
		redundance=mi, combine=difference
			penalized
			penalty=L2
				0.43

Uppsala	2	fmrmr
		perc=0.269, relevance=cindex
		redundance=pearson, combine=difference
			penalized
			penalty=L2
				0.38

Uppsala	3	fmrmr
		perc=0.108, relevance=cindex
		redundance=mi, combine=difference
			penalized
			penalty=L2
				0.44

Jacob	1	only_clinical	mboost m=179, nu=0.344, family=CoxPH	0.28
Jacob	2	kratz	coxboost stepsize.factor=1.888, penalty=275.772	0.27

Jacob	3	fmrmr
		perc=0.061, relevance=cindex,
		redundance=mi, combine=difference
			glmnet
			alpha=0.888
				0.30

models are equally good on the remaining subset of features. On the 'GSE31210' data set in the contrary, the clinical covariates are either not that useful or the genetic covariates hold much information. In all folds many thousand genes are passed to an elastic net with ridge regularization clearly dominating lasso regularization. The 'Uppsala' case is similar in that regard: A pure ridge regression on mRMR pre-selected covariates is the favorable approach here.

To judge the performance gain of the tuning approach the three-fold CV is repeated 30 times for each data set. All reference models as well as the three models with the preliminary found configurations $(\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \boldsymbol{\theta}_3^*)$ are build on each of the $30 \cdot 3 = 90$ training sets. In the next step the best performing reference model w. r. t. to the average C-index across all test sets is selected for each data set. The best reference model is then compared with each tuned model in a pairwise fashion using a two-sided Wilcoxon test for location shift. Table 5 lists mean C-indices and resulting p-values. Tuning yields, with one exception on the 'Jacob' data, always significantly better results than any of the considered reference models. A further noteworthy aspect revealed by the repetitive evaluation is given by the fact that `mboost` outperforms the other reference models on three of four data sets.

6. Conclusions

We have demonstrated how a well-known approach from algorithm configuration can successfully be applied to model selection in high-dimensional survival analysis. Our tuned

Table 5. Mean C-Indices $\bar{C}$ for reference and tuned models evaluated in 30 repetitions of a 3-fold cross-validation. p-values derived from the paired, two-sided Wilcoxon test for location shift: C-indices of best performing reference model vs. C-indices of respective configuration.

Data set	$\bar{C}_{\text{coxboost}}$	$\bar{C}_{\text{coxph}}$	$\bar{C}_{\text{mboost}}$	$\bar{C}_{\text{ridge}}$	$\bar{C}_{\boldsymbol{\theta}_1^*}$	$\bar{C}_{\boldsymbol{\theta}_2^*}$	$\bar{C}_{\boldsymbol{\theta}_3^*}$	$p_{\boldsymbol{\theta}_1^*}$	$p_{\boldsymbol{\theta}_2^*}$	$p_{\boldsymbol{\theta}_3^*}$
GSE31210	0.26	0.32	0.26	0.27	0.18	0.19	0.18	<0.001	<0.001	<0.001
GSE4573	0.51	0.46	0.53	0.51	0.48	0.42	0.41	0.002	<0.001	<0.001
Uppsala	0.49	0.53	0.48	0.50	0.42	0.42	0.42	<0.001	<0.001	<0.001
Jacob	0.31	0.33	0.31	0.31	0.33	0.31	0.30	<0.001	0.631	<0.001

models perform significantly better than the considered reference models. Furthermore, many interesting data set characteristics, otherwise inaccessible in high-dimensional settings, can be derived by examining the configurations.

Unfortunately, the community did not yet agree on a set of benchmark data sets which makes proper scientific comparisons harder than they should be. Only a few articles exist that contain extensive and rigorous benchmark comparisons for survival analysis. One notable exception is [37] which is based on a modelling competition on an exceptionally large breast cancer data set. The combined computational and human resources of all participants led to a comprehensive benchmark study. The rules for accessing this data set are rather restrictive. We plan to obtain it in order to have another source of results for comparison.

One option to mitigate at least the technical problems of having a machine-readable collection of data sets, experimental descriptions, code and results is the recently introduced OpenML [38] project. This projects strives to enable users to organize all relevant parts of a machine learning experiment on an easily query-able online platform and offers integration into many popular machine learning toolkits such R, Weka, RapidMiner and KNIME.

Regarding our current experimental setup, we suspect that the budget of 10 000 function evaluations per configuration run might still be set too low, but as the combined experiments already took several weeks of sequential computing time, we could not increase this any further for the current publication due to time-constraints. We are also interested in comparing the iterative racing approach with a model-based optimization similar to the ones in [8, 9].

In our presented approach, we have only configured survival analysis models for individual data sets. It might also be a worthwhile endeavor to try to configure (parts of) the modeling for a whole domain of survival analysis data sets, e. g., (lung) cancer gene expression data. Another valid option to explore – as proposed in [39] – is to include the runtime cost of the considered configuration into the search, as we usually are under the constraint of a limited, fixed time budget for the whole optimization process.

Acknowledgements

Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Analysis”, project A3 and Collaborative Research Center SFB 823 “Statistical modelling of nonlinear dynamic processes”, project C2.

References

[1] Hoerl A, Kennard R. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics. 1970;12(1):55–67.

[2] Tibshirani R. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society Series B (Methodological). 1996;58(1):267–288.

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

[4] Bøvelstad HM, Nyågard S, Størvold HL, Aldrin M, Borgan Ø, Frigessi A, Lingjaerde OC. Predicting survival from microarray data: A comparative study. Bioinformatics. 2007; 23(16):2080–2087.

[5] Kammers K, Lang M, Hengstler JG, Schmidt M, Rahnenführer J. Survival models with preclustered gene groups as covariates. BMC Bioinformatics. 2011;12(1):478.

[6] López-Ibáñez M, Dubois-Lacoste J, Stützle T, Birattari M. The irace package, iterated race for automatic algorithm configuration. IRIDIA, Université Libre de Bruxelles, Belgium; 2011. Tech Rep TR/IRIDIA/2011-004. Available from: `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`.

[7] Hutter F, Hoos HH, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In: Coello CA, editor. Learning and Intelligent Optimization. Lecture Notes in Computer Science. Springer; 2011. p. 507–523.

[8] Thornton C, Hutter F, Hoos HH, Leyton-Brown K. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2013. p. 847–855.

[9] Koch P, Bischl B, Flasch O, Bartz-Beielstein T, Weihs C, Konen W. Tuning and evolution of support vector kernels. Evolutionary Intelligence. 2012;5(3):153–170.

[10] Heagerty PJ, Zheng Y. Survival model predictive accuracy and ROC curves. Biometrics. 2005;61(1):92–105.

[11] Cox D. Regression models and life-tables. Journal of the Royal Statistical Society Series B (Methodological). 1972;34(2):187–220.

[12] Zou H, Hastie T. Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology). 2005;67(2):301–320.

[13] Friedman J, Hastie T, Tibshirani R. Additive logistic regression: a statistical view of boosting. Annals of Statistics. 2000;28(2):337–407.

[14] Bühlmann P, Hothorn T. Boosting algorithms: Regularization, prediction and model fitting. Statistical Science. 2007;22(4):477–505.

[15] Hothorn T, Buehlmann P, Kneib T, Schmid M, Hofner B. mboost: Model-based boosting. 2013. R package version 2.2-3. Available from: `http://CRAN.R-project.org/package=mboost`.

[16] Binder H, Allignol A, Schumacher M, Beyersmann J. Boosting for high-dimensional time-to-event data with competing risks. Bioinformatics. 2009;25(7):890–896.

[17] Binder H. Coxboost: Cox models by likelihood based boosting for a single survival endpoint or competing risks. 2013. R package version 1.4. Available from: `http://CRAN.R-project.org/package=CoxBoost`.

[18] Mantel N. Chi-square tests with one degree of freedom; extensions of the mantel-haenszel procedure. Journal of the American Statistical Association. 1963;58(303):690–700.

[19] Ishwaran H, Kogalur U. Random survival forests for R. R News. 2007 October;7(2):25–31.

[20] Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS. Random survival forests. The Annals of Applied Statistics. 2008;2(3):841–860.

[21] Saeys Y, Inza In, Larrañaga P. A review of feature selection techniques in bioinformatics. Bioinformatics. 2007;23(19):2507–2517.

[22] Kratz JR, He J, Van Den Eeden SK, Zhu ZH, Gao W, Pham PT, Mulvihill MS, Ziaei F, Zhang H, Su B, Zhi X, Quesenberry CP, Habel La, Deng Q, Wang Z, Zhou J, Li H, Huang MC, Yeh CC, Segal MR, Ray MR, Jones KD, Raz DJ, Xu Z, Jahan TM, Berryman D, He B, Mann MJ, Jablons DM. A practical molecular assay to predict survival in resected non-squamous, non-small-cell lung cancer: development and international validation studies. Lancet. 2012;379(9818):823–832.

[23] Peng H, Long F, Ding C. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2005;27(8):1226–1238.

[24] Tibshirani R, Hastie T, Narasimhan B, Chu G. Diagnosis of multiple cancer types by shrunken centroids of gene expression. Proceedings of the National Academy of Sciences of the United States of America. 2002;99(10):6567–6572.

[25] Hastie T, Tibshirani R, Narasimhan B, Chu G. pamr: Pam: prediction analysis for microarrays. 2013. R package version 1.54.1. Available from: `http://CRAN.R-project.org/package=pamr`.

[26] Birattari M, Stützle T, Paquete L, Varrentrapp K. A racing algorithm for configuring metaheuristics. In: Proceedings of the genetic and evolutionary computation conference. GECCO '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2002. p. 11–18.

[27] Conover WJ. Practical nonparametric statistics. Wiley series in probability and mathematical statistics. Wiley; 1980.

[28] Edgar R, Domrachev M, Lash AE. Gene expression omnibus: NCBI gene expression and hybridization array data repository. Nucleic acids research. 2002;30(1):207–210.

[29] Shedden K, Taylor JMG, Enkemann Sa, Tsao MS, Yeatman TJ, Gerald WL, Eschrich S, Jurisica I, Giordano TJ, Misek DE, Chang AC, Zhu CQ, Strumpf D, Hanash S, Shepherd Fa, Ding K, Seymour L, Naoki K, Pennell N, Weir B, Verhaak R, Ladd-Acosta C, Golub T, Gruidl M, Sharma A, Szoke J, Zakowski M, Rusch V, Kris M, Viale A, Motoi N, Travis W, Conley B, Seshan VE, Meyerson M, Kuick R, Dobbin KK, Lively T, Jacobson JW, Beer DG. Gene expression-based survival prediction in lung adenocarcinoma: a multi-site, blinded validation study. Nature Medicine. 2008;14(8):822–827.

[30] Irizarry RA, Hobbs B, Collin F, Beazer-Barclay YD, Antonellis KJ, Scherf U, Speed TP. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. Biostatistics. 2003;4(2):249–264.

[31] Simon R. Fundamentals of data mining in genomics and proteomics. chap. Resampling Strategies for Model Assessment and Selection. US: Springer; 2007. p. 173–186.

[32] Bischl B, Mersmann O, Trautmann H, Weihs C. Resampling methods for meta-model validation with recommendations for evolutionary computation. Evolutionary Computation. 2012;20(2):249–275.

[33] Therneau TM. A package for survival analysis in S. 2013. R package version 2.37-4. Available from: `http://CRAN.R-project.org/package=survival`.

[34] Therneau TM, Grambsch PM. Modeling survival data: Extending the Cox model. New York: Springer; 2000.

[35] Simon N, Friedman J, Hastie T, Tibshirani R. Regularization paths for Cox's proportional hazards model via coordinate descent. Journal of Statistical Software. 2011;39(5):1–13.

[36] Bischl B, Lang M, Mersmann O, Rahnenführer J, Weihs C. Computing on high performance clusters with R: Packages BatchJobs and BatchExperiments. TU Dortmund University; 2012. Tech Rep 1/2012. available at: `http://sfb876.tu-dortmund.de/PublicPublicationFiles/bischl_etal_2012a.pdf`.

[37] Bilal E, Dutkowski J, Guinney J, Jang IS, Logsdon Ba, Pandey G, Sauerwine Ba, Shimoni Y, Moen Vollan HK, Mecham BH, Rueda OM, Tost J, Curtis C, Alvarez MJ, Kristensen VN, Aparicio S, Børresen-Dale AL, Caldas C, Califano A, Friend SH, Ideker T, Schadt EE, Stolovitzky Ga, Margolin AA. Improving breast cancer survival analysis through competition-based multidimensional modeling. PLoS computational biology. 2013;9(5).

[38] Rijn J, Bischl B, Torgo L, Gao B, Umaashankar V, Fischer S, Winter P, Wiswedel B, Berthold M, Vanschoren J. OpenML: A collaborative science platform. In: Blockeel H, Kersting K, Nijssen S, Železný F, editors. Machine learning and knowledge discovery in databases. Vol. 8190 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2013. p. 645–649.

[39] Snoek J, Larochelle H, Adams R. Practical bayesian optimization of machine learning algorithms. In: Pereira F, Burges C, Bottou L, Weinberger K, editors. Advances in neural information processing systems 25; 2012. p. 2960–2968.