Comput Stat (2016) 31:513–531
DOI 10.1007/s00180-015-0642-2

ORIGINAL PAPER

Boosting in Cox regression: a comparison between the likelihood-based and the model-based approaches with focus on the R-packages *CoxBoost* and *mboost*

Riccardo De Bin[1]

Received: 21 May 2015 / Accepted: 30 December 2015 / Published online: 13 January 2016

Abstract Despite the limitations imposed by the proportional hazards assumption, the Cox model is probably the most popular statistical tool used to analyze survival data, thanks to its flexibility and ease of interpretation. For this reason, novel statistical/machine learning techniques are usually adapted to fit its requirements, including boosting. Boosting is an iterative technique originally developed in the machine learning community to handle classification problems, and later extended to the statistical field, where it is used in many situations, including regression and survival analysis. The popularity of boosting has been further driven by the availability of user-friendly software such as the R packages *mboost* and *CoxBoost*, both of which allow the implementation of boosting in conjunction with the Cox model. Despite the common underlying boosting principles, these two packages use different techniques: the former is an adaptation of model-based boosting, while the latter adapts likelihood-based boosting. Here we contrast these two boosting techniques as implemented in the R packages from an analytic point of view; we further examine solutions adopted within these packages to treat mandatory variables, i.e. variables that—for several reasons—must be included in the model. We explore the possibility of extending solutions currently only implemented in one package to the other. A simulation study and a real data example are added for illustration.

Electronic supplementary material The online version of this article (doi:10.1007/s00180-015-0642-2) contains supplementary material, which is available to authorized users.

✉ Riccardo De Bin
debin@ibe.med.uni-muenchen.de

[1] Department of Medical Informatics, Biometry and Epidemiology, University of Munich, Marhioninistraße 15, 81377 Munich, Germany

Keywords Cox model · Gradient descent · Mandatory variables · Partial likelihood · Survival analysis

1 Introduction

Among the iterative methods exploited during recent years in statistical practice, particular attention has been focused on boosting: originally developed in the machine learning community (Schapire 1990; Freund 1995; Freund and Schapire 1996), primarily to handle classification problems, it has been successfully translated into the statistical field (Breiman 1998; Friedman et al. 2000) and extended to many statistical problems, including regression and survival analysis. Thanks to its resistance to overfitting, it is particularly useful in the construction of prediction models. Its iterative nature, moreover, allows straightforward adaptations to cope with high-dimensional data (Bühlmann and Yu 2003; Bühlmann 2006; Tutz and Binder 2006; Binder and Schumacher 2008), through its component-wise version. Applied in a parametric framework, the basic idea of boosting is to provide estimates of the parameters (e.g., the regression coefficients of a Cox model) by updating their values step by step. At each iteration, a "weak" estimator is fit on a modified version of the data, with the goal of minimizing a pre-specified loss function. The obtained value provides a small contribution used to update the estimate of the parameters: the result of all the contributions is the final estimate. Boosting relies on two tuning parameters. A first parameter controls the "weakness" of the estimator and is usually called *penalty* or *boosting step size* (hereafter, we use the former term). A second much more influential tuning parameter is related to the stopping criterion, i.e. specifies how many boosting iterations are performed. This parameter plays an important role in avoiding overfitting, and, in the case of the component-wise version, it also controls the sparsity of the model, i.e. it is related to variable selection. In this paper we do not investigate the possible ways to choose these tuning parameters. This aspect is highly relevant (see, for example, Mayr et al. 2012), and deserves a dedicated study.

The popularity of the boosting methods benefited from the availability of user-friendly software. The R (R Development Core Team 2014) package *mboost* (Hothorn et al. 2015) provides routines which allow the application of boosting to several statistical problems (for a complete overview, see Bühlmann and Hothorn 2007; Hofner et al. 2014), including survival analysis. Another R package which exploits the boosting method in this context is *CoxBoost* (Binder 2013a). Despite both relying on a boosting method, these two packages implement different techniques: the former uses an adaptation of model-based boosting (Bühlmann and Yu 2003), while the latter adapts the likelihood-based boosting approach (Tutz and Binder 2006).

The goal of this study is to contrast the algorithms used in these two R packages to implement a linear Cox model (hereafter, for simplicity, only "Cox model" or "Cox regression"). Boosting is applicable to many situations which do not conform to the assumptions of the Cox model, offering wide possibilities in survival analysis; however, biomedical applications—especially those involving high-dimensional data—tend to rely on the Cox model. The case of high-dimensional data is highly

relevant for boosting because, in this situation, the traditional statistical tools cease to be appropriate and the boosting is an attractive alternative.

The paper is organized as follows. In Sect. 2 we introduce the Cox model and briefly review the two boosting algorithms implemented in *mboost* and *CoxBoost*. In Sect. 3 we compare these two algorithms showing their similarities and differences as particular cases of the general gradient descending boosting algorithm (Friedman 2001). The comparison continues in Sect. 4, where we focus on the issue of mandatory variables and show how to increase the potential of one boosting package by implementing the solution adopted by the other. A small example with simulated data and a real data application are shown in Sect. 5. Some final considerations are reported in Sect. 6.

2 Methods

2.1 Background

Let us consider the time-to-event data (t, X, δ), where t is the n-dimensional vector of the observed survival times, X the $n \times p$ matrix of the data and δ an n-dimensional vector reporting whether the i-th observed survival time $t^{(i)}$ is censored ($\delta^{(i)} = 0$) or not ($\delta^{(i)} = 1$), $i = 1, \ldots, n$. Hereafter, we suppose, without loss of generalization, that the variables are standardized, i.e. $E[X_j] = 0$ and $Var[X_j] = 1, \forall j = 1, \ldots, p$.

To cope with this kind of data, one usually uses the Cox model (Cox 1972) to describe the hazard function $\lambda(t|X)$,

$$\lambda(t|X) = \lambda_0(t) \exp(X^\top \beta), \tag{1}$$

where $\lambda_0(t)$ is the baseline hazard function and β the p-dimensional vector of the regression coefficients. A nice property of the Cox model is that it is not necessary to consider $\lambda_0(t)$ to estimate β, as β is estimated by maximizing the partial log-likelihood

$$pl(\beta) = \sum_{i=1}^{n} \delta^{(i)} X^{(i)\top} \beta - \log \left(\sum_{l \in R^{(i)}} \exp\{X^{(l)\top} \beta\} \right).$$

Here $X^{(i)}$ denotes the i-th observation, while $R^{(i)}$ is the set of the observations at risk at time $t^{(i)}$. More formally, $R^{(i)} = \{j \in \{1, \ldots, n\} : t^{(j)} < t^{(i)}\}$.

From (1) we note that the hazard function depends multiplicatively on β, i.e. the hazard ratio between two observations is constant over time. Despite this fairly stringent assumption (usually called the proportional hazards assumption), the Cox model is by far the most commonly used tool in biomedical practice, thanks to the relative ease of the interpretation of the regression coefficients. In particular, many approaches related to high-dimensional problems rely on the Cox model (Binder and Schumacher 2008).

The Cox model is also the basis of the two boosting algorithms implemented in the R packages *mboost* and *CoxBoost*. Before analyzing these two specific implementations, we first review the underlying boosting principle, using the concept of the functional gradient descending technique (Friedman 2001). Let $L(y, F(X))$ be a generic loss

function, where $F(X)$ is a statistical model. The goal is to estimate $F(X)$ by iteratively updating its value through a base learner $h(y, X)$. The boosting algorithm can be described as follows:

1. initialize the estimate, e.g., $\hat{F}(X) = \text{constant}$;
2. compute the negative gradient vector, $u = -\left.\frac{\partial L(y,F(X))}{\partial F(X)}\right|_{F(X)=\hat{F}(X)}$;
3. compute the update by:
   3.1 fitting the base learner to the negative gradient vector, $\hat{h}(u, X)$;
   3.2 penalizing it, $\hat{f}(X) = \nu\hat{h}(u, X)$;
4. update the estimate, $\hat{F}(X) = \hat{F}(X) + \hat{f}(X)$.

In the algorithm, steps 2 through 4 are repeated m_{stop} times, where m_{stop} denotes the number of boosting iterations. The penalty ν is the other tuning parameter and can take values between 0 and 1.

This algorithm is very general and can be adapted to numerous statistical problems. As stated earlier, in this paper we focus on applications related to Cox regression that can handle high-dimensional data, thus we mainly consider component-wise boosting. In this version of the boosting the algorithm described above is modified in order to update $\hat{F}(X)$ using only one dimension of X in each boosting iteration. In particular, step 3 is applied separately to the different columns of X, generating p possible updates $\hat{f}_j(X)$. An additional step is then implemented to identify which of the p possible updates should be used in step 4. Since we restrict our analysis to the Cox regression case, hereafter we consider only the parametric version of boosting (which, incidentally, excludes the original version of the method, based on trees), where $F(X)$ is a parametrized class of functions, $F(X, \beta)$, and the update process thus involves only the estimate of the parameter. The parametric component-wise boosting algorithm is:

1. initialize the estimate, e.g., $\hat{\beta} = (0, \ldots, 0)$;
2. compute the negative gradient vector, $u = -\left.\frac{\partial L(y,F(X,\beta))}{\partial F(X,\beta)}\right|_{\beta=\hat{\beta}}$;
3. compute the possible updates by:
   3.1 fitting the base learner to the negative gradient vector, $\hat{h}(u, X_j)$;
   3.2 penalizing it, $\hat{b}_j = \nu\hat{h}(u, X_j)$;
4. select the best update j^* (usually that minimizing the loss function);
5. update the estimate, $\hat{\beta}_{j^*} = \hat{\beta}_{j^*} + \hat{b}_{j^*}$.

Steps 2–5 are repeated m_{stop} times are those between 2 and 5. The quantity $\hat{b}_j$ is the weak estimator.

2.2 *mboost* for Cox regression

The R package *mboost* is a general tool to implement boosting. In particular, its function *glmboost* allows the implementation of model-based boosting for different linear models, by selecting the appropriate loss function via the argument *family*. To perform Cox regression, the pre-built function *CoxPH* is available. Its implementation is based on the work of Ridgeway (1999), who derived the formula for the gradient vector u. The routine *glmboost* is a direct implementation of the functional descending

gradient algorithm, in which $L(y, F(X))$ is the negative partial log-likelihood and $\hat{h}(u, X_j)$ the least squares estimator $(X^\top X)^{-1} X^\top u$. In more detail, the model-based boosting algorithms can be described as follows for the Cox regression:

1. initialize $\hat{\beta} = (0, \ldots, 0)$;
2. compute the negative gradient vector,
$$u^{(i)} = \delta^{(i)} - \sum_{l \in R^{(i)}} \delta^{(l)} \frac{\exp\{X^{(l)\top}\hat{\beta}\}}{\sum_{k \in R^{(l)}} \exp\{X^{(k)\top}\hat{\beta}\}};$$
3. compute the possible updates by applying the least squares estimator to the negative gradient vector, $\hat{b}_j = (X_j^\top X_j)^{-1} X_j^\top u$;
4. select the best update, $j^* = \text{argmin}_j \sum_{i=1}^n (u^{(i)} - X_j^{(i)} \hat{b}_j)^2$;
5. update the estimate, $\hat{\beta}_{j^*} = \hat{\beta}_{j^*} + \nu \hat{b}_{j^*}$.

Steps from 2 to 5 are repeated m_{stop} times.

The algorithm computes the gradient vector of $pl(\beta)$ with respect to $F(X_j, \beta)$, i.e. the direction in which the slope of the partial log-likelihood is locally (in $\hat{\beta}$) steepest (Ridgeway 1999). Multiple univariate linear regressions are then performed to regress this vector (u) on each X_j. The value of $\hat{b}_j$ which minimizes the residual sum of squares, shrunk by ν, is then used to update $\hat{\beta}$. Roughly speaking, the boosting algorithm "climbs" the partial log-likelihood step by step in the direction which is most correlated with the steepest way to "climb" it. This procedure is iteratively performed m_{stop} times. If $p < n$, $\hat{\beta} \to \hat{\beta}_{MPLE}$ as $m_{stop} \to \infty$, where MPLE stands for "maximum partial likelihood estimate". In other words, step by step the boosting estimate $\hat{\beta}$ slowly approaches the MPLE, without never reaching it.

2.3 *CoxBoost*: likelihood-based boosting in the Cox model

The R package *CoxBoost* implements a likelihood-based boosting approach. For the loss function, this approach uses the negative L_2-norm penalized partial log-likelihood,

$$pl_{pen}(\beta) = pl(\beta) - 0.5\lambda\beta^\top P\beta,$$

where P is a $p \times p$ matrix usually corresponding to the identity matrix and λ is the penalty term. An offset term $\hat{\eta} = X^\top \hat{\beta}$ is incorporated into this log-likelihood to keep track of the iterative updates of the parameter estimate, resulting in a function of the form

$$pl^{LB}_{pen}(\beta|\hat{\beta}) = \sum_{i=1}^{n} \delta^{(i)} \left[\hat{\eta}^{(i)} + (X^{(i)\top}\beta - \log\left(\sum_{l \in R^{(i)}} \exp\{\hat{\eta}^{(l)} + X^{(l)\top}\beta\} \right) \right] - \frac{\lambda}{2}\beta^\top P\beta. \tag{2}$$

In each boosting iteration, the maximizer of this function is applied to compute the possible update(s). To better understand the procedure, let us first consider a non-component-wise version, applicable only if $p < n$, and a starting value $\hat{\beta} = (0, \ldots, 0)$.

As an effect of the penalty term, the log-likelihood in the parameter space is "shifted" toward the origin and, as the main consequence, the values of the coordinates of its maximum (i.e. the possible update of $\hat{\beta}$) are a fraction of the coordinates of the MPLE. In this sense, the maximizer of (2) is a "weak estimator", because it provides an estimate that shrinks the MPLE toward 0. The amount of this shrinkage depends on λ.

The new $\hat{\beta}$ is added to the offset term, and the procedure is repeated. Through the penalty term, the partial log-likelihood is now "shifted" toward $\hat{\beta}$ and a new value of the update is computed, moving $\hat{\beta}$ toward the MPLE. In this case as well, $\hat{\beta} \rightarrow \hat{\beta}_{MPLE}$ as $m_{stop} \rightarrow \infty$. It is worth noting that, since λ is constant, the updates to $\hat{\beta}$ become smaller and smaller as we proceed with the boosting iterations. Therefore $\hat{\beta}$ continually approaches the MPLE without reaching it.

The component-wise version follows a similar idea, but the procedure is applied on the p restricted partial log-likelihoods $pl(\beta_j)$. In each boosting iteration, the restricted partial log-likelihoods are "shifted" toward $\hat{\beta}_j$, obtaining the restricted penalized partial log-likelihoods $pl^{LB}_{pen}(\beta_j|\hat{\beta})$. The arguments of the maximums of these functions are the candidate updates, and that which maximizes the penalized partial log-likelihoods is added to the offset term. More precisely:

1. initialize $\hat{\beta} = (0, \ldots, 0)$;
2-3. compute the possible updates by a first order approximation around 0 of the restricted MPLE $\hat{b}^{LB}_j = \frac{pl^{LB}_{\beta_j}(0|\hat{\beta})}{-pl^{LB}_{\beta_j\beta_j}(0|\hat{\beta})}$;
4. select the best update $j^* = \text{argmin}_{1\leq j\leq p} pl^{LB}_{\beta_j}(0|\hat{\beta})^2/[-pl^{LB}_{\beta_j\beta_j}(0|\hat{\beta})]$;
5. update the estimate, $\hat{\beta}_{j*} = \hat{\beta}_{j^*} + \hat{b}^{LB}_{j^*}$.

Steps 2-3–4 are repeated m_{stop} times. Note that we mark the second step as "2-3" to highlight that it corresponds to steps 2 and 3 of the model-based algorithm. Here $pl^{LB}_{\beta_j}(\beta_j|\hat{\beta}) = \frac{\partial pl^{LB}_{pen}(\beta_j|\hat{\beta})}{\partial \beta_j}$ denotes the score and $pl^{LB}_{\beta_j\beta_j}(\beta_j|\hat{\beta}) = \frac{\partial^2 pl^{LB}_{pen}(\beta_j|\hat{\beta})}{\partial \beta_j^2}$ the observed information. The equation in step 4 is the first order approximation around 0 of $pl^{LB}_{pen}(\beta|\hat{\beta})$, implemented in *CoxBoost* for computational reasons (Binder and Schumacher 2008). Hereafter, we use the labels "LB" and "MB" to indicate whether the specific quantities are related to likelihood-based (LB) or model-based (MB) boosting.

3 Comparison

At first sight, the two boosting procedures seem quite different. The updates computed within the model-based boosting are based on the correlation between the observations and the negative gradient vector, while for the likelihood-based boosting this procedure involves the direct maximization of a log-likelihood. Moreover, the penalty term is applied in two completely different ways, to the updates in the former case, directly to the partial log-likelihood in the latter. In general, these aspects would render the two procedures incomparable: for example, the penalty parameters would shrink the estimates obtained in each boosting iteration very differently depending on the correlation structure of X; see "Appendix" for further details. In that they involve only one dimension of X at each iteration, however, the component-wise versions of the

boosting procedures implemented in *mboost* and *CoxBoost* are not affected by this issue. Moreover, we will see that the form of the linear predictor of the Cox model makes the two boosting procedures even more similar.

As a first step of the comparison, we rewrite the likelihood-based boosting procedure as a functional gradient descending technique, making explicit the role of the negative gradient vector in the formula used by *CoxBoost* to compute the possible updates:

1. initialize $\hat{\beta} = (0, \ldots, 0)$;
2. compute the negative gradient vector,
$$u = \left.\frac{\partial pl(F(X,\beta))}{\partial F(X,\beta)}\right|_{\beta=\hat{\beta}} = \left.\frac{\partial pl^{LB}_{pen}(F(X_j,\beta_j)|\hat{\beta})}{\partial F(X_j,\beta_j)}\right|_{\beta_j=0};$$
3. compute the possible updates,
$$\hat{b}^{LB}_j = \left(\left.\frac{\partial F(X_j,\beta_j)}{\partial \beta_j}\right|^{\top}_{\hat{\beta}_j=0} u\right) \Big/ \left(-\left.\frac{\partial \frac{\partial F(X_j,\beta_j)}{\partial \beta_j}}{\partial \beta_j}\right|^{\top}_{\hat{\beta}_j=0} u + \lambda\right); \quad (3)$$
4. select the best update,
$$j^* = \text{argmin}_{1\le j\le p}\left(\left.\frac{\partial F(X_j,\beta_j)}{\partial \beta_j}\right|^{\top}_{\hat{\beta}_j=0} u\right)^2 \Big/ \left(-\left.\frac{\partial \frac{\partial F(X_j,\beta_j)}{\partial \beta_j}^{\top} u}{\partial \beta_j}\right|_{\hat{\beta}_j=0} + \lambda\right);$$
5. update the estimate, $\hat{\beta}_{j*} = \hat{\beta}_{j^*} + \hat{b}^{LB}_{j^*}$.

As before, steps 2–5 are repeated m_{stop} times.

The formula of step 2 shows that, in the case, as that of Cox regression, of linear $F(X,\beta)$, the negative gradient vector of the penalized partial log-likelihood corresponds to that of the unpenalized version. The formula of step 3, moreover, clarifies that the weak estimator used in the likelihood-based approach is a particular form of base learner, which uses the negative gradient vector computed in the previous step to propose possible updates for the estimates of the regression coefficients. In particular, in (3) the score function and the observed information, which are the two terms used to compute the possible updates in the likelihood-based boosting (see Sect. 2.3), are derived from the negative gradient vector by applying the chain rule

$$\frac{\partial pl(\theta)}{\partial \theta} = \frac{\partial pl(F(\theta))}{\partial F(\theta)}\frac{\partial F(\theta)}{\partial \theta}.$$

This formulation makes it clear that both procedures rely on the negative gradient vector to identify the best "direction" in which the estimate can be improved, and both add this improvement, suitably penalized, to the current value of the estimate. The likelihood-based boosting uses the negative gradient vector to derive the score function and the observed information. We saw that these quantities are then used to compute the first order approximation of the maximum penalized partial likelihood estimate around 0. Let us focus on this quantity and contrast it with the update derived from the model-based boosting. The formulas are

$$\hat{b}_j^{LB} = \frac{\left.\frac{\partial F(X_j,\beta_j)}{\partial \beta_j}\right|_{\hat{\beta}_j=0}^{\top} u}{-\left.\frac{\partial \frac{\partial F(X_j,\beta_j)}{\partial \beta_j}^{\top} u}{\partial \beta_j}\right|_{\hat{\beta}_j=0} + \lambda} \quad \text{and} \quad \nu \hat{b}_j^{MB} = \nu \frac{X_j^{\top} u}{X_j^{\top} X_j},$$

respectively. Ignore for the moment the penalty parameters. Again for the linearity of $F(X, \beta)$, $\partial F(X_j, \beta_j)/\partial \beta_j = X_j$, and therefore the two numerators are equal. The same is not true for the denominators: the observed information for the Cox model, including the offset term $\hat{\eta} = X^{\top}\hat{\beta}$, indeed, is

$$\begin{aligned} &-\left.\frac{\partial \frac{\partial F(X_j,\beta_j)}{\partial \beta_j}^{\top} u}{\partial \beta_j}\right|_{\hat{\beta}_j=0} \\ &= \sum_{i=1}^{n} \delta^{(i)} \left\{ \frac{\sum_{l \in R^{(i)}} (X_j^{(l)})^2 e^{X^{(l)\top}\hat{\beta}} \sum_{l \in R^{(i)}} e^{X^{(l)\top}\hat{\beta}}}{\left(\sum_{l \in R^{(i)}} e^{X^{(l)\top}\hat{\beta}}\right)^2} - \frac{\left(\sum_{l \in R^{(i)}} X_j^{(l)} e^{X^{(l)\top}\hat{\beta}}\right)^2}{\left(\sum_{l \in R^{(i)}} e^{X^{(l)\top}\hat{\beta}}\right)^2} \right\}, \end{aligned}$$

clearly different from the simple $X_j^{\top} X_j$ of $\hat{b}_j^{MB}$. Using the negative gradient vector in the denominator as well, indeed, the likelihood-based boosting weak estimator takes into account the concavity of the loss function at the current point in the parametric space ($\hat{\beta}$), while the model-based boosting estimator uses a sort of parabolic approximation.

Remark 1 It is easy to see that in the case of Gaussian linear regression the two boosting techniques provide the same results (Mayr et al. 2014). In this case, indeed, the Gaussian density is used as the loss function and $\frac{\partial^2 l(\beta_j)}{\partial \beta_j^2} = X_j^{\top} X_j$; thus the two denominators are also equal. In the case of the generalized linear model, instead, the weak estimator for the likelihood-based boosting has the form

$$\hat{b}_j^{LB} = \frac{X_j^{\top} u}{V(\mu) X_j^{\top} X_j},$$

where $\mu = g^{-1}(F(X, \beta))$, with g being the canonical link function, and $V(\mu)$ the variance function (for more details, see McCullagh and Nelder 1989, Sect. 2) . Here again the denominator depends on $\hat{\beta}$ and the two algorithms provide different results. In the case of the generalized linear models, for model-based boosting it is possible to obtain the same updates as likelihood-based boosting by using a weighted least squares estimator instead of the simple least squares estimator.

Remark 2 In the previous remark we claimed that the two procedures in the linear Gaussian regression case provide the same results. This is true for suitable values of the penalty parameters ν and λ. With standardized X, simple algebra shows that

the two estimators are equal if $\lambda = n(1-\nu)/\nu$. In the likelihood-based boosting, λ can take values from 0, no penalty, to infinity, while in the model-based boosting, ν takes values between 0 and 1, where 0 corresponds to $\lambda = \infty$, and 1 to $\lambda = 0$. The recommendations of setting λ "sufficiently large" (Binder and Schumacher 2008) and ν "sufficiently small" (Bühlmann and Hothorn 2007), therefore, coincide.

Obviously, one could modify the values of the penalty parameters to force the two boosting procedures to give the same results in the Cox regression case as well. This can be done by setting

$$\lambda = \frac{X_j^\top X_j + \nu pl^{LB}_{\beta_j\beta_j}(0|\hat{\beta})}{\nu}. \tag{4}$$

Since this equation depends on $\hat{\beta}$, it is clear that different values for the penalty parameters should be provided in each boosting iteration. Alternatively, the matrix P should include suitable weights.

Remark 3 The learning paths of the two boosting procedures may also differ due to non-coinciding choices of which dimension should be updated at each boosting step. In *glmboost* the choice is based on the residuals of the regression of u on X_j, while *CoxBoost* selects the dimension which results in the largest decrease of the penalized partial log-likelihood function. The use of a penalized version of the loss function in this step may also be advantageous in the model-based boosting procedure (Mayr et al. 2014).

4 Allowing for mandatory variables

4.1 Background

In recent years, the importance of combining clinical and molecular data in a prediction model has become clear in the biomedical field, and studies have contrasted different methods to profitably exploit both kinds of data in the model building process (Boulesteix and Sauerbrei 2011; De Bin et al. 2014; Truntzer et al. 2014). The main issue related to the combination of clinical and molecular information is the different nature of the data, which belong to the low- and the high-dimensional worlds, respectively. The consequence is that, if not adequately treated, the risk of "losing" the clinical information among the high number of molecular variables is high (Binder and Schumacher 2008; Boulesteix and Sauerbrei 2011). From this point of view, several papers (e.g., Binder and Schumacher 2008; Boulesteix and Sauerbrei 2011; De Bin et al. 2014) show that it is possible to obtain better prediction models by considering the clinical variables as mandatory than by simply merging them with the omics data.

Both the R packages under investigation handle this issue by considering the clinical variables as mandatory: in *CoxBoost* there is the possibility to exclude some variables from the penalization (Binder and Schumacher 2008). With *mboost*, instead, it is possible to perform a two-step procedure in which the mandatory variables are summarized in a score (in survival analysis, typically the linear predictor of a Cox model) that is later used as an offset in the boosting procedure (Boulesteix and Hothorn 2010).

In Boulesteix and Sauerbrei (2011), these two strategies are called "favoring" and "residuals", respectively (De Bin et al. 2014 use the more intuitive term "clinical offset" for the latter strategy). These two strategies have some theoretical differences which may influence the model building process; a particular strategy may thus be more appropriate in some situations. For example, the "clinical offset" strategy implemented in *mboost* may lead to better results when there is a strong consensus among the experts on the clinical (mandatory) variables effect: within this strategy, the clinical regression coefficients are not modified by the boosting procedure, which uses only the molecular data to explain that part of the outcome variability not already explained by the clinical model.

In contrast, the "favoring" strategy implemented in *CoxBoost* allows the coefficients of the clinical variables to change during the boosting procedure: at each iteration, the coefficients of the clinical variables are adapted to take into consideration the information provided by the molecular variables. In this way, it may be possible to better integrate the clinical and the molecular information. Binder and Schumacher (2008) defined two ways to perform the stepwise update of the mandatory variables in the likelihood-based component-wise bootstrap framework. In the first, the $p_0 < n$ mandatory variables are considered in turn with one of the other $p - p_0$ variables, with the matrix P associated with the penalty term containing 0 for all mandatory variables. The second, implemented in *CoxBoost*, instead, consists of updating the regression coefficients of the mandatory variables as a further step before each boosting iteration.

As stated above, the two strategies to deal with mandatory variables have advantages that may depend on the structure of the data. Therefore, it would be valuable to have the chance to apply both strategies in both likelihood- and model-based boosting approaches, to extend the possibilities offered by the two packages. In the following, we consider without loss of generality that the first p_0 columns of X contain the mandatory variables.

4.2 Favoring strategy in *mboost*

We can allow the regression coefficients of the mandatory variables to vary through the iterations in the model-based boosting as in the likelihood-based one. In each boosting iteration, we simultaneously estimate the coefficients of the mandatory and one of the non-mandatory variables, i.e. for each $j = p_0 + 1, \ldots, p$, we regress the negative gradient vector on $X_j^+ = (X_1, \ldots, X_{p_0}, X_j)$. The choice of the best update is performed as in the regular algorithm, while the penalization is applied only to the last component of the update, that corresponding to the non-mandatory variable:

1. initialize $\hat{\beta} = (0, 0, \ldots, 0)$;
2. compute the negative gradient vector u;
3. for each optional variable, compute the possible updates of the coefficients estimates together with the mandatory variables, $\hat{b}_j = (X_j^{+\top} X_j^+)^{-1} X_j^{+\top} u$;
4. select the update which minimizes the residual sum of squares;
5. update the estimate $\hat{\beta}_{[1,\ldots,p_0,j^*]} = \hat{\beta}_{[1,\ldots,p_0,j^*]} + (\hat{b}_{j*[1]}, \ldots, \hat{b}_{j*[p_0]}, \nu \hat{b}_{j*[p_0+1]})$.

Step 2–5 are repeated m_{stop} times. The matrix $(X_j^{+\top} X_j^+)^{-1} X_j^{+\top}$ is common in each boosting iteration, and therefore it is sufficient to compute it only once for the $(p - p_0)$

non-mandatory variables. It is worth noting that in our implementation the regression coefficients of the mandatory variables are not shrunk toward 0. Potentially, one could shrink these coefficients by applying a penalty ν_{clin} to the relative components of the update.

4.3 Clinical offset strategy in *CoxBoost*

To implement the clinical offset strategy within the *CoxBoost* routine we need a preliminary step in which we fit a Cox model which includes the mandatory variables. The linear predictor is then included in $\hat{\eta}$ before the first boosting iteration, and the likelihood-based algorithm proceeds as usual. Since no iteration will involve the mandatory variables, their regression coefficients are not modified by the boosting procedure. It is worth noting that in this way the boosting works using a penalized restricted partial log-likelihood as a loss function, where the parameters related to the mandatory variables are replaced by their maximum partial likelihood estimates.

1. compute the maximum partial likelihood estimate for the coefficients corresponding to the mandatory variables, $\hat{\beta}_0, \ldots, \hat{\beta}_{p_0}$;
2. initialize $\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_{p_0}, 0, \ldots, 0)$;
3. compute, for $j = p_0 + 1, \ldots, p$, the potential updates using $\hat{b}_j^{LB}$;
4. determine which $\hat{b}_j^{LB}$ maximizes the penalized partial log-likelihood;
5. update the parameter estimate $\hat{\beta}_j = \hat{\beta}_j + \hat{b}_j^{LB}$ using the $\hat{b}_j^{LB}$ selected in step 4.

Steps 3–5 are repeated m_{stop} times. Please note that, from a prediction point of view, following this approach we also select the predictor with largest added predictive value.

5 Examples

5.1 Simulated data

In order to illustrate the similarities and differences between the likelihood-based and the model-based boosting outlined in Sect. 3, we conduct a very simple simulation study. We focus on the two-variable regression case, in which it is possible to visualize the likelihood function and show the boosting learning path of the component-wise boosting for the linear and the Cox regressions. We generate $n = 200$ observations (X_1, X_2) from a bivariate Gaussian distribution with mean 0 and covariance matrix

$$\Sigma = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}.$$

For the linear regression simulation, we generate the response $y_1, \ldots, y_n$ from a Gaussian distribution with variance 1 and mean $\beta_1 X_1 + \beta_2 X_2$, where $\beta_1 = 2$ and $\beta_2 = 3$. We center y around its mean and standardize X_1 and X_2. We use the recommended value of 0.1 for the model-based boosting penalty parameter ν (Bühlmann and Hothorn

2007) and the corresponding $\lambda = (n-1)(1-0.1)/0.1 = 1791$ for the likelihood-based boosting. Please note that the $(n-1)$ has replaced n due to the standardization performed with an unbiased estimator of the variance.

For the Cox regression simulation, we generate the survival times t through the formula

$$t = -0.1 \frac{\log(a)}{e^{\beta_1 X_1 + \beta_2 X_2}},$$

where a is generated from a uniform distribution between 0 and 1. If t is smaller than a random draw from an exponential distribution with rate 0.1, it is the observed time, while if it is larger, the observation is considered censored and the value generated from the exponential distribution is the observed time. For more details on this simulation model, see Binder and Schumacher (2008). In this case, we deliberately choose a non-optimal value for the penalty parameter in order to show more efficaciously the different learning paths. We set $\nu = 0.25$ for the model-based boosting and the corresponding $\lambda = (n-1)(1-0.25)/0.25 = 597$ for the likelihood-based boosting.

Figure 1 shows that, for the linear regression, the component-wise versions of the model-based and the likelihood-based boosting procedures provide the same results. Conversely, in Fig. 2, we clearly see that the equivalence does not hold in the case

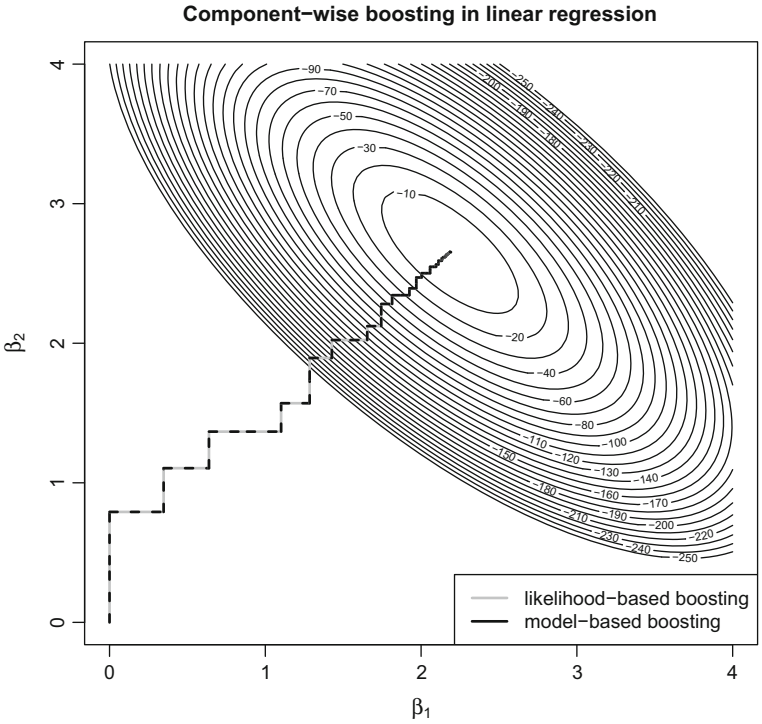

Fig. 1 Learning paths of the component-wise versions of the two boosting approaches in the linear regression simulation. The *contour lines* represent the levels of the normalized log-likelihood

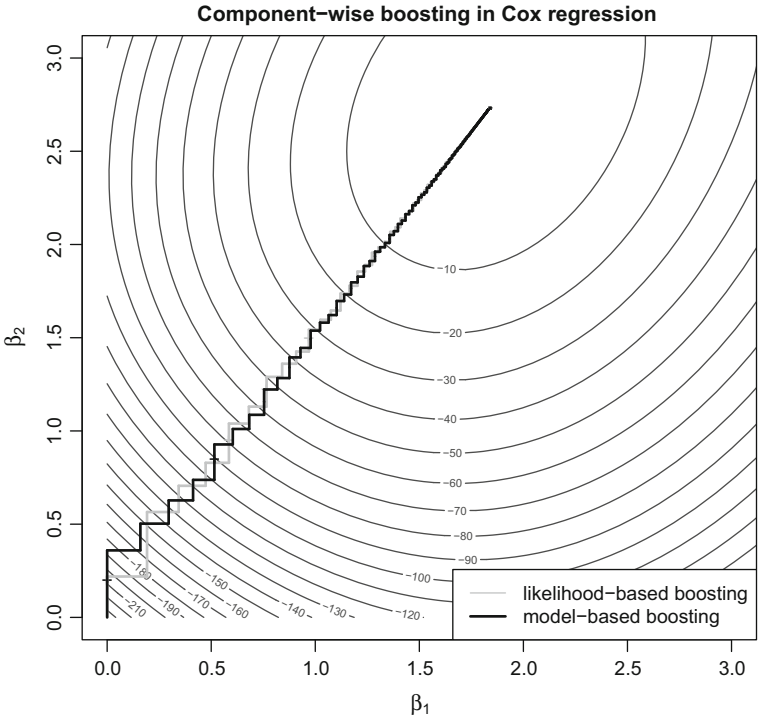

Fig. 2 Learning paths of the component-wise versions of the boosting approaches in the Cox regression simulation. The *contour lines* represent the levels of the normalized partial log-likelihood

of Cox regression. Please note that the magnitude of the difference between the two learning paths depends on the values of the parameters ν and λ. Reducing ν and, consequently, increasing λ, indeed, leads to learning paths increasingly similar, with both approaching the learning path of the least angle regression (Efron et al. 2004) for $\nu \to 0$ and $\lambda \to \infty$, respectively.

Table 1 reports the values obtained for $\hat{\beta}$ in the first 10 boosting iterations. Starting from $\hat{\beta} = (0, 0)$, we obtain the update candidates $\hat{b}_1^{[1]} = 0.204$, $\hat{b}_2^{[1]} = 0.219$ for *CoxBoost* and $\nu\hat{b}_1^{[1]} = 0.178$, $\nu\hat{b}_2^{[1]} = 0.191$ for *glmboost*. As we have seen, the differences lie in the denominator of the two estimators: the numerators coincide and are equal to 142.004 and 151.811, for $\hat{b}_1^{[1]}$ and $\hat{b}_2^{[1]}$, respectively. Here the superscript $[m]$ denotes the m-th boosting iteration. For *CoxBoost* these values are divided by the information, 694.845 and 692.533, respectively, while for *glmboost* they are divided by $X_j^\top X_j/\nu$, which, since we are using standardized predictors, corresponds to $(n-1)/\nu = 796$ for all j. The results are different unless Eq. (4) is satisfied. In particular, keeping λ fixed, this equation requires ν to be equal to 0.286 for $\hat{b}_1^{[1]}$ and to 0.287 for $\hat{b}_2^{[1]}$: the value 0.219 in Table 1 is indeed $0.191 \times 0.287/0.25$. In the first step, both techniques select the second candidate: the values of the loss function is -29.021 versus -33.279 in *CoxBoost* (approximation of the profile penalized partial log-likelihood) and 67.442 versus 52.961 in *glmboost* (residual sum of squares). It is

Table 1 Estimates of the Cox regression coefficients in the first 10 steps of the boosting procedures

Boosting	*CoxBoost*		*mboost*	
Iteration	$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_1$	$\hat{\beta}_2$
0	0.000	0.000	0.000	0.000
1	0.000	0.219	0.000	0.191
2	0.192	0.219	0.000	0.359
3	0.192	0.402	0.160	0.359
4	0.192	0.565	0.160	0.503
5	0.344	0.565	0.296	0.503
6	0.344	0.706	0.296	0.627
7	0.473	0.706	0.413	0.627
8	0.473	0.829	0.413	0.738
9	0.585	0.829	0.515	0.738
10	0.585	0.939	0.515	0.836

not always the case that the two boosting procedure select the same dimension, as we can see in the second step: here, the candidates are $\hat{b}_1^{[2]} = 0.192$, $\hat{b}_2^{[2]} = 0.192$ for *CoxBoost* and $\nu\hat{b}_1^{[2]} = 0.168$, $\nu\hat{b}_2^{[2]} = 0.168$ for *glmboost*, which lead to the values of the loss functions -25.385 versus -25.348 (*CoxBost*) and 54.349 versus 53.486 (*glmboost*). In this case, the former technique update $\hat{\beta}_1$, while the latter again $\hat{\beta}_2$.

5.2 Real data

Here we show the application of the favoring and the clinical offset strategies in a real data example. We use the data on colon cancer presented in a study by Marisa et al. (2013) and publicly available from the *ArrayExpress* web repository, reference number E-GEOD-39582 (http://www.ebi.ac.uk/arrayexpress/experiments/E-GEOD-39582/).

The dataset contains a training and a test set with 443 and 123 observations, respectively. Of these observations, 10 were discarded due to missing values, reducing the sample sizes to 439 (training set) and 117 (validation set) observations. In particular, the effective sample sizes, i.e. the number of observations with an event ($\delta^{(i)} = 1$), are 141 and 36. We have information on 4 clinical variables, namely *sex*, *age*, *subtype* and *stage*. The latter two are categorical variables with 6 and 4 modalities, which are transformed into dummy variables, using the codification $(-1; 1)$ (see "Appendix" for further details). The molecular data consist of 54675 gene expressions determined on Affymetrix U133 Plus 2.0 chips.

We standardize the continuous variables, namely the age and the gene expressions, in order to meet the model-based (centering) and the likelihood-based (homoschedasticity) boosting algorithms requirements. The means and the standard deviations for the standardization process are computed on the training set only. Regarding the tuning parameter, we set the penalty equal to the default value 0.1 for the model-based approach and equal to the result of the routine *optimCoxBoostPenalty* (available in the package *CoxBoost*) for the likelihood-based approach. Please note that the choice

of this tuning parameter is not much relevant, conversely to that of the number of boosting iterations m_{stop}. For both boosting approaches, we select the latter tuning parameter via 15-time repeated tenfold cross-validation, which consists of merging the results of 15 separated tenfold cross-validation replications. This approach makes the choice robust to the specific split of the data in the tenfold (see also Boulesteix et al. 2013).

Using the Brier score (Graf et al. 1999), a time-dependent quadratic score for survival data, we evaluate the performance of the models obtained by following different strategies of combining clinical and molecular data. The results are summarized in the so-called "prediction error curves" generated using the R packages *pec* (Gerds 2014). The curves show the prediction error (Brier score) at each time: the lower the curve, the better is the prediction ability of the model.

Note that all models are trained on the training set and their performance evaluated in the test set only. As seen in Fig. 3, in this example there is a small improvement in performing the favoring strategy (continuous line) rather than the clinical offset strategy (dashed line), for both model-based boosting (left graphic) and likelihood-based boosting (right graphic). Note that for the latter approach the offset-based strategy seems to perform slightly better in the very right part of the curves, but this is not really relevant because the prediction curves for higher values of time are known to be unstable, due to the scarcity (or even absence) of events. This example, moreover, shows very clearly the importance of treating the clinical and the molecular variables differently in the model building process. If we simply merge together the two kinds of data (naive strategy), thus ignoring their differences, we obtain a decidedly worse prediction (dot-dashed line) than those obtained with the aforementioned strategies

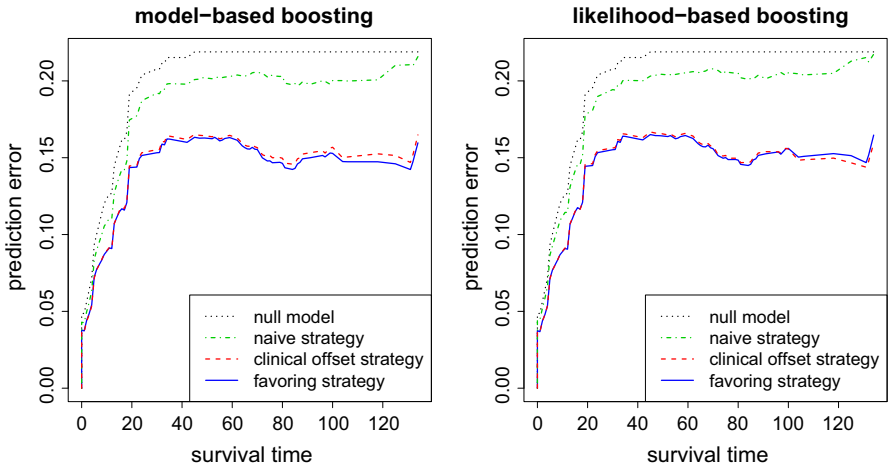

Fig. 3 Prediction error curves—using the Brier score—computed on the test set for the null model (*dotted line*) and for the boosting models (*left graphic* model-based, *right graphic* likelihood-based) obtained following a naive strategy (*dot-dashed line*), a clinical offset strategy (*dashed line*) and a favoring strategy (*continuous line*)

(favoring and clinical offset). Here, the dotted lines show the prediction curve for the null model, in which no variable is used to predict the outcome.

In this specific example, we note that the two algorithms have similar performance. If we compare the prediction error curves, we obtain the best result with the model-based boosting approach within the favoring strategy. Anyway, the differences are very small and, in any case, a single example is not sufficient to drive any conclusion on which algorithm/strategy combination is preferable. From a computational point of view, the algorithm implemented in *mboost* is faster than that implemented in *CoxBoost*. In our analysis, the former needs 6.808 s to compute the model, the latter 83.618; these times are computed following the naive strategy and setting $m_{stop} = 24$ (which is the tuning parameter obtained for *CoxBoost*).

The R code necessary to reproduce the results (both for simulated and real data examples) is available as Electronic Supplementary Material.

6 Discussion

In this paper, we contrasted the model- and likelihood-based boosting algorithms used by the R packages *mboost* and *CoxBoost* to implement the linear Cox model, a simple (but highly relevant) case in which the effects of the regression parameters are relatively easy to understand. To maintain the ease of interpretability, we did not consider more sophisticated boosting versions which include non-linear effects (Schmid and Hothorn 2008; Hofner et al. 2013) or rely on a tree-based approach (Ridgeway 2010). A different possibility to treat survival data using *mboost* is provided by Hothorn et al. (2006): instead of using the partial log-likelihood as the loss function, they modify the boosting algorithm for the linear regression (Bühlmann and Yu 2003, L_2 Boosting), adapting the loss function (Gaussian log-likelihood) and the least squares estimator by adding weights to take into account the censored nature of the data (Hothorn et al. 2006; Bühlmann and Hothorn 2007). Specifically, the inverse probability of censoring weights (Van der Laan and Robins 2003) are used.

We also note that in this paper we considered the theoretical similarities and differences between the two boosting algorithms, without trying to systematically evaluate their performance. The complexity of the real world renders it impossible to identify all situations in which one algorithm performs better than the other: from this point of view, it is of benefit to simply have different solutions to enrich the practitioner's toolbox. As such, an important part of this paper is devoted to the extensions of the solutions available in only one R package to the other. However, it is important to use these possibilities wisely, without falling into the temptation of trying all and then reporting only the results for the method which shows the greatest support for the theory under investigation.

Acknowledgments RDB was financed by Grant BO3139/4-1 from the German Science Foundation (DFG). Special thanks are devoted to Anne-Laure Boulesteix for her advice and suggestions, to Rory Wilson for his help with linguistic improvements and to the two anonymous reviewers for their comments which led to an improved version of the paper.

Appendix

In the paper we showed that, in the case of linear Cox model, the algorithms used by the R packages *mboost* (through the function *glmboost*) and *CoxBoost* follow different learning paths, conversely to the Gaussian linear regression case in which the same result is produced, provided $\lambda = n(1-\nu)/\nu$ (Binder 2013b). Note that the equivalence in the linear regression case only works for the component-wise version of boosting. In the non-component-wise version, when all the dimensions of $\hat{\beta}$ are updated simultaneously, indeed, the two weak estimators have the form

$$\hat{b}^{LB} = (X^\top X + \lambda P)^{-1} X^\top u \qquad \text{and} \qquad \nu \hat{b}^{MB} = \nu (X^\top X)^{-1} X^\top u,$$

for the likelihood- and the model-based boosting, respectively. While the model-based penalty ν affects all dimensions identically, the penalty λ penalizes the dimensions depending on the correlation structure of X. Consider $P = I_p$, the identity matrix used as default in *CoxBoost*. The weak estimator $\hat{b}^{LB}$ is then a ridge estimator: in the regression procedure, when the response is projected onto the orthonormal basis of the explanatory variables (columns of X), the penalty term shrinks the coordinates (inversely) proportionally to the variance of the related principal components. This means, in particular, that the term λ penalizes (shrinks) the coefficients related to principal components with low variance more strongly. If we look at the predictive values obtained through the two algorithms, denoting with B the orthonormal basis of the columns of X, we obtain

$$\hat{y}^{MB} = B \ \mathrm{diag}(1 - (1-\nu)^{m+1}) B^\top y$$
$$\hat{y}^{LB} = B \ \mathrm{diag}(1 - (1 - \frac{d_j}{d_j + \lambda})^{m+1}) B^\top y,$$

where d_j, $j = 1, \ldots, p$ is the j-th eigenvalue of the matrix $X^\top X$ (when divided by n, it is then the variance of the principal component) and m indicates the number of iterations performed. Replacing m by 0 we obtain the formula for the single step update. It is worth noting that, due to its stage-wise nature, the algorithm of boosting ridge regression leads to a different penalization (and, therefore, to different estimates) than the usual ridge regression (Tutz and Binder 2007).

We can obtain a uniform penalization with the likelihood-based boosting by setting $P = (1/n) X^\top X$, provided that the columns of X are centered around 0 and standardized. In this case, $(1/n) X^\top X$ represents the correlation matrix of X.

Note that it is possible to use a penalized least squares estimator within the model-boosting algorithm as well. This solution seems to be gaining popularity (see, e.g., Hofner et al. 2014), because it allows the better handling of categorical variables. Please note that in this case the whole penalization is a combination of the effect of λ and ν. Another issue related to categorical variables is with regard to their variance. In this paper, we considered standardized X, but we saw in the examples that in practical situations we need a standardization process. The likelihood-based boosting, in particular, needs all X_j having variance 1. For this reason, in the real data example

we codified the dummy variables as $(-1; 1)$. In the case of completely balanced observations, the variance of the binary variables is then equal to 1. Unfortunately, this balance occurs rarely in practice. A possible solution which does not require the standardization of X is to replace P by the covariance matrix of X or, for the component-wise version, with $\text{diag}((1/n)X^\top X)$: this standardizes the binary variables using their observed standard deviations as well.

References

Binder H (2013a) CoxBoost: Cox models by likelihood based boosting for a single survival endpoint or competing risks. R package version 1.4. http://CRAN.R-project.org/package=CoxBoost

Binder H (2013b) GAMBoost: generalized linear and additive models by likelihood based boosting. R package version 1.2-3. http://CRAN.R-project.org/package=GAMBoost

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

Boulesteix AL, Hothorn T (2010) Testing the additional predictive value of high-dimensional molecular data. BMC Bioinform 11:78

Boulesteix AL, Sauerbrei W (2011) Added predictive value of high-throughput molecular data to clinical data and its validation. Brief Bioinform 12:215–229

Boulesteix AL, Richter A, Bernau C (2013) Complexity selection with cross-validation for lasso and sparse partial least squares using high-dimensional data. In: Lausen B, Van den Poel D, Ultsch A (eds) Algorithms from and for nature and life. Springer, Cham, Switzerland, pp 261–268

Breiman L (1998) Arcing classifier. Ann Stat 26:801–849

Bühlmann P (2006) Boosting for high-dimensional linear models. Ann Stat 34:559–583

Bühlmann P, Hothorn T (2007) Boosting algorithms: regularization, prediction and model fitting. Stat Sci 22:477–505

Bühlmann P, Yu B (2003) Boosting with the L_2 loss: regression and classification. J Am Stat Assoc 98:324–339

Cox D (1972) Regression models and life-tables. J R Stat Soc Ser B (Methodological) 34:187–220

De Bin R, Sauerbrei W, Boulesteix AL (2014) Investigating the prediction ability of survival models based on both clinical and omics data: two case studies. Stat Med 33:5310–5329

Efron B, Hastie T, Johnstone I, Tibshirani R (2004) Least angle regression. Ann Stat 32:407–499

Freund Y (1995) Boosting a weak learning algorithm by majority. Inf Comput 121:256–285

Freund Y, Schapire R (1996) Experiments with a new boosting algorithm. In: Proceedings of the 13th international conference on machine learning. Morgan Kaufmann Publishers Inc., pp 148–156

Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: a statistical view of boosting. Ann Stat 28:337–407

Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Ann Stat 29:1189–1232

Gerds T (2014) pec: Prediction error curves for risk prediction models in survival analysis. R package version 2.4-4. http://CRAN.R-project.org/package=pec

Graf E, Schmoor C, Sauerbrei W, Schumacher M (1999) Assessment and comparison of prognostic classification schemes for survival data. Stat Med 18:2529–2545

Hofner B, Hothorn T, Kneib T (2013) Variable selection and model choice in structured survival models. Comput Stat 28:1079–1101

Hofner B, Mayr A, Robinzonov N, Schmid M (2014) Model-based boosting in R: a hands-on tutorial using the R package mboost. Comput Stat 29:3–35

Hothorn T, Bühlmann P, Dudoit S, Molinaro A, Van Der Laan MJ (2006) Survival ensembles. Biostatistics 7:355–373

Hothorn T, Buehlmann P, Kneib T, Schmid M, Hofner B, Sobotka F, Scheipl F (2015) mboost: Model-based boosting. R package version 2.5-0. http://CRAN.R-project.org/package=mboost

Marisa L, de Reyniès A, Duval A, Selves J, Gaub MP, Vescovo L, Etienne-Grimaldi MC, Schiappa R, Guenot D, Ayadi M, Kirzin S, Chazal M, Fljou JF, Benchimol D, Berger A, Lagarde A, Pencreach E, Piard F, Elias D, Parc Y, Olschwang S, Milano G, Laurent-Puig P, Boige V (2013) Gene expression classification of colon cancer into molecular subtypes: characterization, validation, and prognostic value. PLoS Med 10(e1001):453

Mayr A, Hofner B, Schmid M (2012) The importance of knowing when to stop. A sequential stopping rule for component-wise gradient boosting. Methods Inf Med 51:178–186
Mayr A, Binder H, Gefeller O, Schmid M (2014) The evolution of boosting algorithms. Methods Inf Med 53:419–427
McCullagh P, Nelder J (1989) General linear models. Chapman and Halls, London
R Development Core Team (2014) R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria
Ridgeway G (1999) Generalization of boosting algorithms and applications of Bayesian inference for massive datasets. Ph.D. thesis, University of Washington
Ridgeway G (2010) gbm: Generalized boosted regression models. R package version 1.6. http://CRAN.R-project.org/package=gbm
Schapire RE (1990) The strength of weak learnability. Mach Learn 5:197–227
Schmid M, Hothorn T (2008) Flexible boosting of accelerated failure time models. BMC Bioinform 9:269
Truntzer C, Mostacci E, Jeannin A, Petit JM, Ducoroy P, Cardot H (2014) Comparison of classification methods that combine clinical data and high-dimensional mass spectrometry data. BMC Bioinform 15:385
Tutz G, Binder H (2006) Generalized additive modeling with implicit variable selection by likelihood-based boosting. Biometrics 62:961–971
Tutz G, Binder H (2007) Boosting ridge regression. Comput Stat Data Anal 51:6044–6059
Van der Laan MJ, Robins JM (2003) Unified methods for censored longitudinal data and causality. Springer, New York