HEC Montréal

Stacking with time-dependent and covariate-dependent weights with survival data

Par

Wei Qin

Département des sciences de la décision

HEC Montréal

Mémoire présenté à HEC Montréal

en vue de l'obtention du grade de Maître ès sciences (M.Sc.)

en Science des données et analytiques d'affaires

Juillet 2021

RÉSUMÉ

L'idée de base du stacking est de combiner k modèles à l’aide d’un modèle linéaire généralisé. Chaque modèle reçoit un poids différent et, en pratique, ces poids doivent être estimés à partir des données, généralement par validation croisée. Ce mémoire se concentre sur les données de survie avec des observations censurées à droite.

Nous savons que la probabilité de survie individuelle diminue avec le temps t, et nous voulons estimer la fonction de survie en utilisant la méthode du stacking. Nous étudions deux extensions de la méthode du stacking de base. La première consiste à laisser les poids être une fonction des covariables x, ce qui peut être intéressant lorsque certains modèles sont bons dans certaines parties de l'espace des covariables mais pas dans d'autres parties. Les poids peuvent s'ajuster d'eux-mêmes en fonction des forces des modèles. La deuxième extension consiste à laisser les poids être une fonction du temps t où le poids du modèle k change au fur et à mesure que le temps t change. Cela peut être intéressant lorsque certains modèles sont bons pour estimer la fonction de survie pour une partie de l'intervalle de temps mais pas dans d'autres parties. Là encore, les poids peuvent s'ajuster d'eux-mêmes en fonction des forces des modèles.

Ce mémoire combine ces deux extensions. Les poids peuvent dépendre à la fois du temps t et des covariables x. Ici, nous pouvons utiliser des "model-based trees" pour diviser le modèle selon x afin d'obtenir les poids à chaque temps t d'une grille. Des pseudo-observations sont utilisées pour traiter la censure et estimer les poids.

Cette approche offre une nouvelle méthode de stacking pour l'analyse de prédiction des données de survie. Les résultats de simulations et des applications à des ensembles de données réelles montrent que cette méthode de stacking pondéré peut effectivement choisir le bon modèle et lui donner un poids plus élevé. En d'autres termes, cette méthode de stacking peut combiner automatiquement les bonnes parties de chaque modèle pour avoir la possibilité d'obtenir un meilleur résultat de prédiction. Une étude plus approfondie sera nécessaire pour définir la grille de temps appropriée à utiliser pour optimiser les performances.

Mots-clés : Données de survie, stacking, modèles d'ensemble, poids, temps, covariables

ABSTRACT

The basic stacking idea is to combine K models by generalized linear model. Each model receives a different weight, and in practice, these weights must be estimated from the data, usually by cross-validation. This thesis focuses on survival data with right-censored observations.

We know that the individual survival probability decreases with time t, and we want to estimate the survival function using the stacking method. We investigate two extensions of the basic stacking method. The first extension is to allow the weights to be a function of the covariate X, which may be interesting when some models are good in some parts of the covariate space, but not in other parts. The weights can adjust themselves according to the strengths of the models. The second extension is to let the weights be a function of time t, where the weight of model k changes as the time t changes. This may be interesting when some models are good to estimate the survival function for some part of the time range, but not in other parts. Again, the weights can adjust themselves according to the strengths of the models.

This thesis combines these two extensions. The weights can depend both on the time t and covariates X. Here, we can use "model-based trees" to split the model according to X to get the weights at each time t of a grid. Pseudo-observations are used to deal with the censoring and to estimate the weights.

This approach provides a new stacking method for survival data prediction analysis. The results of simulations and applications to real data sets show that this weighted stacking method can effectively choose the good performance model and give it a higher weight. In other words, this stacking method can automatically combine the good parts of each model to provide the possibility of getting a better prediction result. Further study will be required to define the appropriate time grid that should be used to optimize the performance.

Keywords: Survival data, stacking, ensemble models, weights, time, covariates

Table of Contents

List of tables

List of figures

Acknowledgements

I would like to thank the following people, without whom I would not have been able to complete this research, and without whom I would have not made it through my master's degree.

First of all, I would like to express my sincere gratitude to my supervisor Professor Denis Larocque for his consistent support and guidance throughout this project and the difficult period of the COVID-19 pandemic. His insight and knowledge always open up new directions and provide me with detailed, easy-to-understand guidance whenever I encounter research difficulties. His world-class expertise and rigorous academic attitude have deeply influenced me for my rest of life.

I would also like to thank my family for all the support they have shown me through this research. I thank my husband Yu for his patience, encouragement, and support in my daily life. A special thanks to my parents in China for their understanding and support.

Finally, I would like to thank the members of the jury for their time.

Introduction

Survival analysis is used to analyze data in which the time until an event is of interest. The response is often referred to as a failure time, survival time or event time. Many medical trials involve following patients for a long time, e.g., time until tumor recurrence or time until cardiovascular death after some treatment intervention. The follow-up time for the study may range from a few weeks to many years. Over the years, survival analysis has been applied in various other domains, such as predicting the churning of customers, estimating the time at which a equipment failure, etc. In the former example, the time the person begins to be a customer with a company can be thought of as the birth event, and the time of leaving the company can be considered the death event.

In many studies, when we perform survival analysis, it is often the case that we may not have exact failure times for all observations, as lifetime data are often "censored". This is because the event has not yet occurred for some subjects at the end of study. This situation is called right-censoring; that is, some subjects may live longer than the duration of the study, or left early before experiencing the event of interest. There are other types of censored data, such as left-censored data, a situation in which when we start our study, the event has already occurred, but we do not know exactly when. Therefore, the event time of a left-censored subject has occurred before a particular time. If we know the event occurred within some interval of time, the observations would be interval censored. Additionally, due to a systematic selection process inherent to the study design, the truncation may cause an observation to be incomplete. Among all these situations of censoring or truncation, right-censoring is the most common situation for survival analysis in practice.

We are interested in how a risk factor or treatment affects time to disease or some other event. Many models to link the covariates to the target time T are available. The most popular model is the Cox model (proportional hazard model), which is semiparametric. This model specifies how the covariates modify the hazard function, but the hazard function itself is not completely specified. Otherwise, the accelerated failure time (AFT) model is fully parametric. We also have advanced methods like survival trees and forests, boosting for survival data, and regularization with survival data.

The prediction of survival analysis can be the survival function or survival time. This thesis focus on survival function to predict the probabilities of event occurrence at time t with enough time points. Among all the methods of survival analysis, each model has its own advantage. This thesis provides a new stacking method which may combine the benefits of the existing models, and its weights can depend both on time t and covariates X to improve prediction performance for right censoring survival data.

Chapter 1

The current practical methods in survival analysis

Models like linear regression, decision tree, random forests and boosting for general data modeling could also be applied to survival data. However, these models should be modified and adjusted to accommodate the properties of survival data, such as censoring.

The Cox model, the AFT model and survival forests are currently the most-used practical methods in survival analysis. The Cox model and the AFT model are developed from linear regression, while survival forests are based on the general random forests. To better understand these models, some fundamental concepts must first be known in survival analysis.

The first basic notion is the survival function. It gives the probability that a subject will survive past time t. If we let $T > 0$ denote the response variable, i.e., the survival time, the survival function is defined as

$$S(t) = \Pr(T > t) = 1 - F(t) \tag{1}$$

Where $F(t)$ is the cumulative distribution function of T.

We could use the time of death of humans as an example. At time $t = 0$, the birth time, the probability that they will survive time 0 is certain, which means $S(0) = 1$. As humans grow up and time passes by, their survival probability will decrease, the probability that they will reach their second birthday is less than the probability of reaching their first birthday, and we can say with certainty that they will not live forever. This means that when time goes to infinity, death will definitely come. Their survival probability is 0, $S(\infty) = 0$. In theory, we could conclude that the survival function is non-increasing. In practice, we observe events on a discrete time scale like hours, days, weeks, etc.

Another definition is the hazard function $h(t)$. It is the instantaneous rate at which an event occurs, with the condition that it survives up to time t. In other words, there is no previous event. Its definition is:

$$h(t) = \lim_{\Delta t \to 0} \frac{\Pr(t < T \leq t + \Delta t | T > t)}{\Delta t} = \frac{f(t)}{S(t)} \qquad (2)$$

where $f(t)$ is the density function. Another interesting function is the cumulative hazard function, or integrated hazard function. It is defined by:

$$H(t) = \int_0^t h(u) du \qquad (3)$$

This is the integral of the hazard function up to time t. It measures the cumulated risk up to a given time. The greater $H(t)$ is, the greater the risk that the event occurs by time *t*.

If any one of the functions $h(t)$, $H(t)$, or $S(t)$ is known, we can derive the other two functions by using the equations:

$$h(t) = -\frac{\partial \log(S(t))}{\partial t} \qquad (4)$$

$$H(t) = -\log(S(t)) \qquad (5)$$

$$S(t) = \exp(-H(t)) \qquad (6)$$

Because survival data often has censoring, it is necessary to have a specific notation to represent it. In this thesis, we define:

- T_i to be the survival time for the ith subject
- C_i to be the censoring time for the ith subject
- δ_i to be the event indicator

$$\delta_i = \begin{cases} 1 \text{ if the event was observed} & (T_i \leq C_i) \\ 0 \text{ if the response was censored} & (T_i > C_i) \end{cases}$$

- The observed response as $Y_i = \min(T_i, C_i)$

These basic definitions related to survival function will be used in Section 1.1 to 1.5, in which we briefly introduce the theory of the current practical estimation methods of survival function, including the non-parametric method, the semi-parametric method Cox model, the full-parametric method AFT model, survival forests and other methods.

1.1 Non-parametric estimation of survival function

To estimate the survival function of one specific group, it may not be necessary to design a model with estimated parameters. As we know each subject's event time or censored time, we could calculate its empirical survival probability in a time range by computing the percentage of survival numbers. The cumulative survival probability could be computed by the multiplication of survival probabilities of consecutive time intervals.

Whether or not the data that exist are censored, the non-parametric Kaplan-Meier method can be used to estimate $S(t)$. We define t_i as the time when the event happened, d_i as the number of events that happened at time t_i, and n_i as the individuals known to have survived, but have not yet had an event or been censored up to time t_i. $(1-\frac{d_i}{n_i})$ is exactly the survival probability for the time interval (t_{i-1}, t_i). By multiplying the survival probabilities of consecutive time intervals for $t_i \leq t$, the Kaplan-Meier estimator of the survival function $S(t)$ can be obtained by:

$$\widehat{S(t)} = \prod_{i:t_i \leq t}(1-\frac{d_i}{n_i}) \tag{7}$$

The Kaplan-Meier estimate is the simplest way of computing the survival over time with censored observations. It involves the computing of probabilities of occurrence of an event at a certain point in time and multiplying these successive probabilities by any earlier computed probabilities to get the final estimate. This method does not involve covariates X. Therefore, it can only generally present the survival function of this group, as we could not get the individual estimated survival function according to specific characteristics of one subject.

1.2 Cox model

The Cox model is a well-recognized statistical technique for analyzing survival data, and likely the model used most often. This model is based on the risk function. The part of the effects of the covariates is similar to linear regression without an intercept. For a subject with covariates values $x_1, x_2, \dots, x_p$, its risk function can be written as the equation $h\left(t \middle| x_1, x_2, \dots, x_p\right) = h_0(t)\exp\left(\beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p\right)$ where $h_0(t)$ is the baseline hazard function that is left unspecified.

If all the x_i are equal to zero, $h\left(t \middle| x_1, x_2, \dots, x_p\right) = h_0(t)\exp\left(\beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p\right) = h_0(t)\exp(0) = h_0(t) * 1 = h_0(t)$, the risk function corresponds to the value of the baseline hazard function $h_0(t)$. $h_0(t)$ may vary over time, as it is a function of t.

The parameters $\beta_1, \beta_2, \dots \beta_p$ are the effects of the covariates on this baseline hazard function. From the formula, it is evident that increasing x_j by 1, with the other covariates fixed, multiplies the risk by $\exp(\beta_j)$. The quantities $\exp(\beta_j)$ are called hazard ratios (HR). A value of β_j greater than zero, or equivalently a hazard ratio greater than one, indicates that as the value of the ith covariate increases, the event hazard increases and thus the length of survival decreases. In summary,

$HR = 1$: No effect

$HR < 1$: Reduction in the hazard, increase of survival probability and survival time

$HR > 1$: Increase in hazard, decrease of survival probability and survival time

For linear regression, we can usually estimate the parameters by MLE (maximum likelihood). Since the Cox model is semi-parametric because of the baseline hazard function, the usual MLE cannot be used to estimate the parameters. However, it is still possible to estimate the parameters by maximizing the "partial likelihood" function. The partial likelihood function is defined as follows:

$$L(\beta) = \prod_{i:\delta_i=1} \frac{h(t_i|X_i)}{\sum_{j:t_j \geq t_i} h(t_j|X_j)} = \prod_{i:\delta_i=1} \frac{\exp(\beta' X_i)}{\sum_{j:t_j \geq t_i} \exp(\beta' X_j)} \quad (8)$$

From the formula above, the term $\frac{h(t_i|X_i)}{\sum_{j:t_j\geq t_i} h(t_j|X_j)}$ can be considered as the probability that subject *i* experiences the event among all the other subjects which are still at risk. To maximize the product of all the terms of the subject without censoring is the theory of maximum likelihood. By eliminating the same term $h_0(t)$, the estimated parameters $\widehat{\beta}'$ for each covariate could be solved through formula (8).

From equation (6), we could get

$$\hat{S}(t) = \exp\left(-\widehat{H}(t)\right) = \exp\left(-\widehat{h_0}(t)\exp\left(\widehat{\beta}'X\right)\right)$$

$$= \exp\left(-\widehat{h_0}(t)\right)^{\exp(\widehat{\beta}'X)} = \left(\widehat{S_0}(t)\right)^{\exp(\widehat{\beta}'X)} \quad (9)$$

where $\widehat{S_0}$ is an estimation of the baseline survival function. Chapter 4 of Kalbfleisch and Prentice (2011)[1] and section 3.5 of Hosmer Jr and Lemeshow (1999)[2] explain how to get the estimated baseline survival function. The Survival package[3] in R provides the function *basehaz* to calculate cumulative baseline hazard for the Cox model which can be used to calculate the survival function for risk-based algorithms.

The Cox model is a relative risk model, and risk estimations for the test data can be predicted. The calculated values are the risk for a person with the given set of covariates relative to an average person. With formula (9), the estimated survival functions of the test data can also be computed. But it is not straightforward to get a predicted expected event time, as the estimated survival curve does not always end at 0, for example when the largest observed time is censored. The estimated median survival time could also be unavailable if the estimated survival function becomes undefined before it reaches 0.5.

The Cox model is also limited by its restrictive application conditions that the hazards should be proportional. If the studied variable consisted of a large number of classes, the hazards were rarely proportional all along the curve, as seen in the example used by F Bugnard et al. (1994)[4], and in the demonstration that the Cox regression model may lead to the creation of a false model that does not include only time-independent predictive factors when violating the proportional hazard assumption in the Cox proportional hazards analysis in Mortality prediction of patients with Acute Coronary Syndrome by Magdalena Babinska et al. (2015)[5].

1.3 AFT model

The most common fully parametric model with survival data is the accelerated failure time model (AFT). Linear regression and the exponential function are used for the Cox proportional hazard model to estimate the effect of the covariates on the risk. The AFT model assumes that the effect of the covariates works directly on the "time to event" by accelerating or decelerating. Hence, the model could be written as:

$$T = \exp\left(\beta_1 x_1 + \beta_1 x_2 + \cdots + \beta_p x_p\right) T_0 \tag{10}$$

Here, it is supposed that the exponential function links the covariates X and the parameters β by multiplication to the reference event time T_0 which equals the value T when all covariates $X = 0$. The linking function could be specified with other functions.

$\log(T_0)$ is usually used to model the reference time T_0 by

$$\log(T_0) = \beta_0 + \sigma\epsilon \tag{11}$$

where ϵ is an error term from a given distribution, and σ is a scale parameter.

Then, we can get

$$\begin{aligned}\log(T) &= \log\left(\exp\left(\beta_1 x_1 + \beta_1 x_2 + \cdots + \beta_p x_p\right) T_0\right) \\ &= \left(\beta_1 x_1 + \beta_1 x_2 + \cdots + \beta_p x_p\right) + \log(T_0) \\ &= \left(\beta_1 x_1 + \beta_1 x_2 + \cdots + \beta_p x_p\right) + \beta_0 + \sigma\epsilon \end{aligned} \tag{12}$$

Assuming that ϵ is the $N(0,1)$ distribution with $\log(T)$ as the dependent variable, then the model is a basic linear regression model, and T follows a log-normal distribution. Other distributions are possible for T_0, thus for T, like exponential, Weibull, log-normal, log-logistic, and generalized gamma.

Compared with the Cox model where it is sometimes difficult or not available to predict expected lifetime or median time, the fully parameterized AFT model could provide solutions.

It can predict a new subject at time 0 and can also compute the predictions for an ongoing subject at any given time t^*, even if it is censored. Thus, the expected lifetime can be obtained by the formula

$$E[T|T > t^*] = t^* + \frac{\int_{t^*}^{\infty} S_T(t)dt}{S_T(t^*)} \tag{13}$$

where $\frac{\int_{t^*}^{\infty} S_T(t)dt}{S_T(t^*)}$ is the estimated expected residual lifetime if $S_T(t)$ is replaced by an estimator from AFT. Similarly, the conditional median given that the subject has survived up to t^* is the value t_m, such that

$$\frac{S_T(t_m)}{S_T(t^*)} = 0.5 \tag{14}$$

Because the AFT model is fully parameterized, it can provide survival function predictions from time $t = 0$ to the end for a new subject, regardless of whether it is censored or not. Hence, it offers a potentially useful statistic approach that is based upon the survival curve rather than the hazard function when the largest observed time is censored. It can estimate the expected lifetime by formula (13) and (14) as its advantage compared to the Cox model. In addition, in the study of aging research by William R. Swindell (2009)[6], most genetic manipulations were found to have a multiplicative effect on survivorship that is independent of age and well-characterized by the AFT model's "deceleration factor". AFT model deceleration factors also provided a more intuitive measure of treatment effect than the hazard ratio, and were robust to departures from modeling assumptions.

1.4 Survival trees and forests

Both semi-parametric Cox and full-parameter AFT models have their formulas to build the model, thus they can fit well with small amounts of data with small variances for the estimated parameters. Furthermore, they can be applied to explain the effects of the effects of the covariates on the survival time and more commonly used in inference.

In data analysis, random forest is another way to build the model, and it usually has a high performance for classification or continuous response variables prediction. It is developed from single decision trees. Under a specifically designed splitting rule, usually dichotomous method, covariates X will be selected for splitting, and the generated tree may have many branches until a certain requirement is satisfied. A new subject could find its position at one node of the tree by its covariates X, and then be predicted.

But a single tree is an unstable learner in the sense that slight modifications in the data set can produce a very different tree with respect to the splitting rule. Hence, a large tree can be considered as a learner with potential for a small bias and a large variance which can be considered as overfitting. Fortunately, random forest can solve this problem.

A random forest, which is an "ensemble" method, generates hundreds or thousands of trees randomly, which could repeat the decision tree model many times to stabilize the result. The combination of the results of these trees may give the best performer in terms of prediction accuracy. There are many ways to combine trees, and one of the most popular is Breiman's (2001)[7] idea. This idea can be described as follows:

1) Independently build B trees with bootstrap samples from the original data.
2) At each node of each tree, randomly select a subset of covariates from all the covariates to find the best split.
3) The final prediction is the average of the predictions of all generated trees from the bootstrap samples.

Trees and random forests have been extended to survival data. If we want it to fit a model with survival data, the continuous event time could be considered as the response variable. A particular split method for survival data is developed as described in the next paragraph because it is not possible to use the least-squares splitting criterion directly to find the best split and grow a tree when some observations are censored.

Generally, the split will create two nodes, left node L and right node R. We want to figure out the best split which could show the biggest difference between the two nodes. With survival data, we are concerned with the survival function. Thus, one strategy is to find splits such that the survival functions in the two nodes, say $S_L(t), S_R(t)$, are very different. One way to quantify this difference is to use a statistic designed to test the hypotheses:

$$H_0: S_L(t) = S_R(t) \text{ for all t}$$

$$H_1:: \ S_L(t) \neq S_R(t) \text{ for at least one t}$$

Ciampi et al. (1986) [8] provided a splitting rule for survival data in 1986 by using the log-rank test.

The log-rank test is a non-parametric way to compare the survival distribution of two samples. The statistic can be represented in the following way:

$$LR = \frac{\sum_{k=1}^{K}(d_{Rk} - E_k)}{\sqrt{\sum_{k=1}^{K} V_k}} \tag{15}$$

where

$$E_k = d_k \frac{Y_R(\tau_k)}{Y(\tau_k)} \tag{16}$$

and

$$V_k = \frac{Y_L(\tau_j)Y_R(\tau_k)d_k(Y(\tau_k) - d_k)}{Y^2(\tau_k)(Y(\tau_k) - 1)} \tag{17}$$

K is the number of all the distinct observed times in the pooled sample with both groups combined. τ_k represent all possible times that were observed. $Y(\tau_k)$ is the total number of subjects at risk at time τ_k. $Y_R(\tau_k)$ is the number of subjects at risk at time τ_k in the right node. d_k is the total number of events at time τ_k. d_{Rk} is the number of events at time τ_k in the right node. Then, we have $Y(\tau_k) = Y_L(\tau_k) + Y_R(\tau_k)$, $d_k = d_{Lk} + d_{Rk}$.

E_k and V_k are the mean and variance of d_{Rj} under the hypotheses H_0. Under H_0, LR has approximately a $N(0,1)$ standard normal distribution.

1.5 Other algorithms

1.5.1 Regularization methods with survival data

In some cases, e.g. when a large number of covariates show multicollinearity, the estimated parameters in the Cox and AFT models may have a big variance and this may affect the prediction performance.

The ridge regression, lasso regression and elastic net are well known as shrinkage methods that work to reduce this problem by adding a penalty parameter before the application of maximum likelihood to optimize the parameters. Thus, for the ridge linear regression, we want the parameters that minimize the sum of squares of the error plus the ridge penalty :

$$\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi}))^2 + \lambda \sum_{j=1}^{p} {\beta_j}^2 \quad (18)$$

For the lasso linear regression, the formula is

$$\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi}))^2 + \lambda \sum_{j=1}^{p} |\beta_j| \quad (19)$$

where $\lambda \sum_{j=1}^{p} |\beta_j|$ is the L_1 penalty.

The elastic net combines ridge and lasso regression by adding both the L_1 and L_2 penalties. It minimizes

$$\frac{1}{n}\sum_{i=1}^{2n}(Y_i - (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi}))^2 + \lambda[\frac{1-\alpha}{2\sum_{j=1}^{p} {\beta_j}^2} + \alpha \sum_{j=1}^{p} |\beta_j|]$$

(20)

As before, λ is the shrinkage parameter, and the new parameter α is a weighting factor between the two penalties. When $\alpha = 0$, we have the ridge penalty, when $\alpha = 1$, we have the lasso penalty, and when $\alpha = 0.5$, we have the elastic net.

These methods are called regularization methods, and this idea could be integrated into the Cox model with survival data and its partial likelihood to optimize the parameters; Simon et al. (2011)[9]. The package glmnet[10] with R is able to fit such models. Benner et al. (2010)[11] analyzes and compares several L_1 penalized Cox regression methods such as "standard" ridge regression, the lasso, the elastic net and some modifications of the lasso like SCAD (smoothly clipped absolute deviation) and adaptive lasso.

1.5.2 Boosting with survival data

Boosting is another effective machine learning method for many kinds of models and data. Boosting can make a "weak learning algorithm" into a "strong learning algorithm". Like random forests, it is an ensemble method that combines the prediction from many models. This is done by building a model from a training data, then creating a second model that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly, or until a maximum number of iterations. Here is a short description of the basic theory of AdaBoost created by Freund et al. (1996)[12]:

Assume that the variable Y is binary and takes the values -1 and 1, and $G(x)$ *is a classifier. For a given vector of covariates* x*, the classifier* $G(x)$ *will return a prediction* $\hat{G}(x) = -1$ *or* 1. *Initialize the weights* $w_i = \frac{1}{n}, for\ i = 1, \dots, n$*, n is the number of the subjects. We decide the number of iterations M. For every iteration* $m = 1\ to\ M$:

1) *Fit the classifier G to the training sample with observations weights* w_i. *Let* $\hat{G}_m$ *be the fitted function.*
2) *Compute the weighted prediction error* $e_m = \frac{\sum_{i=1}^{n} w_i I(y_i \neq \hat{G}_m(x_i))}{\sum_{i=1}^{n} w_i}$
3) *Compute* $\alpha_m = log\ [\frac{1-e_m}{e_m}]$
4) *Update the weights by* $w_i = w_i exp\left(\alpha_m I\left(y_i \neq \hat{G}_m(x_i)\right)\right), i = 1, \dots, n$

5) *The final prediction model is* $\hat{G}(x) = sign[\sum_{m=1}^{M} \alpha_m \hat{G}_m(x)]$

Sourced from course notes by Professor Larocque (2019)[13]

The main idea of boosting is to force the classifier to try to adapt itself to the observations that were misclassified by increasing their weights. But in the end, the final classifier is a weighted average of each individual classifier, with weights that depend on its error. The smaller the error e_m, the larger the weights α_m.

This boosting method, like gradient boosting, can be extended to the Cox model and AFT model with survival data; due to its auto-correction property, it may have better performance compared to another model.

Chapter 2

Stacking methodology

2.1 Ordinary stacking methodology

When prediction accuracy is more essential than inference study, stacking is a widely used ensemble method in prediction. The general idea is to fit several models on the same training data, and then, rather than picking the model with the best performance, these candidate models are aggregated by using another algorithm to make the final prediction.

For example, we have a target Y and p covariates $X = (X_1, X_2, \dots, X_p)$, we have K supervised learning algorithms to develop a predictive model. If we work on survival data, these K models could be Cox, AFT, survival forests, a regularized model like lasso, and boosting. If we fit all the models with the same training data, then according to an evaluation criterion, the best one will be selected with a validation set or by cross-validation method.

Instead of choosing one model, combining them with stacking may be a smart option. Let these K fitted models be $\widehat{m}_1(X), \widehat{m}_2(X), \dots, \widehat{m}_K(X)$. Breiman (1996b)[14] and LeBlanc and Tibshirani (1996)[15] proposed a linear combination as stacking method $\widehat{m}(X)$ with the predictions of candidate models as the inputs, it is defined as

$$\widehat{m}(X) = \sum_{k=1}^{K} \beta_k \widehat{m}_k(X) \tag{21}$$

However, we cannot simply fit a regression with Y as a target and the predictions of candidate models $\widehat{m}_1(X), \widehat{m}_2(X), \dots, \widehat{m}_K(X)$ as predictors to find the coefficients β, because the predictors have been obtained with the training data themselves. In other words, the target Y have already been used to get these predictors, and we should not figure out the coefficients β by reusing them. In this case, new data should be used, with the new target Y and the predictors got with just new covariate X, to get the coefficients β. In this case, it seems that we should split the data into three parts: the training data set, the second data set for finding out the coefficients β, and the third part for validation.

Another way to solve it is to use the leave-one-out (LOO) method for these K models. More precisely, the model is to be fit with all the observations except the i^{th} observation, and then i^{th} observation is predicted with this model. Then, we could fit the regression model to estimate the β's.

$$y_i = \sum_{k=1}^{K} \beta_k \widehat{m}_{k(-i)}(X) \tag{22}$$

It is similar with cross-validation methods. For example, we could divide the training data into 10 folds, and then get the predictions of one of the folds by the model trained with the data of the other folds. The i in the formula (22) could means the fold i, for $i = 1,2, \dots ,10$.

CaretEnsemble[16] and H2O[17] are two of the most-used and popular packages in R for stacking methods or combination of models. For standard data, dozens of models are available to be combined such as XGBoost, random forest, GBM, lasso, SVM, neural networks, etc., with several criteria and methods like mean squared error, AUC, and maximum likelihood to optimize the weights.

The benefit of stacking is obvious, as it allows for the improvement of the performance of the candidate models by taking advantage of the strength of each model compared to a single model. However, due to the variability of the assigned weight for each model and the absence of hyperparameter search for candidate models, the result of stacking may not always be the best. In addition, the predictions of these different models are often correlated. Thus, using regularization methods (e.g., ridge or lasso) to select the appropriate models might be preferable. Moreover, Breiman (1996b)[14] suggested to impose the constraints that all $\beta_k \geq 0$.

2.2 Latest stacking methods for survival data

Due to relaxed assumptions that enable robust estimation, non-parametric estimators (e.g. random survival forest) can often be preferred to parametric (e.g. accelerate failure time) and semi-parametric estimators (e.g. the Cox proportional hazard model). Yet, even when misspecified, parametric and semi-parametric estimators can possess better operating characteristics with small sample sizes due to a smaller variance than non-parametric estimators. Therefore, a bias-variance trade-off is necessary to take advantage of the low bias of non-parametric when the sample size is not large enough. Thus, Andrew Wey et al. (2015)[18] proposed the stacking method to combine parametric, semi-parametric, and non-parametric survival models with stacked survival models via minimizing the inverse probability of censoring weighted (IPCW) Brier Score to obtain constrained weights. The performances of survival models are evaluated by the mean squared error (integrated squared survival error (ISSE)). This stacking method demonstrates that it performs well across a wide range of scenarios, especially with non-linear covariate effects, by adaptively balancing the strengths and weaknesses of individual candidate survival models.

Eric C. Polley and Mark J. van der Laan (2011)[19] introduced a super learner method for right-censored data by cross-validation through converting the data structure into a longitudinal data structure collecting at time t the change in counting processes by a defined time window, then by minimizing the squared error loss function on the hazard to optimize the estimated coefficients of the candidate model for the super learner. The special feature of this algorithm is to add time as the variable and an additional step of smoothing in time with different degrees of freedom for time for the candidate algorithms. This super learner approach elucidates the significance of time to the greatest degree possible.

Another recent study by Andrew Wey et al. (2016)[20] focuses on estimating restricted mean treatment effects with stacked survival models. The difference in restricted mean survival times between two groups is a clinically relevant summary measure in medical research. Since the estimator for the restricted mean difference is defined by the estimator for the covariate-adjusted survival distribution, this method proposes a weighted average of several survival models by minimizing predicted error (the Brier Score) to get a better estimator of the restricted mean difference. The result demonstrates that better performance of the covariate-adjusted survival distribution often leads to better mean-squared error of the restricted mean difference. Additionally, it can perform nearly as well as a Cox regression when the proportional hazard assumption is satisfied, and significantly better when proportional hazards are violated.

Marzieh K.Golmakani et al. (2020)[21] give us a super learner model for survival data prediction by finding the best weighted ensemble of the individual algorithms through minimizing cross-validated risk controls; that is, by minimizing the cross-validated negative log partial likelihood. Two algorithms for optimizing the weights are proposed. One is rewriting the formula by the second order Taylor expansion around the vector of the weights by using the Lagrange formulation. A coarse grid search over tuning parameter space followed by descent algorithm described in Lorber and Ramadge[22] could help us in exhaustive grid search within more reasonable values for reducing the computational challenge when there are more than two candidate models. Another optimization method is to start with two candidate algorithms and solve the one-dimension convex combination parameter, sequentially adding one algorithm until all candidate algorithms are included.

This proposed super learner method could give us the best fit or near-best fit among the candidate models either with simulation studies or with real clinical data examples. However, although these candidate algorithms may range from a basic Cox model to tree-based machine learning algorithms, they must be based on the proportional hazard framework because the optimization is computed on hazard risk for the observations. After that, the survival function for any covariate pattern is calculated with the help of candidate algorithms weights and baseline survival function as in the Cox model.

Because of the candidate model constraints, some algorithms for survival data could not be included for super learner, such as random survival forest, whose estimate cumulative hazard function is calculated using Nelson-Aalen estimator. Thus, when a survival forest algorithm performs best beyond the Cox based algorithms for such a data set, super leaner has its limit.

In the following part of simulation and real data studies, super learner method is compared with stacking with time-dependent and covariate-dependent weights and other single models.

Chapter 3

Evaluation method with survival data

Generally, once a predictive model for an outcome Y is developed, we can estimate the predictions for the validation set. Alternatively, cross-validation may be used to evaluate the performance of this model by calculating the average error with the true outcome values. However, with survival data, the goal may be to estimate the survival function for new subjects, or alternatively, the survival time. But when censoring is present, evaluating the performance of a model becomes difficult, as we do not know the true survival time for some of the observations in the validation set. Furthermore, even without censoring, if the purpose is to estimate the survival function, the true function for each individual is still unknown with only event time as response variable. Fortunately, we have a few useful metrics to solve this difficult situation. For a simulation data set, the survival function for every subject is precisely known, therefore, Integrated Absolute Error (IAE) and Integrated Square Error (ISE) can be used, such as the general average error to evaluate model's performance. For a real data set, the Integrated Brier Score (IBS) (Gerds and Schumacher (2006)[23]; Graf et al. (1999)[24]) is the most popular method. The details of these metrics are described in the following subsections.

3.1 IAE and ISE

This thesis focuses on the estimation of the survival function for the simulation data sets. Its designed function is known exactly; in other words, the survival probabilities at any time t for every test object. Therefore, the general criteria could be used to evaluate the model's performance by generalized average error. Considering that the outputs involve the survival probabilities for a period of time, integrated error is applied. Two commonly used criteria were employed to measure how well the survival function is estimated; see page 676 of Moradian et al. (2017)[25].

Assume that S is the true survival function, and that $\hat{S}$ is the estimated survival function. The two criteria are the Integrated Absolute Error (IAE) and the Integrated Square Error (ISE) and are defined by:

$$IAE = \int_t |S(t) - \hat{S}(t)| dt \tag{23}$$

and

$$ISE = \int_t (S(t) - \hat{S}(t))^2 dt \tag{24}$$

The equations (23) and (24) measure the survival function error between the true and the estimated ones. Here the time points of the survival forests model are used to calculate survival probabilities to get the estimated survival function. The average value of all the test individual IAE or ISE are computed as the criteria value to evaluate the model performance for simulation data set. The codes of computing IAE or ISE are presented in Appendix [1].

3.2 IBS score

IAE and ISE can be applied if the true survival function is well known, i.e. the simulation data set. But with real data, the survival function of any observation is unknown, so we would have a problem even without censoring. Hence, if the goal is to estimate the survival function for the real data set, we cannot use these metrics. There is another method called IBS (Integrated Brier Score) score which could be useful to solve this problem and evaluate the performance of a model with censored data (Gerds and Schumacher (2006)[23]; Graf et al. (1999)[24]).

When the true survival function is known exactly, then its survival probability at any time t is also well known, and its squared error or absolute error could be calculated. But for the real data, we only know its event time or censoring time. If we know its event time τ_i for subject i, let $\hat{S}(t|X)$ denote its estimated survival function, estimated by any model, at time t for a subject with covariate vector X. $\hat{S}(t|X)$ is an estimation of the survival probability $P(T > t|X)$. We denote an indicator $I(\tau_i \ > t)$ that takes a value of 1 if $\tau_i > t$, and 0 otherwise. Thus, a binary target, $I(\tau_i > t)$ can be considered as the empirical survival function for subject i.

In the case where there is no censoring, the Brier Score at any time t is defined as

$$BS(t) = \frac{1}{n}\sum_{i=1}^{n}(I(\tau_i > t) - \hat{S}(t|X_i))^2 \tag{25}$$

where n is the number of subjects.

The Brier Score is a function of t, and it represents the mean of squared error of the survival probability for all the subjects. However, a single time t cannot explain the performance of the model, we want to evaluate the model for all range of time. One way to solve this and get a single performance measure is to compute the integral of $BS(t)$, with respect to t. The Integrated Brier Score is given by

$$IBS = \frac{1}{\max{(\tau_i)}}\int_0^{\max{(\tau_i)}} BS(t)dt \tag{26}$$

where τ_i is the event time for subject i.

The values of IBS have the same meaning as mean squared error with simulated data. Lower IBS indicates better performance. Basically, the IBS is an integrated weighted squared distance between the estimated survival function and the empirical survival curve. The shaded areas of Figure 1 below, sourced from course notes by Professor Larocque (2019)[13], is the IBS score for a single observation. It compares the estimated survival curve to the empirical survival curve for that observation. Since the subject experienced the event at τ_i, the empirical survival curve takes a value of 1 between 0 and τ_i, and a value of 0 after τ_i.

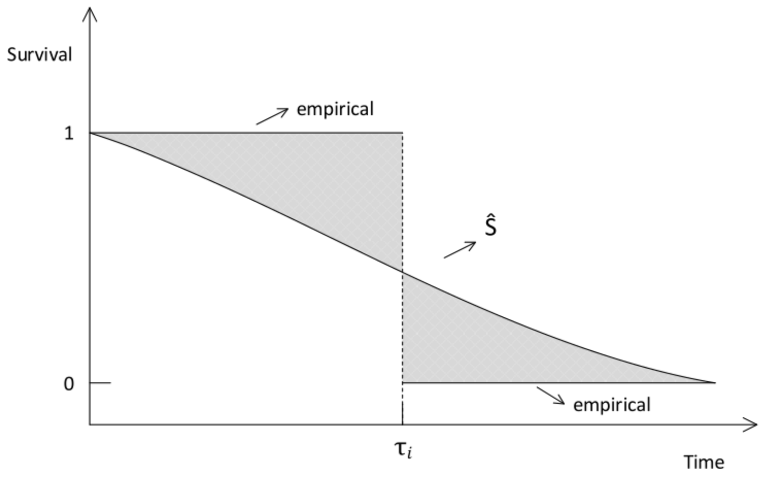


Figure 1 The integrated Brier score with no censoring

If we do not know the event time, that means that censoring is present, a weighting scheme to adjust for it can be applied. It is the inverse probability of censoring weights (IPCW). Let $\hat{G}$ denote an estimate of the survival function of the censoring distribution. It can be the Kaplan-Meier estimate of the censoring distribution, or a more sophisticated approach that uses the covariates. The Brier Score at any time t is computed as

$$BS(t) = \frac{1}{n}\sum_{i=1}^{n}((\hat{S}(t+X_i)^2 I(\tau_i \leq t \text{ and } \delta_i = 1)\hat{G}^{-1}(\tau_i) + (1-\hat{S}(t|X_i))^2\, I(\tau_i > t)\hat{G}^{-1}(t)) \quad (27)$$

To put the weights in evidence, this formula can be written in another way:

$$BS(t) = \frac{1}{n}\sum_{i=1}^{n}\left(I(\tau_i > t) - \hat{S}(t|X_i)\right)^2 w(t) \quad (28)$$

Where $\delta_i = 1$ represent that the event has occurred for observation i , $\delta_i = 0$ represent this observation is censored. And $w(t) = I(\tau_i \leq t\ and\ \delta_i = 1)\hat{G}^{-1}(\tau_i) + I(\tau_i > t)\hat{G}^{-1}(t)$ is the IPCW.

Gerds and Schumacher (2006)[23] show that if the censoring model used for $\hat{G}$ is well specified, then this formula converges to the right value, that is the expected Brier Score, given by

$$E[\left(I(T > t|X) - \hat{S}(t|X)\right)^2] \tag{29}$$

In a way, the IBS is a crude estimate of the performance, because the unknown true survival curve is replaced by the empirical one. But it is hard to do better in this setting.

The package pec in R by Mogensen et al. (2012)[26] can be useful for computing the Brier Score, with the matrix of predicted survival probabilities under a fitted model having as many rows as data and as many columns as times and the response variable Y (event time or censoring time) as inputs.

Chapter 4

Methodology

Before explaining the proposed method in detail, here is first a general description of the key features of the general methodology at a high level. The goal is to have a final stacking model where the model weights are time-dependent and covariate-dependent.

(1) Select a grid of time points. For example, using quantiles form the Kaplan-Meier estimation of the survival function of the training data.
(2) For each time point in the grid, apply a stacking method to estimate the survival function at that point, but such that the weights can depend on the covariates. For example, using a MOB-tree.
(3) Smooth the estimated values at the time points on the grid to get the global survival function in such a way that it is monotonically decreasing. For example, using isotonic regression.

Here is now a detailed description of the method.

4.1 Step 1: Select the $\boldsymbol{K} = \mathbf{3}$ models for stacking method

Three basic, commonly used models have been selected because of their unique advantage as parametric, semi-parametric and non-parametric model representatives. They are the accelerate failure time (AFT) model, the Cox proportional hazard model and random survival forest.

Stacking with these models covers several kinds of classic models for survival data, and it could compare the performance from the aspects of risk hazard, accelerated time and random forests to test if the stacking could have an advantage beyond these basic models.

4.2 Step 2: Select a grid of values for the time range

The objective is to develop a stacking model with time-dependent and covariate-dependent weights. Time-dependent weights are based on the assumption that the weights of candidate models change with time. For example, it is possible that the AFT model performs best in the time interval (t_1, t_2) and the Cox model performs best in the time interval (t_2, t_3) with $(0 < t_1 < t_2 < t_3)$. Thus, time-dependent weight means that each period of time corresponds to a different weight for a given subject.

The non-parametric Kaplan-Meier (KM) method could give us an intuitive, overall description of the survival function for the simulated data set that we will use. Based on the overall curve, we want to estimate the time-dependent weights, and theoretically, the weights may change smoothly and have continuous values from time 0 to the maximum survival time. However, it is not practical to estimate the continuous weights at any time in condition of the limited data number and the limited computation speed. It is more practical to use a grid of time points.

However, it is not appropriate to use time points that are equispaced. For example, we can see from Figure 2 (see section 5.1) that the range of survival time of DGP4 is from 0 to 10, but from time 4 to 10, the survival probability is asymptotically close to 0, and in very few subjects does the event occur in this time range. This is why we use a time grid defined by using equispaced quantiles obtained from the Kaplan-Meier estimate of the survival time.

Define $t_0 = 0 < t_1 < \cdots < t_G$ as this time grid, where G is the number of time intervals. For example, if $G = 9$, then t_1 is the 0.1 quantile of the Kaplan-Meier estimate maximum time (also the maximum observed event time), t_2 is the 0.2 quantile, and so on. Each time in the grid will be used to build a model to get time-dependent weights.

4.3 Step 3: Fit models with training data in two different ways

4.3.1 With all the training data

Firstly, we want to estimate the survival function for each candidate model (Cox, AFT and survival forests). Their predictions will be used to compare the performances with each other and with the new stacking method. Thus, a model for each method with all the training data is fitted. The three models will be used to get estimations for the new test data.

After this step, we have $\hat{S}_k(t|X_{test})$ for the new subjects, the estimated survival function for model k ($k = 1, \dots, K = 3$) trained with the complete sample.

4.3.2 With a cross-validation scheme (e.g. 10 fold)

From the general theory of stacking method, to optimize the weights, the training data predictions of one candidate model $\hat{S}_k(t|X_{train})$ fitted with all the training data cannot be used as predictors because they have been obtained with the data themselves.

Therefore, secondly, each model is fitted with a cross-validation scheme (e.g. 10 folds). We divided the train data into 10 folds, and each fold is predicted with the model fitted by the data of 9 other folds. These models are used to get out-of-sample estimations of the survival function that will be used to optimize the stacking weights.

After this step, we have $\hat{S}_k^{cv}(t|X_i)$, the out-of-sample estimated survival function for the i^{th} observation X_i for model k trained in the CV loop, for $k = 1, \dots, K = 3$ and $i = 1, \dots, n$ where n is the number of training data. These three models will be used for optimizing the time-dependent and covariates-dependent weights.

4.4 Step 4: Optimize the weights with pseudo-observations values

From Step 3, we have $\hat{S}_k^{cv}(t|X_i)$ as the out-of-sample estimated survival function for all the training data, which could be used as the predictors to get the weights. However, due to the censoring, we do not directly have the response variable Y to fit a model to get the weights with survival data. Pseudo-observation gives us a solution to solve this problem.

4.4.1 Pseudo-observation

Per Kargn Andersen et al. (2010)[27] introduces the theory and applications of pseudo-observations in survival analysis including regression models for parameters like the survival functions in a single points, the restricted mean survival time or state occupations probabilities in multi-state models (e.g. the competing risks cumulative incidence function).

Given the information that survival data have special characteristics, right-censoring, or left-truncation, which make it difficult to develop algorithms. If we have complete data, the survival time Y would be observed for all individuals and standard methods for quantitative data could be applied directly with Y as response variable, or methods for binary outcomes could be applied by dichotomizing Y as $I(Y \leq t)$. More generally, methods for repeated binary data could be used for a series of indicators, $I\left(Y \leq t_j\right), j = 1, \dots, m$ (m is the repeated times) as a response variable. Then, we could set up a model to get the weights with the predictors $\hat{S}_k^{cv}(t|X_i)$ obtained in step 3. Furthermore, without censoring, it is possible to get average error for quantitative or binary outcomes.

However, with censored survival data, we fortunately have one way of transforming the event time or censoring time to continuous response variable by the pseudo-observations. In other words, the pseudo-observations could replace the incomplete event time Y.

The basic idea is simple. Let $f(Y_i)$ to be the function of event time as response variable. If the data were complete, $f(Y_i)$ would be observed for each individual i, and the expected value $E(f(Y))$ could be estimated by $1/n \sum_i f(Y_i)$. Conversely, suppose that the data are incomplete, i.e. some observations are censored and therefore not all $f(Y_i)$ are observed, but a well-behaved estimator, $\hat{\theta}$, for the expectation $\theta = E(f(Y))$ is available anyway, e.g. the Kaplan-Meier estimator for $S(t) = E\big(I(Y > t)\big)$. The pseudo-observation for $f(Y)$ for individual $i, i = 1, \dots, n$, is then defined as

$$\hat{\theta}_i = n * \hat{\theta} - (n-1)\hat{\theta}^{-i} \tag{30}$$

where $\hat{\theta}^{-i}$ is the estimator applied to the sample of size $n-1$ obtained by eliminating the i^{th} individual from the data set. Intuitively the i-th pseudo-observation can be viewed as the contribution of the individual i to the $E(f(Y))$ estimate on the sample of size n. The idea now is to replace the incompletely observed $f(Y_i)$ by $\hat{\theta}_i$. For example

(1) $\hat{\theta}_i$ may be used as an outcome variable in a generalized linear regression model with some link function g: $g\big(E(f(Y)|X)\big) = \beta_0 + \sum \beta_j X_j$ or

(2) $\hat{\theta}_i$ may be used to compute residuals or in a scatterplot when assessing model assumptions

The pseudo-observations $\hat{\theta}_i$ will always be used for all n subjects and not only for those where $f(Y_i)$ was unobserved. Note that the models themselves are fitted with the original data. The pseudo-observations are only used as proxy to optimize the weights.

4.4.2 Optimize the weights

In the general method, the stacking weights depend on the covariates and the time. Fix a value in the time grid t_g. We now want to estimate the weight function $\hat{\alpha}_k(t_g, X)$ (of time t_g and covariates X), for $k = 1, \dots, K = 3$ three selected models in section 4.1, to obtain the estimates

$$\hat{S}(t_g|X) = \sum_{i=1}^{K} \hat{\alpha}_k(t_g, X)\hat{S}_k(t_g|X) \tag{31}$$

With the application of pseudo-observation as an outcome variable for each individual—either the observed event time or censored data—candidate model weights will be calculated for each time grid through fitted linear model trees with covariates X as the splitting variables. Through this method, time-dependent weight and covariates-dependent weights can both be obtained simultaneously. The details of the different steps are as follows through the language R.

(1) Using the function *pseudosurv* in the R package pseudo[28], the pseudo-values at t_g for the original sample can be obtained, call them $P_1, P_2, \dots, P_n$. We can then fit a mob-type tree with the function *lmtree* in the R package partykit[29]. A mob, for "model based" tree, is a tree in which a model can be fitted in the nodes. The covariates are divided in two groups, but the groups can have a non-null intersection. The first group contains the variables used to fit the within node model. The second group contains the variables that are used for splitting the tree. In our case, the dependent variables are the pseudo-values (the P_i), $\hat{S}_k^{cv}(t_g|X_i)$ for $k = 1,2,3$ models are the predictors in the within-node linear model, and the covariates X are the splitting variables.

(2) We do this for all time intervals G times separately and end up with estimated weights functions $\hat{\alpha}_k(t_g, X)$, for $k = 1, \dots, K$, and $g = 1, \dots, G$. Notice that for one given time point in the grid, we could get more than one node due to the conditional splitting variables X. In other words, we get covariates X dependent weights for each time point.

However, using *lmtree* to optimize the weights can produce weights that can be negative or greater than 1. It is more reasonable to have weights that satisfy:

(1) The individual weights for model k are not negative
(2) The sum of weights of each model is 1
(3) Remove the intercept in the regression model

In fact, this method refers to a regression with constraints so that the weight must be between 0 and 1. With the response variable pseudo-observation and $\hat{S}_k^{cv}(t_g|X_i)$ for $k = 1,2,3$ as the predictors for each node of each time point, the function solve of the package quadprog[30] in R can help us compute the constrained weights. The R code of the function for computing the weights is detailed in Appendix [2]. Basically, we build the mob-tree as usual, but instead of fitting an unconstrained linear model as default in the terminal nodes, a constrained model is fitted.

4.5 Compute the final estimation for test data

4.5.1 Compute new data estimate at time grid $\boldsymbol{t_g}$

After the last step, we have the weights $\hat{\alpha}_k$ of each model k for each time point in the grid t_g, $g = 1, \dots, G$. Then, the final estimation for a new point X_{new} can be computed. For each time value in the grid, we can get

$$\hat{S}(t_g|X_{new}) = \sum_{i=1}^{K} \hat{\alpha}_k(t_g, X_{new})\hat{S}_k(t_g|X_{new}) \tag{32}$$

We thus have $\hat{S}(t_1|X_{new}), \dots, \hat{S}(t_G|X_{new})$.

4.5.2 Compute new data estimate for all time $\boldsymbol{t}$

We want to estimate the survival function for the new individual data, so it is important and necessary to estimate the survival probability values for all time t, not only at the chosen time grid points t_g. There is a simple way to extend the weights to the other times. If we want to estimate $S(t_j)$ at time t_j which is not in the time grid, we can use the weights for the grid point that is the nearest to t_j. For example, suppose that the time grid is 1.2, 2.6, 4.5, 6.7, 10.1. We want to estimate $S(t_j = 4.1)$. The nearest time grid value for $t = 4.1$ is 4.5, thus we use the estimated weights for $t_g = 4.5$.

In order to better compare the performances among the selected models, the time interests of the survival forests model are chosen as the time points to describe the survival probabilities for all time t. The R code of the function for predicting the test data is detailed in Appendix [3].

4.5.3 Adjustment of estimated survival probability value

Because these $\hat{S}(t_1|X_{new}), \dots, \hat{S}(t_{max}|X_{new})$ come from the combination of candidate models, it is likely to have a case like $\hat{S}(t_1|X_{new}) < \hat{S}(t_2|X_{new})$ with $t_1 < t_2$ that goes against the monotonically decreasing nature of the survival function.

One way to solve this problem is to apply a monotone regression method like isotonic regression to these values for every new individual to obtain the complete estimated survival function $\hat{S}(t|X_{new})$ for all t. One simple isotonic regression method is to make sure the survival probability at the next time point is no bigger than the previous time value; if it is not the case, the probability at next time point will be replaced by the value of previous time point. The R code of the function for isotonic regression is detailed in Appendix [4].

4.6 Clustering stacking methodology

Sections 4.1 to 4.5 present the steps of the general stacking method with time-dependent weights and covariates-dependent weights, which assumes the weights may depend on the time and covariates simultaneously. For a given time point in the grid, a MOB-tree is used to get covariate-dependent weights. We will also investigate another method as described here.

For a given time point in the grid, the terminal nodes of the tree provide a grouping of the observations. Instead of using a tree, a cluster analysis algorithm could be used to perform the grouping. More precisely, we could apply the clustering K-means algorithm to do the grouping work. It is possible that it could do better than the linear model tree.

The number of terminal nodes in the tree is used as the group number for the K-means algorithm for each time point. The package clue[31] in R can predict the K-means groups for new subject data according to the covariates X. Within the group, we can get the constrained weights (positive and sum equal to 1) with the same method proposed in section 4.4.2 with the package quadprog[30]. Then, the prediction for the test data with these weights can be obtained.

With all other steps being the same, this clustering stacking method is very similar to the general stacking method in sections 4.1 to 4.5. It is only slightly different based on the grouping with covariates X. Therefore, the prediction results of these two methods may not differ much.

Chapter 5

Simulation study

5.1 Simulation design

In the main simulation study, four Data Generating Processes (DGPs) are used to generate artificial data. DGP1 is from a master course note by Professor Larocque (2019)[13] (Page 188) by Professor Denis Larocque. The other three—DGP2, DGP3 and DGP4—are from "L1 splitting rules in survival forests" by Hoora Moradian et al. (2017)[25].

Each model is fitted with a training sample of size 100. Then, the performance of the fitted models is evaluated with an independent test set of size 500. Each simulation is repeated 50 times. In section 5.3, additional simulation results using training samples of size 500 will be presented.

The parameter $parcens$ controls the proportion of censoring of DGP 1, while the parameter α controls the proportion of censoring of the other three DGP. With adjustment of parameter parcens and α, DGP1 has censoring proportion 0.36. The other three DGPs have censoring proportion 0.25, 0.3, 0.35, respectively. Here are the detailed descriptions of the DGPs.

5.1.1 DGP 1

There are 15 covariates. Let $X = (X_1, X_2, \dots, X_{15})$ have a multivariate normal distribution with mean 0, variance 1 and correlation 0.3 for each pair of variables. Let $V = (V_1, V_2, \dots, V_{15})$ be a transformation of X. Each element of V is a function of the corresponding element of X. V_6, V_{12} are the absolute values of X_6, X_{12} respectively. V_7, V_8, V_{11} are binary variables that take a value of 1 if the corresponding X is larger than 0.5, 0.2, and 0.1 respectively. V_9 is $\log(X_9 + 5)$, and let V_{10} is $\exp\left(\frac{X_{10}}{3}\right)$. The other variables remain untransformed. The true model is $\log(T) = f(V) + \log(\epsilon)$ where f is a function of the covariates and ϵ is an error term from the gamma distribution with shape parameter 3 and rate parameter 5. The function f is as follows:

$$f(V) = -0.5 + \frac{V_1 + V_2 + 0.3 * {V_2}^2 + V_5 + V_6 - 0.3 * V_5 * V_6 + V_7 + \frac{1}{V_{10} + 3} + (V_3 > 0) * (V_1 > 0) - (V_3 < 0) * (V_1 > 0)}{3}$$

In addition, $f(V)$ is bounded to be within the range of minimum -1.8 and maximum 1.7. The censoring times are exponentially distributed with rate of $(\frac{1}{parcens=1.5})$ to make DGP 1 have censoring proportion 0.36.

5.1.2 DGP 2

This is an altered version of scenario 2 from Sec. 4.1 of Zhu and Kosorok (2012)[32]. Ten IID (independent and identically distributed) uniform covariates on the interval (0,1) are available, $X_1, \ldots, X_{10}$. Survival times are drawn from an exponential distribution with mean μ where $\mu = 10|\sin(X_1\pi - 1)\ | + 3|X_2 - 0.5| + X_3$. The censoring times are uniformly distributed on the interval $(0, \alpha = 29.1)$ to make DGP 2 have censoring proportion 0.25.

5.1.3 DGP 3

This is adapted from scenario 3 in Sect. 4.1 of Zhu and Kosorok (2012)[32]. Twenty-five covariates $X_1, \ldots, X_{25}$ are generated from a multivariate normal distribution with covariance matrix $\sigma_{ij} = 0.75^{|i-j|}$. The survival time follows a gamma distribution with shape parameter $\mu = 0.5 + 0.3|\sum_{i=11}^{15} X_i|$ and scale parameter of 2. The censoring times are uniformly distributed on the interval $(0, \alpha = 9.5)$ to make DGP 3 have censoring proportion 0.3.

5.1.4 DGP 4

This is a dependent censoring DGP. It is adapted from scenario 1 in Sect. 4.1 of Zhu and Kosorok (2012)[32]. Twenty-five covariates $X_1, \ldots, X_{25}$ are generated from a multivariate normal distribution with covariance matrix $\sigma_{ij} = 0.9^{|i-j|}$. The survival time follows an exponential distribution with mean of $\mu = 0.1|\sum_{i=11}^{20} X_i|$. The censoring times are drawn from an exponential distribution with mean $\mu/(\alpha = 0.54)$ to make DGP 4 have censoring proportion 0.35.

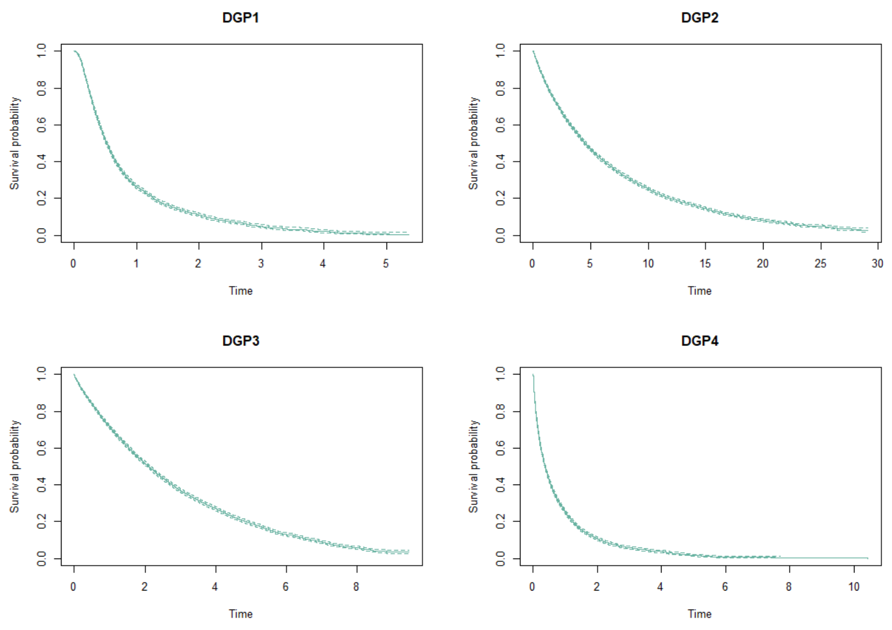


Figure 2 Kaplan-Meier estimate of the survival curve for 10000 observations of four simulations data sets with censoring proportion 0.36, 0.25, 0.3,0.35 respectively

Figure 2 above shows a Kaplan-Meier estimate of the survival curve for 10000 observations of four simulation data sets, respectively. We could observe that the maximum survival time for the four data sets is different, approximatively 5, 30, 10, 10, respectively. And even though DGP3 and DGP4 have similar maximum survival time—but their survival probabilities change at very different rates—DGP3 falls gently with time, while DGP4 falls rapidly, and by time 4, the survival probability is almost down to 0. Overall, the survival curve of DGP1 and DGP4 decreases sharply, while the survival curve of DGP2 and DGP3 decreases gently. The survival time range for each DGP is shown in the figure, and then we can define the maximum time of time grid for each DGP. Section 4.2 explains how the maximum time for making the time grids should be selected; considering that the training data has only 100 subjects, there are very few subjects whose event time exceeds time 1.5 if we take DGP1 as an example, and the same value of pseudo-observation is calculated when time is over 1.5. Thus, we define 1.5 as the maximum reasonable time for DGP1 to fit model for weighting, and with the same analogy, 15, 5, and 1.5 as max time grid for DGP2, DGP3 and DGP4, respectively. The code used to generate the simulation data DGPs is detailed in Appendix [5].

5.2 Simulation results

5.2.1 Weights presentation

After the five methodological steps from section 4.1 to section 4.5, the estimated predictions of three base survival models (Cox, AFT with loglogistic distribution and survival forests), stacking method, clustering stacking method and super learning method for the 50 test data sets. The time points at which the survival probabilities are predicted in the random forest model are used for the other two single models and stacking methods for the purposes of the same standard of evaluation, more than hundreds of time points probabilities are sufficient to reflect the estimated survival function.

In the stacking part, a linear model tree is fitted with constrained linear coefficients to optimize the weights. At each time interval, the constrained weights of the three models are received, where the weights are between 0 and 1 and their sum is exactly 1.

In order to better explain how the weights of the three candidate models vary as a function of time t, the quantiles 0.2, 0.4, 0.6 and 0.8 of the time grids are selected to view the changes. Each group of boxplots corresponds to the distribution for all the repetition of simulation of one DGP. The weights vary between 0 and 1 due to the constraints. As a result, the sum of average weights of the candidate models for a given block of time, such as quantile 0 to 0.2 of the max time grid, should be 1. the average weights are not shown in the boxplots below, but the median weights can reflect average model weight for different time intervals.

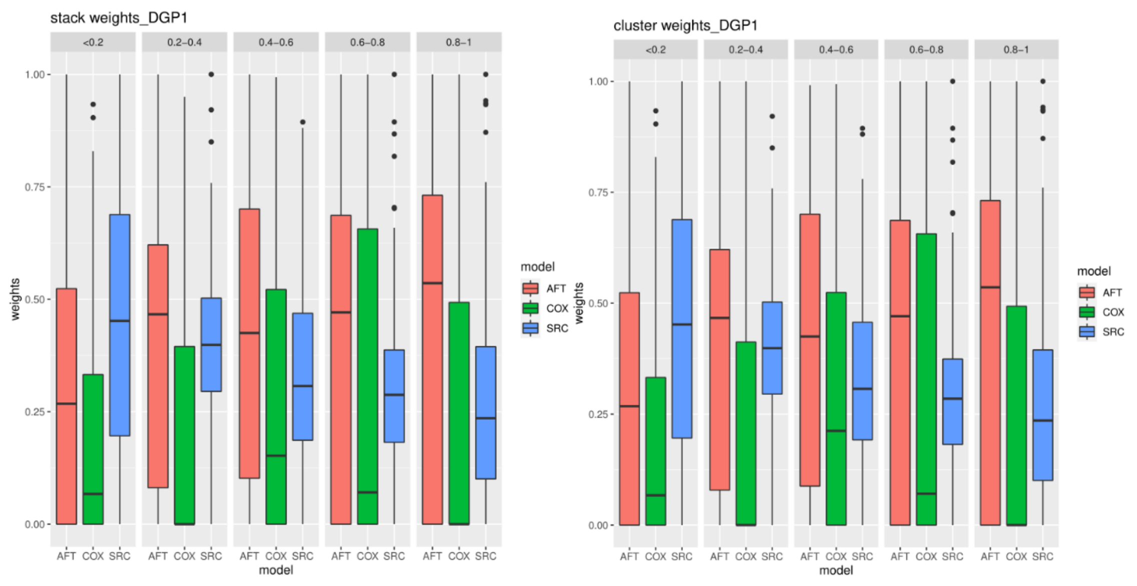


Figure 3 Time quantiles weights distribution of stacking method for DGP1 (ntrain=100)

(Left: general stacking, Right: clustering stacking)

Figure 3 (Left) shows that for DGP 1, the median weights of the algorithm survival forest (SRC) are close to 0.5 as time ranges from 0 to 20% maximum time grid 1.5, i.e. from time 0 to time $0.2 * 1.5 = 0.3$, compared to 0.25 for model AFT and nearly 0 for Cox. The weight of each model shifts as the survival time increases, with the survival forest losing weight ans the other two models gaining. This means that the weights of the models are time dependent. However, the way that the weights change with time depends on the data set. For example, for DGP2, DGP3 and DGP4, throughout all their survival time ranges, the stacking method gives more weights (almost 1) to suvival forest (SRC), and much less weights to the other two models. Because of the structure of those DGPs, survival forest outperforms the three basic models for simulation data

DGP2, DGP3 and DGP4 across all time ranges. Figure 4, Figure 5 and Figure 6 show the time quantiles weights distributions for DGP2, DGP3 and DGP4.

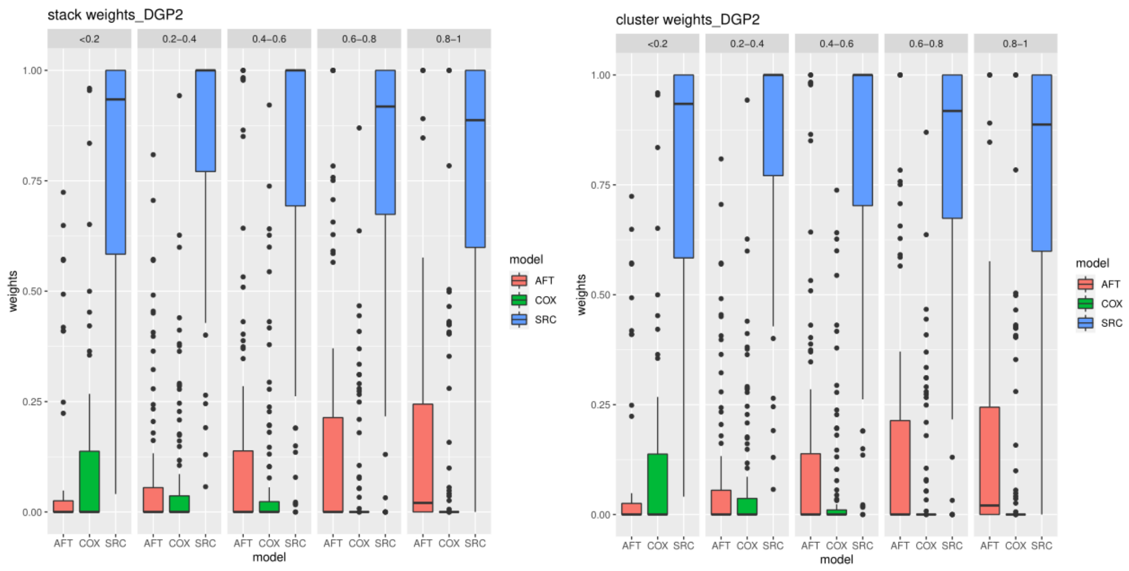


Figure 4 Time quantiles weights distribution of stacking method for DGP2 (ntrain=100)

(Left: general stacking, Right: clustering stacking)

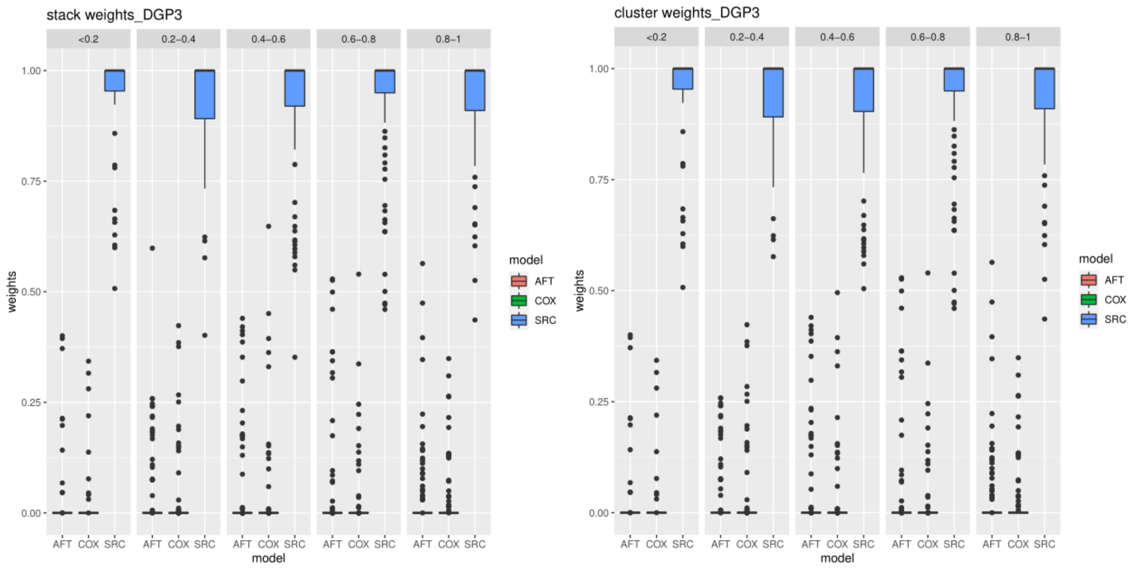


Figure 5 Time quantiles weights distribution of stacking method for DGP3 (ntrain=100)

(Left: general stacking, Right: clustering stacking)

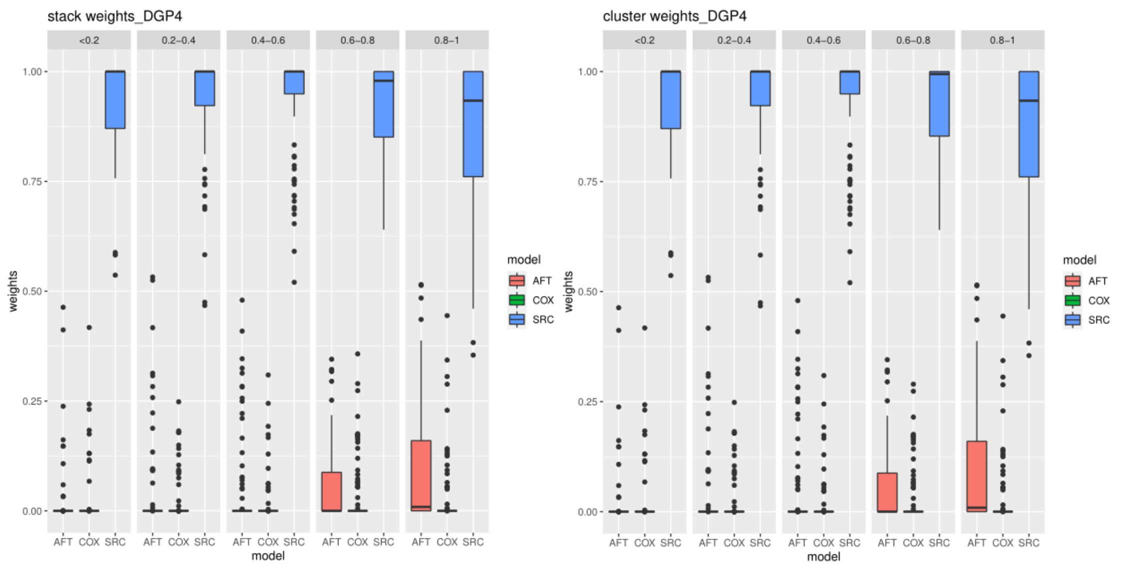


Figure 6 Time quantiles weights distribution of stacking method for DGP4 (ntrain=100)

(Left: general stacking, Right: clustering stacking)

The left figures above using general stacking method show how the model weights change as time goes by, and in fact, the linear model tree algorithm regroups the subjects. For each time grid, nodes (groups) are produced in function of the characteristics of covariates X. The number of nodes varies depending on the time grid and the data sets used. Clustering is another way to optimize the group weights. Section 4.6 describes the technique of clustering stacking methodology, and the K-means clustering algorithm could give us another option to regroup them with the number of groups for each time grid the same as the number of nodes of the general stacking method. The results of this clustering stacking algorithm's time quantiles weights distribution are shown in the right figures above.

When we compare these rights figures to the left figures of the general stacking method, It is obvious that they are extremly similar, especially for DGP2, DGP3 and DGP4, where the figures of the two algorithms are exactly the same. That is because for each time grid, there is only one root node generated by the linear model tree method. In other words, all the subjects could be treated as a single group, and the weights for this time interval have not noted the difference in aspect of the covariates X. As a consequence, the result of clustering is identical to the result of general stacking.

5.2.2 Performance comparison

Because the accurate survival function for each observation for the simulated data sets is known, we could calculate the survival probabilities at any time point for each subject. IAE and ISE can be easily computed as prediction performance criteria for test data sets. The smaller the integrated error, the better it performs. The following two figures illustrate IAE and ISE results of three basic models (Cox, AFT, SRC) for survival data, as well as the new stacking method with time-dependent weights and covariates-dependent weights and its clustering method. To compare, the super learner proposed by Marzieh K.Golmakani et al. (2020)[21] is also included, and only two algorithms (Cox and GBM boosting) are used for risk coefficient optimization to save computational time.

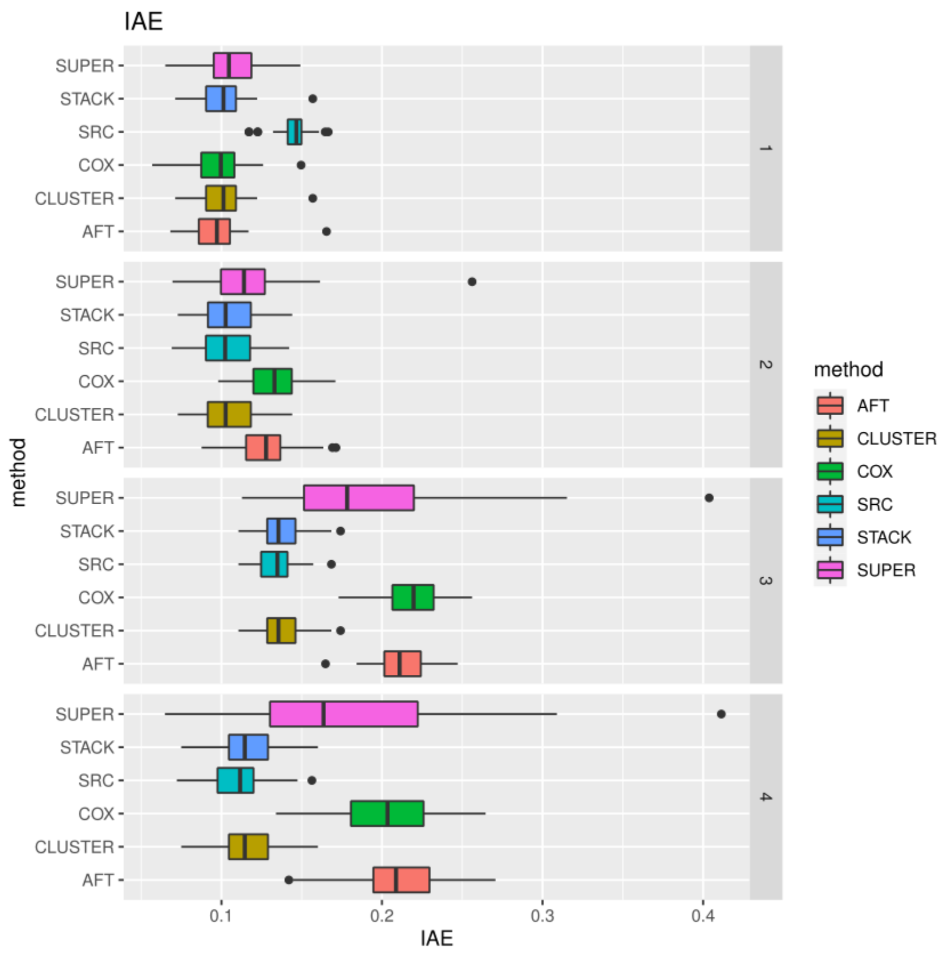


Figure 7 Integrated Absolute Error (IAE) boxplots of four simulation data sets (ntrain=100)

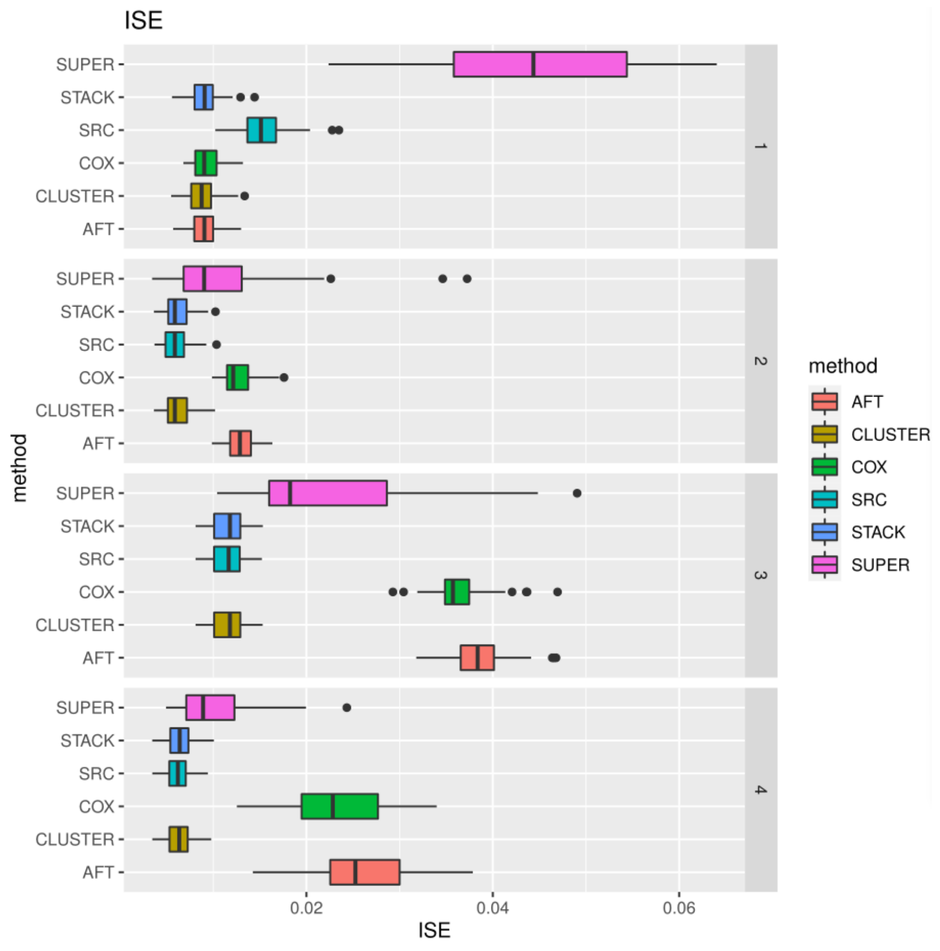


Figure 8 Integrated Square Error (ISE) boxplots of four simulation data sets (ntrain=100)

As performance criteria, IAE and ISE are only applied for simulation data because its true survival function for each observation is defined. The two results are strikingly similar. It is frequent that the lower the error, the better the method's efficiency. Likewise, the lower the variance, the more stable are the performances of the method. From the figures of IAE and ISE, stacking and cluster method with time-dependent weights and covariates-dependent weights performs best for DGP1 and DGP2, and performs nearly best as survival forest for DGP3 and DGP4. While the super learner method can outperform the basic Cox model, it cannot outperform the survival forest because its risk optimization algorithm does not include survival forest model, which has unique advantages for DGP3 and DGP4. Similar to our weights distribution analysis, the clustering stacking method produces very similar results as compared to general stacking method.

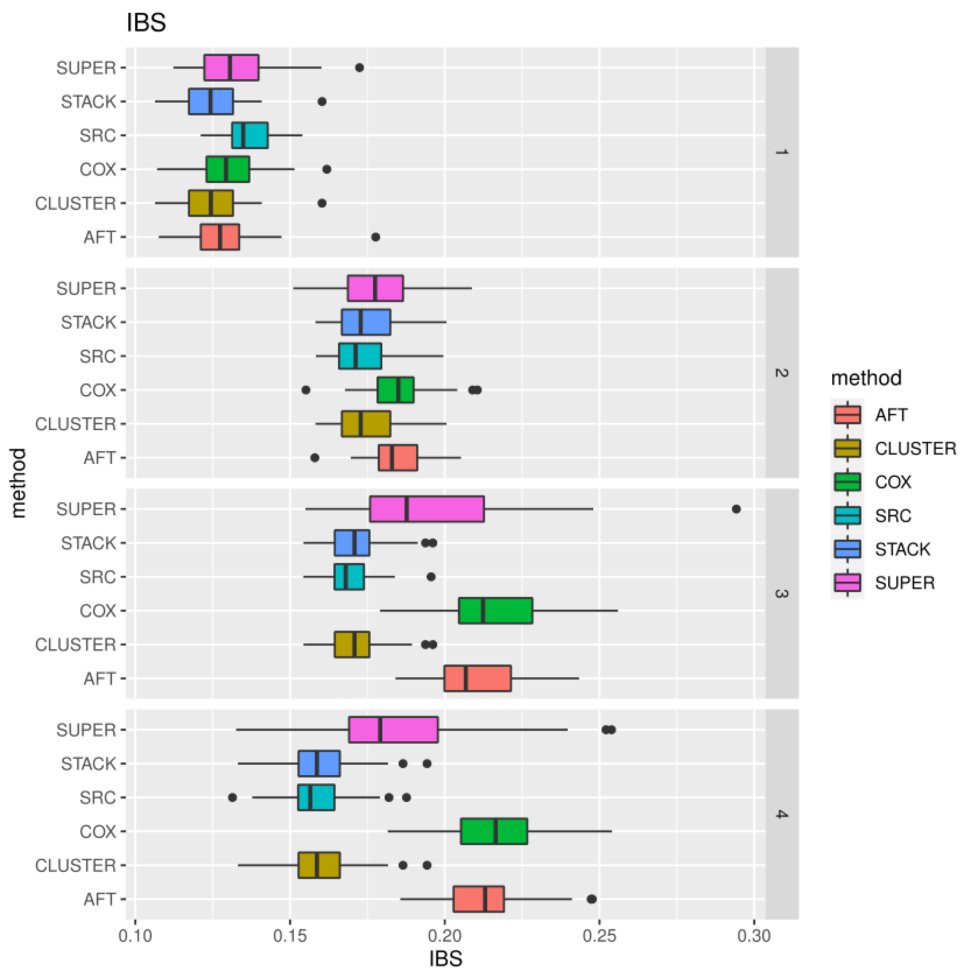


Figure 9 Integrated Brier Score (IBS) boxplots of four simulation data sets (ntrain=100)

In real data, the survival function for any observation is unknown, we just know its event time or censoring time. Therefore, IAE and ISE criteria could not be applied for real survival data. Despite the fact that the accurate error for the four simulations can be calculated, we also want to compare their IBS. Figure 9 give us the similar results with IAE and ISE. For DGP1, the stacking method and clustering stacking method have the smallest IBS, and for DGP2, DGP3 and DGP4, they are very close to the best performance of the survival forest model. This means that we will be able to trust the IBS-based comparisons with the real data sets.

In conclusion, for the simulation data sets, the two stacking methods could detect automatically which basic classic model performs best and incorporate the ideal part of each candidate model to produce better predictions.

5.3 Supplementary simulation analysis

The above simulation results are based on a train size 100 and a test size of 500 with 50 repeats. We are also interested in seeing how this stacking method performs as there are more training data. The same stacking algorithms mentioned in Section 4 are used in a supplementary experiment.

The supplementary simulation entails increasing the training size from 100 to 500, keeping the test size constant at 500, using the same maximum time grid (1.5, 15, 5, 1.5 for DGP1 to DGP4) and repeating the simulation 50 times, as described in Table 1.

Table 1 Description of the simulations

	Initial simulation	Supplementary simulation
Train size	100	500
Test size	500	500
Maximum time grid for DGP1 to DGP4	1.5, 15, 5, 1.5	1.5, 15, 5, 1.5

Theoretically, the more train data we use to fit a model, the more reliable and precise the prediction becomes, and the smaller the error becomes. To begin, we look at the weight's charts for the comparison of 100-training size initial simulation and 500-training size supplementary simulation.

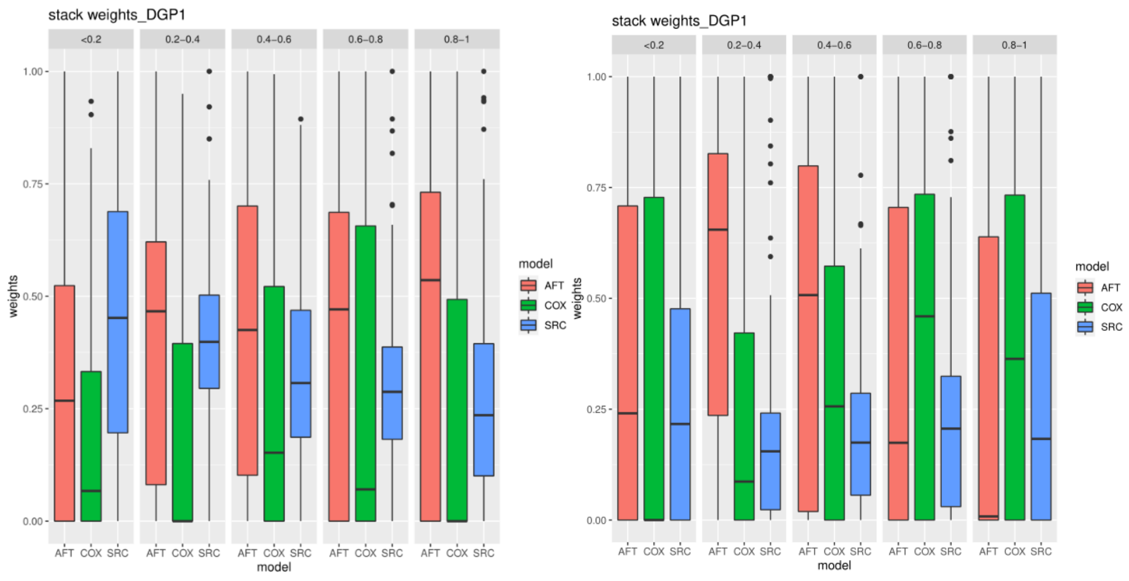
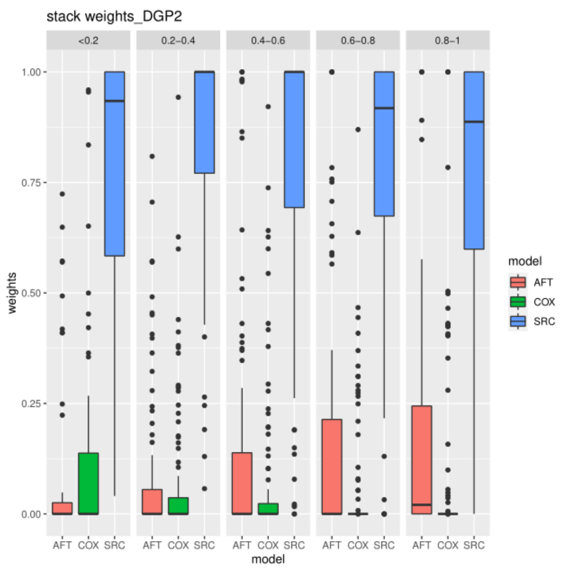


Figure 10 Time quantiles weights distribution of general stacking method for DGP1

(Left: ntrain=100, Right: ntrain=500)

When comparing the two weights charts (train size 100 and train size 500) in Figure 10 of DGP1, there is no discernible difference. In other words, even though there are more data for training, the weights of each model are so divergent (from 0 to 1) that we cannot be certain which one is better at a given time point. However, since the src boxplot is more clearly clustered at the bottom of the right chart (train size 500) with a small weight than left chart (train size 100) with more train data, the simple conclusion can still be drawn that the SRC model does not work well for this set of simulated data.



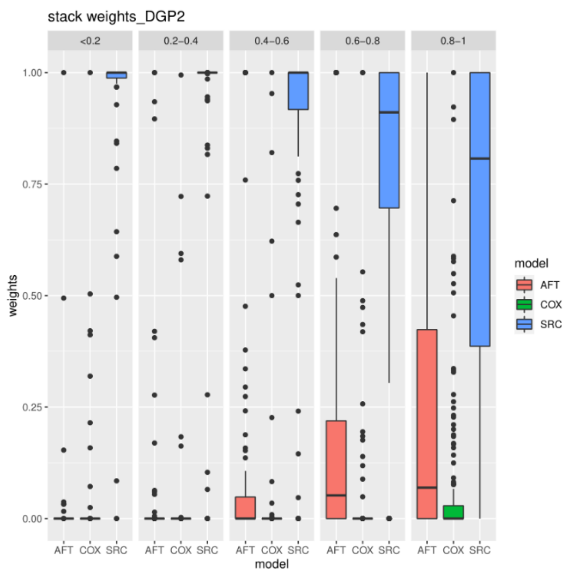
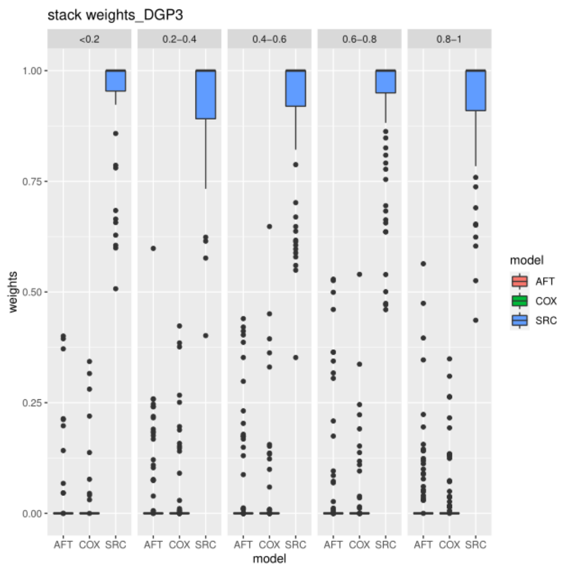


Figure 11 Time quantiles weights distribution of general stacking method for DGP2

(Left: ntrain=100, Right: ntrain=500)



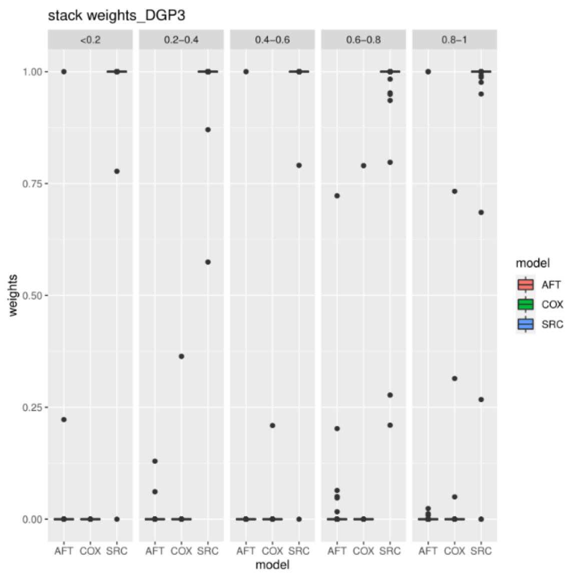


Figure 12 Time quantiles weights distribution of general stacking method for DGP3

(Left: ntrain=100, Right: ntrain=500)

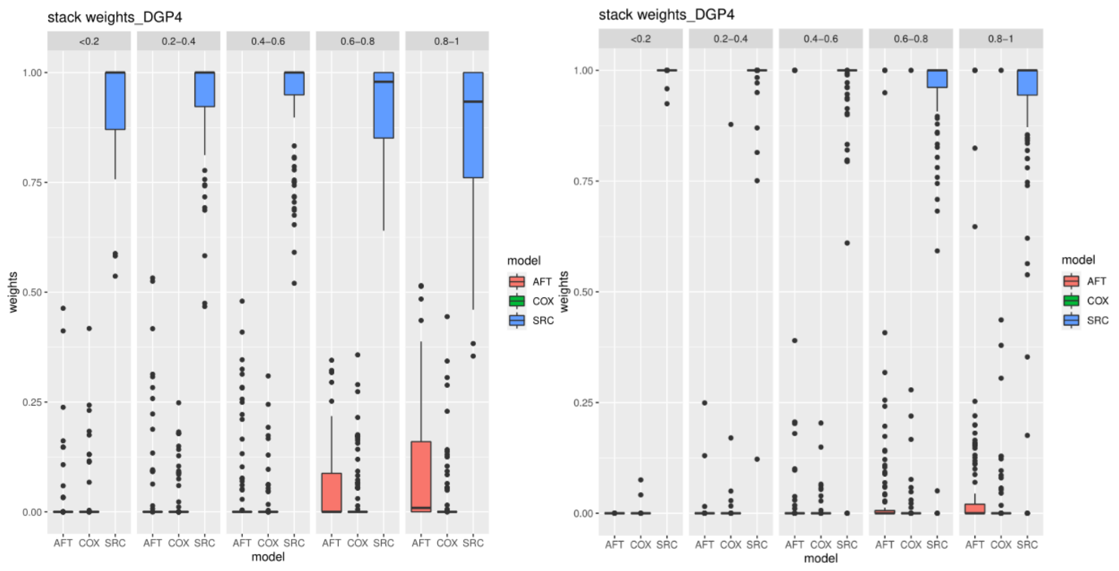


Figure 13 Time quantiles weights distribution of general stacking method for DGP4

(Left: ntrain=100, Right: ntrain=500)

We can see that the weights are more centralized with less dispersion in right chart (train size 500) compared with left chart (train size 100) in Figure 11 to Figure 13 of DGP 2 to DGP4. More training data allows us to be more accurate in determining which model performs best. SRC is clearly the best for DGP2, DGP3 and DGP4. At any time point, this new stacking approach automatically selects the optimal algorithm SRC and assigns almost all the weight to this model. The clustering stacking figures (train size 500) for the four DGPs are not displayed, because they are quite similar to general stacking method (train size 500).

Now we look at the error and see how the outcome changes as the training size goes from 100 to 500. Theoretically, test error decreases with training size from a large value to the stable value. When training size increases, it is normal to see the range of IAE decrease from level 0.1 (Figure 7) to level 0.05 (Figure 14). The same is true for ISE which drops from level 0.15 (Figure 8) to level 0.0125 (Figure 15). This is simple to comprehend because the more data we have, the better the model can do. When compared to the previous simulation with a training size of 100, IAE and ISE produce nearly the same results: the stacking or clustering stacking approach can give us nearly the best performance of all the models. When Cox or AFT performs better, as in the case of DGP1, stacking or clustering stacking provides predictions that are very close to Cox or AFT. When SRC model performs well, such as DGP2, DGP3 and DGP4, stacking or clustering stacking provides us predictions that are very close to SRC.

On the one hand, this new stacking method appears to be unable to outperform the best candidate model while using IAE as the criterion. By using ISE as the criterion, however, a pleasant surprise can be noticed in DGP1: the ISE of clustering stacking gets slightly lower as compared to the median, average, or standard deviation than the best model Cox. This demonstrates that the advantages of different algorithms can be absorbed by this new stacking approach. However, when one of the candidates has a overwhelming advantage, such as model SRC in DGP2, DGP3 and DGP4, the stacking method assign it a near-total 1 weight to it, particularly when there are more training data (500). As a result, the output of this new stacking is nearly identical to that of the best candidate algorithm SRC.

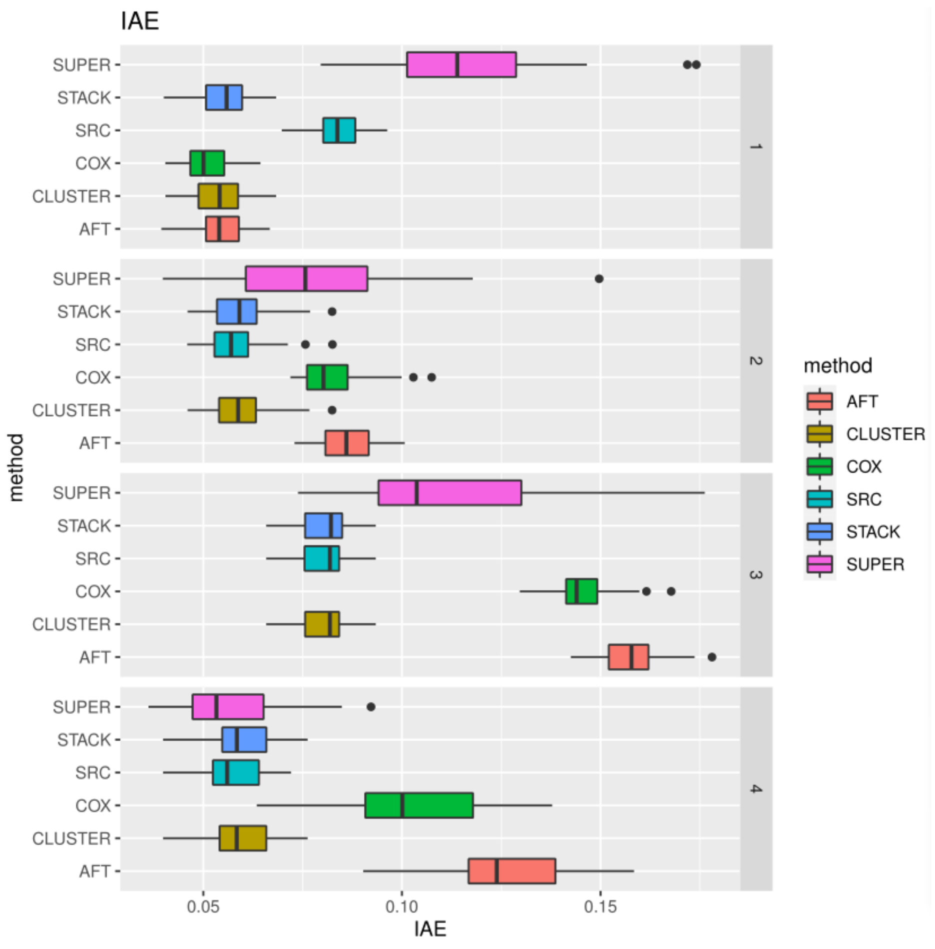


Figure 14 Integrated Absolute Error (IAE) boxplots of four simulation data sets (ntrain=500)

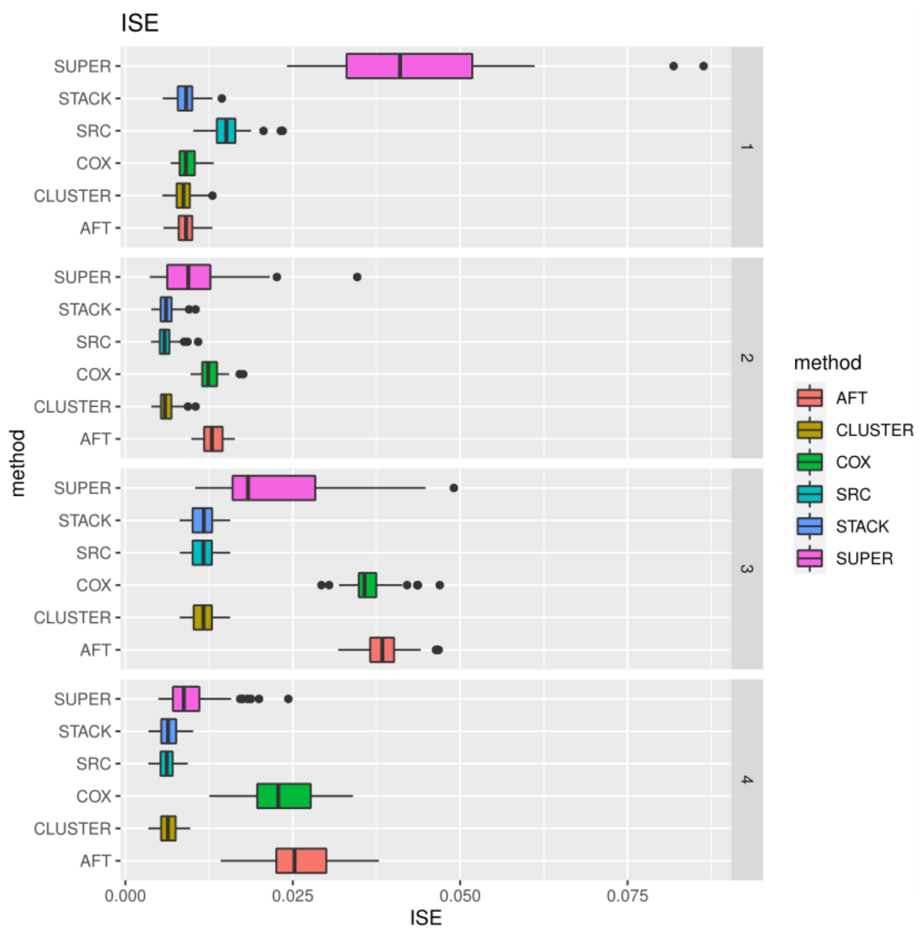


Figure 15 Integrated Square Error (ISE) boxplots of four simulation data sets (ntrain=500)

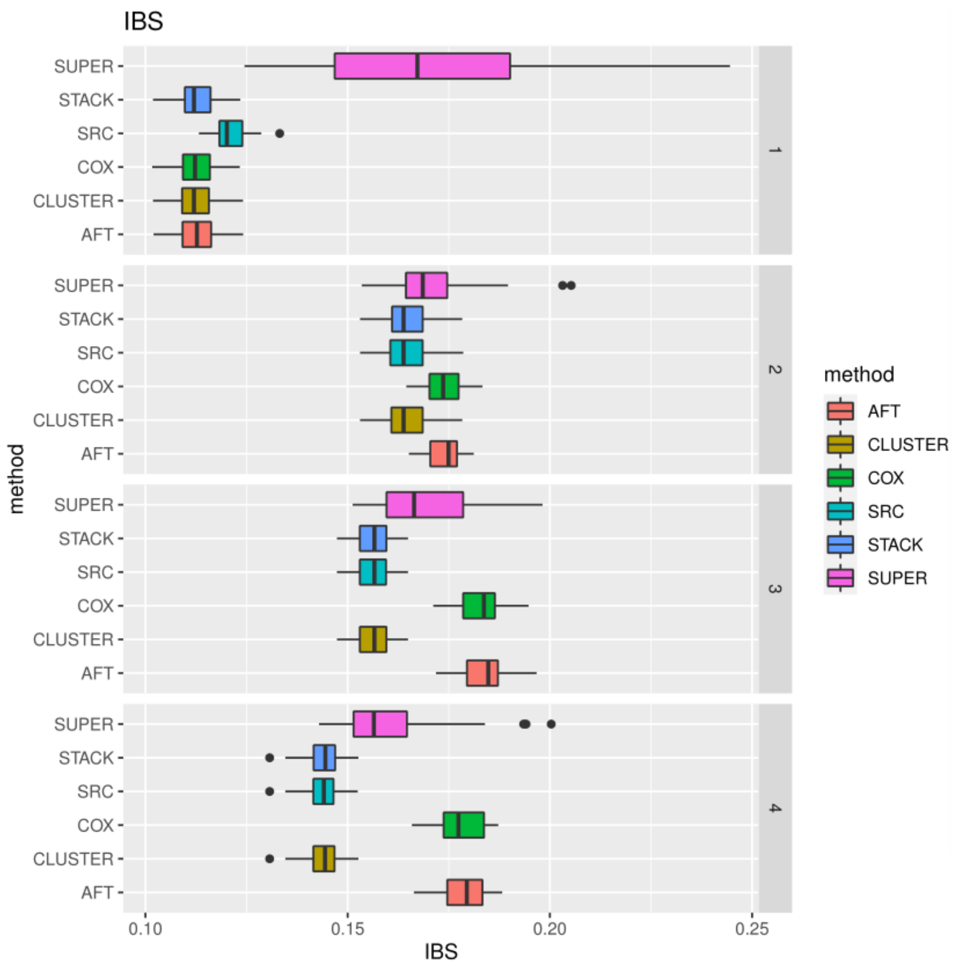


Figure 16 Integrated Brier Score (IBS) boxplots of four simulation data sets (ntrain=500)

It is not shocking that when the training size is increased from 100 to 500, IBS values also decrease marginally (Figure 16). Since IBS is not as precise as IAE or ISE, it appears that for DGP1, Cox and AFT are nearly identical, and both stacking work nearly identically with the best candidate for all four DGPs.

Chapter 6

Real data study

6.1 Introduction of six selected real data sets

In this section, we compare the performance of the same methods used in the simulation study with six real data sets, the same as the ones used in "L_1 splitting rules in survival forests" by Hoora Moradian et al. (2017)[25]: The Primary Biliary Cirrhosis (PBC) data, the CSL Liver Cirrhosis data, the German Breast Cancer (GBC) study group data, the Wisconsin Breast Cancer Prognostic (WPBC) data, the Veteran data and the National Wilm's Tumor Study (NWTCO) data. A brief description of these data sets is presented in Table 1.

The PBC data is described in the monograph by Fleming and Harrington (1991)[33]. We use all twelve of its covariates used by Bou-Hamad et al. (2011)[34] plus *copper, sgot and stage*. The same 312 patients who participated in the randomized trial are used here. Missing values are replaced by the median as in Bou-Hamad et al. (2011)[34] and Fleming and Harrington (1991)[33]. The data is from the R package SMPracticals[35].

The CSL data was obtained by Schlichting et al. (1983)[36] and is provided in the package timereg[37]. In this example, we only use the six invariant covariates (*time, prot, sex, treat, prot.base and prot.prev*). Records are grouped by ID variable, so the number of observations used is 446 (2481 rows).

The GBSG data (Schumacher et al. (1994)[38]) is obtained from the package mfp[39]. The data contains 686 observations and eight covariates. There is no missing data.

The WPBC data is available in the UCI machine learning repository (Bache and Lichman (2013)[40]). We can also find it in package TH.data[41]. There are 198 observations in the data. However, four missing values are replaced by the average. Thirty-two covariates are used in this example.

The Veteran data (Kalbfleisch and Prentice (1980)[42]) is obtained from the package randomForestSRC[43]. There are 137 observations with no missing values. It contains six covariates.

Finally, the NWTCO data (Breslow and Chatterjee (1999)[44]) is available in the package survival[3]. The four relevant covariates, *instit, histol, age and stage*, are used here. The data consists of 4088 observations and no missing values.

Table 2 Descriptions of the real data sets

No.	Name	# Covariates	Sample size	% Censoring	Source
1	PBC	15	312	60	Package SMPracticals
2	CSL	6	446	39	Package timereg
3	GBSG	8	686	56	Package mfp
4	WPBC	32	198	76	Package TH.data
5	Veteran	6	137	7	Package randomForestSRC
6	NWTCO	4	4088	85	Package survival

We can generate and replicate a simulation data set 50 times to obtain reliable results for comparing the performances, but we cannot generate data for real data. Therefore, the cross-validation method is used to evaluate and obtain results for real data sets. Here, each real data set is split into 10 groups. The observations of each group can be used as test data to be estimated, while the other observations can be used as training data. Thus, 10 results for each real data set are obtained.

Figure 17 below shows Kaplan-Meier estimate of the survival curve for each real data set. To ensure that each time block has enough subjects with event time to generate the stacking model, we set the maximum time grid as 3000 for PBC data, 8 for CSL data, 2000 for GBSG data, 50 for WPBC data, 300 for Veteran data and 500 for NWTCO data.

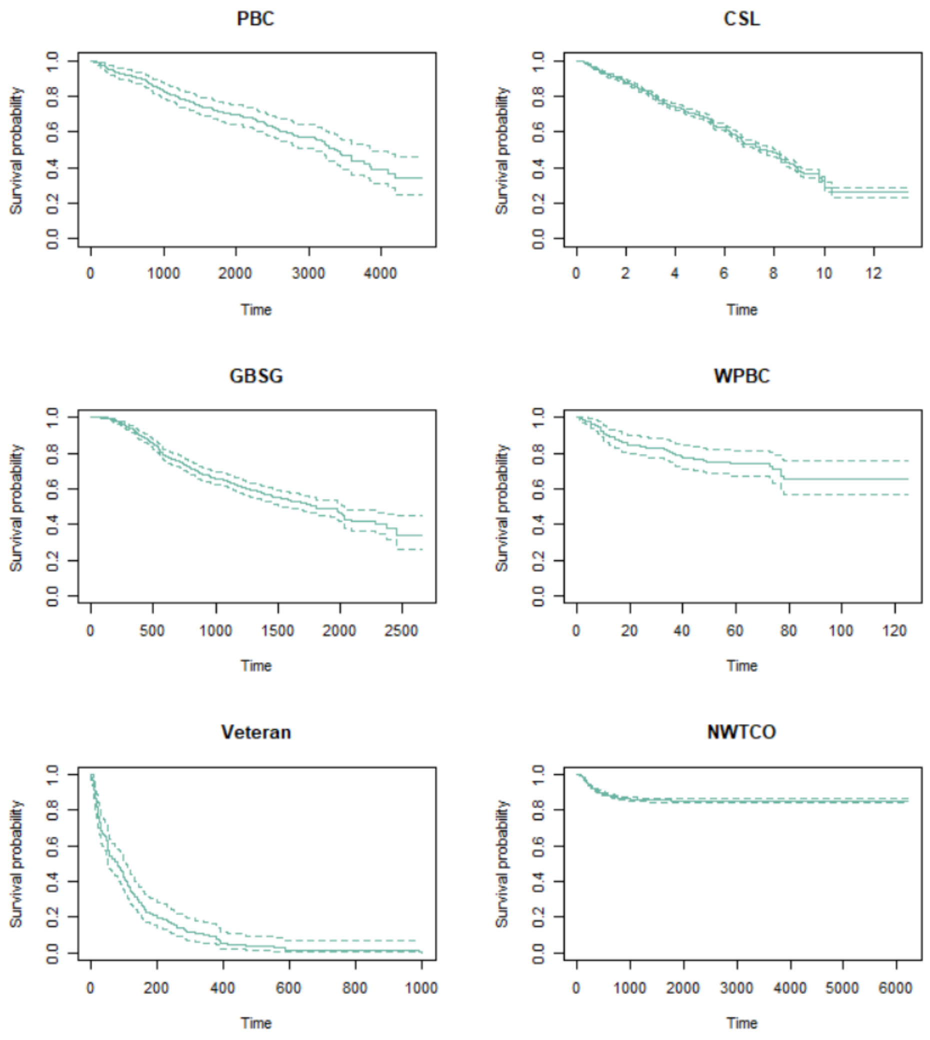


Figure 17 Kaplan-Meier estimate of the survival curve for six real data sets with censoring proportion 0.6, 0.39, 0.56, 0.76, 0.07, 0.85 respectively

(From left to right, from top to bottom)

6.2 Real data sets results

6.2.1 Weights presentation

The simulation data sets have proven that the weights for each candidate model depend on the time range, and general stacking weights are very similar to clustering stacking method. However, when it comes to the real data set, things change a bit. As we all know, clustering regroups the observations based on their different characteristics. For example, the age or the cell size. Each group would have similar properties that may be more apparent, and K-means clustering may be a better option for grouping to obtain constrained weights than linear model tree. That is why the clustering stacking method is introduced and evaluated.

According to Figure 18 below for PBC data, it appears that the weights of the Cox method are barely 0, especially for smaller than 0.8 quantiles of maximum time grid 3000. The weights of the AFT method have higher importance (the medians are over 0.5 for each time grid).

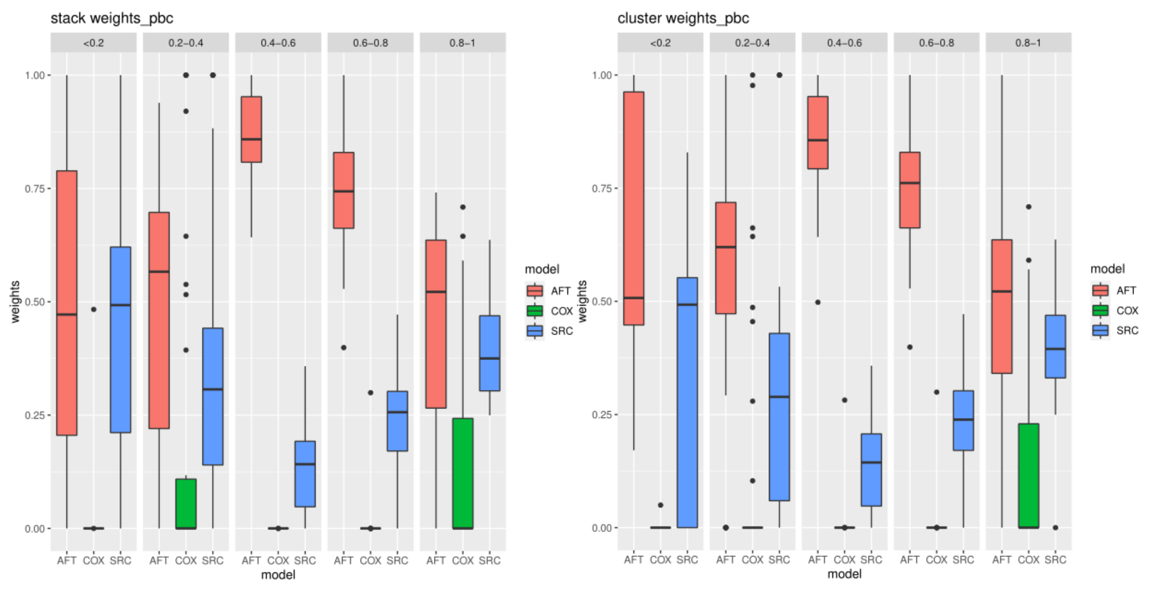


Figure 18 Time quantiles weights distribution of stacking method for PBC data

(Left: general stacking, Right: clustering stacking)

However, CSL data (Figure 19) is another case. We can only conclude that the Cox method may be ineffective since its weight is early 0 for most time intervals, but we cannot discern the importance between the AFT method and survival forest because the weights have a wide range from 0 to 1 with large variances. Because there is no clear difference, it seems difficult to tell which candidate algorithm is more appropriate for this data set, even for a small time range.

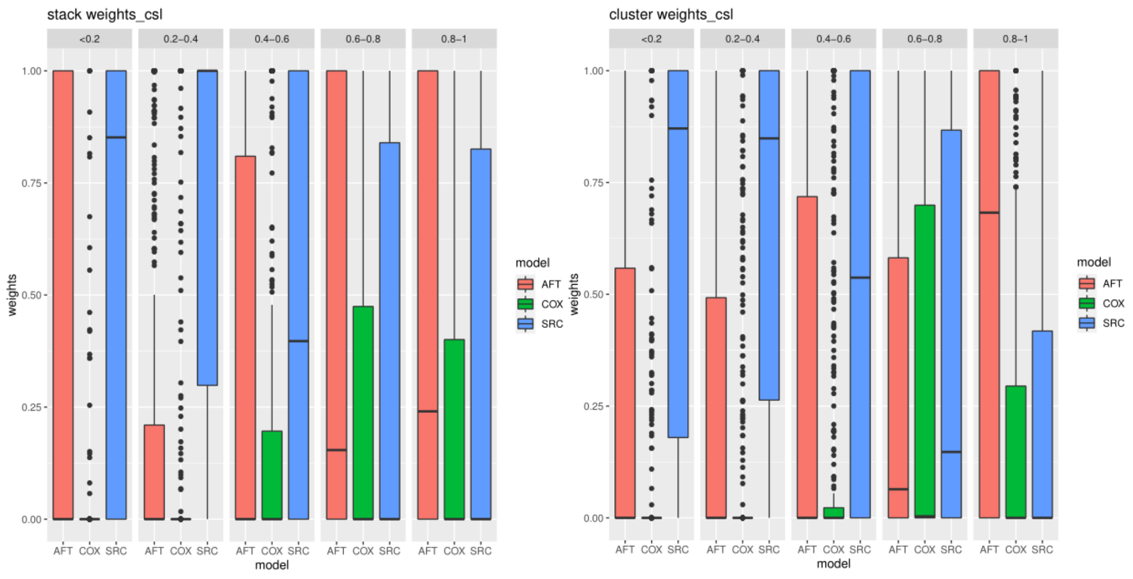


Figure 19 Time quantiles weights distribution of stacking method for CSL data

(Left: general stacking, Right: clustering stacking)

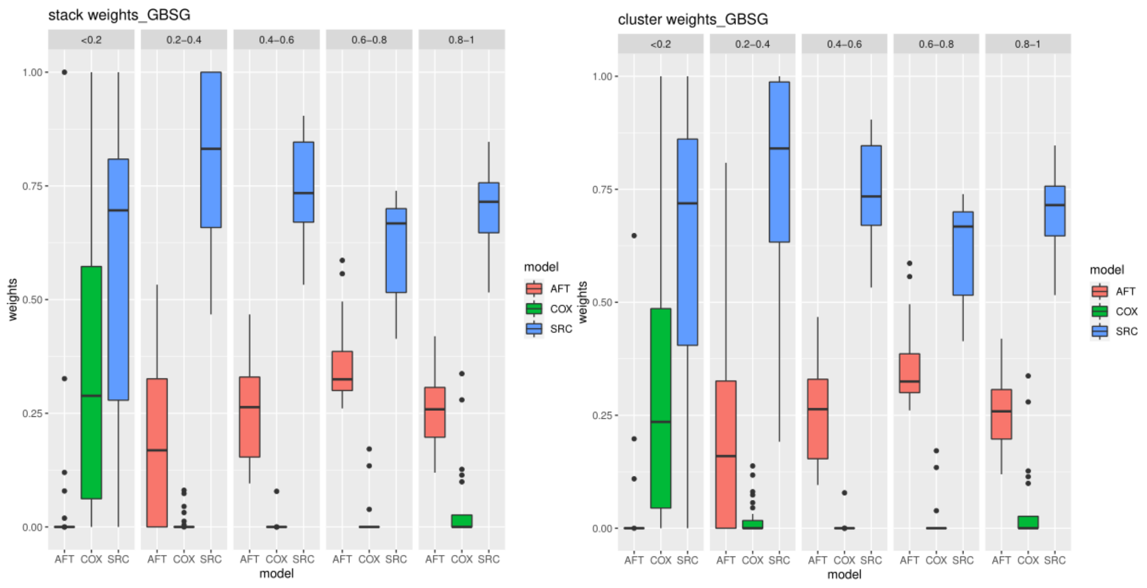


Figure 20 Time quantiles weights distribution of stacking method for GBC data

(Left: general stacking, Right: clustering stacking)

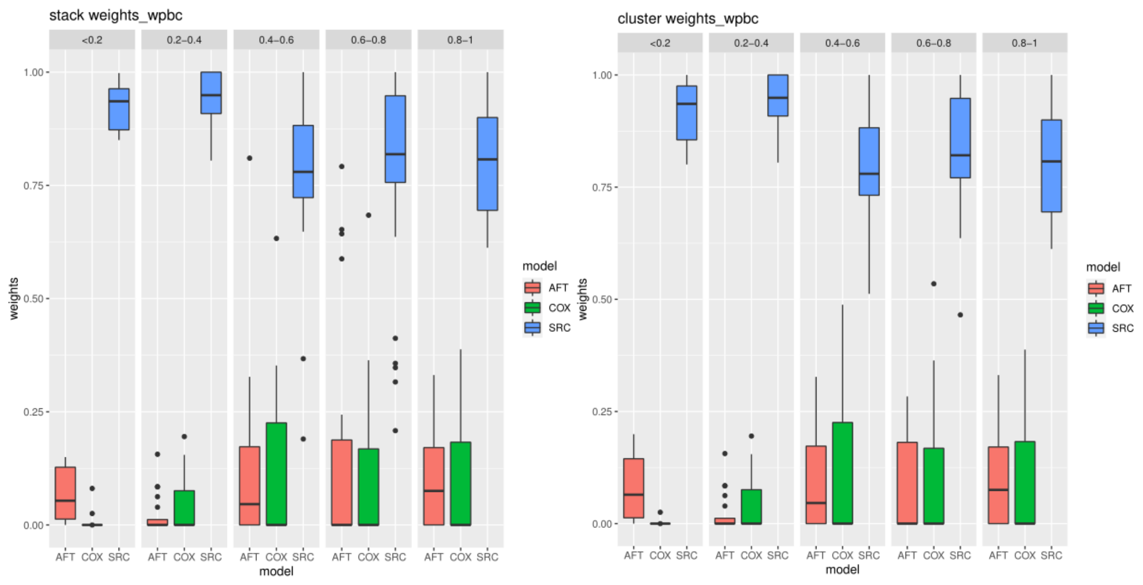


Figure 21 Time quantiles weights distribution of stacking method for WPBC data

(Left: general stacking, Right: clustering stacking)

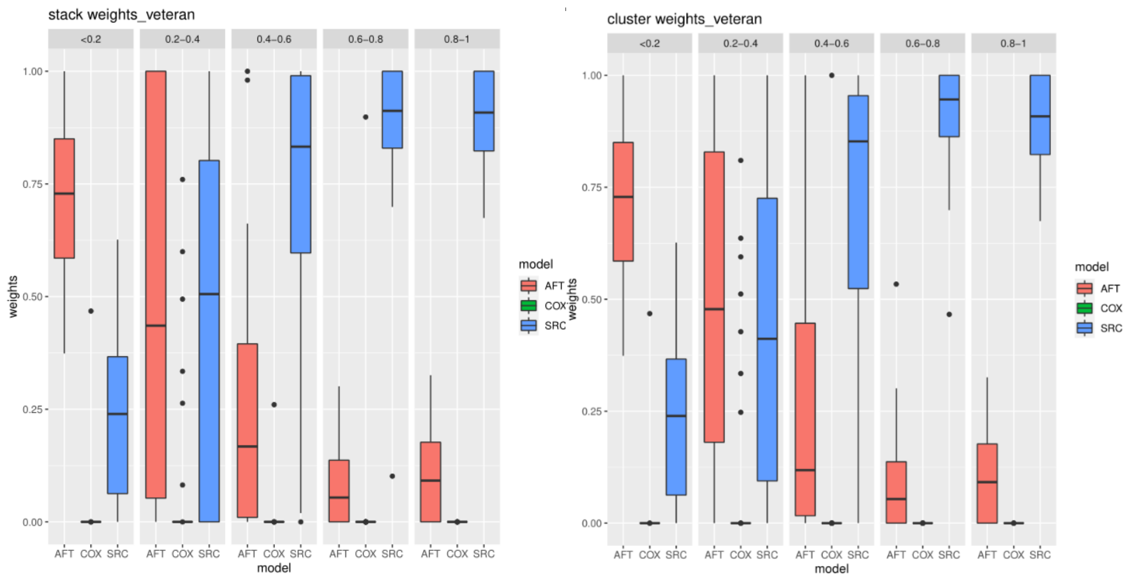


Figure 22 Time quantiles weights distribution of stacking method for Veteran data

(Left: general stacking, Right: clustering stacking)

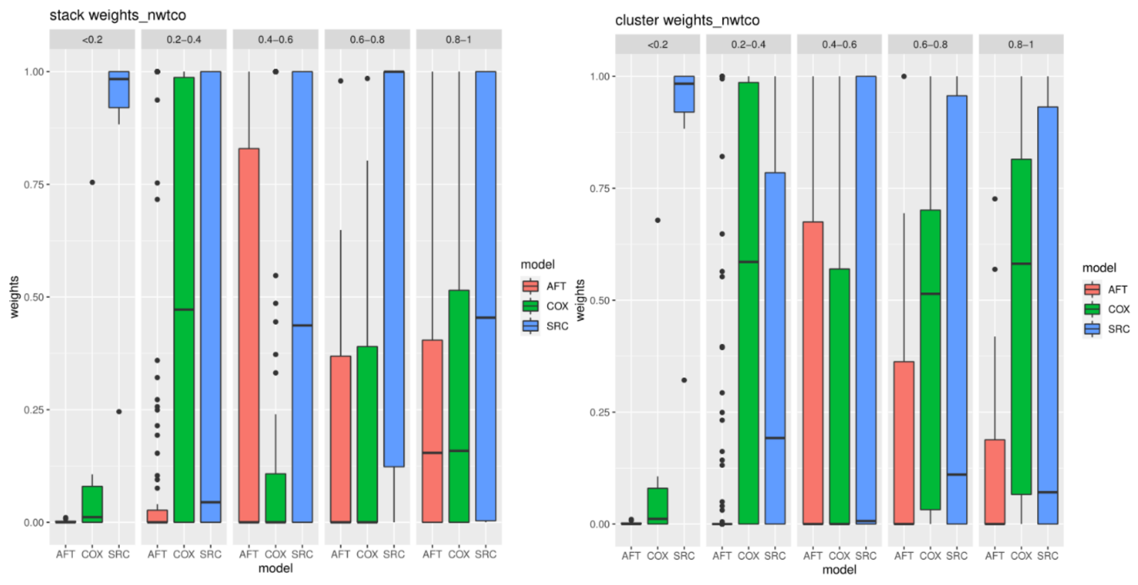


Figure 23 Time quantiles weights distribution of stacking method for NWTCO data

(Left: general stacking, Right: clustering stacking)

Figure 20 to Figure 23 show IBS results for GBSG data, WPBC data, Veteran data and NWTCO data. GBSB, WPBC, and Veteran data have an obvious preference of choosing models by combing their balance weights or with a single high weight. For example, in all time ranges, the weight of SRC (survival forests model) is very high for the WPBC data, it could be explained by survival forests model's excellent performance. NWTCO is a little different from the others, when time is less than the 0.2 quantile, the random forests model gets a very high weight with a slight variance, as shown in Figure 23. However, due to its high censoring proportion of 0.85, it is difficult to tell which model outperforms the others when time is greater than the 0.2 quantile. One explanation may be the lack of data for large time grids. It emphasizes the importance of having enough non-censoring data at each time interval to implement the new stacking method.

6.2.2 Performance comparison

Since the true survival function is unknown for real data, The IAE and ISE cannot be calculated as we did with the artificial data sets. Instead, our primary criterion is the integrated Brier Score. It can be considered as weighted mean squared error, the smaller a model's integrated Brier Score is, the better it performs. The graphic below shows the result of IBS for each candidate method with the stacking method and super learner algorithm (Marzieh K.Golmakani et al. 2020[21]). Because of multiple iterations of super learner when optimizing risk, super learner algorithm costs too much time for a large number of observations, particularly when choosing many candidate models or complexes models. Here, in order to save time, only the Cox and lasso models are used to optimize. Nonetheless, NWTCO data took approximately a week to obtain the results for 4088 observations.

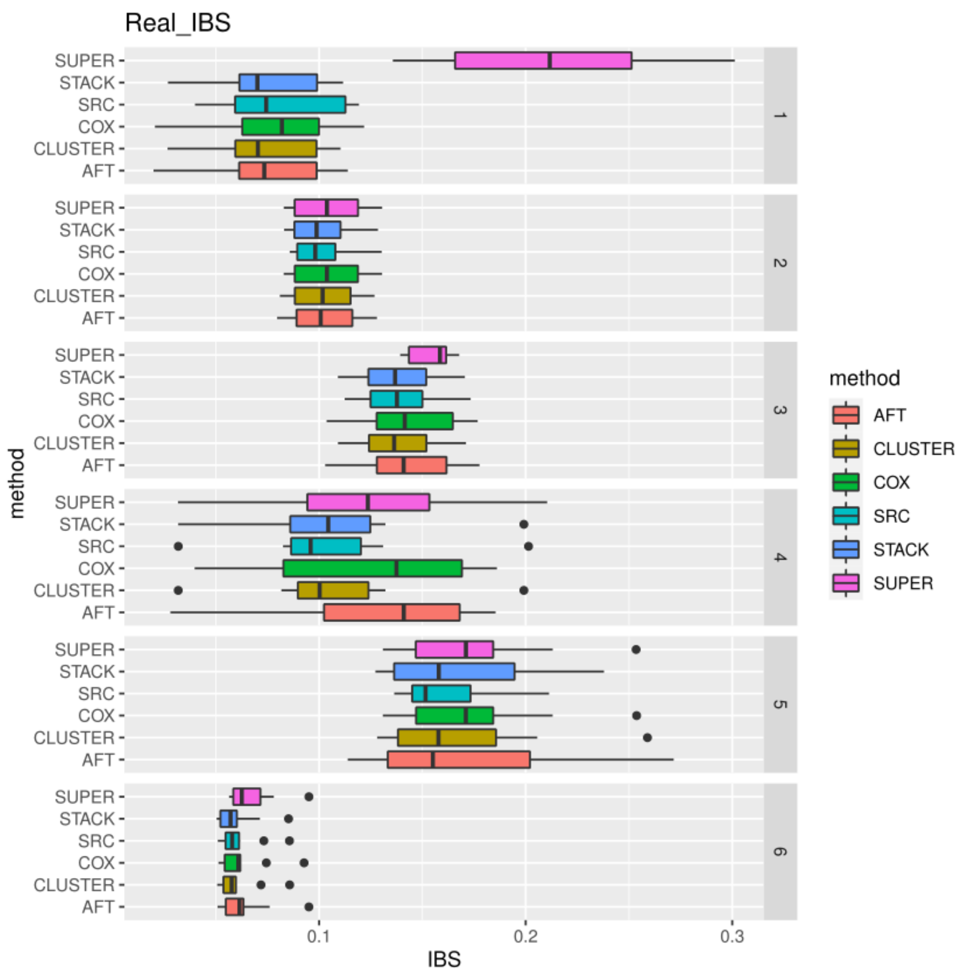


Figure 24 Integrated Brier Score (IBS) boxplots of six real data sets

(1: PBC   2: CSL   3: GBSG   4: WPBC 5: Veteran 6: NWTCO)

IBS follows the same criteria analogy with IAE and ISE; the smaller IBS, the better it perfoms in terms of prediction. Six real data sets all show that the stacking method (or clustering stacking) performs best or nearly best with low variance. For PBC data (No.1), super leaner perform the worst, while the other methods have almost the same performances. For CSL data 2, survival forests performs best because of its smallest variance, but overall, these models do not have significant difference under the IBS criteria. For GBSG data 3 and WPBC data 4, stacking method performs much better than the Cox, AFT and super learner methods, and they perform almost as well as the survial forest. Although AFT has smaller IBS than the Cox and stacking methods, it has the largest variance. SRC (survival forests model) perfoms better than the Cox model and the AFT model with Veteran data because of its lowest median IBS and lowest variance. We also could reach the same conclusion by the analysis of the weights presentation figures. SRC get the hightest weight among all the three basic models. For the NWTCO data, because of its high proportion of censoring 0.85, it is difficult to tell the difference among all the algorithms.

Overall, this new stacking method with time-dependent and covariate-dependent weights can automatically detect the good parts of each candidate model to avoid performing the worst, but it is unlike the traditional stacking method with fixed weights for each candidate model whose simple linear regression usually gives us the middle performance. Since it can combine ideal parts to produce a new estimated survival function for each new observation, it can perform near-best of these candidate models. Furthermore, when the data has grouping properties, such as in WPBC and Veteran data, the clustering stacking method can be useful and performs better than general stacking.

Chapter 7

Discussion and conclusion

This new stacking method with time-dependent and covariate-dependent weights works effectively for survival data, and it proves that survival function depends on the time and covariates at the same time. According to the fixed time grid, it could automatically choose the ideal parts of each candidate model and combine them together with time-dependent and covariate-dependent weights. In other words, this new stacking method can perform nearly the best, no matter if the performance of the candidate algorithm is good or not. Another advantage of this new stacking method is that we could analyze its weights representation figures to find out which model is more appropriate for a certain time range.

However, the number of time grids could affect the performance of the stacking method. Perhaps selecting 9 as the size of the grid is not optimal. Maybe using equispaced grid values based on the Kaplan-Meier is not optimal also. Because we cannot know in advance which time grid that we choose is the best, the result for stacking method is not perfect, although it performs good enough in our experiments on simulation data sets and on real data sets. Future studies could investigate the time grid selection to optimize the prediction results.

Another inconvenience is that this stacking method is somewhat complex, and time-consuming. Here we use three basic methods; if more candidate models are added in, it will undoubtedly complicate the process. Due to the weights and grouping work, the computation of prediction for new observations at hundreds of time points also takes a lot of time.

In conclusion, with the goal of improving the prediction performance, how to optimize this new stacking method with time-dependent and covariate-dependent weights is important for future study and research work, either from the direction of choosing time grids, or from computing work.

Appendix

Appendix [1]

The function *iaeise* is used to calculate IAE (Integrated absolute error) and ISE (Integrated square error) for one subject as the criteria value to evaluate the model performance for the simulation data set with the inputs s (vector of true survival probabilities), shat (vector of estimated survival probabilities), time (vector of time points for s and shat) and endpoint (last value for the integration). The function *res* is used to calculate IAE and ISE for all test subjects with the inputs true_prob (vector of true survival probabilities, values followed by next subject at all time points), estimated (vector of estimated survival probabilities, values followed by next subject at all time points), time (vector of time points) and npoints (number of time points).

```
# function to compute the IAE and ISE for one subject
iaeise=function(s,shat,time,endpoint){
  # s = vector with the true S (s[1] should be 1)
  # shat = vector with the estimated S (shat[1] should be 1)
  # time = time points for s and shat  (time[1] should be 0)
  # endpoint = last value for the integration (must be > last value of time)
  time=c(time,endpoint)
  timediff=diff(time)
  iae=sum(timediff*abs(s-shat))/endpoint
  ise=sum(timediff*(s-shat)^2)/endpoint
  c(iae,ise)
}
######################################################
# Function to get IAE and ISE for all subjects
res=function(true_prob,estimated,time,npoints)
{
  res=NULL
  max_time=max(time)+time[2]-time[1]
  for(i in 1:(length(true_prob)/npoints))
  {
    true_i=true_prob[((i-1)*npoints+1):(i*npoints)]
    test_i=estimated[((i-1)*npoints+1):(i*npoints)]
    time_i=time[((i-1)*npoints+1):(i*npoints)]
    addres=iaeise(true_i,test_i,time_i,max_time)
    res=rbind(res,addres)
  }
  return(res)
}
```

Appendix [2]

The function *get_weights* is used to get the weights based on Cox, AFT and SRC with the new stacking method as described in Section 4.4. The inputs dattrain (training data set), dattest (test data set that needed to be standardized with training data set before being clustered with k-means), k (number of folders of cross-validation scheme), t_grid (vector of time grids) and formule (formula of target Y and covariates X) are needed. A list of general stacking weights, liner model tree, k-means model and clustering stacking weights fitted with training data are returned.

```
# function to get the constrained weights, lmtree models and
# constrained cluster weights
get_weights=function(dattrain,dattest,k,t_grid,formule)
{
  ntrain=nrow(dattrain)
  per=c(1:ntrain)
  tl=1
  nntest=ceiling(ntrain/k)
  predsrc_cv=list()
  indtest=list()
  for(i in 1:(k-1))
  {
    indtest[[i]]=per[tl:(tl+nntest-1)]
    tl=tl+nntest
  }
  indtest[[k]]=per[(nntest*(k-1)+1):ntrain]
  # k folds
  for(i in 1:k)
  {
    cind=indtest[[i]]
    fitsrc_cv=rfsrc(Surv(y,status)~.,data = dattrain[-cind,],ntree = 200)
    predsrc_cv[[i]]=predict(fitsrc_cv,newdata=dattrain[cind,])
  }
  # AFT model
  pre_median=NULL
  pre_scale=NULL
  # k=10 folds
  for (i in 1:k)
  {
    cind=indtest[[i]]
    fitsreg6_cv=survreg(Surv(y,status)~.,data = dattrain[-cind,],dist =
"loglogistic")
    add=predict(fitsreg6_cv,newdata = dattrain[cind,],type = "response")
    add2=fitsreg6_cv$scale
    pre_median=c(pre_median,add)
    pre_scale=c(pre_scale,add2)
  }
  # Cox model
  predcox_cv=list()
  for(i in 1:k)
  {
    cind=indtest[[i]]
    fitcox_cv=coxph(Surv(y,status)~.,data = dattrain[-cind,],x=TRUE)
    predcox_cv[[i]]=survfit(fitcox_cv,newdata=dattrain[cind,])
  }
  # get the combined table for each individual at time t_grid
  # the table result_train contains true survival prob, estimated prob
src,aft,cox and time t_grid.
  result_train=NULL
  for(i in 1:ntrain)
  {
    output_src=NULL
    output_aft=NULL
    output_cox=NULL
    output_t=NULL
    folder=ceiling(i/nntest)
    for(t in 1:length(t_grid))
    {
      num_src=which.min(abs(predsrc_cv[[folder]]$time.interest-t_grid[t]))
      num_cox=which.min(abs(predcox_cv[[folder]]$time-t_grid[t]))
      add1=predsrc_cv[[folder]]$survival[(i-(folder-1)*(nntest)),][num_src]
      add2=1/(1+((t_grid[t]/pre_median[i])^(1/pre_scale[folder])))
      add3=predcox_cv[[folder]][(i-(folder-1)*(nntest))]$surv[num_cox]
      add4=t_grid[t]
      output_src=c(output_src,add1)
      output_aft=c(output_aft,add2)
      output_cox=c(output_cox,add3)
      output_t=c(output_t,add4)
    }
    add5=bind_rows(replicate(length(t_grid), dattrain[i,], simplify = FALSE))

result_train_i=data.frame(add5,output_src=output_src,output_aft=output_aft,
                               output_cox=output_cox,t=output_t)
    result_train=rbind(result_train,result_train_i)
  }
  # Get pseudo-observations of train group
  pseudo_surv=pseudosurv(time=dattrain$y,event=dattrain$status,tmax = t_grid)
  # From the summary table, we can see that the pseudo-observations based
  # on the Kaplan-Meier estimator have negative value or more than 1.
  # put the pseudo-observations in the table
  pseudo_s=NULL
  for(i in 1:nrow(pseudo_surv$pseudo))
  {
    pseudo_s=c(pseudo_s,pseudo_surv$pseudo[i,])
  }
  result_train$pseudo_s=pseudo_s
  mob_train=list()
  pre_train=list()
  node_train=list()
  nodeid_train=list()
  for(i in 1:length(t_grid))
  {
    mydata=result_train[result_train$t==t_grid[i],]
    mob_train[[i]]<- lmtree(formule,data = mydata)
    pre_train[[i]]=predict(mob_train[[i]],mydata,type = "response")
    node_train[[i]]=predict(mob_train[[i]],mydata,type="node")
    nodeid_train[[i]]=sort(unique(node_train[[i]]))
  }
  train_node=NULL
  for (i in 1:ntrain)
  {
    for( j in 1:length(t_grid))
    {
      add=node_train[[j]][i]
      train_node=c(train_node,add)
    }
  }
  result_train$node=train_node
  ntype=3 # number of algorithms
  # To get constrained weights(sum=1,and be positives)
  weights_train=list()
  #predict_train=rep(0,ntrain)
  for(i in 1:length(t_grid))
  {
    weights_train[[i]]=matrix(0,length(nodeid_train[[i]]),3)
    for(j in 1:length(nodeid_train[[i]]))
    {
      indi=which(result_train$t==t_grid[i] &
result_train$node==nodeid_train[[i]][j])
      dati=result_train[indi,]

Rinv=solve(chol(as.matrix(t(dati[,c("output_src","output_aft","output_cox")])
) %*% as.matrix(dati[,c("output_src","output_aft","output_cox")])))
      c=cbind(rep(1,ntype),diag(ntype))
      b=c(1,rep(0,ntype))
      d=as.matrix(t(dati[,"pseudo_s"])) %*%
        as.matrix(dati[,c("output_src","output_aft","output_cox")])
      weights_train[[i]][j,]=solve.QP(Dmat = Rinv,factorized = TRUE,
                                dvec = d,Amat = c,bvec =
b,meq=1)$solution
      # compute the predictions with the new weights
      #predict_train[indi]=apply(t(weights_train[[i]][j,] *
      #
as.matrix(t(dati[,c("output_src","output_aft","output_cox")]))),1,sum)
    }
  }
  ###### get cluster weights
  ngroup=NULL
  for(i in 1:length(t_grid))
  {
    ngroup[i]=nrow(weights_train[[i]])
  }
  mydata_all=rbind(dattrain[,!(names(dattrain) %in%
c("y","status"))],dattest[,!(names(dattest) %in% c("y","status"))])
  mydata_all_sta=as.data.frame(scale(mydata_all,center = TRUE,scale = TRUE))
  mydata=dattrain[,!(names(dattrain) %in% c("y","status"))]
  mydata_standalise=mydata_all_sta[c(1:nrow(dattrain)),]
  result=list()
  weights_cluster=list()
  my_data=list()

  for(i in 1:length(t_grid))
  {
    result[[i]]=kmeans(mydata_standalise,ngroup[i])
    my_data[[i]]=result[[i]]$cluster
    weights_cluster[[i]]=matrix(0,ngroup[i],ntype)
    for(j in 1:ngroup[i])
    {
      indi=which(my_data[[i]]==j)
      dati=result_train[result_train$t==t_grid[i],][indi,]

Rinv=solve(chol(as.matrix(t(dati[,c("output_src","output_aft","output_cox")])
) %*%

as.matrix(dati[,c("output_src","output_aft","output_cox")])))
      c=cbind(rep(1,ntype),diag(ntype))
      b=c(1,rep(0,ntype))
      d=as.matrix(t(dati[,"pseudo_s"])) %*%
as.matrix(dati[,c("output_src","output_aft","output_cox")])
      weights_cluster[[i]][j,]=solve.QP(Dmat = Rinv,factorized = TRUE,dvec =
d,Amat = c,bvec = b,meq=1)$solution
    }
  }
return(list(weights_train=weights_train,mob_train=mob_train,mob_cluster=resul
t,weights_cluster=weights_cluster))
}
```

Appendix [3]

The function *pre_test* can get predictions for test data of three candidate models Cox, AFT, SRC, general stacking and clustering stacking models with the inputs dattrain (training data set that needed to be standardized with test data set), dattest (test data set), mob_train (linear model tree obtained from function *get_weights*), weights_train (general stacking weights obtained from function *get_weights*), mob_cluster (k-means model obtained from function *get_weights*), weights_cluster (clustering stacking weights obtained from function *get_weights*) and t_grid (vector of time grids). The time points from prediction of SRC model are used for all the other models.

```
# function to get predictions of all the models
pre_test=function(dattrain,dattest,mob_train,weights_train,mob_cluster,weight
s_cluster,t_grid)
{
  ntest=nrow(dattest)
  ntype=3
  # fit 3 models for test group
  # AFT model
  fitsreg6=survreg(Surv(y,status)~.,data = dattrain,dist = "loglogistic")
  test_median=predict(fitsreg6,newdata = dattest,type = "response")
  test_scale=fitsreg6$scale
  # Survival tree
  fitsrc=rfsrc(Surv(y,status)~.,data = dattrain,ntree = 200)
  predsrc=predict(fitsrc,newdata=dattest)
  # Cox model
  fitcox=coxph(Surv(y,status)~.,data = dattrain,x=TRUE)
  predcox=survfit(fitcox,newdata=dattest)
  # Test group frame date set
  result_test=NULL
  t_points=c(0,predsrc$time.interest)
  for(i in 1:ntest)
  {
    output_aft=NULL
    output_cox=NULL
    id=NULL
    for(t in 1:length(t_points))
    {
      num_cox=which.min(abs(predcox$time-t_points[t]))
      add2=1/(1+((t_points[t]/test_median[i])^(1/test_scale)))
      add3=predcox$surv[num_cox,i]
      output_aft=c(output_aft,add2)
      output_cox=c(output_cox,add3)
    }
result_test_i=data.frame(id=rep(i,length(t_points)),output_src=c(1,predsrc$su
rvival[i,]),output_aft=output_aft,output_cox=output_cox,t=t_points)
    result_test=rbind(result_test,result_test_i)
  }
  # put the estimated values(stack) with constraints in the data set
  t_med=0.5*(t_grid[-1]+t_grid[-length(t_grid)])
  t_pre=NULL
  for(i in 1:length(t_points))
  {
    if(t_points[i]<=t_med[1]){add=1}
    if(t_points[i]>t_med[length(t_med)]){add=length(t_med)+1}
    for(j in 1:(length(t_med)-1))
    {
      if(t_points[i]>t_med[j] & t_points[i]<=t_med[j+1]){add=j+1}
    }
    t_pre=c(t_pre,add)
  }
  result_test$t_pre=t_pre
  predict_test=NULL
  node_test=NULL
  for(i in 1:nrow(result_test))
  {
    j=result_test[i,]$t_pre
    mydata=dattest[result_test[i,]$id,]
    i_node=predict(mob_train[[j]],mydata,type = "node")
if(length(coef(mob_train[[j]]))>ntype){num_node=which(row.names(coef(mob_trai
n[[j]]))==i_node)}
    if(length(coef(mob_train[[j]]))==ntype){num_node=1}
    add_test=apply(t(weights_train[[j]][num_node,] *
as.matrix(t(result_test[i,c("output_src","output_aft","output_cox")]))),1,sum
)
    node_test=c(node_test,num_node)
    predict_test=c(predict_test,add_test)
  }
  result_test$node=node_test
  result_test$stack=predict_test
  mydata_all=rbind(dattrain[,!(names(dattrain) %in%
c("y","status"))],dattest[,!(names(dattest) %in% c("y","status"))])
  mydata_all_sta=as.data.frame(scale(mydata_all,center = TRUE,scale = TRUE))
  test_standalise_unique=mydata_all[c((nrow(dattrain)+1):nrow(mydata_all)),]

test_standalise=test_standalise_unique[rep(seq_len(nrow(test_standalise_uniqu
e)), each = length(t_points)),]
  test_standalise$t_pre=result_test$t_pre
  pre_group=list()
  pre_prob=list()
  test_i=list()
  t_sort=unique(test_standalise$t_pre)
  for(i in 1:length(t_sort))
  {
    test_i[[i]]=test_standalise[test_standalise$t_pre==t_sort[i],]
    pre_group[[i]]=cl_predict(mob_cluster[[t_sort[i]]],
test_i[[i]][,!(names(test_i[[i]]) %in% c("t_pre"))])

pre_prob[[i]]=rep(0,nrow(test_standalise[test_standalise$t_pre==t_sort[i],]))
    for (j in 1:nrow(weights_cluster[[i]]))
    {
      indi=which(pre_group[[i]]==j)
      dati=result_test[result_test$t_pre==t_sort[i],][indi,]
      pre_prob[[i]][indi]=apply(t(weights_cluster[[t_sort[i]]][j,] *

as.matrix(t(dati[,c("output_src","output_aft","output_cox")]))),1,sum)
    }
  }
  stack_cluster=rep(0,nrow(result_test))
  for(p in 1:length(t_sort))
  {
    tra=which(result_test$t_pre==t_sort[p])
    stack_cluster[tra]=pre_prob[[p]]
  }
  result_test$stack_cluster=stack_cluster
  return(result_test)
}
```

Appendix [4]

The function *isotonic* is used to render the estimated survival probabilities non-increasing with time. For any model, the estimated survival probability at time $t = 0$ should be 1, and it should not be greater than the previous time point. We have the result_test (a table of estimated survival probabilities at all time points fitted by the Cox, AFT, SRC, general stack, cluster stack and super learner models for all test individuals) and npoints (number of time points) as inputs.

```
# function to make shat[1]=1 and isotonic regression
isotonic=function(result_test,npoints){
  result=NULL
  for(i in 1:(nrow(result_test)/npoints))
  {
    individu=result_test[(i-1)*npoints+seq(npoints),]
    individu$output_src[1]=1
    individu$output_aft[1]=1
    individu$output_cox[1]=1
    individu$stack[1]=1
    individu$stack_cluster[1]=1
    individu$super[1]=1
    for(j in 2:npoints)
    {
      if(individu$stack[j]>individu$stack[j-
1]){individu$stack[j]=individu$stack[j-1]}
      if(individu$stack[j]<0){individu$stack[j]=0}
      if(individu$stack_cluster[j]>individu$stack_cluster[j-
1]){individu$stack_cluster[j]=individu$stack_cluster[j-1]}
      if(individu$stack_cluster[j]<0){individu$stack_cluster[j]=0}
    }
    result=rbind(result,individu)
  }
  return(result)
}
```

Appendix [5]

DGP1-DGP4 are four Data Generating Processes (DGPs) that produce simulated date sets. The DGPs are described in detail in Section 5.1. True survival probabilities for simulated data are calculated using the *trueS1-trueS4* functions, which take values from DGPs and a vector of time points as inputs.

```
# Function to generate artificial survival data DGP1
# n = sample size
# parcens = parameter(lambda) of the exponential censoring time.
#          Use it to get the desired proportion of censoring
# output = data frame with 21 columns
# col1 - col15 = covariates V1-V15
#            (only V1,V2,V3,V5,V6,V7,V10 are related to the event time)
# col16 = y = observed time(true or censored). This is the target variable
# col17 = status = 1 = dead(event occured); 0 = alive (censored)
# col18 -  col21 are used to evaluate the models but are not available for
training and in fact
#                 would not be available in a real applicaiton
# col18 = truey = true event time
# col19 = cens = ture cencoring time
# col20 = truefx = true function of the covariates (log scale)
#                 log(Y) = truefx+epsilon
#                 epsilon if from a gamma(3,5) distribution
# col21 = exptruefx = exp(truefx)
#                 Y = exptruefx*exp(epsilon)
DGP1=function(n,parcens)
{
  library(mvtnorm)
  sigma=matrix(0.3,15,15)
  sigma=sigma+0.7*diag(15)
  x=rmvnorm(n,mean = rep(0,15),sigma = sigma)
  x[,6]=abs(x[,6])
  x[,7]=as.numeric(x[,7]>0.5)
  x[,8]=as.numeric(x[,8]>0.2)
  x[,9]=log(x[,9]+5)
  x[,10]=exp(x[,10]/3)
  x[,11]=as.numeric(x[,11]>0.1)
  x[,12]=abs(x[,12])
  truefx=-0.5+(x[,1]+x[,2]+0.3*x[,2]^2+x[,5]+x[,6]-0.3*x[,5]*x[,6]+x[,7]+
               1/(x[,10]+3)+(x[,3]>0)*(x[,1]>0)-(x[,3]<0)*(x[,1]>0))/3
  truefx=apply(cbind(truefx,1.7),1,min)
  truefx=apply(cbind(truefx,-1.8),1,max)
  exptruefx=exp(truefx)
  # true value of Y(time) if no censoring
  truey=exptruefx*rgamma(n,3,5)
  # censoring time
  cens=rexp(n,1/parcens)
  # observed time
  y=apply(cbind(truey,cens),1,min)
  # censoring indicator (1 = event occured (dead); 0 = censored (alive))
  status=as.numeric(truey<cens)
  out=data.frame(cbind(x,y,status,truey,cens,truefx,exptruefx))
  out
}
# function to compute the true survival function: DGP1
trueS1=function(exptruefx,vectime)
{
  # exptruefx = value from DGP1 function
  # vectime = vector of points where to evaluate S
  pgamma(vectime/exptruefx,3,5,lower.tail = FALSE)
}

# Functions to reproduce DGP 2 to 4 from Moradian et al. (2017)[17]
####################### DGP2
DGP2=function(n,alpha)
{
  # n = sample size
  # alpha = parameter for the censoring distribution
  # status = 1 = dead (event occured); 0 = alive (censored)
  dat=data.frame(matrix(runif(n*10),ncol=10))
  names(dat)=paste("x",1:10,sep="")
  dat$u=10*abs(sin(dat$x1*pi-1)) + 3*abs(dat$x2-.5) + dat$x3
  dat$truetime=rexp(n,1/dat$u)
  dat$censor=alpha*runif(n)
  dat$y=apply(dat[,c("censor","truetime")],1,min)
  dat$status=as.numeric(dat$truetime<=dat$censor)
  dat
}
# function to compute the true survival function DGP2
trueS2=function(u,vectime)
{
  # u = value from DGP2 function
  # vectime =  vector of points where to evaluate S
  pexp(vectime, rate = 1/u, lower.tail = FALSE, log.p = FALSE)
}
####################### DGP3
DGP3=function(n,alpha)
{
  # n = sample size
  # alpha = parameter for the censoring distribution
  sigma=diag(25)
  for(i in 1:25){
    for(j in 1:25){
      sigma[i,j]=0.75^{abs(i-j)}
    }
  }
  dat=data.frame(rmvnorm(n,sigma=sigma))
  names(dat)=paste("x",1:25,sep="")
  dat$u=0.5 + 0.3*abs(dat$x11+dat$x12+dat$x13+dat$x14+dat$x15)
  dat$truetime=rgamma(n,shape=dat$u,scale=2)
  dat$censor=alpha*runif(n)
  dat$y=apply(dat[,c("censor","truetime")],1,min)
  dat$status=as.numeric(dat$truetime<=dat$censor)
  dat
}
# function to compute the true survival function: DGP3
trueS3=function(u,vectime)
```

```
{
  # u = value from DGP3 function
  # vectime =  vector of points where to evaluate S
  pgamma(vectime,shape=u,scale=2,lower.tail = FALSE)
}
####################### DGP4
DGP4=function(n,alpha)
{
  # n = sample size
  # alpha = parameter for the censoring distribution
  sigma=diag(25)
  for(i in 1:25){
    for(j in 1:25){
      sigma[i,j]=0.9^{abs(i-j)}
    }
  }
  dat=data.frame(rmvnorm(n,sigma=sigma))
  names(dat)=paste("x",1:25,sep="")

dat$u=0.1*abs(dat$x11+dat$x12+dat$x13+dat$x14+dat$x15+dat$x16+dat$x17+dat$x18
+dat$x19+dat$x20)
  dat$truetime=rexp(n,1/dat$u)
  dat$censor=rexp(n,alpha/dat$u)
  dat$y=apply(dat[,c("censor","truetime")],1,min)
  dat$status=as.numeric(dat$truetime<=dat$censor)
  dat
}
# function to compute the true survival function: DGP4
trueS4=function(u,vectime)
{
  # u = value from DGP4 function
  # vectime =  vector of points where to evaluate S
  pexp(vectime, rate = 1/u, lower.tail = FALSE, log.p = FALSE)
}
```

References

[1] John D Kalbfleisch and Ross L Prentice, *The statistical analysis of failure time data*, vol. 360, John Wiley & Sons, 2011.

[2] David W Hosmer Jr and Stanley Lemeshow, *Applied survival analysis: regression modelling of time to event data*, Eur Orthodontic Soc, 1999, p.561-562.

[3] Therneau TM, *Survival analysis*, R package version 3.2-7, 2020. *https://cran.r-project.org/web/packages/survival/survival.pdf*

[4] F Bugnard, C Ducrot, D Calavas, *Advantages and inconveniences of the Cox model compared with the logistic model: application to a study of risk factors of nursing cow infertility*, Veterinary Research, BioMed Central, 1994, vol. 25 (2-3), p.134-139.

[5] Magdalena Babińska et al., *Limitations of Cox Proportional hazards analysis in Mortality prediction of patients with acute coronary syndrome*, Studied in logic, grammar and rhetoric, 2015, vol. 43 (56).

[6] William R. Swindell, *Accelerated failure time models provides a useful statistic framework for aging reseache*, Exp Gerontol, 2009, Mar, vol. 44 (3), p.190-200.

[7] Leo Breiman, *Random forests*, Machine learning, 2001, vol. 45 (1), p.5-32.

[8] Antonio Ciampi, Johanne Thiffault, Jean-Pierre Nakache, and Bernard Asselain, *Stratification by stepwise regression, correspondence analysis and recursive partition: a comparison of three methods of analysis for survival data with covariates*, Computational statistics & data analysis, 1986, vol. 4 (3) p.185-204.

[9] Noah Simon, Jerome Friedman, Trevor Hastie, and Rob Tibshirani, *Regularization paths for cox's proportional hazards model via coordinate descent*, Journal of statistical software, 2011, vol. 39 (5), p.1.

[10] Jerome Friedman et al., *Lasso and Elastic-Net Regularized Generalized Linear Models*, R package version 4.1, 2021. https://cran.r-project.org/web/packages/glmnet/glmnet.pdf

[11] Axel Benner, Manuela Zucknik, Thomas Hielscher, Carina Ittrich, Ulrich Mansmann, *High-dimensional Cox models: The choice of penalty as part of the model building process*, Biometrical journal, 2010, Feb, vol. 52(1), p.50-69.

[12] Yoav Freund, Robert E Schapire, et al., *Experiments with a new boosting algorithm*, Icml, 1996, vol.96, p.148-156.

[13] Denis Larocque, *Course notes (Version 3) of advanced statistical learning*, HEC Montreal, 2019.

[14] Leo Breiman, *Stacked regression*, Marchine learning, 1996b, vol. 24 (1), p.49-64.

[15] Michael LeBlanc and Robert Tibshirani, *Combining estimates in regression and classification*, Journal of the American Statistical Association, 1996, vol. 91 (436), p.1641-1650.

[16] Zachary A. Deane-Mayer and Jared E, Knowles, *Ensembles of Caret Models*, R package version 2.0.1, 2019. https://cran.r-project.org/web/packages/caretEnsemble/caretEnsemble.pdf

[17] Erin LeDell et al., *R Interface for the 'H2O' Scalable Machine Learning Platform*, R package version 3.32.0.1, 2020. https://cran.r-project.org/web/packages/h2o/h2o.pdf

[18] Andrew Wey, John Connett and Kyle Rudser, *Combining parametric, semi-parametric, and non-parametric survival models with stacked survival models*, Biostatistics, 2015, vol. 16 (3), p.537-549.

[19] Eric C. Polley, Mark J. van der Laan, *Super Learning for Right-Censored Data, Targeted Learning: Causal Inference for Observational and Experimental Data*, Springer Series in Statistics, 2011, Chapter 16, p.249-258.

[20] Andrew Wey, David M. Vock, John Connett, and Kyle Rudser, *Estimating restricted mean treatment effects with stacked survival models*, Stat Med, 2016, August 30, vol.35 (19), p.3319-3332.

[21] Marzieh K.Golmakani and Eric C.Polley, *Super Learner for Survival Data Prediction*, The International Journal of Biostatistics, 2020.

[22] Lorbert A, Ramadge P, *Decent methods for tuning parameter refinement*, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, p.469-476.

[23] Thomas A Gerds and Matin Schumacher, *Consistent estimation of the expected brier score in general survival models with right-censored event times*, Biometrical journal, 2006, 48 (6), p.1029-1040.

[24] Erika Graf, Claudia Schmoor, Willi Sauerbrei, and Martin Schumacher, *Assessment and comparison of prognostic classification schemes for survival data*, Statistics in medicine, 1999, vol. 18 (17-18), p.2529-2545.

[25] Hoora Moradian, Denis Larocque and François Bellavance, *L1 splitting rules in survival forests*, Lifetime Data Anal, 2017, p. 671-691.

[26] U.B.Mogensen, H.Ishwaran, and T.A. Gerds, *Evaluating random forests for survival analysis using prediction error curves*, Journal of statistical software, 2012, vol. 50 (11), p.1.

[27] Per Kragh Andersen and Maja Pohar Perme, *Pseudo-observations in survival analysis*, Statistical Methods in Medical Research, 2010, p.71-99.

[28] Maja Pohar Perme and Mette Gerster, *Computes Pseudo-Observations for Modeling*, R package version 1.4.3, 2017. https://cran.r-project.org/web/packages/pseudo/pseudo.pdf

[29] Torsten Hothorn et al., *A Toolkit for Recursive Partytioning*, R package version 1.2-12, 2021.
https://cran.r-project.org/web/packages/partykit/partykit.pdf

[30] S original by Berwin A. Turlach et al., *Functions to Solve Quadratic Programming Problems*, R package version 1.5-8, 2019.
https://cran.r-project.org/web/packages/quadprog/quadprog.pdf

[31] Kurt Hornik, *Cluster Ensembles*, R package version 0.3-58, 2020.
https://cran.r-project.org/web/packages/clue/clue.pdf

[32] Zhu R and Kosorok MR, *Recursively imputed survival trees*, J Am Stat Assoc, 2012, vol. 107 (497), p.331-340.

[33] Fleming TR, Harrington DP, *Counting processes and survival analysis*, Wiley, Hoboken, 1991.

[34] Bou-Hamad I, Larocque D, Ben-Ameur H, *A review of survival trees*, Stat Surv, 2011, vol. 5, p.44-71.

[35] Anthony Davison, *Practicals for Use with Davison (2003) Statistical Models*, R package version 1.4-3, 2018.
https://cran.r-project.org/web/packages/SMPracticals/SMPracticals.pdf

[36] Schlichting P, Christensen E, Andersen PK, Fauerholdt L, Juhl E, Poulsen H, Tygstrup N, *Prognostic factors in cirrhosis identified by Cox's regression model*, Hepatology, 1983, vol. 3 (6), p.889-895.

[37] Thomas Scheike with contributions from Torben Martinussen, Jeremy, *Flexible Regression Models for Survival Data*, R package version 1.9.8, 2020.
https://cran.r-project.org/web/packages/timereg/timereg.pdf

[38] Schumacher M, Bastert G, Bojar H, Huebner K, Olschewski M, Sauerbrei W, Schmoor C, Beyerle C, Neumann RL, Rauschecker HF, *Randomized 2 × 2 trial evaluating hormonal treatment and the duration of chemotherapy in node-positive*

breast cancer patients, German breast cancer study group, J Clin Oncol, 1994, vol. 12 (10), p.2086-2093.

[39] original by Gareth, modified by Axel Benner. Multivariable Fractional Polynomials. R package version 1.5.2, 2015. *https://cran.r-project.org/web/packages/mfp/mfp.pdf*

[40] Bache K, Lichman M, *UCI machine learning repository*, 2013. *http://archive.ics.uci.edu/ml*

[41] Torsten Hothorn. *TH's Data Archive*, R package version 1.0-10, 2019. *https://cran.r-project.org/web/packages/TH.data/TH.data.pdf*

[42] Kalbfleisch JD, Prentice RL, *The statistical analysis of failure time data*, Wiley series in probability and mathematical statistics, Wiley,1980.

[43] Hemant Ishwaran and Udaya B. Kogalur, *Fast Unified Random Forests for Survival, Regression, and Classification (RF-SRC)*, Rpackage version 2.10.1, 2021. *https://cran.r-project.org/web/packages/randomForestSRC/randomForestSRC.pdf*

[44] Breslow NE, Chatterjee N, *Design and analysis of two-phase studies with binary outcome applied to wilms tumour prognosis*, J R Stat Soc Ser C (Appl Stat), 1999, vol. 48 (4), p.457-468.