Implementing Optimal Survival Tree Algorithm

Bryan Zang[*] Linsi Zhong[†]

Abstract

As tree-based methods have become increasingly popular, an obvious question of interest would be about the comparison of such methods to more traditional regression models. In this paper we conduct a naive comparison between trees built from the `rfsrc` and `iai` packages with the results of a Cox proportional hazards model.

Introduction

With the progression in development of statistical models, it is evident that tree-based methods have become increasingly popular due to the ease of interpretation, high accuracy, and high stability in comparison to traditional models in the case of supervised learning. However, of the publicly available packages, there are relatively few that are suitable for analyzing survival data. To better understand the advantages and disadvantages of these methods, we will conduct a simple comparison between models resulting from three different methods: (1) the Cox proportional hazards model, (2) a survival tree built using Ishwaran et al.'s Random Survival Forest (RSF) algorithm[1], and (3) a survival tree built from the recently published Optimal Survival Tree (OST) algorithm[2].

Methodology

The Cox proportional hazards model is a well-known and commonly used regression model for survival analysis, it explains the hazard rate of an event as a linear combination of the effects of the covariates. Tree-based methods (also referred to by recursive partitioning techniques) on the other hand work off of an algorithm since they are nonparametric. The core idea is to repeatedly split the covariate space, based on some loss function, into small regions (nodes) containing homogeneous observations. The smallest nodes at the end of the tree are called leaves and the first node the tree begins with is called the root. This construction allows for easy-to-interpret structures and as well as the ability to explain complex, nonlinear relations between covariates. However, the training of trees based on traditional algorithms require complete observations of data, which is not guaranteed in survival analysis due to censoring. Hence there are various algorithms developed to bypass this issue, in this paper we will only consider trees constructed from the RSF algorithm and the OST algorithm. The main focus is on the OST algorithm so in-depth details relating to the RSF algorithm will be omitted. Note that we force the RSF algorithm to output a tree by setting the maximum amount of trees in the forest to be 1; this is done mostly due to the availability of accuracy metric computations, with packages such as `rpart` and `ctree` we found it difficult/impossible to obtain the same statistics as that of the `iai`[3] package.

[*]University of Waterloo, bszang@uwaterloo.ca
[†]University of Waterloo, l33zhong@uwaterloo.ca

[1]Ishwaran H. and M. (2008)
[2]Bertsimas D. and A. (2022)
[3]Bezanson J. and V. (2017)

Tree Construction

Tree algorithms in general consist of two main parts: (1) the splitting rule and (2) the pruning rule. The splitting rule selects the partitions to incorporate into the tree model; mainly we consider node-distance or node-purity splitting rules, where node-distance states the selection should maximize the difference between separate nodes and node-purity states the selection should group similar observations in a single node. Since under node-distance splitting rules it is not possible to assess singular nodes in isolation, the OST algorithm considers only node-purity splitting rules. Note that the Cox model's score function is in fact an example of a node-distance splitting rule. Once a splitting rule is established, the model must also state a pruning rule to dictate when to stop adding more partitions. Cost-complexity pruning is mainly used when considering node-purity splitting rules since cost-complexity seeks to select trees with minimum weighted combination of the total error and tree complexity. Here, total error is defined by the sum of each leaf node error and tree complexity is defined by the number of leaf nodes — the corresponding weights for these values are determined through cross validation.

Optimal Trees Algorithm

The OST algorithm extends the optimal trees algorithm also by Bertsimas and Dunn to accommodate for survival data. Traditionally, trees are constructed by a top-down, greedy manner, so each split is determined independently without considering the effect of the selection in regards to future splits — note that the RSF algorithm employs this type of method. This is obviously a main disadvantage since not every problem is convex and so a solution from a greedy algorithm is not guaranteed to be globally optimal. Issues such as poor generalization and faulty interpretations can arise from this drawback. In order to fix this problem, solutions such as random forests have been proposed. Random forests aggregate on the results of multiple trees, and although they can produce results with higher accuracy, the non-linearity can lead to black-box models with are un-interpretable. Bertsimas et al., proposes to replace this framework with a one-step approach — instead of a greedy optimization problem they instead consider a mixed-integer optimization (MIO) problem that is to be solved using coordinate descent. Each split in the tree is decided with full knowledge of all other splits and hence this new framework allows for preservation of interpretability and the scalability to larger datasets without resulting in computational complexity issues. At a high-level understanding, this framework (optimal tree algorithm) randomly visits each node k and repeats the following steps: - if k is not leaf, delete split at k - if k is not leaf, find optimal split to use at k and update current split - if k is leaf, create new split at k for each change induced, the objective value of the new tree is calculated with respect to the loss function

$$\min_T \texttt{error}(T, D) + \alpha \cdot \texttt{complexity}(T)$$

where T is the tree model, D is the training data, α is a tuning parameter that determines the cost-complexity trade-off (also called complexity parameter), `error` is a function measuring how well T fits D, and `complexity` is a function penalizing the complexity of T. When no more changes are done, the algorithm outputs the final model as a local optimum, let us denote this solution as τ. Since the problem of optimizing tree models is non-convex, this process is repeated with multiple, randomly-generated initial trees for which we will denote the series of solutions to be $\{\tau_i\}_{i=1}^m$ for n randomly-generated initial trees. The optimal tree algorithm will output the solution τ_j if the initial tree j results in a solution with the minimum objective value.

Optimal Survival Trees

Suppose we have survival data with observations $(t_i, \delta_i)_{i=1}^n$ where t_i denotes the time of last observation of i and δ_i is an indicator for i taking value of 1 if the last observation of i was a death and 0 if it was a censoring. The algorithm uses a splitting rule derived from a proportional hazards model, which recall assumes that the true survival distribution for each observation is explained by

$$P(S_i \leq t) = 1 - e^{-\theta_i \Lambda(t)}$$

where $\Lambda(t)$ is the baseline cumulative hazards function and θ_i is the adjustment to the baseline cumulative hazard for each observation i. Empirically, we estimate the cumulative hazard function with the Nelson-Aalen estimator

$$\hat{\Lambda}(t) = \sum_{i:t_i \leq t} \frac{\delta_i}{\sum_{j:t_j \geq t} 1}$$

and assuming that all the coefficients for observations in the same node are equal, then we have the within-leaf sample likelihood to be

$$L = \prod_{i=1}^{n} \left(\theta_i \frac{d}{dt} \Lambda(t_i) \right)^{\delta_i} e^{-\theta_i \Lambda(t_i)}$$

maximizing with respect to the coefficients θ_i, we obtain the maximum within-leaf sample likelihood estimates for node k to be

$$\hat{\theta}_k = \frac{\sum_i \delta_i \mathbf{1}_{\{T_i = k\}}}{\sum_i \hat{\Lambda}(t_i) \mathbf{1}_{\{T_i = k\}}}$$

To compare how well these estimates from the corresponding splits are, we first define a fully saturated tree to be a tree where every observation has a single coefficient, this means there is a unique parameter for each observation in this model; the coefficients for each node follows to be given by

$$\hat{\theta}_i^{\text{sat}} = \frac{\delta_i}{\hat{\Lambda}(t_i)}, \quad \text{for } i = 1, ..., n$$

The algorithm then computes the prediction error at each node as the difference between the log-likelihood of the current fitted node and that of the saturated model's coefficients, which can be found to be

$$\texttt{error}_k = \sum_{i:T(i)=k} \left(\delta_i \log \left(\frac{\delta_i}{\hat{\Lambda}(t_i)} \right) - \delta_i \log \hat{\theta}_k - \delta_i + \hat{\Lambda}(t_i) \cdot \hat{\theta}_k \right)$$

with this prediction error, the algorithm updates the overall error function to be the sum of the leaf-node errors

$$\texttt{error}(T, D) = \sum_k \texttt{error}_k(D)$$

which in turn replaces the overall loss function. Thus, it can be seen that the coordinate descent optimization process consists of repeatedly computing $\hat{\theta}_k$ using the maximum within-leaf sample likelihood estimates for which the within-leaf prediction error is calculated to continuously update the overall loss function in order to guide the descent towards a minimum.

Data Analysis

The dataset under examination is derived from a study on Monoclonal Gammopathy of Undetermined Significance (MGUS), encompassing the natural history records of 1,341 patients consecutively enrolled up to the year 1994, with follow-up data extending through 1999. This dataset is sourced from the `survival` package in `R`, representing a comprehensive cohort aimed at investigating the progression and outcomes associated with MGUS. The primary outcome of interest is the time to death, with the analysis incorporating several critical covariates such as age at diagnosis, sex, and the size of the monoclonal serum spike. This is a long-term study since the time until death or last contact varies from 1 month to 424 months and it involves 631 females, which is about half of the subjects. Also, age at diagnosis of subjects varies from 24 to 96 years old, with a mean of 70.42 years.

```r
head(mgus2)
```

```
##   id age sex dxyr  hgb creat mspike ptime pstat futime death
## 1  1  88   F 1981 13.1   1.3    0.5    30     0     30     1
## 2  2  78   F 1968 11.5   1.2    2.0    25     0     25     1
## 3  3  94   M 1980 10.5   1.5    2.6    46     0     46     1
## 4  4  68   M 1977 15.2   1.2    1.2    92     0     92     1
## 5  5  90   F 1973 10.7   0.8    1.0     8     0      8     1
## 6  6  90   M 1990 12.9   1.0    0.5     4     0      4     1
```

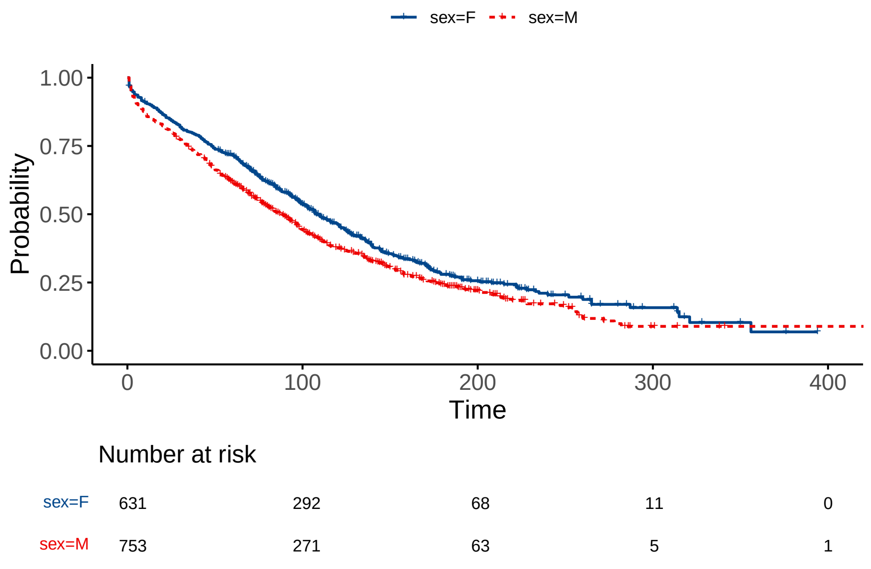
```
sum(is.na(mgus2))
```

```
## [1] 54
```

there are 54 incomplete entries which we will omit in our analysis. We can construct a life table for time points 10-50 jumping by 10 to get

```
## Call: survfit(formula = survival_object ~ 1, data = mgus2)
##
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##     0   1384       0    1.000 0.00000        1.000        1.000
##    10   1235     158    0.886 0.00855        0.869        0.903
##    20   1173      60    0.842 0.00980        0.823        0.862
##    30   1104      67    0.794 0.01088        0.773        0.815
##    40   1040      60    0.750 0.01164        0.728        0.774
##    50    966      74    0.697 0.01237        0.673        0.721
```

from which we can say that the decline in survival rate seems relatively even and that there are no sudden gaps at least in the first 50 time points. In addition, we notice there is a sex variable, and so an intuitive idea is to see if there is a significant difference between the data for males and females; plotting two Kaplan-Meier curves, we have



both sexes follow a similar trend and do not differ in value by a lot, however it is the case that females generally have a slightly higher probability of event at each time point. Specific to the following models, we do not incorporate the variables `ptime` and `pstat` and note the variables of interest is defined by the variables `futime`, denoting the time until death/last contact, and

Model Fitting

We randomly split the data using the `iai::split_data` function into a 70:30 split where 70% of the data is used for training and the remaining 30% is used for testing. Now, the output of this function is suited for training the OST with `iai::optimal_tree_survival_learner` but it is not suitable for training the RSF tree nor the Cox model; so we manipulate the data to obtain

```r
head(train.df)
```

```
##   time status age sex dxyr  hgb creat mspike
## 1   92   TRUE  68   M 1977 15.2   1.2    1.2
## 2    4   TRUE  90   M 1990 12.9   1.0    0.5
## 3  151   TRUE  89   F 1974 10.5   0.9    1.3
## 4    2   TRUE  87   F 1974 12.3   1.2    1.6
## 5   57  FALSE  86   F 1994 14.5   0.9    2.4
## 6  136   TRUE  79   F 1981  9.4   1.1    2.3
```

```r
head(test.df)
```

```
##   time status age sex dxyr  hgb creat mspike
## 1   30   TRUE  88   F 1981 13.1   1.3    0.5
## 2   25   TRUE  78   F 1968 11.5   1.2    2.0
## 3   46   TRUE  94   M 1980 10.5   1.5    2.6
## 4    8   TRUE  90   F 1973 10.7   0.8    1.0
## 5    2   TRUE  86   M 1972 11.8   1.0    2.3
## 6   14   TRUE  80   F 1985 13.1   1.0    1.3
```

then fitting the Cox model on both sets we have

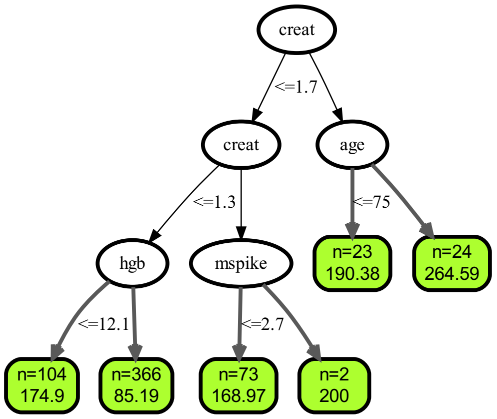
```
## Call:
## coxph(formula = Surv(time, status) ~ ., data = train.df)
##
##   n= 937, number of events= 663
##
##             coef exp(coef)  se(coef)      z Pr(>|z|)
## age     0.053993  1.055477  0.004060 13.300  < 2e-16 ***
## sexM    0.413555  1.512184  0.081023  5.104 3.32e-07 ***
## dxyr   -0.001417  0.998584  0.007012 -0.202    0.840
## hgb    -0.124954  0.882538  0.020903 -5.978 2.26e-09 ***
## creat   0.028903  1.029324  0.021991  1.314    0.189
## mspike -0.022054  0.978187  0.074665 -0.295    0.768
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##        exp(coef) exp(-coef) lower .95 upper .95
## age       1.0555     0.9474    1.0471    1.0639
## sexM      1.5122     0.6613    1.2901    1.7724
## dxyr      0.9986     1.0014    0.9850    1.0124
## hgb       0.8825     1.1331    0.8471    0.9194
## creat     1.0293     0.9715    0.9859    1.0747
## mspike    0.9782     1.0223    0.8450    1.1323
##
```

```
## Concordance= 0.684  (se = 0.011 )
## Likelihood ratio test= 293.4  on 6 df,   p=<2e-16
## Wald test            = 273.2  on 6 df,   p=<2e-16
## Score (logrank) test = 276.7  on 6 df,   p=<2e-16
```

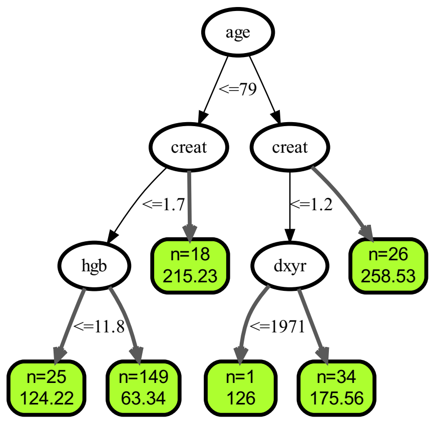
```
## Call:
## coxph(formula = Surv(time, status) ~ ., data = test.df)
##
##   n= 401, number of events= 275
##
##              coef  exp(coef)   se(coef)      z Pr(>|z|)
## age     0.0618157  1.0637663  0.0067407  9.171  < 2e-16 ***
## sexM    0.4765634  1.6105301  0.1333594  3.574 0.000352 ***
## dxyr    0.0000652  1.0000652  0.0115061  0.006 0.995479
## hgb    -0.1263165  0.8813359  0.0379628 -3.327 0.000877 ***
## creat   0.2153625  1.2403115  0.0552401  3.899 9.67e-05 ***
## mspike  0.1593153  1.1727076  0.1118157  1.425 0.154214
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##        exp(coef) exp(-coef) lower .95 upper .95
## age       1.0638     0.9401    1.0498    1.0779
## sexM      1.6105     0.6209    1.2401    2.0916
## dxyr      1.0001     0.9999    0.9778    1.0229
## hgb       0.8813     1.1346    0.8181    0.9494
## creat     1.2403     0.8062    1.1130    1.3821
## mspike    1.1727     0.8527    0.9419    1.4600
##
## Concordance= 0.711  (se = 0.017 )
## Likelihood ratio test= 157.5  on 6 df,   p=<2e-16
## Wald test            = 153.4  on 6 df,   p=<2e-16
## Score (logrank) test = 163.9  on 6 df,   p=<2e-16
```

and as well as the RSF tree

```
##                          Sample size: 937
##                     Number of deaths: 663
##                      Number of trees: 1
##            Forest terminal node size: 15
##        Average no. of terminal nodes: 6
## No. of variables tried at each split: 3
##               Total no. of variables: 6
##        Resampling used to grow trees: swor
##     Resample size used to grow trees: 592
##                             Analysis: RSF
##                               Family: surv
##                       Splitting rule: logrank *random*
##        Number of random split points: 10
##                           (OOB) CRPS: 0.14247195
##    (OOB) Requested performance error: 0.40324684
```



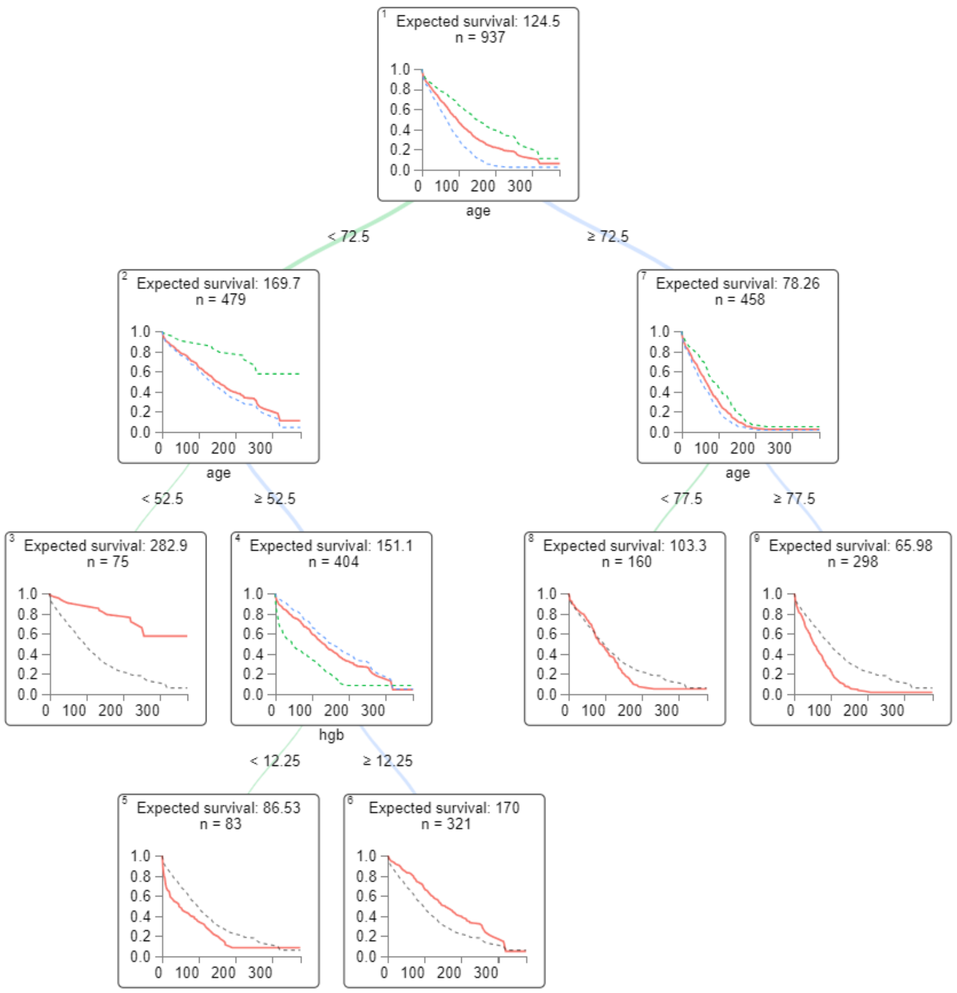
```
##                          Sample size: 401
##                     Number of deaths: 275
##                      Number of trees: 1
##            Forest terminal node size: 15
##        Average no. of terminal nodes: 6
## No. of variables tried at each split: 3
##               Total no. of variables: 6
##        Resampling used to grow trees: swor
##     Resample size used to grow trees: 253
##                             Analysis: RSF
##                               Family: surv
##                       Splitting rule: logrank *random*
##        Number of random split points: 10
##                         (OOB) CRPS: 0.1365458
##    (OOB) Requested performance error: 0.31276115
```



and lastly the OST

```
## Julia Object of type OptimalTrees.OptimalTreeSurvivalLearner.
## Fitted OptimalTreeSurvivalLearner:
##   1) Split: age < 72.5
##     2) Split: age < 52.5
##       3) Expected survival: 282.9, 75 points, error 0.5555
##       4) Split: hgb < 12.25
##         5) Expected survival: 86.53, 83 points, error 0.8506
##         6) Expected survival: 170, 321 points, error 0.6392
##     7) Split: age < 77.5
##       8) Expected survival: 103.3, 160 points, error 0.5076
##       9) Expected survival: 65.98, 298 points, error 0.4782
```

```
##   max_depth          cp train_score valid_score rank_valid_score
## 1         1 0.054381689   0.1087634   0.6205184                3
## 2         2 0.012588035   0.1568805   0.6557620                2
## 3         3 0.008430298   0.2016222   0.6835868                1
```



We can thus obtain the following concordance statistics

```
##              training testing
## ost.c.stat     0.6700  0.6673
## tree.c.stat    0.5968  0.6872
## cox.c.stat     0.6837  0.7114
```

OST performs better than the RSF tree however the Cox model outperforms both tree models by a small amount. Then taking a look at the dynamic AUC values

```
##   ost.auc tree.auc   cox.auc
##    0.7402   0.3794    0.7468
```

we see that again the results from the Cox model perform the best and that the OST results are better than that of the RSF tree. We note that the dynamic AUC for the RSF tree model is particularly low, this may be because of how the AUC values are calculated in the `survAUC` package.

Conclusion

We confirm the results of Bertsimas et al. in which the model yielding from the OST algorithm indeed is optimal relative to another tree-based model. In particular we compared the OST to a tree constructed from the traditional top-down, greedy approach; we see not only the better performance in the corresponding Harrell's C statistics but also complete domination in terms of dynamic AUC values. Overall we conclude that the Cox model displays the best performance, followed by the OST, and then lastly the RSF tree model. However, it is worth noting that this comparative analysis is very limited and naive in nature since we only considered one dataset and one construction of each model. A better comprehensive study would be to conduct the analysis on multiple datasets, using multiple constructions of the models, and possibly even including accuracy metrics from other packages. This problem was addressed earlier where we noted the difficulty in obtaining certain metrics from packages such as `rpart` and `ctree`. In addition, some other problems encountered include the discrepancy in value-naming, specifically, the `iai` package for the OST model abuses terminology and claims certain function outputs to be something it is not. For example, the official documentation states that for some time `t`, the command

```r
pred_curves <- iai::predict(grid, test_X)
sapply(pred_curves, `[`, t)
```

should query all the survival probabilities for time `t`, however it was found to be instead the empirical CDF values. This type of issue occurred more than once throughout the course of this project, hence why only two accuracy metrics are considered in this paper; so it is obvious more time and more resources would be needed in order to conduct a well-structured comprehensive analysis.

Acknowledgements

We would like to thank Professor Diao Liqun for the amazing job she has done over this past semester, we have learnt a lot and found the course to be very fun and informational.

References

Bertsimas D., Gibson E., Dunn J., and Orfanoudaki A. 2022. "Optimal Survival Trees." *Machine Learning* 111: 2951–3023.

Bezanson J., Karpinski S., Edelman A., and Shah V. 2017. "Julia: A Fresh Approach to Numerical Computing." *SIAM Review* 59 (1): 65–98.

Ishwaran H., Blackstone E., Kogalur U., and Lauer M. 2008. "Random Survival Forests." *Annals of Applied Statistics* 2 (3): 841–60.