Survival Stacking: Implementation and Assessment with Heart Failure Data

Hanning Su, Hanzhang Zhao, Ruochong Fan

May 2023

Abstract

Survival stacking is a technique that casts survival analysis problems into classification tasks. We implement this data reorganization technique and demonstrate, with a heart failure survival time dataset, that a binary classifier (Boosting, Random Forest(RF) or Artificial Neural Net(ANN)) trained on stacked data brings us interpretable results and better discriminatory power.

1 Introduction

Survival stacking (Craig et al. 2021) is a data reorganization technique that enables us to apply machine learning algorithms designed for classifications to conduct survival analysis. This project covers some theoretical material illustrating how machine learning on stacked data relates to traditional statistical models, including the Kaplan-Meier estimator, Cox Proportional Hazard regression and Logistic regression. Then, with a heart failure dataset (Ahmad et al. 2017), we demonstrate that learning algorithms (including Boosting, RF and ANN) applied on stacked data, similar to Cox regression, can produce interpretable results, such as variable importance scores. In addition, we compare their discriminatory performance with an adapted Harrell's C metric.

2 Simple Example of Survival Stacking

$$\begin{array}{cccc} \tilde{T} & \Delta & Z_1 & Z_2 \\ \end{array} \quad \begin{array}{ccccc} \delta & I_1 & I_2 & Z_1 & Z_2 \end{array}$$

$$\begin{bmatrix} 1 & 1 & 21 & 2 \\ 2 & 0 & 16 & 5 \\ 3 & 1 & 19 & 3 \\ 3 & 1 & 25 & 4 \end{bmatrix} \xRightarrow[\text{stack}]{\text{survival}} \begin{bmatrix} 1 & 1 & 0 & 21 & 2 \\ 0 & 1 & 0 & 16 & 5 \\ 0 & 1 & 0 & 19 & 3 \\ 0 & 1 & 0 & 25 & 4 \\ 1 & 0 & 1 & 19 & 3 \\ 1 & 0 & 1 & 25 & 4 \end{bmatrix}$$

This toy example (with right censoring and ties) is representative of the dataset we adopt. Given our initially collected data of form $\{\tilde{T}_i, \Delta_i, \mathbf{Z}_i\}_{i=1}^4$. We have two at-risk sets, $R(t_1 = 1) = \{1, 2, 3, 4\}$ and $R(t_2 = 3) = \{3, 4\}$, with I_1 and I_2 being indicators respectively. Within each at-risk set, we let event indicator $\delta = 1$ if a given member experience event at the event time corresponding to the at-risk set. Stacking row-wise of every member of every at-risk set we have the stacked dataset on the right hand side.

3 Model Assumption

3.1 Modeling Assumptions for Survival Stacking

We assume our response (equivalently, outcome, event indicator, or status) for subject i to be distributed as:

$$\delta_i | I_{i1}, I_{i2}, ..., I_{iK}, \mathbf{Z}_i \sim \text{Bernoulli}(h(I_{i1}, I_{i2}, ..., I_{iK}, \mathbf{Z}_i))$$

Here $\mathbf{Z}_i$ is the covariate vector representing characteristics such as gender, age and other characteristics of subject i. We let $I_{ij} = 1$ if the subject belongs to at risk set j and let $I_{ij} = 0$ otherwise. In the stacked data, for each observation, exactly one indicator equals 1 and we treat entries of a subject in different at risk sets as independent observations. K is the number of distinct event times $t_1, t_2, ...$

We recall that for survival time random variable T discrete hazard at event time t_j is defined as

$$h_j = P(T = t_j | T \geqslant t_j) = \frac{P(T = t_j)}{P(T \geqslant t_j)}$$

Now, if we let $h_{i1} = h(I_{i1} = 1, h(I_{i2} = 0, ..., h(I_{iK} = 0), \mathbf{Z}_i)), h_{i2} = h(I_{i1} = 0, h(I_{i2} = 1, ..., h(I_{iK} = 0), \mathbf{Z}_i)), ...$ Then $h_{i1}, h_{i2}, ..., h_{iK}$ represent discrete hazards for subject i at the event times $t_1, t_2, ...$ Therefore the survival curve for subject i provided by our model is:

$$S(t|\mathbf{Z}_i) = \sum_{j:t_j \leqslant t} (1 - h_{ij})$$

, which has a similar form as a Kaplan Meier product limit estimator.

3.2 Bridging from Kaplan Meier Estimator to Machine Learning

If we assume $h(I_{i1}, I_{i2}, ..., I_{iK}, \mathbf{Z}_i) = \alpha_{t_1} I_{i1} + \alpha_{t_2} I_{i2} + ... + \alpha_{t_K} I_{iK} := \alpha^T \mathbf{I}_i$ and estimate α with MLE, we shall have $\hat{\alpha}_{t_j} = \frac{d_j}{n_j}$, where d_j is the number of events in at risk set j and n_j is the number of individuals in at risk set j, giving rise to $\hat{S}(t|\mathbf{Z}_i)$ coinciding with the Kaplan Meier estimator. To utilize information in $\mathbf{Z}_i$, we can let $h(I_{i1}, I_{i2}, ..., I_{iK}, \mathbf{Z}_i) = \frac{\exp(\alpha^T \mathbf{I}_i + \beta_1 Z_{i1} + \beta_2 Z_{i2} + ... + \beta_p Z_{ip})}{1 + \exp(\alpha^T \mathbf{I}_i + \hat{\beta}^T \mathbf{Z}_i)}$ and estimate parameters with MLE, then we are performing a Logistic regression that closely resembles the Cox proportional-hazard regression with $\mathbf{Z}_i$ as covariate vector. Mathematically, we can show approximately equality between baseline hazard and parameter estimates of β. Statistical inferences results largely coincide as well.

So far, we are still in the realm of statistics. What if we want to go beyond logistic regression and do not want to assume a specific form for h? We can find solutions in machine learning. Equipped with several well-developed machine learning algorithms, we can be totally flexible with the form of h by letting

$$h(I_{i1}, I_{i2}, ..., I_{iK}, Z_{i1}, ..., Z_{ip}) = f(I_{i1}, ..., I_{iK}, Z_{i1}, ..., Z_{ip})$$

, for some arbitrary function f taking values in $[0, 1]$, and obtain $\hat{f}$ by training a binary classifier for δ with binary cross entropy loss.

4 Dataset Introduction

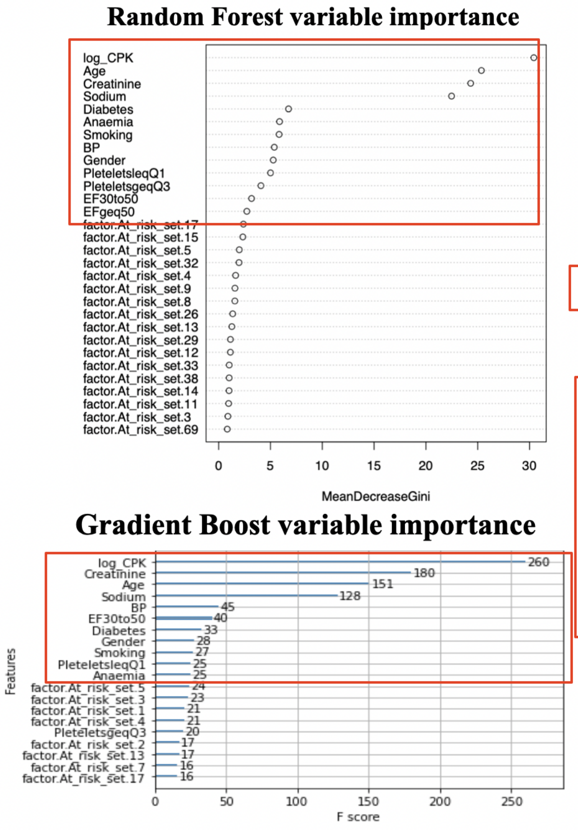
We adopt the original dataset and the Cox model built in the paper by Craig et al. (2021) The data originated from heart failure patients admitted to Institute of Cardiology and Allied hospital Faisalabad-Pakistan between April and December, 2015. According to the paper, we applied the same variable transformation, including dividing variables to multiple levels and giving logarithm to the CPK variable.

5 Variable Importance Scores and Baseline Survival Curves

5.1 Variable Importance Scores

We want to compare variable importance between survival methods (Cox proportional hazard model), logistic regression, and machine learning methods. We look at p-values for regression-based models since they describe the significance of variables' marginal impacts on the survival experience. We then apply feature selection for random forest and gradient boost to learn about the contributions of variables to the reduction in Gini impurity and increase in accuracy, respectively.

We can see that Random Forest and Boosting have very similar feature importance scores, where variables log(CPK), Creatinine, Age, and Sodium have a much larger value than other variables. We can also



	Dependent variable:	
	TIME	Event_indicator
	Cox prop. hazards	*logistic*
	(1)	(2)
Gender	-0.143 (0.249)	-0.142 (0.251)
Smoking	0.099 (0.250)	0.092 (0.252)
Diabetes	0.180 (0.223)	0.178 (0.224)
BP	0.473** (0.218)	0.483** (0.221)
Anaemia	0.224 (0.219)	0.230 (0.221)
Age	0.049*** (0.009)	0.050*** (0.010)
EF30to50	-1.108*** (0.238)	-1.131*** (0.240)
EFgeq50	-1.080*** (0.319)	-1.101*** (0.322)
Sodium	-0.053** (0.024)	-0.054** (0.024)
Creatinine	0.257*** (0.071)	0.261*** (0.072)
log_CPK	0.093 (0.101)	0.097 (0.102)
PleteletsleqQ1	0.235 (0.257)	0.243 (0.259)
PleteletsgeqQ3	0.228 (0.265)	0.230 (0.268)

Figure 1: Variable Importance

see from both Cox PH and logistic regression that variables BP, Age, EF30to50, EFgeq50, Sodium, and Creatinine have significant effects on the survival experience. The two sets of variables determined to be important/significant have a considerable overlap.

5.2 Baseline Survival Curve

One can derive the baseline survival curve from machine learning models directly by predicting $h_2, h_2, \ldots$ given $\mathbf{Z} = \mathbf{0}$. We do not need additional assumptions for this derivation. In contrast, we need to assume a baseline hazard function (e.g. Breslow baseline) in Cox regression.

Figure 2 shows the cumulative baseline survival probabilities evaluated at $t \in [0, 250]$, by different methods, on the y-axis. We can see that all curves are showing a similar trend. The survival probabilities drop fast as the x variable (time) increases. Gradient boost's survival curve is showing exceptional similarity to the Cox proportional hazard's curve. In summary, we believe machine learning algorithms can provide some reasonable inference on variable importance, and we can plot survival curves with them without too much difficulty.

6 Methods Comparison with Adapted Harrell's C

In order to compare the various methods we adopt, we divide our data set into a training set (80% of 299 subjects) and a testing set (20% of 299 subjects). We run each of those methods on the training set and calculate Harrell's C at median and 75% quantile of the training set event times.

The Harrell's C index is a goodness of fit measure for models which produces risk scores. The risk score, in survival stacking context, is $-\hat{S}_i(t)$, negative of subject i's estimated survival curve value at t. A generic definition of Harrell's C is $C = \frac{\#\text{ concordant pairs}}{\#\text{ concordant pairs} + \#\text{ discordant pairs}}$. Intuitively, concordance occurs when higher risk individuals, with $-\hat{S}_i(t)$ higher, has shorter survival time; meanwhile, discordance occurs when higher

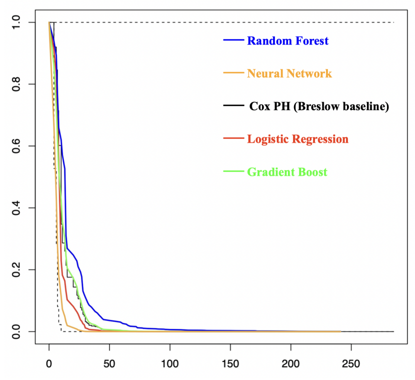


Figure 2: Baseline survival curve

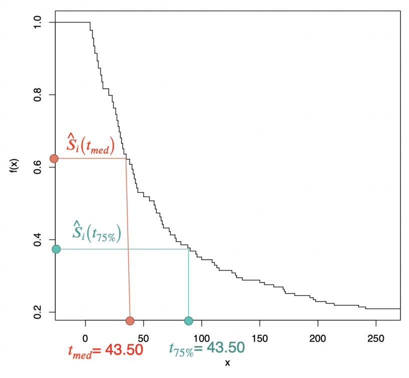


Figure 3: $\hat{S}(t_{\text{med}}), \hat{S}(t_{75\%})$'s calculation illustration

risk individual has longer survival time. According to this intuition, we define our adapted Harrell's C index as:

$$C(t) = \frac{\sum_{i \neq j} \mathbb{1}\{\hat{S}_i(t) > \hat{S}_j(t)\}\mathbb{1}\{\tilde{T}_i > \tilde{T}_j\}\Delta_j}{\sum_{i \neq j} \mathbb{1}\{\tilde{T}_i > \tilde{T}_j\}\Delta_j}$$

We evaluate C at $t = t_{\text{med}} :=$ median of training set event times and at $t = t_{75\%} :=$ upper quartile of training set event times. We remark that $\Delta_j = 1$ if $\tilde{T}_j$ is event time (subject j experiences event) and $\Delta_j = 0$ if $\tilde{T}_j$ is censoring time.
To illustrate how we calculate $\hat{S}_i(t_{\text{med}})$ and $\hat{S}_i(t_{75\%})$, take ANN fit as an example. We estimate and store the survival curve as a step function object in R and use it to evaluate $\hat{S}_i(t)$ at different t's. An illustration is shown in Figure 3.

	C(t_{med})	C($t_{75\%}$)
Cox PH	0.607	0.607
Logistic	0.606	0.606
Random Forest	0.667	0.646
Boosting	0.639	0.639
ANN	0.652	0.653

Figure 4: Harrell's C / Time dependent AUC

Neural network is the overall top performer but random forest is the top performer at t_{med}

7 Conclusion

If we are interested in ranking the risks of in a group of individuals, we have demonstrated with Harrell's C that machine learning techniques can outperform Cox PH regression by a sizable amount. Some machine learning methods (even neural nets in recent years (Liang et al. 2018)) can produce variable importance scores, providing some level of interpretability.

For survival data sets that contain a small number of event times, positive cases (outcome = 1) will be extremely rare (0.00685 positive-negative ratio in our case) after stacking. This can be an issue for some ML algorithms, for instance, our empirical experience shows that neural nets can be very unstable (it can produce a 0.3 to 0.5 Harrell's C if we attempt to regularize with batch normalization).

If we want to use machine learning techniques to generate survival curves in practice such as in an insurance company, care must be taken. the survival curves they produce for the same individual can be very different, e.g. baseline survival curves in Figure 2. Deeper knowledge about machine learning algorithms or the development of ad hoc techniques is needed before we can actually choose and utilize the survival curves they produce.

References

Ahmad, Tanvir, et al. ”Survival analysis of heart failure patients: A case study.” PloS one 12.7 (2017): e0181001.

Craig, Erin, Chenyang Zhong, and Robert Tibshirani. ”Survival stacking: casting survival analysis as a classification problem.” arXiv preprint arXiv:2107.13480 (2021).

Liang, Faming, Qizhai Li, and Lei Zhou. ”Bayesian neural networks for selection of drug sensitive genes.” Journal of the American Statistical Association 113.523 (2018): 955-972.

Survival Stacking Implementation

Hanning Su, Hanzhang Zhao, Ruochong Fan
Student ID: hs3375, hz2832, rf2826

May 3, 2023

Defining stack function and simulation example

```
## No longer assuming no ties
survival_stack <- function(Dataset, Observed_time, Status, Truncation = TRUE,
                           L) { #Status must be a binary vector
  ##
  #Dataset must be sorted before provided as argument
  ##
  New_dataset <- NULL
  at_risk_index <- 0
  if (Truncation == FALSE) {
    L = rep(0, dim(Dataset)[1])
  }

  # Binary outcome, indicating which individual expereienced event in a given at risk
  y_R <- NULL

  # Unique event times
  unique_event_time <- unique(Observed_time[Status == 1])

  for (i in 1:dim(Dataset)[1]) {
    if (Status[i] == 1 & Observed_time[i] %in% unique_event_time) { # if we have a even
      at_risk_index <- at_risk_index + 1 # we define a at risk set with this index
      for (j in 1:dim(Dataset)[1]) { # then go through entire dataset again
        if (Observed_time[i] > L[j] & Observed_time[j] >= Observed_time[i]) {
          # geting at risk set at
          # the given event time
          if (Observed_time[j] == Observed_time[i] & Status[j] == 1) {
            New_dataset <- rbind(New_dataset, c(c(t(Dataset[j,])),
                                                at_risk_index))
            y_R<- c(y_R, 1)
          } else{
             New_dataset <- rbind(New_dataset, c(c(t(Dataset[j,])),
                                                 at_risk_index))
            y_R <- c(y_R, 0)
          }
        }
      }
      unique_event_time <- unique_event_time[ !unique_event_time == Observed_time[i]]
      #print(unique_event_time)
    }
  }
  return(cbind(New_dataset, y_R))
}

## Example in paper
X <- data.frame("Observed_time" = c(5, 1, 1, 2, 3, 3, 5),
                "Status" = c(0, 1, 1,  0, 1, 0, 1),
                "C1" = c(123, 7, 100,  8, 9, 10, 111),
                "C2" = c(456, 9, 0.5, 10, 5, 3, 222))

X <- X[order(X$Observed_time),]

X_star <- survival_stack(Dataset=X, Observed_time = X$Observed_time,
              Status = X$Status,
               Truncation =FALSE)

X_star <- data.frame(X_star)

names(X_star) <- c('Observed_time','Status','C1','C2','At_risk_set', 'Event_indicator')

# Wroks correctly # Ties handled as well
X_star

##    Observed_time Status  C1    C2 At_risk_set Event_indicator
## 1              1      1   7   9.0           1               1
## 2              1      1 100   0.5           1               1
## 3              2      0   8  10.0           1               0
## 4              3      1   9   5.0           1               0
## 5              3      0  10   3.0           1               0
## 6              5      0 123 456.0           1               0
## 7              5      1 111 222.0           1               0
## 8              3      1   9   5.0           2               1
## 9              3      0  10   3.0           2               0
## 10             5      0 123 456.0           2               0
## 11             5      1 111 222.0           2               0
## 12             5      0 123 456.0           3               0
## 13              5      1 111 222.0          3                1
## Heart data https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5519051/
Heart <- read.csv("Heart.csv")


Heart <- Heart[order(Heart$TIME),]

## Divide Ejection Fraction and platelets into levels log transform
Heart$EF30to50 <- rep(0, dim(Heart)[1])
Heart$EFgeq50 <- rep(0, dim(Heart)[1])
Heart$log_CPK<- rep(0, dim(Heart)[1])
Heart$PleteletsleqQ1<- rep(0, dim(Heart)[1])
Heart$PleteletsgeqQ3<- rep(0, dim(Heart)[1])


for (i in 1:dim(Heart)[1]){
  if (30 < Heart$Ejection.Fraction[i] & Heart$Ejection.Fraction[i] <= 45) {
    Heart$EF30to50[i] = 1
  } else if (Heart$Ejection.Fraction[i] > 45) {
    Heart$EFgeq50[i] = 1
  }

  Heart$log_CPK[i] <- log(Heart$CPK[i])

  if (Heart$Pletelets[i] <= quantile(Heart$Pletelets)[2]){
    Heart$PleteletsleqQ1[i] = 1
  } else if (Heart$Pletelets[i] >= quantile(Heart$Pletelets)[4])
    Heart$PleteletsgeqQ3[i] = 1
}

# Remove original Ejection.Fraction and pletelets and CPK
Heart <- Heart[,-c(9, 12, 13)]

#write.csv(Heart, "Heart_cox.csv")

Heart1 <- survival_stack(Dataset=Heart, Status =Heart$Event,
                              Observed_time =Heart$TIME,
               Truncation =FALSE)


Heart2 <- data.frame(Heart1)

names(Heart2) <- c("TIME", "Event", "Gender", "Smoking", "Diabetes", "BP",
                   "Anaemia", "Age", "Sodium", "Creatinine", "EF30to50",
                   "EFgeq50", "log_CPK", "PleteletsleqQ1", "PleteletsgeqQ3",
                   "At_risk_set", "Event_indicator")

Heart3 <- Heart2[,-c(1, 2)]
#write.csv(Heart3, "Heart_atrisk.csv")


matrix <- model.matrix(~ factor(At_risk_set) + Gender + Smoking + Diabetes + BP +
                   Anaemia + Age + Sodium + Creatinine + EF30to50 +
                   EFgeq50 + log_CPK + PleteletsleqQ1 + PleteletsgeqQ3 - 1
                  , Heart3 )


Heart4 <- data.frame(matrix)
Heart4$Event_indicator <- Heart3$Event_indicator
#write.csv(Heart4, "Heart_at_risk_dummy.csv")
```

Preparing Training set and Test set for calculating Harrell's C

Note: We variable importance/significance and survival curve plotting portion of the project, we use entire Heart data. In addition, we use a validation set to select neural network structure but eventually we train neural net with the entire dataset.
Note: For calculating Harrel's C, every ML methods get trained on the test set and perform Harrell's C calculation with the test set.

```
## Randomly select 80% of subjects from Heart as Training set
# Attaching the index column
Heart$index <- as.numeric(row.names(Heart))

set.seed(3375)
test_indices <- sample(c(1:dim(Heart)[1]), floor(dim(Heart)[1]*0.2))
Train_set <- Heart[-test_indices, ]
Test_set <- Heart[test_indices, ]

#write.csv(Train_set, "Train_set_cox.csv")
#write.csv(Test_set, "Test_set.csv") # we only need this so call it THE test set

# Storing training and testing set index
Train_set_index <- Train_set$index
Test_set_index <- Test_set$index

# Stacking indexed Heart data
Heart_ind1 <- survival_stack(Dataset=Heart, Status =Heart$Event,
```

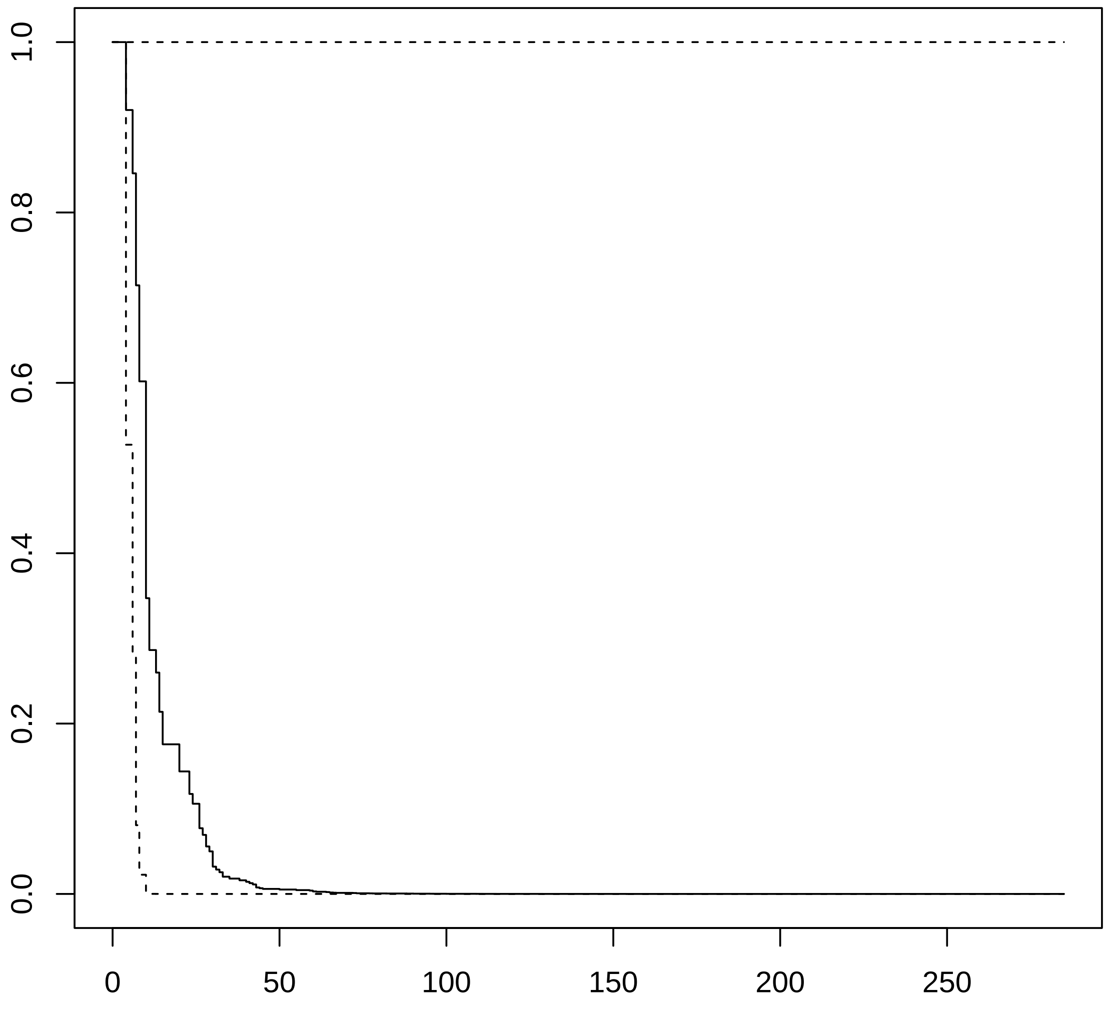
```r
                              Observed_time =Heart$TIME,
               Truncation =FALSE)

Heart_ind2 <- data.frame(Heart_ind1)

names(Heart_ind2) <- c("TIME", "Event", "Gender", "Smoking", "Diabetes", "BP",
                   "Anaemia", "Age", "Sodium", "Creatinine", "EF30to50",
                   "EFgeq50", "log_CPK", "PleteletsleqQ1", "PleteletsgeqQ3",
                   "index", "At_risk_set", "Event_indicator")


Heart_ind3 <-Heart_ind2[,-2]

Train_set3 <- Heart_ind3[Heart_ind3$index %in% Train_set_index,]
#Test_set3 <- Heart_ind3[Heart_ind3$index %in% Test_set_index,]

#write.csv(Train_set3, "Train_set_atrisk.csv")
#write.csv(Test_set3, "Test_set_atrisk.csv") # we do not need this !!

matrix <- model.matrix(~ TIME + factor(At_risk_set) + Gender + Smoking + Diabetes + BP +
                   Anaemia + Age + Sodium + Creatinine + EF30to50 +
                   EFgeq50 + log_CPK + PleteletsleqQ1 + PleteletsgeqQ3 +
                   index - 1
                  , Heart_ind3)

Heart_ind4 <- data.frame(matrix)
Heart_ind4$Event_indicator <- Heart_ind3$Event_indicator

Train_set4 <- Heart_ind4[Heart_ind4$index %in% Train_set_index,]
#Test_set4 <- Heart_ind4[Heart_ind4$index %in% Test_set_index,]

#write.csv(Train_set4, "Train_set_atrisk_dummy.csv")
#write.csv(Test_set4, "Test_set_atrisk_dummy.csv") # we do not need this !!
```

Cox fit

```r
Heart_cox <- read.csv("Heart_cox.csv")
library(survival)
cox_fit <- coxph(Surv(TIME, Event)~Gender + Smoking + Diabetes + BP + Anaemia +
                   Age + EF30to50 +  EFgeq50 + Sodium + Creatinine + log_CPK +
                   PleteletsleqQ1 + PleteletsgeqQ3, data=Heart_cox)
```

```
summary(cox_fit)

## Call:
## coxph(formula = Surv(TIME, Event) ~ Gender + Smoking + Diabetes +
##     BP + Anaemia + Age + EF30to50 + EFgeq50 + Sodium + Creatinine +
##     log_CPK + PleteletsleqQ1 + PleteletsgeqQ3, data = Heart_cox)
##
##   n= 299, number of events= 96
##
##                      coef exp(coef)  se(coef)      z Pr(>|z|)
## Gender          -0.143427  0.866384  0.249313 -0.575 0.565096
## Smoking          0.098519  1.103536  0.250074  0.394 0.693610
## Diabetes         0.179882  1.197076  0.222877  0.807 0.419614
## BP               0.473381  1.605413  0.218393  2.168 0.030192 *
## Anaemia          0.223993  1.251062  0.218977  1.023 0.306351
## Age              0.049048  1.050271  0.009449  5.191 2.09e-07 ***
## EF30to50        -1.107605  0.330349  0.238049 -4.653 3.27e-06 ***
## EFgeq50         -1.080066  0.339573  0.319293 -3.383 0.000718 ***
## Sodium          -0.052943  0.948434  0.023650 -2.239 0.025184 *
## Creatinine       0.257369  1.293522  0.070762  3.637 0.000276 ***
## log_CPK          0.093106  1.097578  0.101209  0.920 0.357604
## PleteletsleqQ1  0.234843  1.264710  0.256969  0.914 0.360771
## PleteletsgeqQ3  0.228468  1.256673  0.265349  0.861 0.389232
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##                exp(coef) exp(-coef) lower .95 upper .95
## Gender            0.8664     1.1542    0.5315    1.4123
## Smoking           1.1035     0.9062    0.6760    1.8016
## Diabetes          1.1971     0.8354    0.7734    1.8528
## BP                1.6054     0.6229    1.0464    2.4631
## Anaemia           1.2511     0.7993    0.8145    1.9216
## Age               1.0503     0.9521    1.0310    1.0699
## EF30to50          0.3303     3.0271    0.2072    0.5267
## EFgeq50           0.3396     2.9449    0.1816    0.6349
## Sodium            0.9484     1.0544    0.9055    0.9934
## Creatinine        1.2935     0.7731    1.1260    1.4860
## log_CPK           1.0976     0.9111    0.9001    1.3384
## PleteletsleqQ1    1.2647     0.7907    0.7643    2.0928
## PleteletsgeqQ3    1.2567     0.7958    0.7471    2.1139
##
## Concordance= 0.742  (se = 0.026 )
## Likelihood ratio test= 82.96  on 13 df,   p=3e-12
## Wald test            = 84.56  on 13 df,   p=2e-12
## Score (logrank) test = 91.5  on 13 df,   p=7e-14
```

```r
# Gender = 1 Age = 40
new_data_base <- data.frame("Gender" = 0,  "Smoking" = 0,  "Diabetes" = 0,  "BP" = 0,
                           "Anaemia" = 0, "Age" = 0, "EF30to50" = 0,
                         "EFgeq50" = 0, "Sodium" = 0,  "Creatinine" = 0,
                         "log_CPK" = 0 , "PleteletsleqQ1" = 0 , "PleteletsgeqQ3" = 0)

#mod.3 <- coxph(Surv(time, status) ~ (age + wt.loss)*sex, data=cancer)
surv_fit_base <- survfit(cox_fit, newdata = new_data_base)
plot(surv_fit_base)
```

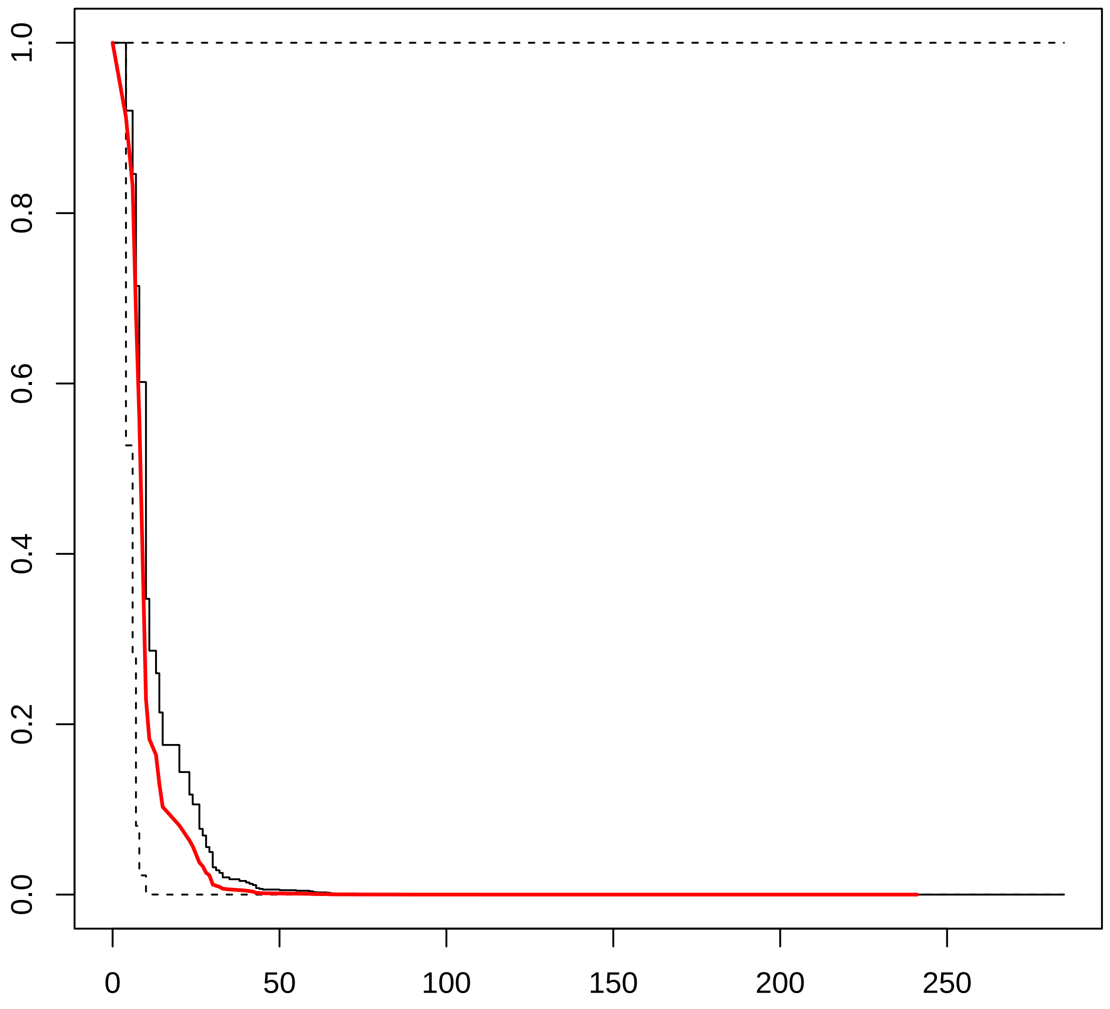


Cox Regression Harrell's C calculation

```
library(survival)
library(pec)
Test_set_cox <- read.csv("Test_Set.csv")[-c(1, 17)]
Train_set_cox <- read.csv("Train_set_cox.csv")[-c(1, 17)]
Train_set_cox$TIME <- as.numeric(Train_set_cox$TIME)
cox_fit_train <- coxph(Surv(TIME, Event) ~ ., data = Train_set_cox, x = TRUE)
median <- quantile(Train_set_cox$TIME[Train_set_cox$Event==1])[3]
upper_quartile <- quantile(Train_set_cox$TIME[Train_set_cox$Event==1])[4]
cox_surv_pred <- predictSurvProb(cox_fit_train,newdata=Test_set_cox,
                            times=c(median, upper_quartile))





## ETAS in the Harrell's c formula calculate
ETA_median <- cox_surv_pred[,1]

# Prepare a data frame for calculating Harrell's c (Median)
Harrell_c_median <- data.frame(cbind(ETA_median, ETA_median, Test_set_cox$TIME,
                                     Test_set_cox$TIME, Test_set_cox$Event))

ETA_upper_quartile <- cox_surv_pred[,2]

# Prepare a data frame for calculating Harrell's c (Upper Quartile)
Harrell_c_up_qautile <- data.frame(cbind(ETA_upper_quartile, ETA_upper_quartile,
                                            Test_set_cox$TIME, Test_set_cox$TIME,
                                            Test_set_cox$Event))

numerator_med <- 0
denominator_med <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_med = numerator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4]) *
        (Harrell_c_median[j,1] < Harrell_c_median[i,2]) *
        (Harrell_c_median[j,5] == 1)

      denominator_med = denominator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4])*
        (Harrell_c_median[j,5] == 1)
    }
  }
}

numerator_med / denominator_med

## [1] 0.6072261

numerator_upquar <- 0
denominator_upquar <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_upquar = numerator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4]) *
        (Harrell_c_up_qautile[j,1] < Harrell_c_up_qautile[i,2]) *
        (Harrell_c_up_qautile[j,5] == 1)

      denominator_upquar = denominator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4])*
        (Harrell_c_up_qautile[j,5] == 1)
    }
  }
}

numerator_upquar / denominator_upquar

## [1] 0.6072261
```

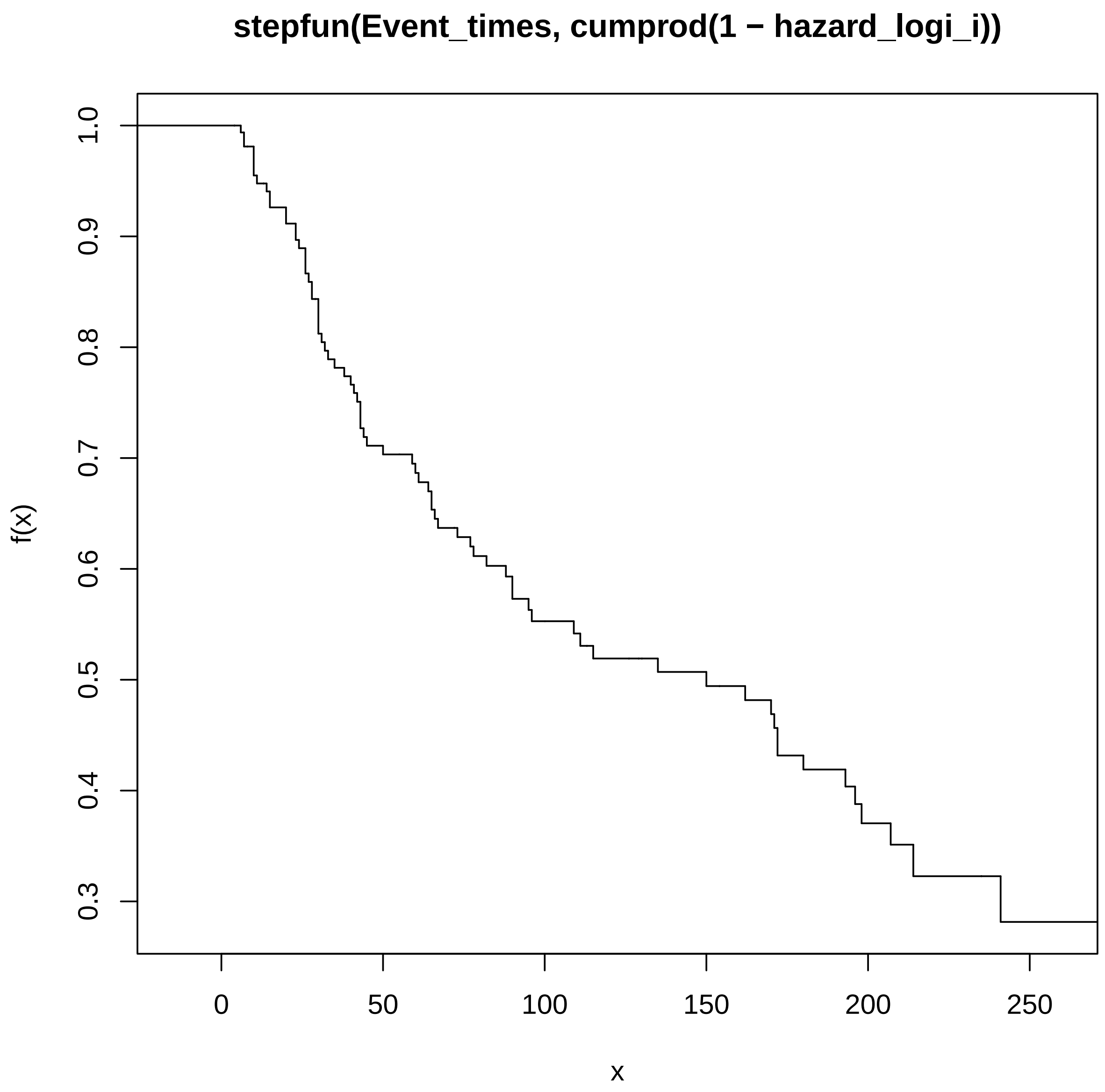
Logistic fit

```
Heart_atrisk <- read.csv("Heart_atrisk.csv")
model = glm(Event_indicator ~ factor(At_risk_set) +  Gender + Smoking +
              Diabetes + BP + Anaemia + Age + EF30to50 +  EFgeq50  +
              Sodium + Creatinine + log_CPK + PleteletsleqQ1 + PleteletsgeqQ3
             - 1, family = 'binomial', data = Heart_atrisk)

#summary(model)
summary(model)$coefficients[c(70:82),]

##                   Estimate Std. Error    z value     Pr(>|z|)
## Gender         -0.14216224 0.25149238 -0.5652746 5.718870e-01
## Smoking         0.09170852 0.25249541  0.3632087 7.164490e-01
## Diabetes        0.17820685 0.22443997  0.7940067 4.271915e-01
## BP               0.48259091 0.22066995  2.1869353 2.874725e-02
## Anaemia          0.23045967 0.22084326  1.0435440 2.966964e-01
## Age              0.04991062 0.00958346  5.2079960 1.908911e-07
## EF30to50        -1.13050006 0.24030891 -4.7043618 2.546612e-06
## EFgeq50         -1.10099997 0.32221594 -3.4169631 6.332386e-04
## Sodium          -0.05389204 0.02380831 -2.2635808 2.359991e-02
## Creatinine       0.26108715 0.07211293  3.6205316 2.939984e-04
## log_CPK          0.09723950 0.10203553  0.9529965 3.405918e-01
## PleteletsleqQ1   0.24278596 0.25903486  0.9372714 3.486190e-01
## PleteletsgeqQ3   0.22996116 0.26760878  0.8593184 3.901649e-01

alpha <- as.numeric(summary(model)$coefficients[c(1:69),][,1])

hazard <- exp(alpha)

hazard[hazard > 1] <- 1

hazard_logi <- c(0, c(hazard))


### Baseline survival curve
Event_times <- c(0, c(Heart$TIME[Heart$Event == 1]))

# cox baseline survival curve
plot(surv_fit_base)

# logistic baseline survival curve
lines(unique(Event_times), cumprod(1 - hazard_logi), lwd=2, col="red")
```



Logistic Regression Harrell's C calculation

```
Test_set_logi <- read.csv("Test_set.csv")[,-1]
Train_set_logi <- read.csv("Train_set_atrisk_dummy.csv")[,-c(1,2, 85)]
NEW_DATA <- NULL

for (i in 1:dim(Test_set_logi)[1]) {
  new_data <- data.frame(cbind(diag(rep(1, 69)),
                               matrix(rep(as.numeric(Test_set_logi[i,]), 69),
                                    69, 16, byrow=TRUE)))
  NEW_DATA <- rbind(NEW_DATA, new_data)
}

colnames(NEW_DATA) <- c(colnames(Train_set_logi)[1:69],
                        colnames(Test_set_logi))


logi_trained <- glm(Event_indicator ~ . - 1,
                    family = 'binomial', data = Train_set_logi)

prediction_logi <- predict(logi_trained, newdata = data.frame(NEW_DATA) ,
                         type = "response")

Event_times <- unique(c(Heart$TIME[Heart$Event == 1]))
library(stats)
Survival_curve <- list()
for (i in 0:(dim(Test_set_logi)[1] - 1)){
  j = i * 69
  hazard_logi_i <- c(0, c(prediction_logi[(j+1):(j+69)]))
  Survival_curve <- append(Survival_curve,
                         stepfun(Event_times, cumprod(1 - hazard_logi_i)))
}

Survival_curve_testsub1 <- list()
# Plotting the first survival curve (sanity check)
plot(Survival_curve[[1]], do.points=FALSE)
```



```
Survival_curve_testsub1 <- append(Survival_curve_testsub1, Survival_curve[[1]])

# Calculating the quantiles of observed event times in the train set
Train_set_cox <- read.csv("Train_set_cox.csv")
quantiles <- quantile(Train_set_cox$TIME[Train_set_cox$Event == 1])

median <- quantiles[3]
upper_quartile <- quantiles[4]

## ETAS in the Harrell's c formula calculate
ETA_median <- NULL
```

```r
# Evaluate survival curve at median event time in train set
for (i in 1:59) {
  Surv_curve <- Survival_curve[[i]]
  ETA_median <- c(ETA_median, Surv_curve(median))
}
# Prepare a data frame for calculating Harrell's c (Median)
Harrell_c_median <- data.frame(cbind(ETA_median, ETA_median, Test_set_logi$TIME,
                                     Test_set_logi$TIME, Test_set_logi$Event))

ETA_upper_quartile <- NULL
# Evaluate survival curve at upper quartile event time in train set
for (i in 1:59) {
  Surv_curve <- Survival_curve[[i]]
  ETA_upper_quartile  <- c(ETA_upper_quartile , Surv_curve(upper_quartile))
}
# Prepare a data frame for calculating Harrell's c (Upper Quartile)
Harrell_c_up_qautile <- data.frame(cbind(ETA_upper_quartile, ETA_upper_quartile,
                                         Test_set_logi$TIME, Test_set_logi$TIME,
                                         Test_set_logi$Event))

numerator_med <- 0
denominator_med <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_med = numerator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4]) *
        (Harrell_c_median[j,1] < Harrell_c_median[i,2]) *
        (Harrell_c_median[j,5] == 1)

      denominator_med = denominator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4])*
        (Harrell_c_median[j,5] == 1)
    }
  }
}

numerator_med / denominator_med
```

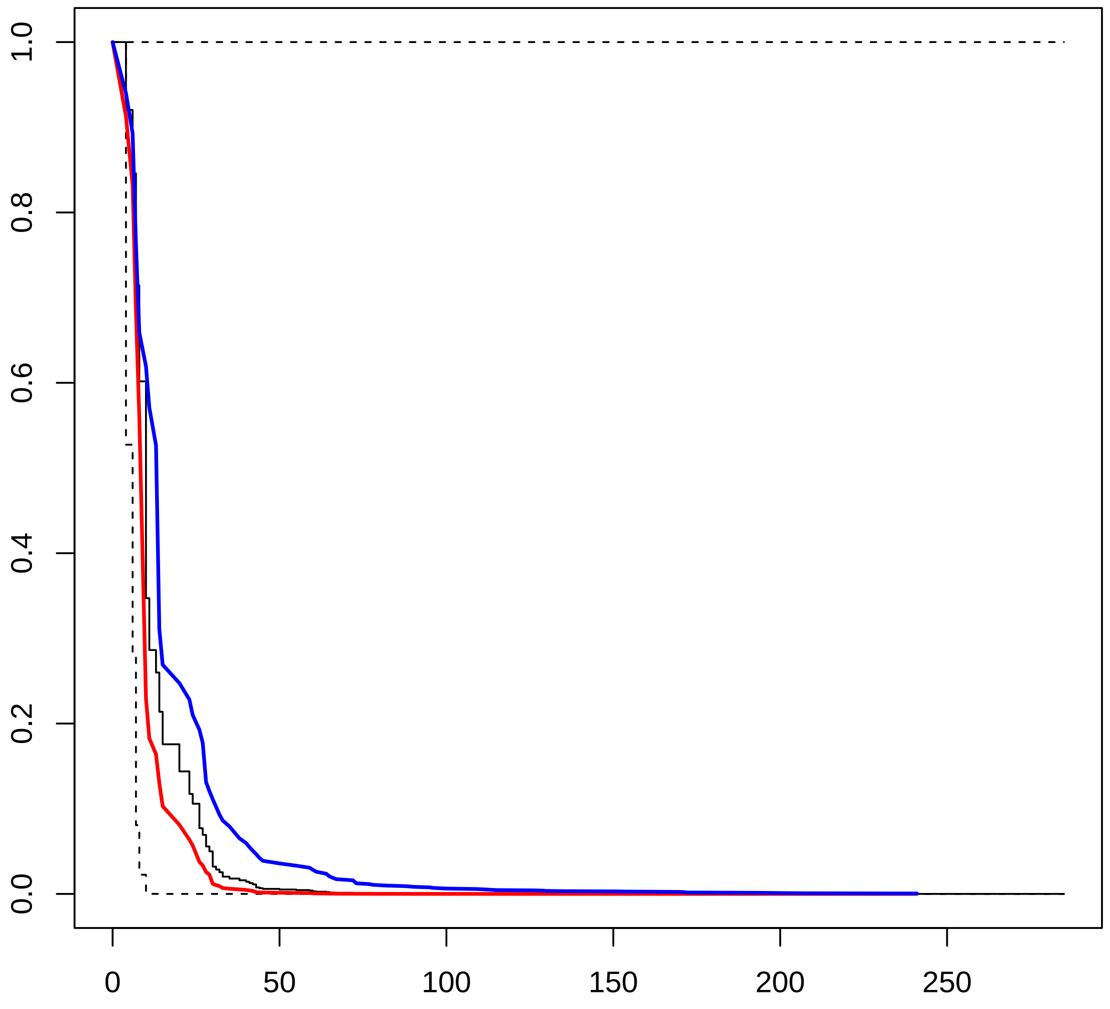
```
## [1] 0.6060606
```

```r
numerator_upquar <- 0
denominator_upquar <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_upquar = numerator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4]) *
        (Harrell_c_up_qautile[j,1] < Harrell_c_up_qautile[i,2]) *
        (Harrell_c_up_qautile[j,5] == 1)

      denominator_upquar = denominator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4])*
        (Harrell_c_up_qautile[j,5] == 1)
    }
  }
}

numerator_upquar / denominator_upquar

## [1] 0.6060606
```

Random Forest fit

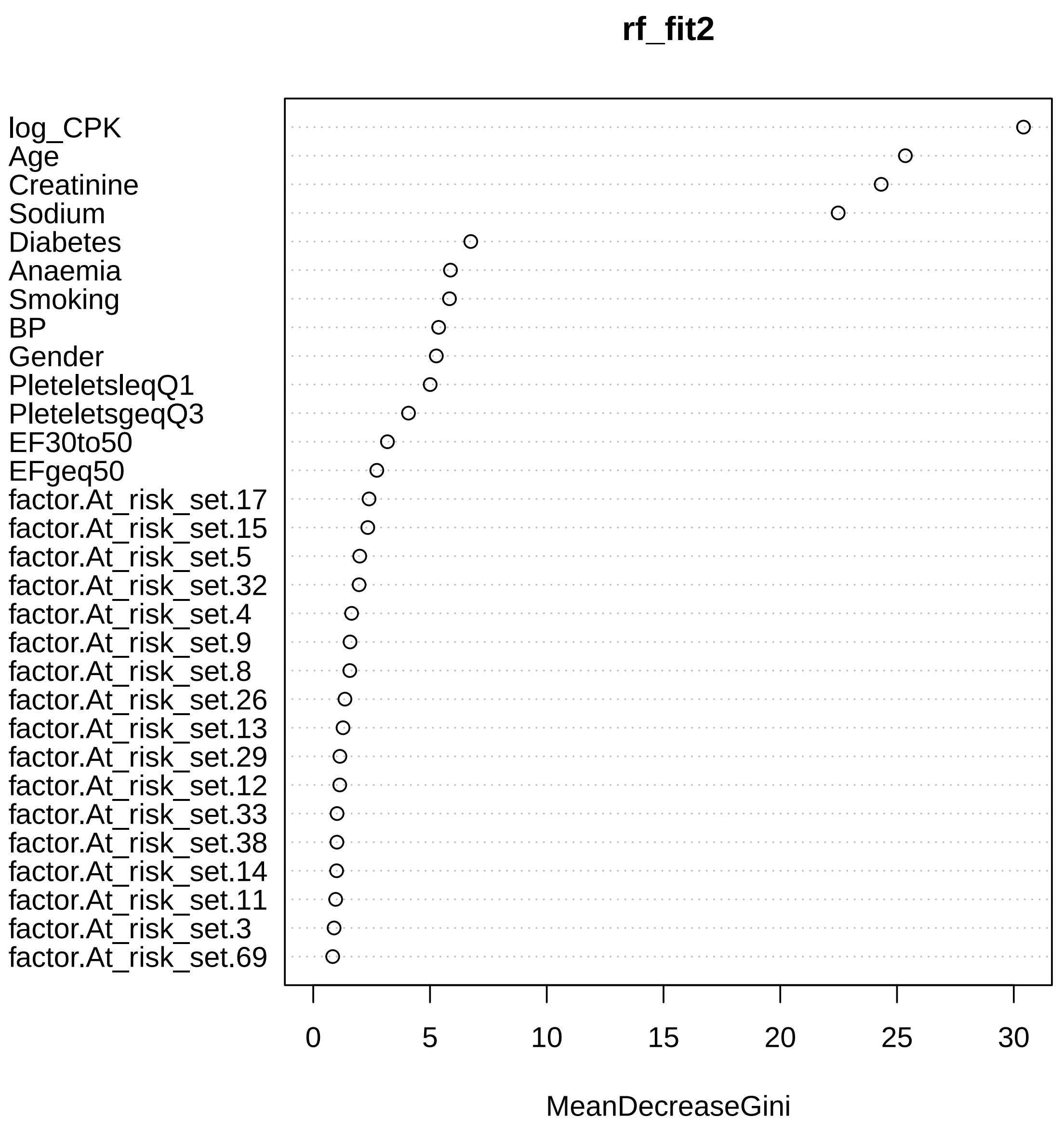
```
Heart_at_risk_dummy <- read.csv("Heart_at_risk_dummy.csv")[,-1]
library(randomForest)

# Each split randomly select 1/3 of predictors
#rf_fit1 <- randomForest(factor(Event_indicator) ~., data = Heart_at_risk_dummy,
#                        mtry=28)

# Each split randomly select 2/3 of predictors
rf_fit2 <- randomForest(factor(Event_indicator) ~., data = Heart_at_risk_dummy,
                       mtry=56)



new_data <- data.frame(cbind(diag(rep(1, 69)),
                             matrix(rep(0, 13 * 69), 69, 13)))
colnames(new_data) <- colnames(Heart_at_risk_dummy)[1:82]

pred_fit2 <- predict(rf_fit2, newdata = new_data , type = "prob")

hazard_rf <- c(0, c(pred_fit2[,2]))

Event_times <- c(0, c(Heart$TIME[Heart$Event == 1]))
```

```
# cox baseline survival curve
plot(surv_fit_base)

# logistic baseline survival curve
lines(unique(Event_times), cumprod(1 - hazard_logi), lwd=2, col="red")

# Random Forest baseline survival curve
lines(unique(Event_times), cumprod(1- hazard_rf), lwd=2, col="blue")
```



Random Forest Variable Importance

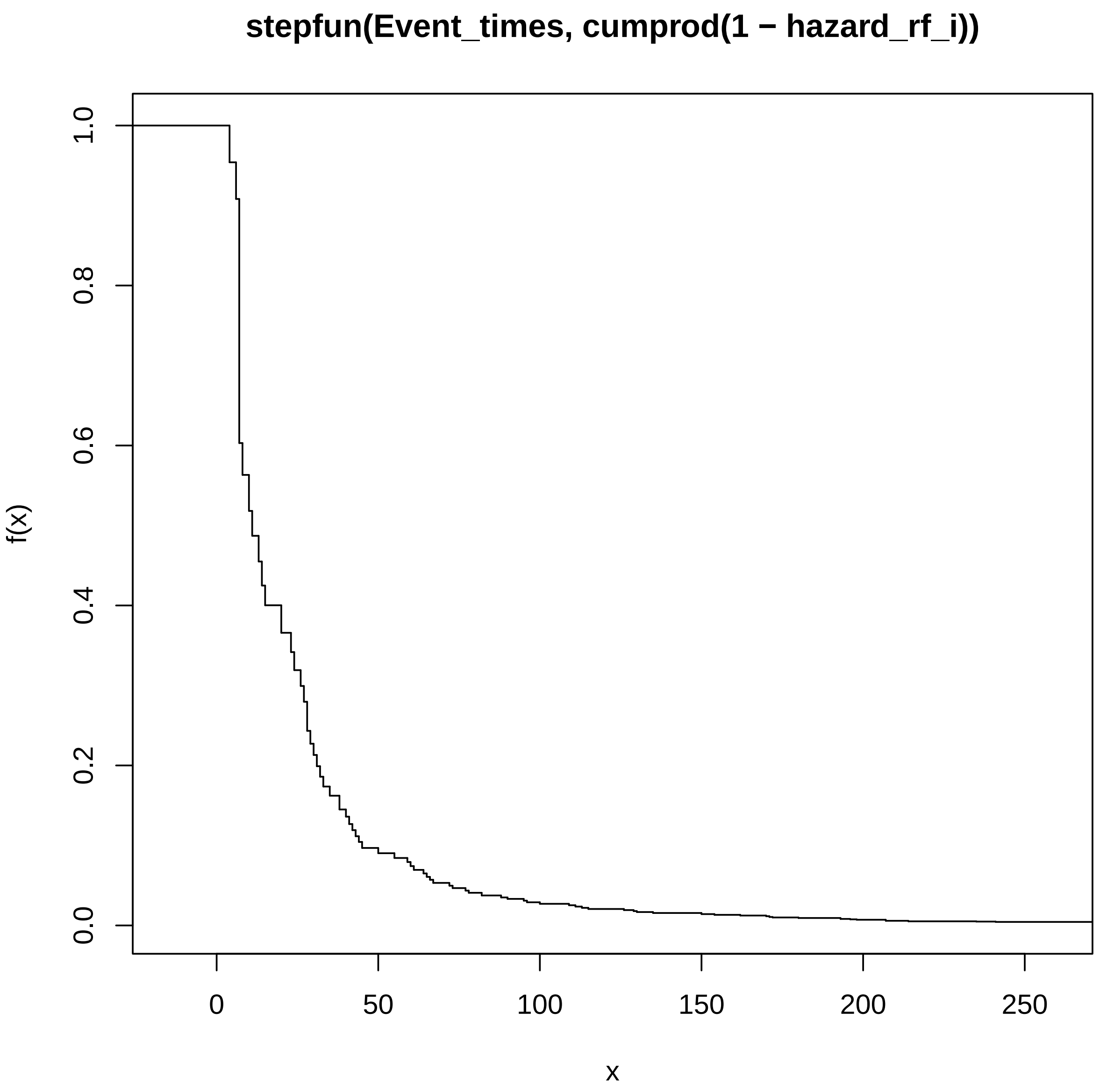
```r
#importance(rf_fit2, type = "2")  # Gini importance value
varImpPlot(rf_fit2, type = "2") # Gini importance plot
```



Mean Decrease Gini - Measure of variable importance based on the Gini impurity index used for the calculation of splits in trees.

Random Forest Harrell's C calculation

```r
Test_set_rf <- read.csv("Test_set.csv")[,-1]
Train_set_rf <- read.csv("Train_set_atrisk_dummy.csv")[,-c(1,2, 85)]
NEW_DATA <- NULL

for (i in 1:dim(Test_set_rf)[1]) {
  new_data <- data.frame(cbind(diag(rep(1, 69)),
                             matrix(rep(as.numeric(Test_set_rf[i,]), 69),
                                    69, 16, byrow=TRUE)))
  NEW_DATA <- rbind(NEW_DATA, new_data)
}

colnames(NEW_DATA) <- c(colnames(Train_set_rf)[1:69], colnames(Test_set_rf))

library(randomForest)
rf_trained <- randomForest(factor(Event_indicator) ~., data = Train_set_rf,
                       mtry=56)

prediction_rf <- predict(rf_trained, newdata = data.frame(NEW_DATA) ,
                         type = "prob")

Event_times <- unique(c(Heart$TIME[Heart$Event == 1]))
library(stats)
Survival_curve <- list()
for (i in 0:(dim(Test_set_rf)[1] - 1)){
  j = i * 69
  hazard_rf_i <- c(0, c(prediction_rf[(j+1):(j+69),2]))
  Survival_curve <- append(Survival_curve,
                         stepfun(Event_times, cumprod(1 - hazard_rf_i)))
}

# Plotting the first survival curve (sanity check)
plot(Survival_curve[[1]], do.points=FALSE)
```

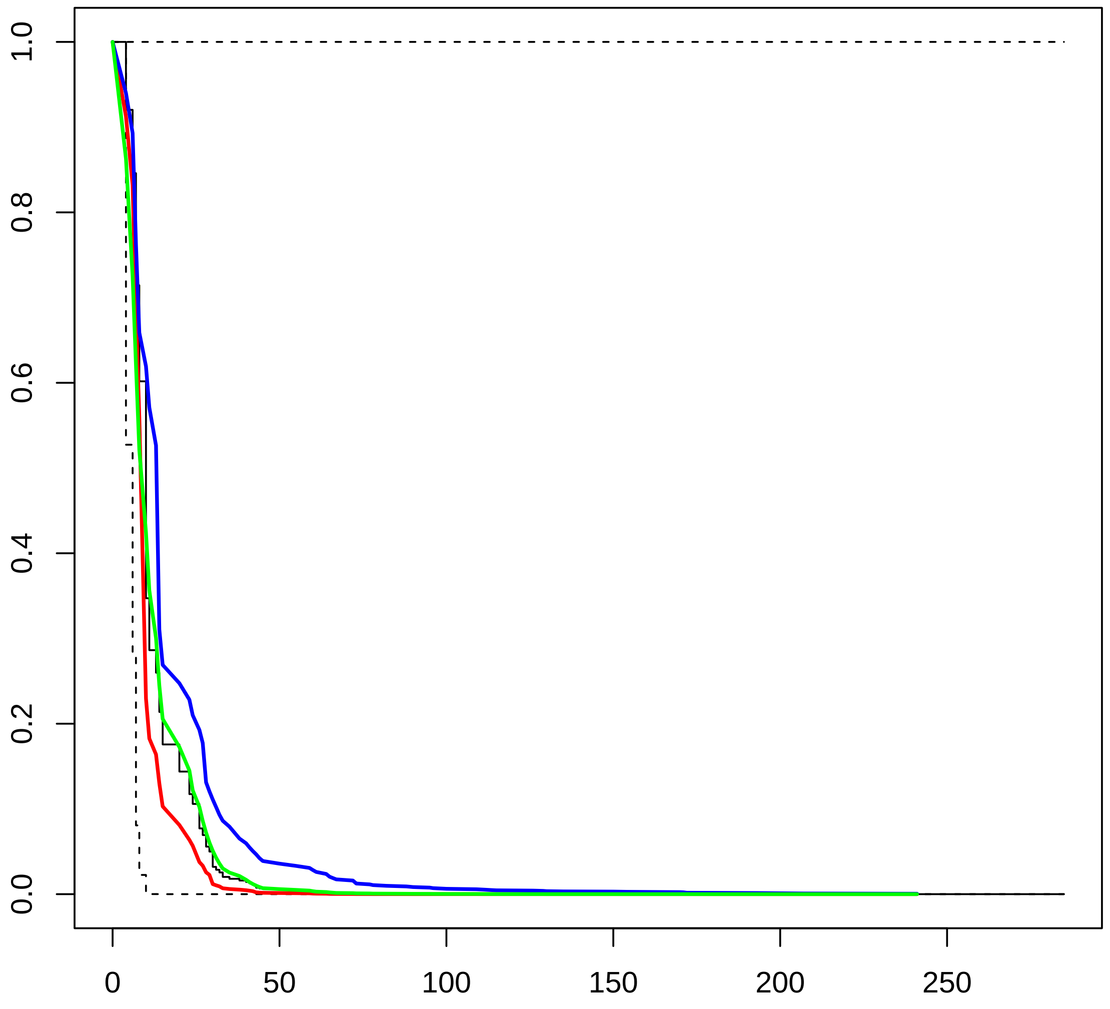


```r
# Calculating the quantiles of observed event times in the train set
Train_set_cox <- read.csv("Train_set_cox.csv")
quantiles <- quantile(Train_set_cox$TIME[Train_set_cox$Event == 1])

median <- quantiles[3]
upper_quartile <- quantiles[4]

## ETAS in the Harrell's c formula calculate
ETA_median <- NULL
# Evaluate survival curve at median event time in train set
for (i in 1:59) {
  Surv_curve <- Survival_curve[[i]]
  ETA_median <- c(ETA_median, Surv_curve(median))
}
# Prepare a data frame for calculating Harrell's c (Median)
Harrell_c_median <- data.frame(cbind(ETA_median, ETA_median, Test_set_rf$TIME,
                                     Test_set_rf$TIME, Test_set_rf$Event))

ETA_upper_quartile <- NULL
# Evaluate survival curve at upper quartile event time in train set
for (i in 1:59) {
  Surv_curve <- Survival_curve[[i]]
  ETA_upper_quartile  <- c(ETA_upper_quartile , Surv_curve(upper_quartile))
}
# Prepare a data frame for calculating Harrell's c (Upper Quartile)
Harrell_c_up_qautile <- data.frame(cbind(ETA_upper_quartile, ETA_upper_quartile,
                                            Test_set_rf$TIME, Test_set_rf$TIME,
                                            Test_set_rf$Event))

numerator_med <- 0
denominator_med <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_med = numerator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4]) *
        (Harrell_c_median[j,1] < Harrell_c_median[i,2]) *
        (Harrell_c_median[j,5] == 1)

      denominator_med = denominator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4])*
        (Harrell_c_median[j,5] == 1)
    }
  }
}

numerator_med / denominator_med

## [1] 0.6666667

numerator_upquar <- 0
denominator_upquar <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_upquar = numerator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4]) *
        (Harrell_c_up_qautile[j,1] < Harrell_c_up_qautile[i,2]) *
        (Harrell_c_up_qautile[j,5] == 1)

      denominator_upquar = denominator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4])*
        (Harrell_c_up_qautile[j,5] == 1)
    }
  }
}

numerator_upquar / denominator_upquar

## [1] 0.6456876
```

Boosting fit

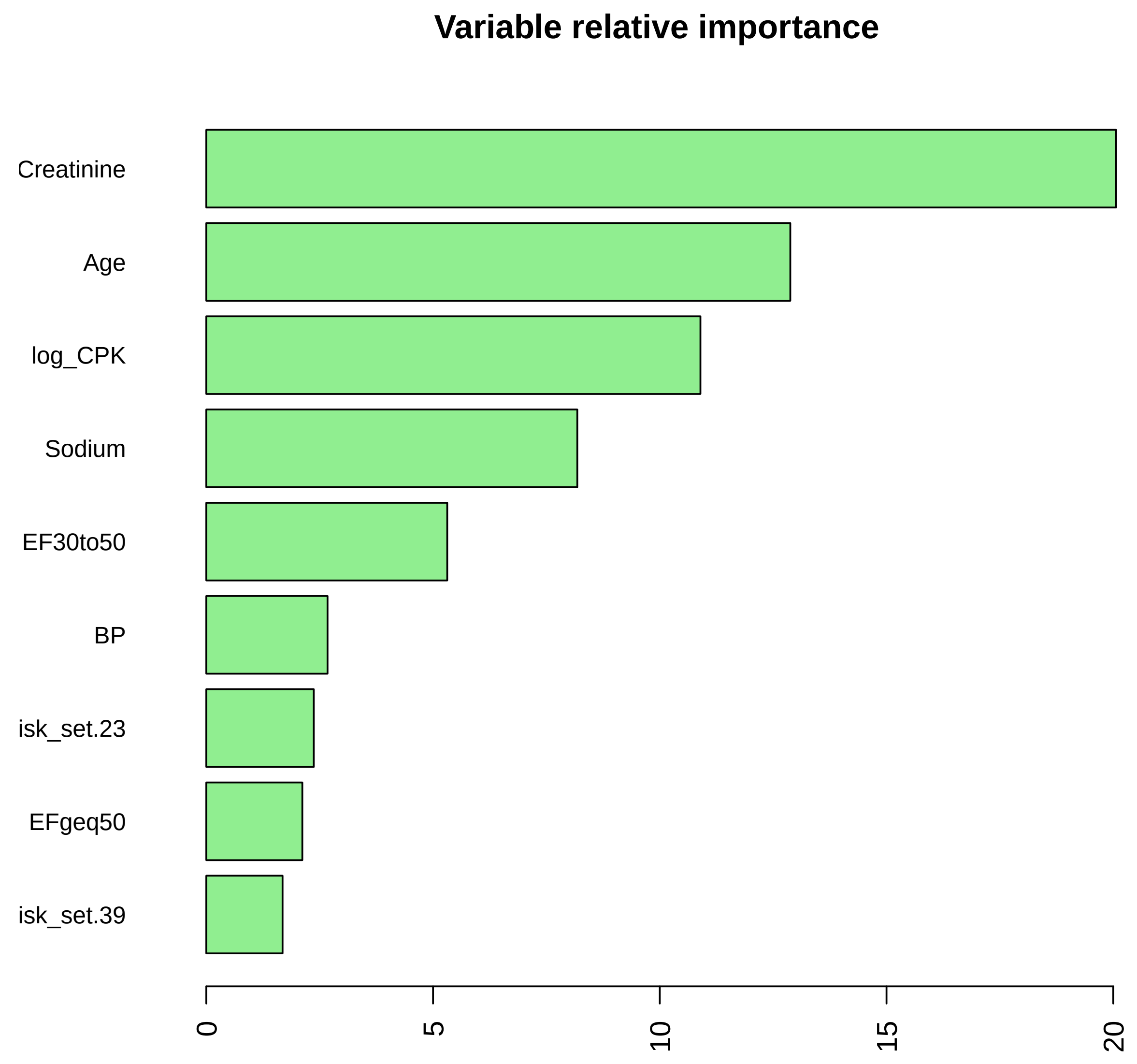
```
Heart_at_risk_dummy <- read.csv("Heart_at_risk_dummy.csv")[,-1]
Heart_at_risk_dummy$Event_indicator <- as.factor(Heart_at_risk_dummy$Event_indicator)

library(adabag)
Boosting_fit1 <- boosting(Event_indicator~., data = Heart_at_risk_dummy,
                          boos=TRUE, mfinal=50)

new_data <- data.frame(cbind(diag(rep(1, 69)),
                             matrix(rep(0, 13 * 69), 69, 13)))
colnames(new_data) <- colnames(Heart_at_risk_dummy)[1:82]

pred_fit_boost <- predict(Boosting_fit1, newdata = new_data , type = "prob")

hazard_boost <- c(0, c(pred_fit_boost$prob[,2]))

Event_times <- c(0, c(Heart$TIME[Heart$Event == 1]))

# cox baseline survival curve
plot(surv_fit_base)

# logistic baseline survival curve
lines(unique(Event_times), cumprod(1 - hazard_logi), lwd=2, col="red")

# Random Forest baseline survival curve
lines(unique(Event_times), cumprod(1- hazard_rf), lwd=2, col="blue")
```

```
# Boosting baseline survival curve
lines(unique(Event_times), cumprod(1- hazard_boost), lwd=2, col="green")
```



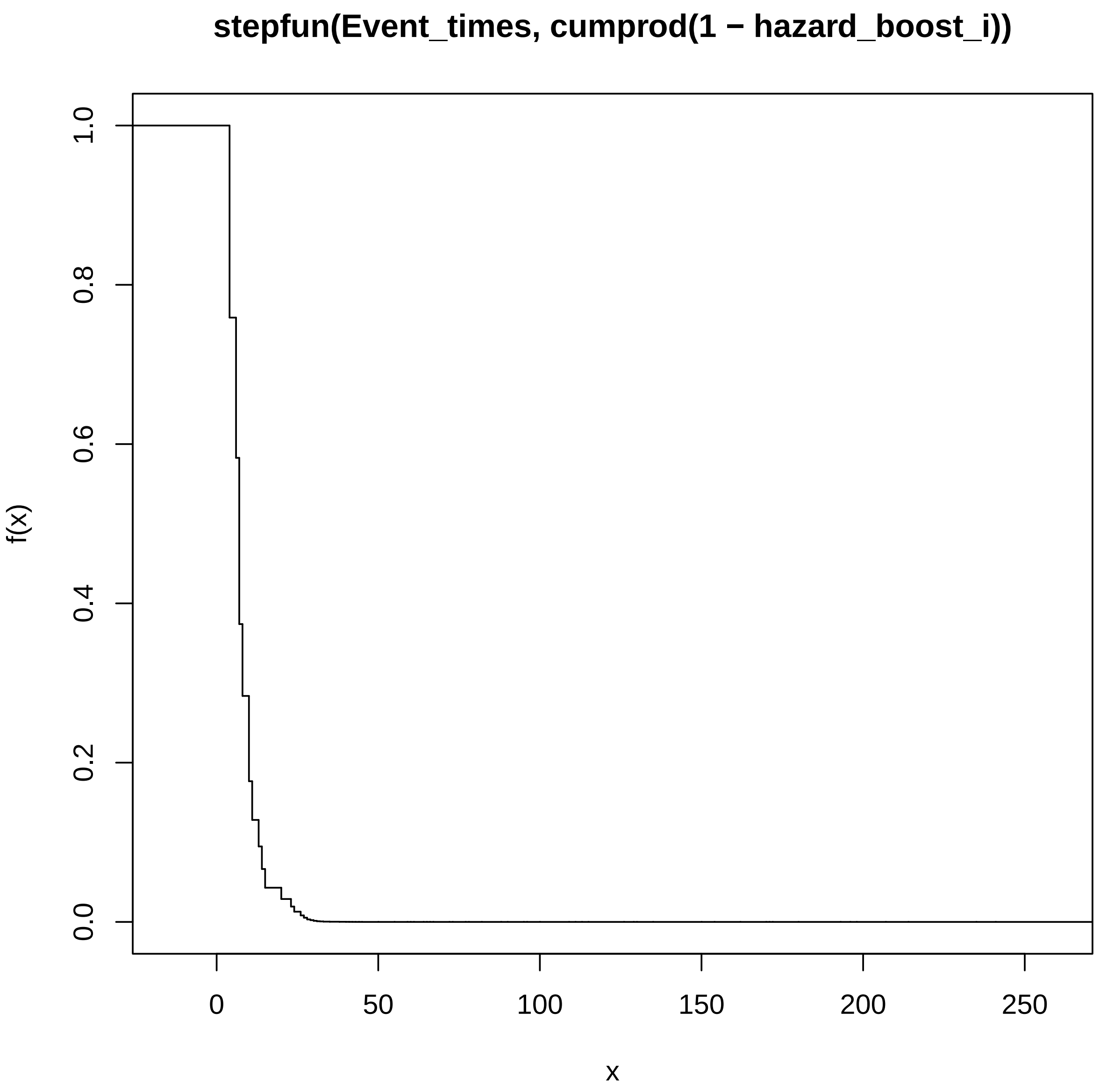
Boosting varaible importance

```
object <- Boosting_fit1
barplot(sort(object$imp, decreasing=TRUE)[9:1], horiz=TRUE,
        main = "Variable relative importance", col = "lightgreen",
        las = 2, xaxs = "r", cex.names=0.8)
```



Boosting C calculation

```
Test_set_boost <- read.csv("Test_set.csv")[,-1]
Train_set_boost <- read.csv("Train_set_atrisk_dummy.csv")[,-c(1,2, 85)]
Train_set_boost$Event_indicator <- as.factor(Train_set_boost$Event_indicator)
NEW_DATA <- NULL

#for (i in 1:dim(Test_set_boost)[1]) {
#  new_data <- data.frame(cbind(diag(rep(1, 69)),
#                              matrix(rep(as.numeric(Test_set_boost[i,]), 69),
#                                     69, 16, byrow=TRUE)))
#  NEW_DATA <- rbind(NEW_DATA, new_data)
#}

#colnames(NEW_DATA) <- c(colnames(Train_set_boost)[1:69],
#                         colnames(Test_set_boost))


#Boosting_trained <- boosting(Event_indicator~., data = Train_set_boost,
#                           boos=TRUE, mfinal=50)

#prediction_boost <- predict(Boosting_trained, newdata = data.frame(NEW_DATA) ,
#                          type = "prob")£prob

# Reading stored prediction
prediction_boost <- read.csv("prediction_boost.csv")[,-1]

prediction_boost[1:5,]

##          V1        V2
## 1 0.7588143 0.2411857
## 2 0.7679302 0.2320698
## 3 0.6418088 0.3581912
## 4 0.7588143 0.2411857
## 5 0.6224581 0.3775419

Event_times <- unique(c(Heart$TIME[Heart$Event == 1]))
library(stats)
Survival_curve <- list()
for (i in 0:(dim(Test_set_boost)[1] - 1)){
  j = i * 69
  hazard_boost_i <- c(0, c(prediction_boost[(j+1):(j+69),2]))
  Survival_curve <- append(Survival_curve,
                           stepfun(Event_times, cumprod(1 - hazard_boost_i)))
}

# Plotting the first survival curve (sanity check)
plot(Survival_curve[[1]], do.points=FALSE)
```



```r
# Calculating the quantiles of observed event times in the train set
Train_set_cox <- read.csv("Train_set_cox.csv")
quantiles <- quantile(Train_set_cox$TIME[Train_set_cox$Event == 1])

median <- quantiles[3]
upper_quartile <- quantiles[4]

## ETAS in the Harrell's c formula calculate
ETA_median <- NULL
# Evaluate survival curve at median event time in train set
for (i in 1:59) {
  Surv_curve <- Survival_curve[[i]]
  ETA_median <- c(ETA_median, Surv_curve(median))
}
# Prepare a data frame for calculating Harrell's c (Median)
Harrell_c_median <- data.frame(cbind(ETA_median, ETA_median, Test_set_boost$TIME,
                                     Test_set_boost$TIME, Test_set_boost$Event))

ETA_upper_quartile <- NULL
# Evaluate survival curve at upper quartile event time in train set
for (i in 1:59) {
  Surv_curve <- Survival_curve[[i]]
  ETA_upper_quartile  <- c(ETA_upper_quartile , Surv_curve(upper_quartile))
}
# Prepare a data frame for calculating Harrell's c (Upper Quartile)
Harrell_c_up_qautile <- data.frame(cbind(ETA_upper_quartile, ETA_upper_quartile,
                                         Test_set_boost$TIME, Test_set_boost$TIME,
                                         Test_set_boost$Event))

numerator_med <- 0
denominator_med <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_med = numerator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4]) *
        (Harrell_c_median[j,1] < Harrell_c_median[i,2]) *
        (Harrell_c_median[j,5] == 1)

      denominator_med = denominator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4])*
        (Harrell_c_median[j,5] == 1)
    }
  }
}

numerator_med / denominator_med

## [1] 0.6386946

numerator_upquar <- 0
denominator_upquar <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_upquar = numerator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4]) *
        (Harrell_c_up_qautile[j,1] < Harrell_c_up_qautile[i,2]) *
        (Harrell_c_up_qautile[j,5] == 1)

      denominator_upquar = denominator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4])*
        (Harrell_c_up_qautile[j,5] == 1)
    }
  }
}

numerator_upquar / denominator_upquar

## [1] 0.6386946
```

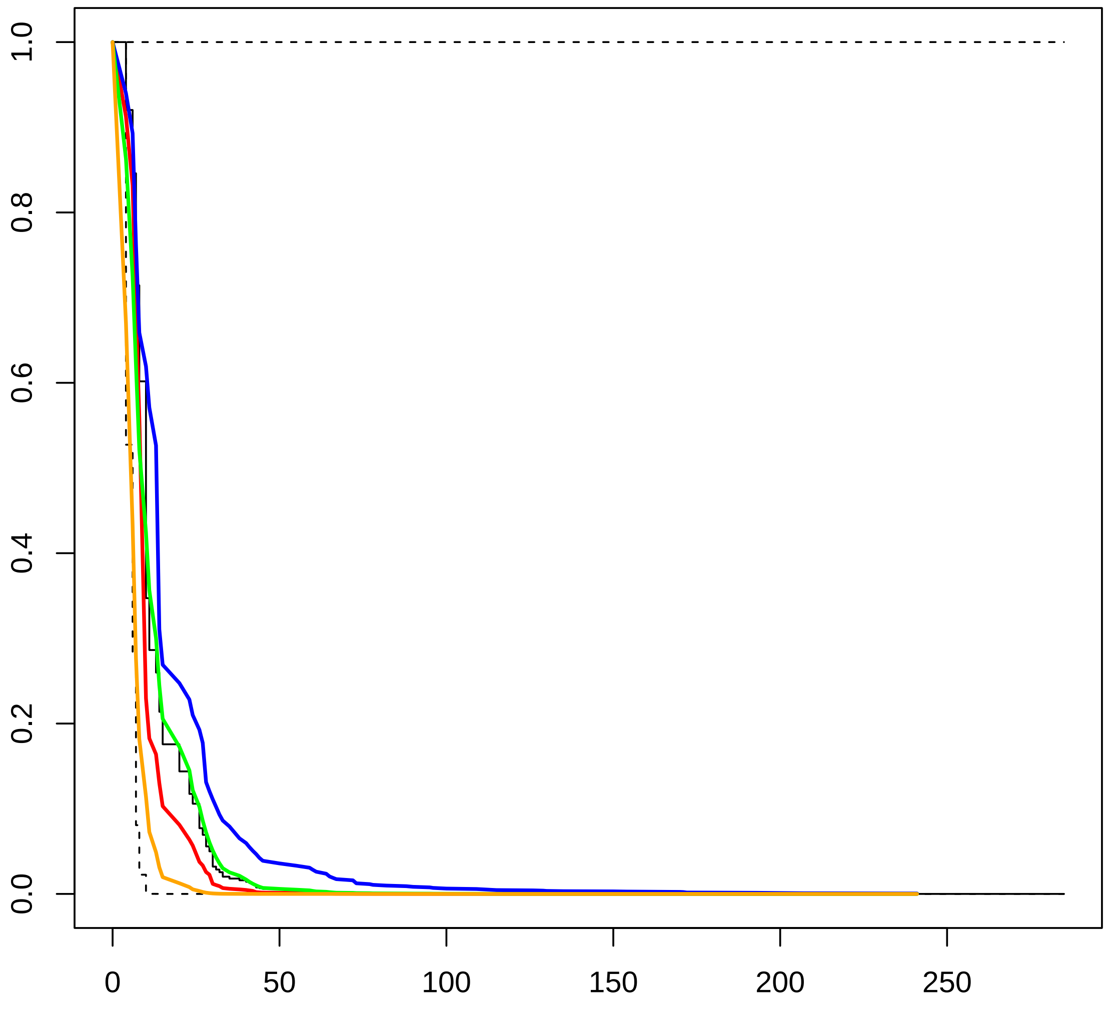
Neural Network (ANN) Model fit

```
Heart_at_risk_dummy <- read.csv("Heart_at_risk_dummy.csv")[,-1]
library("keras")
library("tensorflow")
#batch normalization parameters
alpha = .1
epsilon = 1e-4
```

```
## Ignore validation set
train_data <- Heart_at_risk_dummy
x_train <- as.matrix(train_data[,-83])
y_train <- train_data[,83]

NN_model2 <- keras_model_sequential()
NN_model2 %>%
  layer_dense(units = 128, activation = 'relu', input_shape = c(82)) %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 8, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1, activation = 'sigmoid')

summary(NN_model2)

## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense_3 (Dense)                    (None, 128)                      10624
##  dropout_2 (Dropout)                (None, 128)                      0
##  dense_2 (Dense)                    (None, 32)                       4128
##  dropout_1 (Dropout)                (None, 32)                       0
##  dense_1 (Dense)                    (None, 8)                        264
##  dropout (Dropout)                  (None, 8)                        0
##  dense (Dense)                      (None, 1)                        9
## ================================================================================
## Total params: 15,025
## Trainable params: 15,025
## Non-trainable params: 0
## ________________________________________________________________________________

NN_model2 %>% compile(
  loss = "binary_crossentropy",
  optimizer = 'adam',
  metrics = c('accuracy')
)

history <- NN_model2 %>% fit(
  x_train, y_train,
  epochs= 30, batch_size = 100, verbose=1
)
```

```
NN_fit <- NN_model2
new_data <- data.frame(cbind(diag(rep(1, 69)),
                             matrix(rep(0, 13 * 69), 69, 13)))
colnames(new_data) <- colnames(Heart_at_risk_dummy)[1:82]

hazard <- predict(NN_fit, x = as.matrix(new_data))

hazard_NN <- c(0, c(hazard))

Event_times <- c(0, c(Heart$TIME[Heart$Event == 1]))

# cox baseline survival curve
plot(surv_fit_base)

# logistic baseline survival curve
lines(unique(Event_times), cumprod(1 - hazard_logi), lwd=2, col="red")
```

```r
# Random Forest baseline survival curve
lines(unique(Event_times), cumprod(1- hazard_rf), lwd=2, col="blue")

# Boosting baseline survival curve
lines(unique(Event_times), cumprod(1- hazard_boost), lwd=2, col="green")

# Neural Network baseline survival curve
lines(unique(Event_times), cumprod(1- hazard_NN), lwd=2, col="orange")
```



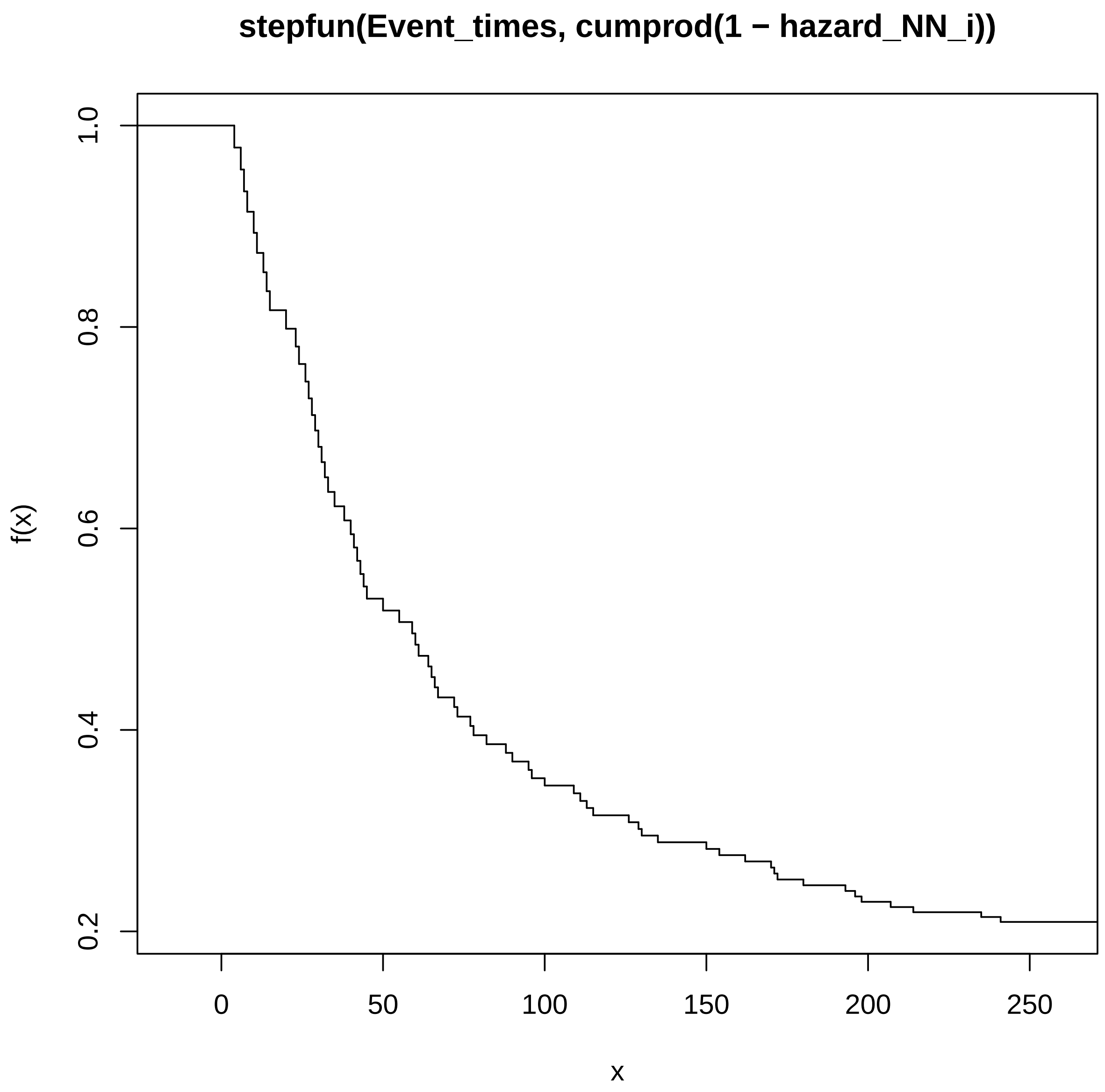
ANN Harrell's C calculation

```r
Test_set_NN <- read.csv("Test_set.csv")[,-c(1,17)]
Test_set_NN2 <- read.csv("Test_set.csv")[,-c(1, 2, 3, 17)]
Train_set_NN <- read.csv("Train_set_atrisk_dummy.csv")[,-c(1,2, 85)]
NEW_DATA <- NULL

for (i in 1:dim(Test_set_NN2)[1]) {
  new_data <- data.frame(cbind(diag(rep(1, 69)),
                               matrix(rep(as.numeric(Test_set_NN2[i,]), 69),
                                      69, 13, byrow=TRUE)))
  NEW_DATA <- rbind(NEW_DATA, new_data)
}

colnames(NEW_DATA) <- c(colnames(Train_set_NN)[1:69], colnames(Test_set_NN2))
```

```r
x_train <- as.matrix(Train_set_NN[,-83])
y_train <- Train_set_NN[,83]

NN_model_train <- keras_model_sequential()
NN_model_train %>%
  layer_dense(units = 128, activation = 'relu', input_shape = c(82)) %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 8, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1, activation = 'sigmoid')

NN_model_train %>% compile(
  loss = "binary_crossentropy",
  optimizer = 'adam',
  metrics = c('accuracy')
)

history <- NN_model_train %>% fit(
  x_train, y_train,
  epochs= 30, batch_size = 100, verbose=1
)
```

```r
NN_fit_train <- NN_model_train
```

```r
prediction_NN <- predict(NN_fit_train, x = as.matrix(NEW_DATA))

Event_times <- unique(c(Heart$TIME[Heart$Event == 1]))

library(stats)
Survival_curve <- list()
for (i in 0:(dim(Test_set_NN)[1] - 1)){
  j = i * 69
  hazard_NN_i <- c(0, c(prediction_NN[(j+1):(j+69)]))
  Survival_curve <- append(Survival_curve,
                          stepfun(Event_times, cumprod(1 - hazard_NN_i)))
}

# Plotting the first survival curve (sanity check)
plot(Survival_curve[[1]], do.points=FALSE)
```



```r
Survival_curve_testsub1 <- append(Survival_curve_testsub1, Survival_curve[[1]])

# Calculating the quantiles of observed event times in the train set
Train_set_cox <- read.csv("Train_set_cox.csv")
quantiles <- quantile(Train_set_cox$TIME[Train_set_cox$Event == 1])

median <- quantiles[3]
upper_quartile <- quantiles[4]

## ETAS in the Harrell's c formula calculate
ETA_median <- NULL
```

```r
# Evaluate survival curve at median event time in train set
for (i in 1:59) {
  Surv_curve <- Survival_curve[[i]]
  ETA_median <- c(ETA_median, Surv_curve(median))
}
# Prepare a data frame for calculating Harrell's c (Median)
Harrell_c_median <- data.frame(cbind(ETA_median, ETA_median, Test_set_NN$TIME,
                                     Test_set_NN$TIME, Test_set_NN$Event))

ETA_upper_quartile <- NULL
# Evaluate survival curve at upper quartile event time in train set
for (i in 1:59) {
  Surv_curve <- Survival_curve[[i]]
  ETA_upper_quartile  <- c(ETA_upper_quartile , Surv_curve(upper_quartile))
}
# Prepare a data frame for calculating Harrell's c (Upper Quartile)
Harrell_c_up_qautile <- data.frame(cbind(ETA_upper_quartile, ETA_upper_quartile,
                                          Test_set_NN$TIME, Test_set_NN$TIME,
                                          Test_set_NN$Event))

numerator_med <- 0
denominator_med <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_med = numerator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4]) *
        (Harrell_c_median[j,1] < Harrell_c_median[i,2]) *
        (Harrell_c_median[j,5] == 1)

      denominator_med = denominator_med +
        (Harrell_c_median[i,3] > Harrell_c_median[j,4])*
        (Harrell_c_median[j,5] == 1)
    }
  }
}

numerator_med / denominator_med
```

```
## [1] 0.6515152
```

```r
numerator_upquar <- 0
denominator_upquar <- 0
for (i in 1:59) {
  for (j in 1:59) {
    if (i != j) {
      numerator_upquar = numerator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4]) *
        (Harrell_c_up_qautile[j,1] < Harrell_c_up_qautile[i,2]) *
        (Harrell_c_up_qautile[j,5] == 1)

      denominator_upquar = denominator_upquar +
        (Harrell_c_up_qautile[i,3] > Harrell_c_up_qautile[j,4])*
        (Harrell_c_up_qautile[j,5] == 1)
    }
  }
}

numerator_upquar / denominator_upquar

## [1] 0.6526807
```