Prediction of Survival Outcome

Carlos S Traynor

2018-05-02

This vignette serves as an approach to the modelling of survival data in medical statistics. We will review the most relevant methods including adaptations to the use of high throughput genomic data. There are a vast number of packages focused in survival analysis and our goal is make a summary of the most useful functions and the best ways to combine them to obtain a meaningfull analysis.

The main outcome in survival analysis is the time untill an event occurs. The difference between survival analysis and other statistical analysis is that some events are censored, or not observed because the actual event time is longer than the follow-up of the study.

We will need various packages that can be installed through the installation of predsurv package. In addition we will work with a simulated dataset, which is a list containing a gene expression matrix and the survival outcome of an hypothetical experiment, you can check the simulation algorithm in data-raw.

```r
#devtools::install_github("csetraynor/predsurv")
library(predsurv)
```

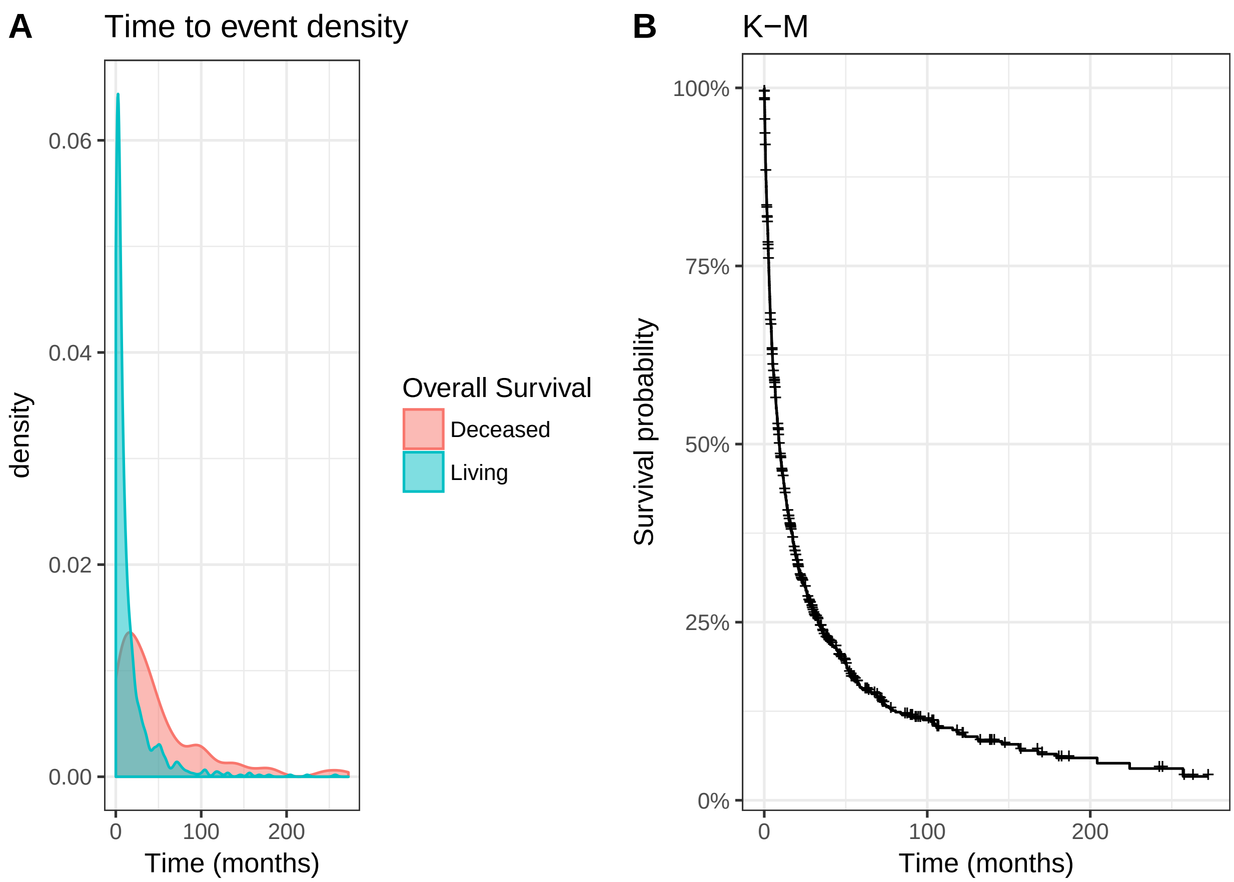
```
## Loading required package: ggplot2
```

```
## Loading required package: survival
```

First step is to simulate some data.

```r
####Simulation study
set.seed(12318)
survdata <- surv_sim_data(N = 1000, features = 100, CenRate = 1/10)

#Exploratory Analysis
tte <- plot_tte_dist(survdata, time = os_months, status = os_deceased)
km <- plot_km(survdata, time = os_months, status = os_deceased)
ggpubr::ggarrange(tte, km,
          labels = c("A", "B"),
          ncol = 2)
```



Because this is a simulated dataset we can skip various steps of data manipulation, however we want to review a first data exploration, for example the time to event distribution and the Kaplan and Meier plots. Besides, create a test and training splits. We will use a function of the package survdata.

```
##Create train-test split
set.seed(83742)
fold = create_training_test_set(survdata, p = 0.5, status = os_deceased)
train = fold[["train"]]
test = fold[["test"]]
head(train[,1:6]); rm(fold)
```

```
##    os_months os_deceased   feature1    feature2    feature3   feature4
## 2  1.8841725        TRUE  1.3403761  0.50205205 -0.96813780  0.7163509
## 3  0.3014296        TRUE  1.2644508  0.01232113  0.49439537  1.8088983
## 4  4.1180839        TRUE -0.2822482 -0.23557265  0.38338585  0.5260858
## 5  7.8922908        TRUE -0.5493864  0.70255017  0.04593234 -1.6291111
## 6  0.4372697        TRUE -1.1830843 -0.11865939 -0.09117216 -0.9333777
## 7 13.6881960        TRUE  0.1403592 -0.11056636 -2.43884780  0.2206866
```

Now is as simple as train the models to the training set:

```
##Train Models##
mod_uni <- predsurv::fun_train(train = train, fit = "Univariate")
mod_lasso <- predsurv::fun_train(train = train, fit = "Lasso")
mod_bst <- predsurv::fun_train(train = train, fit = "Random forest")
mod_ridge <- predsurv::fun_train(train = train, fit = "Ridge regression")
mod_enet <- predsurv::fun_train(train = train, fit = "Elastic net")
mod_iter_enet <- predsurv::fun_train(train = train, fit = "Elastic net", iterative = TRUE)
```

Second step moves on assesing model performance. For instance one can get the AUC by:

```
##ROC
roc_lasso <- predsurv::fun_test(obj = mod_lasso, train_data = train,  pred = "ROC",
                                integrated = FALSE, noboot = 10, test_data = test)
roc_ridge <- predsurv::fun_test(obj = mod_ridge, train_data = train, pred = "ROC",
                                integrated = FALSE, noboot = 10 , test_data = test)
roc_uni <- predsurv::fun_test(obj = mod_uni, train_data = train,  pred = "ROC",
                              integrated = FALSE, noboot = 10 , test_data = test)
roc_enet <- predsurv::fun_test(obj = mod_enet, train_data = train,  pred = "ROC",
                               integrated = FALSE, noboot = 10 , test_data = test)
roc_iter_enet <-  predsurv::fun_test(obj = mod_iter_enet, train_data = train,
                                     pred = "ROC",
                                     integrated = FALSE, noboot = 10 ,
                                     test_data = test)
```

Or the Brier score

```
##brier
brier_lasso <- predsurv::fun_test(obj = mod_lasso, train_data = train,
                                  test_data = test,
                                  pred = "Brier",
                                  integrated = FALSE)
```

```
## No covariates  specified: Kaplan-Meier for censoring times used for weighting.
```

```
brier_ridge <- predsurv::fun_test(obj = mod_ridge, train_data = train,
                                  test_data = test ,
                                  pred = "Brier",
                                  integrated = FALSE)
```

```
## No covariates  specified: Kaplan-Meier for censoring times used for weighting.
```

```
brier_uni <- predsurv::fun_test(obj = mod_uni, train_data = train,
                                test_data = test ,
                                pred = "Brier",
                                integrated = FALSE)
```

```
## No covariates  specified: Kaplan-Meier for censoring times used for weighting.
```

```
brier_enet <- predsurv::fun_test(obj = mod_enet, train_data = train,
                                 test_data = test,
                                 pred = "Brier",
                                 integrated = FALSE)
```

```
## No covariates  specified: Kaplan-Meier for censoring times used for weighting.
```

```
brier_bst<- predsurv::fun_test(obj = mod_bst, train_data = train,
                               test_data = test,
                               pred = "Brier",
                               integrated = FALSE)
```

```
## No covariates  specified: Kaplan-Meier for censoring times used for weighting.
```

```
brier_iter_enet <- predsurv::fun_test(obj = mod_iter_enet,
                                      train_data = train,
                                      test_data = test,
```

```
                                          pred = "Brier", integrated = FALSE)
```

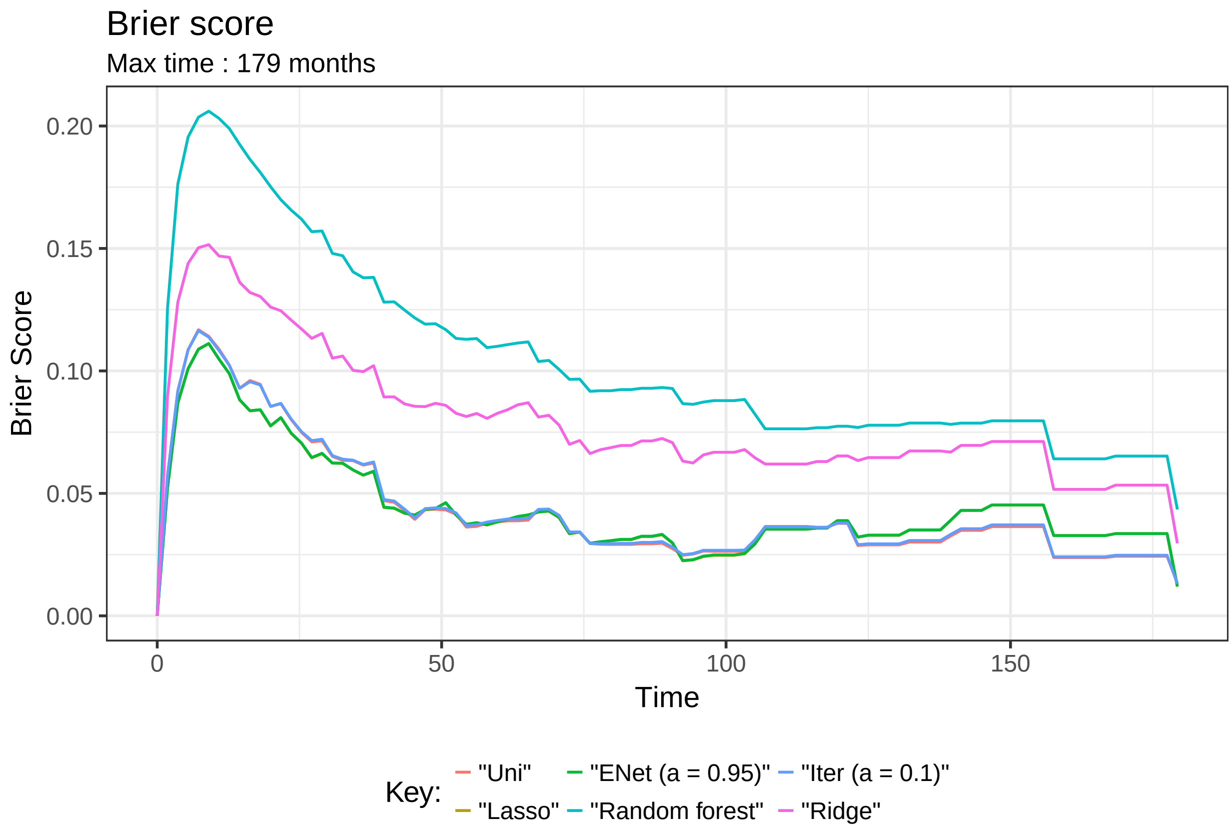
```
## No covariates  specified: Kaplan-Meier for censoring times used for weighting.
```

And lastly we would like to visualise the results, right now is a little tedious but more improvements are on the way, this is how a plot for ROC can be made:

```
#Prepare for plot
attr(roc_uni, 'prediction.of.model') <- "Uni"
attr(roc_lasso, 'prediction.of.model') <- "Lasso"
attr(roc_enet, 'prediction.of.model') <- paste0("ENet (a = ",
                                                attr(mod_enet,
                                                     'chosen.alpha'), ")")
attr(roc_iter_enet, 'prediction.of.model') <- paste0("Iter (a = ",
                                                     attr(mod_iter_enet,
                                                          'chosen.alpha'), ")")
attr(roc_ridge, 'prediction.of.model') <- "Ridge"
###Plot
rocplot <- roc.plot2(roc_uni, roc_lasso, roc_enet, roc_ridge, roc_iter_enet) +
  labs(subtitle = paste0("Time = ", round(quantile(unlist(test[test$os_deceased ,
                                                               "os_months"])
                                                   , .73),0 ) , " months"))
rocplot
```

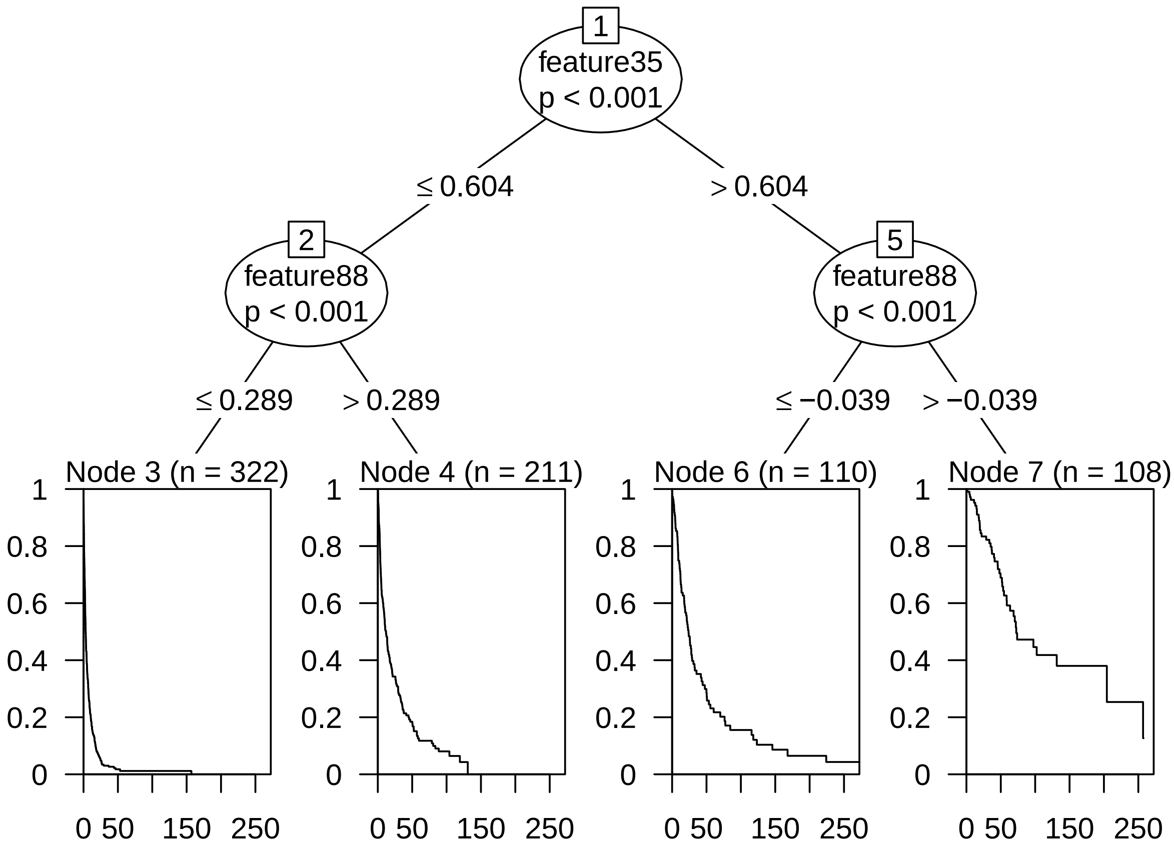
Similarly the Brier score:

```
### Gives attr for plot
attr(brier_uni, 'prediction.of.model') <- "Uni"
attr(brier_lasso, 'prediction.of.model') <- "Lasso"
attr(brier_enet, 'prediction.of.model') <- paste0("ENet (a = ",
                                                  attr(mod_enet,
                                                       'chosen.alpha'),
                                                  ")")
attr(brier_ridge, 'prediction.of.model') <- "Ridge"
attr(brier_bst, 'prediction.of.model') <- "Random forest"
attr(brier_iter_enet, 'prediction.of.model') <- paste0("Iter (a = ",
                                                       attr(mod_iter_enet,
                                                            'chosen.alpha'),
                                                       ")")
#Plot
brierplot <- plot_brier2(brier_uni, brier_lasso, brier_enet, brier_bst,
                         brier_iter_enet, brier_ridge) +
  labs(title = "Brier score", subtitle =
         paste0("Max time : " , round(max(unlist(test[test$os_deceased ,
                                                      "os_months"])), 0),
                " months") )
brierplot
```



Survival trees are easy to visualise and understand, we can fit a survival tree by:

```
stree = fun_train(train = train, fit = "Tree")
plot(stree)
```



Nevertheless, decision trees can create overcomplex trees that don't generalise well.

In this vignette we have shown an introduction of how fit and assess model performance for different methods, however, are these analyses reliable? May be it is possible to obtain more reliable model performance assessment via MC-Cross-Validation. We should review these methods and more in the next vignette.