Supplementary Material for Two Stage Design for Survival Endpoint

Biyue Dai

August 21, 2019

This article provides a guidance of the supplementary R functions for the manuscript “Two-Stage Adaptive Design for Prognostic Biomarker Signatures with a Survival Endpoint”

Documentation of Functions

two_stage_sim()

Description

Conduct simulation for the proposed two-stage design for given scenarios.

Arguments

nn: Total sample size

rate_l: rate parameter in the Exponential Distribution that generates the Time of Event Occurs (default = 0.025)

rate_c: rate parameter in the Exponential Distribution that generates the Time of Censoring (default = 0.025)

beta: the true regression coefficients

mean: parameter in generating the dependent variables (default = 0)

sd: parameter in generating the dependent variables (default = log(1.5))

itrB: number of iterations used for bootstrap or permutation resampling (default = 200)

nfold: number of folds for cross-validation (default = 10)

S: proportion of samples used in first stage (default: 0.5)

alpha_1: level of test in first stage (default: 0.25)

alpha_2: level of test in second stage (default: 0.2)

null_C: The value of C Index under the null hypothesis of interest for bootstrap (default: 0.5)

method: character that takes value of “permutation” or “bootstrap”.

rep = 500: total number of iterations for the Monte Carlo Simulation

seedID: seed number for reproducing results (default: NULL)

two_stage_permutation()

Description

Analyze a given data set using the two-stage design. Inference in the first stage is conducted via permutation.

Arguments

survY: output from surv function in survival package

X: a matrix or data frame of variables of interest, without missing value

itrB: number of iteration used in permutation

seed: seed number for reproducing results (default: NULL)

nfold: number of folds used for cross-validation (default: 10)

S: proportion of samples used in first stage (default: 0.5)

alpha_1: level of test in first stage (default: 0.25)

alpha_2: level of test in second stage (default: 0.2)

printmodel: logical value. TRUE indicates the final signature will be print out (default: FALSE)

two_stage_bootstrap()

Description

Analyze a given data set using the two-stage design. Inference in the first stage is conducted via bootstrap.

Arguments

survY: output from surv function in survival package

X: a matrix or data frame of variables of interest, without missing value

itrB: number of iteration used in permutation

seed: seed number for reproducing results (default: NULL)

nfold: number of folds used for cross-validation (default: 10)

S: proportion of samples used in first stage (default: 0.5)

alpha_1: level of test in first stage (default: 0.25)

alpha_2: level of test in second stage (default: 0.2)

null_C : The value of C Index under the null hypothesis of interest (default: 0.5)

tuning_C()

Description

Function that helps tune the beta coefficients to hit the target level of C for the alternative hypothesis. For a given vector of beta, only the last coefficient will be tuned and all other coefficients will be fixed.

Arguments

nn: number of samples used as the population (default = 10000)

mean: parameter in generating the dependent variables (default = 0)

sd: parameter in generating the dependent variables (default = log(1.5))

rate_l: rate parameter in the Exponential Distribution that generates the Time of Event Occurs (default = 0.025)

p: number of covariates

initial_beta: a vector of beta coefficients to start with (for p and initial_beta, only of the two arguments is needed)

targetC: target value of C Index of interest

by, start, end: parameters that create a sequence of candidate values for the last coefficient. start: where the sequence starts; end: where the sequence ends; by: the increments for the sequence.

seedID: seed number for reproducing results (default: NULL)

tuning_censoring()

Function that helps tune the distribution of censoring time so that censoring hits the targete proportion.

Arguments

nn: number of samples used as the population (default = 1000)

mean: parameter in generating the dependent variables (default = 0)

sd: parameter in generating the dependent variables (default = log(1.5))

rate_l: rate parameter in the Exponential Distribution that generates the Time of Event Occurs (default = 0.025)

by, start, end: parameters for creating the sequence that can be used to tune the last coefficient. start: where the sequence starts; end: where the sequence ends; by: the increments for the sequence.

p: number of covariates

beta: a vector of beta coefficients

target: target value of censoring proportion

seed: seed number for reproducing results (default: NULL)

.cvFolds()

Lower level function that assigns fold IDs given number of folds.

cvcox()

Lower level function that calculates cross-validated linear predictors from Cox Regression

Examples

Example 2: Simulation Under the Null Hypothesis: C = 0.5

We first illustrate how to use the permutation version

```
source("functions.R")
```

```
result1 <- two_stage_sim(nn = 300, # sample size
              rate_l = 0.025, rate_c = 0.025,
              beta = rep(0,10),mean = 0, sd = log(1.5),
              nfold = 10,
              itrB = 200, #recommend >= 200
              S = 0.5,
              alpha_1 = 0.25, alpha_2 = 0.2,
              null_C = 0.5,
              method = "permutation",
              rep = 10, #recommend >= 500
              seedID = 999
)
head(result1)
```

```
##          survC threshold early_stop       C_2       se_2 final_validation
## [1,] 0.5825983 0.5388558          0 0.5142684 0.04230755                0
## [2,] 0.4709064 0.5316113          1        NA         NA                0
## [3,] 0.4957204 0.5362874          1        NA         NA                0
## [4,] 0.4377654 0.5307089          1        NA         NA                0
## [5,] 0.4698146 0.5282047          1        NA         NA                0
## [6,] 0.5383999 0.5228074          0 0.5715279 0.03922860                1
##           cens     cens1     cens2
## [1,] 0.5433333 0.5666667 0.5200000
## [2,] 0.4733333 0.4933333 0.4533333
## [3,] 0.4966667 0.5000000 0.4933333
## [4,] 0.4766667 0.4600000 0.4933333
## [5,] 0.4733333 0.4400000 0.5066667
## [6,] 0.4666667 0.4266667 0.5066667
```

For simplicity, we only specified 5 iterations for the Monte Carlo simulation. Usually we need >= 500 iterations to evaluate operation characteristics

To compute type one error rate:

```
mean(result1[,"final_validation"])
```

```
## [1] 0.2
```

To compute probability of early stopping:

```
mean(result1[,"early_stop"])
```

```
## [1] 0.5
```

We now illustrate how to use the Bootstrap version:

```
result2 <- two_stage_sim(nn = 300, # sample size
              rate_l = 0.025, rate_c = 0.025,
              beta = rep(0,10),mean = 0, sd = log(1.5),
              nfold = 10,
              itrB = 50, #recommend >= 200
              S = 0.5,
              alpha_1 = 0.25, alpha_2 = 0.2,
              null_C = 0.5,
              method = "bootstrap",
              rep = 5, #recommend >= 500
              seedID = 999
)
head(result2)
```

Example 3: Simulate Samples with the population C Index > 0.5

Generating data with C > 0.5 is less straignthfoward than generating data with C = 0.5. The difficulty comes from hitting the right level of the population C Index and also maintaining the correct proportion of censoring.

We provided two functions \textbf{tuning_C} and \textbf{tuning_censoring} to select the parameters that can generate the targeted level of C.

Step 1: Tune the Regression coefficients with the \textbf{tuning_C} function so that the Population C Index reaches a targeted value.

In this example, we aim at having the C Index to be **0.6**. We want to have **5** covariates in the model.

```
tuned_C <- tuning_C(nn = 10000,
                    mean = 0,
                    sd = log(1.5),
                    rate_l = 0.025,
                    p = 5,
                    by = 0.01,
                    start = 0.5,
                    end = 1.1,
                    targetC = 0.6,
                    seed = 123)
```

The first two outputs yield the values of the beta coefficients that should be used later:

```
tuned_C$beta
```

```
## [1] 0.2 0.2 0.2 0.2
```

```
tuned_C$beta_j
```

```
## [1] 0.83
```

Step 2: Tune Censoring Distribution with the \textbf{tuned_censor} function to Hit the Targeted Censoring Percentage

Given the beta coefficients produced by \textbf{tuned_C}, We need to tune the rate parameter in the censoring distribution so that 30 percent of the data is censored.

```
tuned_censor<- tuning_censoring(nn = 10000,
                                mean = 0,
                                sd = log(1.5),
                                rate_l = 0.025,
                                start = 0.001,
                                end = 0.025*0.3*(1-0.3) + 0.05,
                                by = 0.0001,
                                beta = c(tuned_C$beta,tuned_C$beta_j),
                                target = 0.3,
                                seed = 111)
tuned_censor
```

```
## $rate_c
## [1] 0.0101
## 
## $cpct
## [1] 0.3005
## 
## $beta
## [1] 0.20 0.20 0.20 0.20 0.83
## 
## $target
## [1] 0.3
## 
## $mean
## [1] 0
## 
## $rate_l
## [1] 0.025
## 
## $sd
## [1] 0.4054651
```

Here we take rate_c from the output as the rate parameter for the censoring distribution.

```
tuned_censor$rate_c
```

```
## [1] 0.0101
```

Step 3: Apply the tuned parameters in simulation:

We can use these data to conduct a simulation with the permutation version of the design. In this scenario, the null Hypothesis is C = 0.5. Since the data is generated under C = 0.6, we will be looking at the power of the study.

```
result_P <- two_stage_sim(nn = 500, # sample size
              rate_l = 0.025, rate_c = tuned_censor$rate_c,
              beta = c(tuned_C$beta,tuned_C$beta_j),mean = 0, sd = log(1.5),
              nfold = 10,
              itrB = 200, #recommend >= 200
              S = 0.5,
              alpha_1 = 0.25, alpha_2 = 0.2,
              method = "permutation",
              rep = 500, #recommend >= 500
              seedID = 999
              )
head(result_P)
```

To compute Power:

```
mean(result_P[,"final_validation"])
```

```
## [1] 0.892
```

To compute probability of Early Stopping:

```
mean(result_P[,"early_stop"])
```

```
## [1] 0.082
```

We can use these data to conduct a simulation with the bootstrap version. In this scenario, we can set the null Hypothesis to be C = 0.6. Since the data is generated under C = 0.6, we will be looking at the type one error rate of the study.

```
result_B <- two_stage_sim(nn = 500, # sample size
              rate_l = 0.025, rate_c =  tuned_censor$rate_c,
              beta = c(tuned_C$beta,tuned_C$beta_j),mean = 0, sd = log(1.5),
              nfold = 10,
              itrB = 200, #recommend >= 200
              S = 0.5,
              alpha_1 = 0.25, alpha_2 = 0.2,
              null_C = 0.6,
              method = "bootstrap",
              rep = 500, #recommend >= 500
              seedID = 999
              )
head(result_B)
```

To compute Type one error rate:

```
mean(result_B[,"final_validation"])
```

```
## [1] 0.018
```

To compute probability of Early Stopping:

```
mean(result_B[,"early_stop"])
```

```
## [1] 0.778
```