Supplementary Material for Two Stage Design for Survival Endpoint

Biyue Dai

July 22, 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_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)

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)

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 1: Analyze Real Data

```r
# load functions
source("functions.R")
# load data
data(lung) # built-in data set in survival package

lung <- na.omit(lung) # for simplicity, I only took the complete cases
survY <- Surv(lung$time, lung$status)
X <- lung[,c("age","sex","ph.ecog","ph.karno","meal.cal","wt.loss")]
two_stage_permutation(survY, X, itrB = 100, S =0.5, printmodel = TRUE)
```

```
## [[1]]
##         survC threshold.75%    early_stop           C_2          se_2 
##    0.65324963    0.52372600    0.00000000    0.59566929    0.04306117 
##  final_reject          cens         cens1         cens2 
##    1.00000000    0.28143713    0.27380952    0.28915663 
## 
## $final_model
## Call:
## coxph(formula = survY_1 ~ ., data = df)
## 
##                coef  exp(coef)   se(coef)      z       p
## age       2.339e-02  1.024e+00  1.549e-02  1.510 0.13105
## sex      -7.954e-01  4.514e-01  2.830e-01 -2.810 0.00495
## ph.ecog   5.871e-01  1.799e+00  3.666e-01  1.601 0.10927
## ph.karno -9.260e-03  9.908e-01  1.987e-02 -0.466 0.64122
## meal.cal -6.624e-05  9.999e-01  3.557e-04 -0.186 0.85227
## wt.loss  -1.798e-02  9.822e-01  1.022e-02 -1.759 0.07856
## 
## Likelihood ratio test=24.17  on 6 df, p=0.0004866
## n= 84, number of events= 61
```

```
two_stage_bootstrap(survY, X, itrB = 100, seed = 666, S =0.5)
```

```
##         survC threshold.25%    early_stop           C_2          se_2 
##    0.54031650    0.57047312    0.00000000    0.60646900    0.04470738 
##  final_reject          cens         cens1         cens2 
##    1.00000000    0.28143713    0.28571429    0.27710843
```

```
two_stage_bootstrap(survY, X, itrB = 100, seed = 666, S =0.5, null_C = 0.6)
```

```
##         survC threshold.25%    early_stop           C_2          se_2 
##     0.5403165     0.5704731     1.0000000            NA            NA 
##  final_reject          cens         cens1         cens2 
##     0.0000000     0.2814371     0.2857143     0.2771084
```

The output contains the following values:

survC: cross-validated C Index of the model built in Stage 1

threshold: the rejection threshold in permutation and bootstrap distribution

early_stop: indicator of whether the trial is stopped early or not. 1: stopped early; 2: continue to second stage

C_2: C Index in stage 2. It would be NA if the study is stopped early.

se_2: standard error of C Index in stage 2. It would be NA if the study is stopped early.

final_reject: Indicator of whether the signature is finally validated. 1: validated; 2: validation has failed.

cens, cens1, cens2: censoring proportion overall, for stage 1 and for stage 2.

Example 2: Simulation Under the Null Hypothesis

Permutation:

```
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 = 50, #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,] 0.4487941 0.5187871          1        NA         NA                0
## [2,] 0.4681314 0.5464667          1        NA         NA                0
## [3,] 0.4892921 0.5236636          1        NA         NA                0
## [4,] 0.5369328 0.5186862          0 0.5229261 0.03931562                0
## [5,] 0.4863903 0.5281089          1        NA         NA                0
## [6,] 0.4601738 0.5442161          1        NA         NA                0
##           cens     cens1     cens2
## [1,] 0.5066667 0.5000000 0.5133333
## [2,] 0.5066667 0.5466667 0.4666667
## [3,] 0.5066667 0.4733333 0.5400000
## [4,] 0.5066667 0.5133333 0.5000000
## [5,] 0.5066667 0.5000000 0.5133333
## [6,] 0.5066667 0.5066667 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] 0.1
```

To compute probability of early stopping:

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

```
## [1] 0.7
```

We can also use the same function for the Bootstrap:

```
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: Simulation Under the Alternative Hypothesis

Simulation under the alternative hypothesis is less straignthfoward than under the null. The difficulty comes from generating data that hit the right level of targeted C Index in the alternative hypothesis; and also maintain the correct proportion of censoring.

Step 1: Tune the correlation coefficients 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 **4** covariates in the model.

```
tuned_C <- tuning_C(nn = 10000,
         mean = 0,
         sd = log(2),
         rate_l = 0.025,
         p = 4,
         by = 0.01,
         start = 0.5,
         end = 1.5,
         targetC = 0.6,
         seed = 111)
tuned_C
```

```
## $beta
## [1] 0.1 0.1 0.1
##
## $beta_j
## [1] 0.51
##
## $C
## [1] 0.6
##
## $targetC
## [1] 0.6
##
## $mean
## [1] 0
##
## $rate_l
## [1] 0.025
##
## $sd
## [1] 0.6931472
```

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

```
tuned_C$beta
```

```
## [1] 0.1 0.1 0.1
```

```
tuned_C$beta_j
```

```
## [1] 0.51
```

Step 2: Tune Censoring Distribution to Hit Targeted Censoring Percentage

In this step, we take the vector of beta coefficients. We need to tune for 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(2),
                 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] 0.0104
##
## $cpct
## [1] 0.3012
##
## $beta
## [1] 0.10 0.10 0.10 0.51
##
## $target
## [1] 0.3
##
## $mean
## [1] 0
##
## $rate_l
## [1] 0.025
##
## $sd
## [1] 0.6931472
```

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

```
tuned_censor$rate_c
```

```
## [1] 0.0104
```

Step 3: Input the parameters obtained from Step 1 and 2 into the simulation function:

Permutation:

```
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 = 50, #recommend >= 200
              S = 0.5,
              alpha_1 = 0.25, alpha_2 = 0.2,
              method = "permutation",
              rep = 10, #recommend >= 500
              seedID = 999
              )
head(result_P)
```

```
##          survC threshold early_stop       C_2       se_2 final_validation
## [1,] 0.5542788 0.5245119          0 0.5505896 0.02582355                1
## [2,] 0.5405665 0.5186702          0 0.5565712 0.02584874                1
## [3,] 0.5570210 0.5231370          0 0.5415237 0.02664440                1
## [4,] 0.5421736 0.5124696          0 0.5655999 0.02530248                1
## [5,] 0.5574156 0.5107324          0 0.5492082 0.02407200                1
## [6,] 0.5411765 0.5209601          0 0.5521056 0.02558661                1
##       cens cens1 cens2
## [1,] 0.322 0.304 0.340
## [2,] 0.322 0.340 0.304
## [3,] 0.322 0.292 0.352
## [4,] 0.322 0.308 0.336
## [5,] 0.322 0.336 0.308
## [6,] 0.322 0.320 0.324
```

To compute Power:

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

```
## [1] 0.9
```

To compute probability of Early Stopping:

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

```
## [1] 0.1
```

Bootstrap:

```
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 = 50, #recommend >= 200
              S = 0.5,
              alpha_1 = 0.25, alpha_2 = 0.2,
              null_C = 0.55,
              method = "bootstrap",
              rep = 10, #recommend >= 500
              seedID = 999
              )
head(result_B)
```

```
##          survC threshold early_stop C_2 se_2 final_validation  cens cens1
## [1,] 0.5542788 0.5441762          1  NA   NA                0 0.322 0.304
## [2,] 0.5405665 0.5348054          1  NA   NA                0 0.322 0.340
## [3,] 0.5570210 0.5465522          1  NA   NA                0 0.322 0.292
## [4,] 0.5421736 0.5412734          1  NA   NA                0 0.322 0.308
## [5,] 0.5574156 0.5415776          1  NA   NA                0 0.322 0.336
## [6,] 0.5411765 0.5344045          1  NA   NA                0 0.322 0.320
##      cens2
## [1,] 0.340
## [2,] 0.304
## [3,] 0.352
## [4,] 0.336
## [5,] 0.308
## [6,] 0.324
```