BOOTSTRAPPING THE CROSS-VALIDATION ESTIMATE USING BOOTCV

The **BootCV** package for `R` provides functions of fast bootstrap methods to quantify the uncertainty of cross-validation estimate, which quickly estimates the standard error of the cross-validation estimate and produces valid confidence intervals for a population parameter measuring average model performance. The method overcomes the computational challenge inherent in bootstrapping the cross-validation estimate by estimating the variance component within a random effects model and is flexible for evaluating the performance of general predictive models. In this document, we introduce how to implement the R package to bootstrap the cross-validation estimates of general predictive models. Details about the implemented algorithms can be found in Cai et al. (2023).

1. Introduction.

Cross-validation is a widely used technique for evaluating the performance of predictive models, which involves splitting observed data D_n with n i.i.d. observations into a training set D_{train} and testing set D_{test} and using D_{test} to evaluate the performance of the model trained by D_{train}. The output of the training procedure is usually a parameter estimate $\psi(\widehat{D_{train}})$, which can be a finite-dimensional vector or infinite-dimensional function. The evaluation of its performance is usually conducted by calculating a summary statistic $L\left\{D_{test}, \hat{\psi}(D_{train})\right\}$ and repeat the aforementioned step many times to reduce the variability of random splits and the average summary statistic is obtained as the final cross-validation estimate of the model performance:

$$\widehat{Err}^{CV} = \frac{1}{B_{CV}} \sum_{b=1}^{B_{CV}} L\left\{D_{test}^b, \hat{\psi}(D_{train}^b)\right\},$$

where $D_n = D_{train}^b \cup D_{test}^b$ represents the bth split. Many cross-validation applications can fit into the very general framework.

In general, the resulting cross-validation estimate is a random quantity, dependent on both the random splitting of the data and the observed data itself. **BootCV** can accurately quantify the uncertainty associated with the estimate, which is especially important when comparing the performance of two models using cross-validation. Compared with existing methods to make inferences on cross-validation estimates, **BootCV** has fewer restrictions on predictive models and is more computationally efficient.

This document is organized into three main sections (Sections 2,3 and 4). Before introducing the details of this package, we present a simple example of implementing **BootCV** to help users get a better understanding (Section 2). Then we describe the usage of functions in **BootCV** in detail (Section 3) and present the PEACE study as an example of analyzing real data using **BootCV** (Section 4).

2. Quick Start.

In this section, we give users a general sense of **BootCV** with a basic example of using this package to generate a cross-validation estimate and the corresponding confidence interval. We will briefly go over the key steps, basic operations, inputs and outputs.

First we need to load **BootCV** (and dependent R package **lme4** (Bates, 2014)):

```
# library(lme4)
library(BootCV)
```

We generate the observed data D_n as a $n \times p$ matrix, where n is the number of observations and each observation is a p-dimensional vector:

```
n <- 90
p <- 10
x <- matrix(rnorm(n * p), ncol = p)
beta <- rnorm(p)
y <- x %*% beta + rnorm(n)
data <- cbind(y, x)
```

Users need to specify the summary statistics L for evaluating the model performance, which is calculated as a function of training data and testing data. Here we use the mean square error in linear regression as an example:

```
L <- function(train.data, test.data) {
  y <- train.data[, 1]
  x <- train.data[, -1]
  yt <- test.data[, 1]
  xt <- test.data[, -1]

  fit <- lm(y ~ x)
  beta <- fit$coef

  return(mean((yt - cbind(1, xt) %*% beta)^2))
}
```

Both `train.data` and `test.data` are in the same form of data matrix as `data` and `data = rbind(train.data, test.data)`. After specifying the sample size of training set m in cross-validation, we can use the function `Boot.CV` to estimate the standard error of cross-validation estimate of summary statistics L:

```
m <- 50 # training set size
boot = Boot.CV(data, L, m)
# result with adjustment
result1 = CV.confint(boot,data,L,m,method='Boot.CV',adj=T,print=T)
# result without adjustment
result2 = CV.confint(boot,data,L,m,method='Boot.CV',adj=F,print=T)
```

where `boot` is a list that contains all the relevant information for calculating the cross-validation estimate and confidence interval. Users can adjust the number of bootstraps, the number of cross-validations and the tuning parameter in determining the adjusted sample size of training set flexibly. Based on `boot`, users can use the function `CV.confint` to generate cross-validation estimation and confidence interval. Users can determine whether to adjust for the reduced training sample size and whether to print the results with the bool inputs `adj` and `print` and confidence level with `alpha`.

Sometimes, training the predictive model can be very expensive in terms of computation, and it may not be feasible to conduct many times bootstraps. In this case, users may consider using the function `Boot.Cal`. One thing to note is that we need to specify the confidence level when using `Boot.Cal` and keep it the same as that used in `CV.confint` (default choice is 0.95).

```
m <- 50 # training set size
boot = Boot.Cal(data, L, m)
# result with adjustment
result1 = CV.confint(boot,data,L,m,method='Boot.Cal',adj=T,print=T)
# result without adjustment
result2 = CV.confint(boot,data,L,m,method='Boot.Cal',adj=F,print=T)
```

3. Package Usage.

3.1. *Functions.* **BootCV** consists of three main functions: `Boot.CV` and `Boot.Cal` to implement the algorithms and `CV.confint` to generate the results.

```
Boot.CV(data, L, m, B.bt = 400, B.cv = 20, lambda0 = 0.368)
```

`Boot.CV` implements the proposed algorithm (algorithm 2 in the reference paper) and returns the bootstrap standard error of the cross-validation estimate and inflation factor. `data` is a $n \times p$ data matrix, which consists of n observations and each observation is a p-dimensional vector. `L` is a function with two inputs, `train.data` and `test.data`, which are in the same form as `data`. It is the summary statistics specified by the user to evaluate the performance of the model fitted with `train.data` in `test.data`. The construction of summary statistics varies across different settings of predictive models and more examples are given in the following subsection. `m` is the size of training set, `B.bt` is the number of bootstraps (default choice is 400) and `B.cv` is the number of cross-validations (default choice is 20). `lambda0` is the tuning parameter to determine the adjusted sample size of training set (default choice is 0.368), which leads to a desirable balance between effective training sample size and testing sample size. `Boot.CV` returns a list with the bootstrap standard error of cross-validation estimate `sigma` and inflation factor to account for the reduced sample size in bootstrapped training set `factor`.

```
Boot.Cal(data, L, m, B.bt = 20, B.cv = 50, Brm.bt = 1000, alpha = 0.05,

lambda0 = 0.368)
```

`Boot.Cal` implements the proposed algorithm with a small number of model fittings. Besides the parameters in `Boot.CV`, `Boot.Cal` may need additional parameters, such as the number of bootstraps for the variance estimator `Brm.bt` (default choice is 1000) and 1-confidence level `alpha` (default choice is 0.05, consistent with that in `CV.confint`). `Boot.Cal` returns a list with the bootstrap standard error of cross-validation estimate `sigma`, the cutoff for the confidence interval `cutoff` and the inflation factor to account for the reduced sample size in bootstrapped training set `factor`.

```
CV.confint(boot, data, L, m, method = c("Boot.CV", "Boot.Cal"),

adj = TRUE, B.cv = 400, alpha = 0.05, print = FALSE)
```

Based on the list `boot` obtained from `Boot.CV` or `Boot.Cal`, `CV.confint` returns the final cross-validation estimation and confidence interval. Users need to specify which algorithm they use (`Boot.CV` or `Boot.Cal`) with `method`, whether to adjust for the reduced training sample size with bool value `adj` (default choice is TRUE) and whether to print the result with bool value `print` (default is TRUE).

3.2. *Examples of Summary Statistics.* This subsection provides several typical examples of how to construct the summary statistics L in different settings:

3.2.1. *Example 1 (Precision Medicine).* When the predictive model is a precision medicine strategy which is a binary classification rule to recommend treatment to a patient based on his or her baseline characteristics to maximize the treatment benefit, we can construct an individualized treatment response (ITR) score by minimizing a loss function based on a training dataset D_{train}, (Tian et al., 2014)

$$\frac{1}{m}\sum_{X_i\in D_{train}}\left\{Y_i-\gamma'\tilde{Z}_i-(G_i-\pi)\beta'\tilde{Z}_i\right\}^2, \tag{1}$$

where $X_i=(Z_i,G_i,Y_i)$, Y_i is the response of interest with a higher value being desirable, Z_i is the baseline covariate, $\tilde{Z}_i$ is its counterpart including an intercept, $G_i\in\{0,1\}$ is a binary treatment indicator and independent of Z_i (i.e., the treatment is randomly assigned to patients in the training set), and $\pi=\mathrm{pr}(G_i=1)$. Let $\hat{\gamma}(D_{train})$ and $\hat{\beta}(D_{train})$ be the minimizer of (1). The sample average of cross-validated treatment effect estimators is our final cross-validation estimate measuring the performance of the treatment recommendation system and the summary statistics measuring the prediction performance is:

$$L\left(D_{test},\psi\right)=\frac{\sum_{X_i\in D_{test}}Y_iG_iI(\psi'Z_i>0)}{\sum_{X_i\in D_{test}}G_iI(\psi'Z_i>0)}-\frac{\sum_{X_i\in D_{test}}Y_i(1-G_i)I(\psi'Z_i>0)}{\sum_{X_i\in D_{test}}(1-G_i)I(\psi'Z_i>0)},$$

where $\hat{\psi}(D_{train})=\hat{\beta}(D_{train})$.

3.2.2. *Example 2 (Binary Outcomes).* When the predictive model is constructed via fitting a logistic regression model, i.e., calculating a regression coefficient vector $\hat{\beta}(D_{train})$ by maximizing the log-likelihood function

$$\sum_{(Z_i,Y_i)\in D_{train}}\left[\beta'\tilde{Z}_iY_i-\log\left\{1+\exp(\beta'\tilde{Z}_i)\right\}\right],$$

based on a training dataset D_{train}, where Z_i is the predictor, $\tilde{Z}_i=(1,Z_i')'$, and $Y_i\in\{0,1\}$ is the binary outcome. If the dimension of Z_i is high, a lasso-regularization can be used in estimating β. In any case, the c-index in a testing set D_{test} can be calculated as

$$\hat{\theta}(D_{train},D_{test})=\frac{1}{\tilde{n}_{test,0}\tilde{n}_{test,1}}\sum_{X_i\in D_{test}(0)}\sum_{X_j\in D_{test1}(1)}I\left(\hat{\beta}(D_{train})'\tilde{Z}_i<\hat{\beta}(D_{train})'\tilde{Z}_j\right),$$

where $\tilde{n}_{test,g}$ is the number of observations in the set $D_{test}(g)=\{X_i=(Z_i,Y_i)\in D_{test}:Y_i=g\},g\in\{0,1\}$. The sample average of cross-validated c-index estimators is our final cross-validation estimate measuring the classification performance of the logistic regression and the summary statistics measuring the prediction performance is:

$$L\left(D_{test},\psi\right)=\frac{1}{\tilde{n}_{test,0}\tilde{n}_{test,1}}\sum_{X_i\in D_{test}(0)}\sum_{X_j\in D_{test}(1)}I\left(\psi'\tilde{Z}_i<\psi'\tilde{Z}_j\right),$$

where $\hat{\psi}(D_{train})=\hat{\beta}(D_{train})$, which can also be fitted with other methods such as random forest.

3.2.3. *Example 3 Mean Absolute Prediction Error (MAPE).* When the predictive model is constructed by fitting a standard linear regression model, i.e., calculating a regression coefficient vector $\hat{\beta}(D_{train})$ by minimizing a L_2 loss function

$$\sum_{X_i \in D_{train}} \left(Y_i - \beta' \widetilde{Z}_i\right)^2,$$

based on a training dataset D_{train}, where Z_i is the baseline covariate for the ith patient and $\widetilde{Z}_i = (1, Z_i')'$ including an intercept. When the prediction error in testing set D_{test} is calculated by mean absolute prediction error (Tian et al., [2007](https://doi.org/10.1093/biomet/asm036))

$$\hat{\theta}(D_{train}, D_{test}) = \frac{1}{n_{test}} \sum_{X_i \in D_{test}} |Y_i - \hat{\beta}(D_{train})' \tilde{Z}_i|,$$

where n_{test} is the number of observations in the testing set. The sample average of cross-validated mean absolute prediction error estimators is our final estimator measuring the predictive performance of the linear model and the summary statistics measuring the prediction performance is:

$$L\left(D_{test}, \psi\right) = \frac{1}{n_{test}} \sum_{X_i \in D_{test}} |Y_i - \psi' \widetilde{Z}_i|,$$

where $\hat{\psi}(D_{train}) = \hat{\beta}(D_{train})$. If nonlinear predictive models are considered, then one may construct the predictive model via a more flexible machine learning algorithm such as the random forest or neural network and the summary statistic is similar.

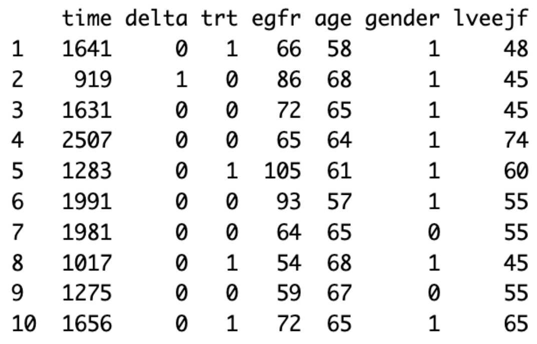
4. Real Data Analysis.

In this section, we will use **BootCV** to analyze the real data from the clinical trial “Prevention of Events with Angiotensin Converting Enzyme Inhibition” (PEACE) (Investigators, 2004). The PEACE trial was designed to examine the effect of ACEi on reducing future cardiovascular events in patients with stable coronary artery disease and normal or slightly reduced left ventricular function. It was inconclusive whether ACEi therapy can reduce mortality in the entire study population but ACEi has been reported to still be effective in reducing future cardiovascular risk in selected patients (Solomon et al., 2006). Therefore, it is desirable to identify a high-value subgroup of patients who may benefit from ACEi and build a scoring system to capture the individualized treatment effect. In the PEACE example, we derived a precision medicine strategy recommending ACEi use and evaluated its performance with **BootCV**.

First, we need to import **BootCV** with other needed packages and read the peace dataset. While a total of 8290 patients are enrolled in the study, we focus on a subgroup of 7865 patients with complete covariate information, in which 3947 and 3603 patients were assigned to receive ACEi and placebo, respectively. To build a candidate scoring system capturing the ITR, we used 4 baseline covariates previously identified as statistically and clinically important predictors of overall mortality (Solomon et al., 2006): `age`, `gender`, eGFR for renal function (`egfr`), and left ventricular ejection fraction (`lveejf`). `time` represents the survival time, `delta` represents the censoring status and `trt` represents the treatment indicator.

```
library(survival)
library(survRM2)
library(lme4)
library(BootCV)

data <- read.csv("peace.data.csv", head=T)
peace_data <- data[,2:8]
```



```
   time delta trt egfr age gender lveejf
1  1641     0   1   66  58      1     48
2   919     1   0   86  68      1     45
3  1631     0   0   72  65      1     45
4  2507     0   0   65  64      1     74
5  1283     0   1  105  61      1     60
6  1991     0   0   93  57      1     55
7  1981     0   0   64  65      0     55
8  1017     0   1   54  68      1     45
9  1275     0   0   59  67      0     55
10 1656     0   1   72  65      1     65
```

Fig 1: The peace trial data preview.

Specifically, separate Cox proportional hazards models were fitted in the ACEi and placebo arms, and the between-group difference in estimated restricted mean survival time (RMST) was used to compute the ITR score. The calculation of RMST can be conducted by the following function:

```
rmst1fun <- function(time, status, tau)
{fit <- survfit(Surv(time, status)~1)
 time0 <- fit$time
 surv0 <- fit$surv
 time0 <- c(0, time0)
 surv0 <- c(1, surv0)

 surv0 <- surv0[time0<tau]
 time0 <- time0[time0<tau]

 surv0 <- c(surv0, min(surv0))
 time0 <- c(time0, tau)

 rmst <- sum(surv0[-length(surv0)]*(time0[-1]-time0[-length(surv0)]))
 return(rmst)
}
```

We considered the average treatment effect (ATE) as the hazard ratio and the summary statistics to evaluate the performance of the precision medicine strategy can be specified as follows.

```
L <- function(train.data, test.data) {
  # training set
  y <- train.data$time
  y[y==0] <- 1
  d <- train.data$delta
  G <- train.data$trt
  x <- as.matrix(train.data[,4:7])

  # testing set
  yt <- test.data$time
  yt[yt==0] <- 1
  dt <- test.data$delta
  Gt <- test.data$trt
  xt <- as.matrix(test.data[,4:7])

  tau <- 2000
  fit1 <- coxph(Surv(y, d)~x, subset=(G==1))
  fit1.detail <- coxph.detail(fit1)
  fit0 <- coxph(Surv(y, d)~x, subset=(G==0))
  fit0.detail <- coxph.detail(fit0)

  time1 <- fit1.detail$time
  hazard1 <- fit1.detail$hazard
  surv1 <- exp(-cumsum(hazard1))
  mu1 <- apply(x[G==1,], 2, mean)
  beta1 <- fit1$coef

  scoren1 <- exp(t(t(xt)-mu1)%*%beta1)
  m1 <- length(scoren1)
  rmst1 <- rep(NA, m1)
  for(i in 1:m1)
  {surv <- c(1, surv1^scoren1[i])
  time <- c(0, time1)
  surv <- c(surv[time<=tau], min(surv[time<=tau]))
  time <- c(time[time<=tau], tau)
  rmst1[i] <- sum(surv[-length(surv)]*(time[-1]-time[-length(surv)]))
  }

  time0 <- fit0.detail$time
  hazard0 <- fit0.detail$hazard
  surv0 <- exp(-cumsum(hazard0))
  mu0 <- apply(x[G==0,], 2, mean)
  beta0 <- fit0$coef

  scoren0 <- exp(t(t(xt)-mu0)%*%beta0)
  m0 <- length(scoren0)
  rmst0 <- rep(NA, m0)
  for(i in 1:m0)
  {surv <- c(1, surv0^scoren0[i])
  time <- c(0, time0)
  surv <- c(surv[time<=tau], min(surv[time<=tau]))
  time <- c(time[time<=tau], tau)
  rmst0[i] <- sum(surv[-length(surv)]*(time[-1]-time[-length(surv)]))
  }

  Dt <- rmst1-rmst0
  D0 <- median(Dt)
  hr1 <- exp(coxph(Surv(yt, dt)~Gt, subset=(Dt>=D0))$coef)
  hr0 <- exp(coxph(Surv(yt, dt)~Gt, subset=(Dt<D0))$coef)

  return(hr1/hr0)
}
```

Based on the summary statistics, we derive the ITR score from a training set of size m = 6292, which represents 80% of the entire study population. With 500 cross-validations, the cross-validated estimate of hazard ratio $\widehat{Err}_m^{CV}$ was 0.8 in the high-value subgroup, and 1.18 in the complement of the high-value subgroup. Their ratio was 0.68. Now, we use `BootCV` to estimate the standard errors of these estimators and construct the confidence interval with $(B_{BOOT}, B_{CV}) = (500, 20)$.

```
m <- round(nrow(data)*4/5)
boot <- Boot.CV(peace_data, L, m, B.bt = 500, B.cv = 20)
result <- CV.confint(boot, peace_data, L, m, method = "Boot.CV",
B.cv = 500, print = TRUE)
```

The 95% CI of the hazard ratio in the high-value subgroup was [0.66, 0.98] and that in the complement of the high-value subgroup was [0.85, 1.65]. The 95% CI of their ratio was [0.44, 1.04], suggesting that the ITR score had a statistically significant interaction with the treatment, i.e. the ATE in the high-value subgroup was higher than in the remaining patients. Similarly, we can also consider other measurements of ATE such as the RMST difference.

REFERENCES

BATES, D. (2014). Fitting linear mixed-effects models using lme4. *arXiv preprint arXiv:1406.5823*.

CAI, B., PELLEGRINI, F., PANG, M., DE MOOR, C., SHEN, C., CHARU, V. and TIAN, L. (2023). Bootstrapping the Cross-Validation Estimate. *arXiv preprint arXiv:2307.00260*.

INVESTIGATORS, P. T. (2004). Angiotensin-converting–enzyme inhibition in stable coronary artery disease. *New England Journal of Medicine* **351** 2058–2068.

SOLOMON, S. D., RICE, M. M., A. JABLONSKI, K., JOSE, P., DOMANSKI, M., SABATINE, M., GERSH, B. J., ROULEAU, J., PFEFFER, M. A. and BRAUNWALD, E. (2006). Renal function and effectiveness of angiotensin-converting enzyme inhibitor therapy in patients with chronic stable coronary disease in the Prevention of Events with ACE inhibition (PEACE) trial. *Circulation* **114** 26–31.

TIAN, L., ALIZADEH, A. A., GENTLES, A. J. and TIBSHIRANI, R. (2014). A simple method for estimating interactions between a treatment and a large number of covariates. *Journal of the American Statistical Association* **109** 1517–1532.