Shortest

Yingchao Zhou

2022-05-23

```
library(lava)

#' Calculate the upper quantile for data with weights
#'
#' @param v the data that we want to find quantile
#' @param prob find the upper 100*prob% quantile
#' @param w weight for data
#' @param sorted whether v has been sorted
#'
#' @export
upper.quantile = function(v, prob, w=NULL, sorted=FALSE) {
  if (is.null(w)) w = rep(1,length(v))
  if (!sorted) { o = order(v); v = v[o]; w = w[o] }
  i = which(cumsum(w/sum(w)) >= prob)
  if (length(i)==0) return(Inf)
  else return(v[min(i)])
}

#' Calculate the lower quantile for data with weights
#'
#' @param v the data that we want to find quantile
#' @param prob find the lower 100*prob% quantile
#' @param w weight for data
#' @param sorted whether v has been sorted
#'
#' @export
lower.quantile = function(v, prob, w=NULL, sorted=FALSE) {
  if (is.null(w)) w = rep(1,length(v))
  if (!sorted) { o = order(v); v = v[o]; w = w[o] }
  i = which(cumsum(w/sum(w)) <= prob)
  if (length(i)==0) return(-Inf)
  else return(v[max(i)])
}

#' Calculate shortest prediction interval
#'
#' @param x data
#' @param alpha nominal error rate
#'
#' @export
shortest.pi = function(x, alpha){
  alpha_iter = seq(alpha/50, alpha, alpha/50)
```

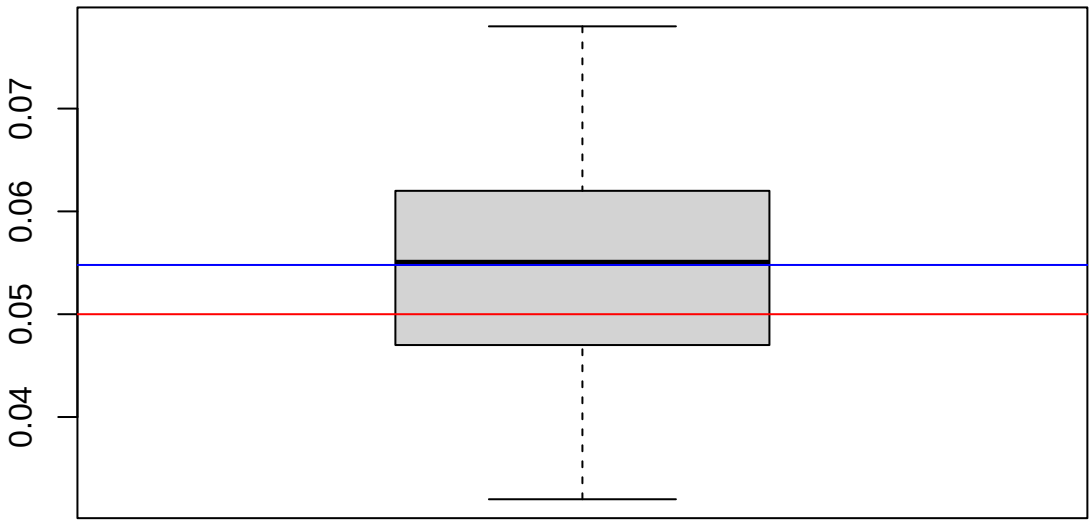
```
  j <- 1
  prev_length = Inf
  find_finite  = 0
  #Find the quantiles producing the shortest 1-alpha interval
  while(j < length(alpha_iter)){
    lower_q = lower.quantile(c(x, Inf), prob = alpha_iter[j], w = rep(1, length(x)+1), sorted = FALSE)
    upper_q = upper.quantile(c(x, Inf), prob = alpha_iter[j] + (1 - alpha), w = rep(1, length(x)+1), so
    if((upper_q-lower_q) < prev_length){
      find_finite = 1
      shortest_lower_q = lower_q
      shortest_upper_q = upper_q
      prev_length = upper_q-lower_q
    }
    j = j+1
  }
  if(find_finite){
    PI = c(shortest_lower_q, shortest_upper_q)
  }else {
    stop("No finite interval at given coverage rate")
  }
  return(PI)
}

#' Cross validation for the boost of shortest prediction interval
#'
#' @param x data
#' @param alpha nominal error rate
#' @param K fold
#' @importFrom lava foldr
#'
#' @export
cv.pi = function(x, alpha, K = 2){
  n = length(x)
  t.iter = seq(0, min(1.12*sqrt(alpha/n), alpha - 4/n),
               by = min(1.12*sqrt(alpha/n), alpha - 4/n)/10)
  folds = foldr(n, K, rep=1)
  pre.cov = 0
  best.rate = t.iter[1]
  for(j in 1:length(t.iter)){
    nominal.rate = alpha - t.iter[j]
    cov.k = 0
    for(k in 1:K){
      fold = as.vector(folds[[1]][[k]])
      xtest = x[fold]
      xtrain  = x[-fold]
      PI = shortest.pi(xtrain, nominal.rate)
      cov.k = cov.k + sum(PI[1]<= xtest & PI[2] >=xtest)
    }
    cov.k  = cov.k/n
    if(abs(cov.k-1+alpha)< abs(pre.cov-1+alpha)){# more accurate
      best.rate = t.iter[j]
      pre.cov = cov.k
    }
```

```
  }
  PI = shortest.pi(x, alpha-best.rate)
  return(PI)
}
```

Shortest prediction interval: See the Data-driven nonparametric prediction intervals.pdf. For a skewed distribution, the equal-tailed prediction interval is usually not the shortest. For given miscoverage rate α, a natural idea is to search among the intervals with left tail β and right tail $\alpha - \beta$, where $\beta \in (0, \alpha)$. However, this search results in undercoverage.

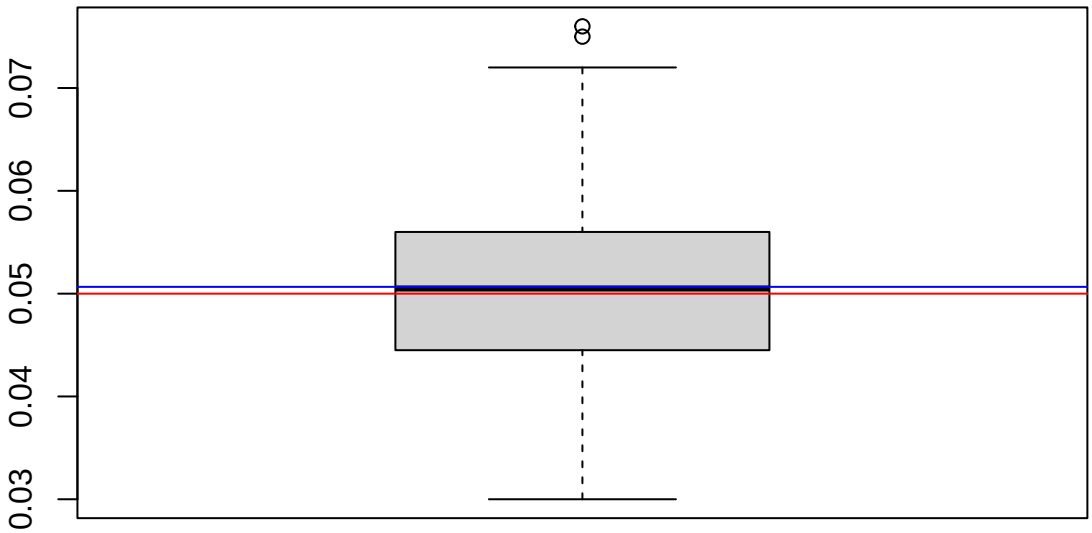
With no correction on the nominal error rate, the prediction interval is going to undercover:

```
n = 1000 ## training sample size
n0 = 1000 ## test sample size
alpha = 0.05 ## nonimal error rate
alpha_iter = seq(alpha/50, alpha, alpha/50) ## the left tails we will search
cov.box = c()

for(i in 1:100){
  x = rnorm(n, 0, 1)  ## training data
  x0 = rnorm(n0, 0, 1) ## test data

  j <- 1
  prev_length = Inf
  #Find the quantiles producing the shortest 1-alpha interval
  while(alpha_iter[j]<alpha){
    lower_q = lower.quantile(c(x, Inf), prob = alpha_iter[j], w = rep(1, length(x)+1), sorted = FALSE)
    upper_q = upper.quantile(c(x, Inf), prob = alpha_iter[j] + (1 - alpha), w = rep(1, length(x)+1), so
    if((upper_q-lower_q) < prev_length){
      shortest_lower_q = lower_q
      shortest_upper_q = upper_q
      prev_length = upper_q-lower_q
    }
    j = j+1
  }

  coverage.x0 = 1-mean(shortest_lower_q<= x0 & shortest_upper_q >=x0)
  cov.box = c(cov.box, coverage.x0)
}

boxplot(cov.box, main = paste(expression(alpha), '=', alpha, 'n=', n))
lines(x = 0:5, y = rep(alpha,6), col = 'red')
lines(x = 0:5, y = rep(mean(cov.box),6), col = 'blue')
```



Illustrate the dependence on "searching for the shortest interval" and "undercover": if we choose a interval position at random, it's not going to be the shortest, but it's valid in coverage.

```
n = 1000
n0 = 1000
alpha = 0.05
cov.box = c()

for(i in 1:100){
  x = rnorm(n, 0, 1)
  x0 = rnorm(n0, 0, 1)

  beta = runif(1, min = 0, max = alpha)

  lower_q = lower.quantile(c(x, Inf), prob = beta, w = rep(1, length(x)+1), sorted = FALSE)
    upper_q = upper.quantile(c(x, Inf), prob = beta + (1 - alpha), w = rep(1, length(x)+1), sorted = FA


  coverage.x0 = 1-mean(lower_q<= x0 & upper_q >=x0)
  cov.box = c(cov.box, coverage.x0)
}

boxplot(cov.box, main = paste(expression(alpha), '=', alpha, 'n=', n))
lines(x = 0:5, y = rep(alpha,6), col = 'red')
lines(x = 0:5, y = rep(mean(cov.box),6), col = 'blue')
```



Our cross validation method

```
n = 1000
n0 = 1000
alpha = 0.05
K = 2
alpha_iter = seq(alpha/50, alpha, alpha/50)
cov.box = c()

for(i in 1:100){
  x = rnorm(n, 0, 1)
  x0 = rnorm(n0, 0, 1)

  pre.cov = 0
  beta = alpha/2
  #Find the quantiles producing the shortest 1-alpha interval through cross-validation
  folds = foldr(length(x), K, rep=1)
  for(k in 1:K){
    fold = as.vector(folds[[1]][[k]])
    xtest = x[fold]
    xtrain  = x[-fold]
    j <- 1
    prev_length = Inf
    while(alpha_iter[j]<alpha){
    lower_q = lower.quantile(c(xtrain, Inf), prob = alpha_iter[j], w = rep(1, length(xtrain)+1), sorted
    upper_q = upper.quantile(c(xtrain, Inf), prob = alpha_iter[j] + (1 - alpha), w = rep(1, length(xtra
    if((upper_q-lower_q) < prev_length){
      shortest_lower_q = lower_q
      shortest_upper_q = upper_q
      prev_length = upper_q-lower_q
      k.beta = alpha_iter[j]
    }
    j = j+1
    }
```

```
    k.cov = mean(shortest_lower_q<= xtest & shortest_upper_q >=xtest)
    if(k.cov>pre.cov){
      pre.cov = k.cov
      beta = k.beta
    }
  }

  shortest_lower_q = lower.quantile(c(x, Inf), prob = beta, w = rep(1, length(x)+1), sorted = FALSE)
  shortest_upper_q = upper.quantile(c(x, Inf), prob = beta + (1 - alpha), w = rep(1, length(x)+1), sort
  coverage.x0 = 1-mean(shortest_lower_q<= x0 & shortest_upper_q >=x0)
  cov.box = c(cov.box, coverage.x0)
}

boxplot(cov.box, main = paste(expression(alpha), '=', alpha, 'n=', n))
lines(x = 0:5, y = rep(alpha,6), col = 'red')
lines(x = 0:5, y = rep(mean(cov.box),6), col = 'blue')
```