Simulations - Data Generation and Analysis

Nir Keret

2023-03-13

Load the required packages and the rcpp file:

```
library(survival)
library(MASS)
library(Rcpp)
library(Epi)
library(multipleNCC)
library(msm)
sourceCpp("cox-subsampling.cpp", verbose=TRUE)
```

Define the estimating function that will be used for the subsampling:

```
build_score = function(ret)
{
  tmp = ret$hess
  ret_hess = tmp + t(tmp)
  diag(ret_hess) = diag(tmp)
  ret[["hess"]] = -ret_hess
  return(ret)
}

information_score_matrix = function(beta,weights=NULL,times,truncs=NULL,status,covariates,
                                    samp_prob=NULL,information_mat=T,score_res_mat=T) {
  if(is.null(weights)) {weights = c(-1)}
  if(is.null(truncs)) {truncs = c(-1)}

  ret = rcpp_score_wrapper(beta,weights,times,truncs,status,covariates,
                           1*information_mat,1*score_res_mat)
  if(information_mat)
  {
    ret = build_score(ret)
  }

  if(is.null(samp_prob)) return(ret)
  if(samp_prob == "A")
  {
    ret[["samp_prob"]] = rcpp_A_OPT(ret[["residual"]],solve(ret[["hess"]]))
  }
  if(samp_prob == "L")
  {
    ret[["samp_prob"]] = rcpp_L_OPT(ret[["residual"]])
  }

  return(ret)
```

```
}

CoxSubsample = function(V, D, X, R=NULL, q, q0 = q, method)
{
  if(!(method %in% c("U","L","A")))
  {
    stop("method has to be one of U, L or A")
  }
  n = length(V)
  cens_ind = which(D == 0)
  n_cens = length(cens_ind)
  n_events = n - n_cens

  #uniform sampling
  unif_cont_ind = sample(cens_ind,q0,replace = T)  #sampling q0 censored observations
  samp_ind_unif = c(unif_cont_ind,setdiff(1:n,cens_ind)) #joining  censored with all events
  cens_weights_unif = length(cens_ind)/ q0
  weights_unif = ifelse(D,1,cens_weights_unif)

  if(is.null(R))
  {
    fit_samp_unif = coxph(Surv(time = V[samp_ind_unif], event = D[samp_ind_unif]) ~
                            X[samp_ind_unif,],weights = weights_unif[samp_ind_unif],
                          robust = F)
  }else
  {
    fit_samp_unif = coxph(Surv(R[samp_ind_unif],v[samp_ind_unif],delta[samp_ind_unif],
                               type = "counting") ~ X[samp_ind_unif,],
                          weights = weights_unif[samp_ind_unif],robust = F)
  }
  U_coef = coef(fit_samp_unif)
  names(U_coef) = colnames(X)
  if(method == "U")
  {
    if(is.null(R))
    {
      tmpU = information_score_matrix(U_coef,weights = weights_unif[samp_ind_unif],
                                      times = v[samp_ind_unif],status = delta[samp_ind_unif],
                                      covariates = X[samp_ind_unif,])
    }else
    {
      tmpU = information_score_matrix(U_coef,weights = weights_unif[samp_ind_unif],
                                      times = v[samp_ind_unif],truncs = R[samp_ind_unif],
                                      status = delta[samp_ind_unif],
                                      covariates = X[samp_ind_unif,])
    }

    Score_U = tmpU$residual * n_cens
    phi_mat_U = cov(Score_U)
    I_inv_U = solve(tmpU$hess)
    var_unif = I_inv_U + I_inv_U %*% phi_mat_U %*% I_inv_U/q0
    ret = list("coef" = U_coef, "var" = var_unif)
    return(ret)
```

```r
}
if(method == "L")
{
  if(is.null(R))
  {
    tmp_L1 = information_score_matrix(U_coef,times = V,status = D,covariates = X,
                                      samp_prob = "L",information_mat = F)
  }else
  {
    tmp_L1 = information_score_matrix(U_coef,times = V,truncs = R,status = D,
                                      covariates = X, samp_prob = "L",information_mat = F)
  }
  D_L = D[tmp_L1$ord]
  cens_ind_ord = which(!D_L)

  # random sampling with L-optimal probabilities from censored
  #sampling q censored observations:
  samp_ind_cens = sample(1:n_cens,q,replace = T,prob = tmp_L1$samp_prob)
  #join the sampled censored with the failure times
  samp_ind_opt = c(cens_ind_ord[samp_ind_cens],which(D_L))
  cens_weights_opt = (1/(tmp_L1$samp_prob * q))[samp_ind_cens]
  weights_opt = c(cens_weights_opt,rep(1,n_events))

  samp_ord = tmp_L1$ord[samp_ind_opt]

  if(is.null(R))
  {
    fit_samp_opt_L = coxph(Surv(time=V[samp_ord],event=D[samp_ord]) ~ X[samp_ord,],
                           weights = weights_opt,robust = F,init = U_coef)
    tmp_L2 = information_score_matrix(coef(fit_samp_opt_L),weights = weights_opt,
                                      times = V[samp_ord],status = D[samp_ord],
                                      covariates = X[samp_ord,])
  }else
  {
    fit_samp_opt_L = coxph(Surv(R[samp_ord],V[samp_ord],D[samp_ord],type = "counting")
                           ~ X[samp_ord,],weights = weights_opt,robust = F,init = U_coef)
    tmp_L2 = information_score_matrix(coef(fit_samp_opt_L),weights = weights_opt,
                                      truncs = R[samp_ord],times = V[samp_ord],
                                      status = D[samp_ord], covariates = X[samp_ord,])
  }
  L_coef = coef(fit_samp_opt_L)
  names(L_coef) = colnames(X)
  ind_rm = (1:n_events)+q
  order_rm = tmp_L2$ord[!(tmp_L2$ord %in% ind_rm)]
  Score_L = tmp_L2$residual / tmp_L1$samp_prob[samp_ind_cens][order_rm]
  phi_mat_L = cov(Score_L)
  I_inv_L = solve(tmp_L2$hess)
  var_opt_L = I_inv_L + I_inv_L %*% phi_mat_L %*% I_inv_L/q
  ret = list("coef" = L_coef, "var" = var_opt_L)
  return(ret)
}
if(method == "A")
{
```

```r
    if(is.null(R))
    {
      tmp_A1 = information_score_matrix(U_coef,times = V,status = D,covariates = X,
                                        samp_prob = "A")
    }else
    {
      tmp_A1 = information_score_matrix(U_coef,times = V,truncs = R,status = D,
                                        covariates = X, samp_prob = "A")
    }
    D_A = D[tmp_A1$ord]
    cens_ind_ord = which(!D_A)

    # random sampling with A-optimal probabilities from censored
    #sampling q censored observations:
    samp_ind_cens = sample(1:n_cens,q,replace = T,prob = tmp_A1$samp_prob)
    #join the sampled censored with the failure times:
    samp_ind_opt = c(cens_ind_ord[samp_ind_cens],which(D_A))
    cens_weights_opt = (1/(tmp_A1$samp_prob * q))[samp_ind_cens]
    weights_opt = c(cens_weights_opt,rep(1,n_events))

    samp_ord = tmp_A1$ord[samp_ind_opt]
    if(is.null(R))
    {
      fit_samp_opt_A = coxph(Surv(time=V[samp_ord],event=D[samp_ord]) ~ X[samp_ord,],
                             weights = weights_opt,robust = F,init = U_coef)
      tmp_A2 = information_score_matrix(coef(fit_samp_opt_A),weights = weights_opt,
                                        times = V[samp_ord],status = D[samp_ord],
                                        covariates = X[samp_ord,])
    }else
    {
      fit_samp_opt_A = coxph(Surv(R[samp_ord],V[samp_ord],D[samp_ord],type = "counting")
                             ~ X[samp_ord,],weights = weights_opt,robust = F,init = U_coef)
      tmp_A2 = information_score_matrix(coef(fit_samp_opt_A),weights = weights_opt,
                                        truncs = R[samp_ord],times = V[samp_ord],
                                        status = D[samp_ord],covariates = X[samp_ord,])
    }
    A_coef = coef(fit_samp_opt_A)
    names(A_coef) = colnames(X)
    ind_rm = (1:n_events)+q
    order_rm = tmp_A2$ord[!(tmp_A2$ord %in% ind_rm)]
    Score_A = tmp_A2$residual / tmp_A1$samp_prob[samp_ind_cens][order_rm]
    phi_mat_A = cov(Score_A)
    I_inv_A = solve(tmp_A2$hess)
    var_opt_A = I_inv_A + I_inv_A %*% phi_mat_A %*% I_inv_A/q
    ret = list("coef" = A_coef, "var" = var_opt_A)
    return(ret)
  }
}
```

Next we define the number of sampled censored observations per observed event, the number of samples to generate and the number of covariates. We create some objects to hold the simulation results.

```r
r = 6 #number of covariates
N_samples = 500
```

```
n_controls = 3 #number of sampled censored observations per failure time


q_vec = rep(NA,N_samples)
PL_res = unif_res = opt_A_res = opt_L_res= ncc_naive_res = ncc_samu_res =
  matrix(NA,nrow = N_samples,ncol = r)
coverage = matrix(NA,nrow = N_samples,ncol = 6)

var_PL = var_opt_L = var_opt_A = var_unif = var_NCC_c = var_NCC_S =
  vector("list",length = N_samples)

cens_rate = rep(NA,N_samples)
```

Now is the main "for" loop that performs the simulations. We generate N samples, and analyze each one using the full-data partial-likelihood estimator, the subsampling-based uniform/L-opt/A-opt **(using the "estimating function" provided in the repository)**, and the NCC methods (the classic one and the improved one due to Samuelsen). The data generation process can be changed so that it corresponds to any of the settings A/B/C in the paper.

```
#regression coefficients:
b <- c(3,-5,1,-1,1,-3,rep_len(c(-1,1/2),r-6))/10
#covariates upper bound:
upper_unif = c(4,4,4,4,4,4,rep_len(c(1,1),r-6))
# upper_unif = c(1,6,2,2,1,6,rep_len(c(1,6),r-6)) for setting "B"

for(m in 1:N_samples)
{
  n = 35000
  c = rexp(n,0.2)  #sampling censoring times
  X_vec = runif(n*r,0,rep(upper_unif,each = n))  #sampling covariates
  X = matrix(X_vec,nrow = n,ncol = r)
  #creating correlation:  for setting "C"
  # X[,4] = 0.5*X[,2] + 0.5*X[,1] +rnorm(n,0,0.1)
  # X[,5] = X[,1] + rnorm(n,0,1)
  # X[,6] = X[,1] + rnorm(n,1,1.5)

  linear_comb = X %*% b
  rates = c(0.001,0.025)  #baseline hazard rates
  knots = c(0,6)  #baseline hazard change points
  obs_rates = exp(linear_comb) %*% rates
  y = apply(obs_rates,1,rpexp,n=1,t=knots)  #sampling event times from piecewise exponential
  v = pmin(c,y)
  delta = v == y

  R = runif(n,0,quantile(v,0.75))  #sampling recruitment (truncation) times
  n = 15000
  #sampling n observations out of those we manage to observe (due to truncation):
  obs = sample(which(v > R),n)
  v = v[obs]
  R = R[obs]
  X = X[obs,]
  delta = delta[obs]
```

```r
n_events = sum(delta)
n_cens <- n - n_events
q = n_events*n_controls  #number of sampled censored observations per failure time

q_vec[m] = q

## full sample - full partial-likelihood
# fit = coxph(Surv(v,delta) ~ X)   #no truncation
fit = coxph(Surv(R,v,delta,type = "counting") ~ X) #with truncation
tmp = toc(quiet = T)
PL_res[m,] = coef(fit)
var_PL[[m]] = fit$var

cover1 = b <= PL_res[m,] + qnorm(0.975)*sqrt(diag(var_PL[[m]]))
cover2 = b >= PL_res[m,] - qnorm(0.975)*sqrt(diag(var_PL[[m]]))

coverage[m,1] = sum(cover1 & cover2)

cens_ind = which(delta == 0)

#pilot estimate - uniform sampling
fitUsubsample = CoxSubsample(V=v, D=delta, X=X, R=R, q, method = "U")
unif_res[m,] = fitUsubsample$coef
var_unif[[m]] = fitUsubsample$var

cover1 = b <= unif_res[m,] + qnorm(0.975)*sqrt(diag(var_unif[[m]]))
cover2 = b >= unif_res[m,] - qnorm(0.975)*sqrt(diag(var_unif[[m]]))

coverage[m,4] = sum(cover1 & cover2)

# L-opt sampling

fitLsubsample = CoxSubsample(V=v, D=delta, X=X, R=R, q, method = "L")
opt_L_res[m,] = fitLsubsample$coef
var_opt_L[[m]] = fitLsubsample$var

cover1 = b <= opt_L_res[m,] + qnorm(0.975)*sqrt(diag(var_opt_L[[m]]))
cover2 = b >= opt_L_res[m,] - qnorm(0.975)*sqrt(diag(var_opt_L[[m]]))

coverage[m,2] = sum(cover1 & cover2)


#A-opt sampling

fitAsubsample = CoxSubsample(V=v, D=delta, X=X, R=R, q, method = "A")
opt_A_res[m,] = fitAsubsample$coef
var_opt_A[[m]] = fitAsubsample$var

cover1 = b <= opt_A_res[m,] + qnorm(0.975)*sqrt(diag(var_opt_A[[m]]))
cover2 = b >= opt_A_res[m,] - qnorm(0.975)*sqrt(diag(var_opt_A[[m]]))

coverage[m,3] = sum(cover1 & cover2)
```

```r
    #  NCC - naive NCC and Samuelsen's NCC

  # ncc_samp = ccwc(exit=v,fail = delta,controls = n_controls,silent = T) # no truncation
  ncc_samp = ccwc(entry = R,exit=v,fail = delta,controls = n_controls,silent = T)  #with truncation

  # fit_naive_ncc = coxph(Surv(time=v[ncc_samp$Map],event=ncc_samp$Fail)~
  #X[ncc_samp$Map,] + strata(ncc_samp$Set))  # no truncation
  fit_naive_ncc = coxph(Surv(R[ncc_samp$Map],v[ncc_samp$Map],
                           ncc_samp$Fail,type = "counting")~X[ncc_samp$Map,] +
                        strata(ncc_samp$Set))  #with truncation

  ncc_naive_time[m] = tmp$toc - tmp$tic
  ncc_naive_res[m,] <- coef(fit_naive_ncc)
  var_NCC_c[[m]] = fit_naive_ncc$var

  cover1 = b <= ncc_naive_res[m,] + qnorm(0.975)*sqrt(diag(var_NCC_c[[m]]))
  cover2 = b >= ncc_naive_res[m,] - qnorm(0.975)*sqrt(diag(var_NCC_c[[m]]))

  coverage[m,5] = sum(cover1 & cover2)

  sampstat = rep(0,n)
  sampstat[ncc_samp$Map[ncc_samp$Fail==0]] = 1
  sampstat[delta] = 2
  # fit_samu_ncc = wpl(Surv(time=v,event=delta) ~ .,data.frame(X),
  #m=n_controls,samplestat = sampstat)  # no truncation
  fit_samu_ncc = wpl(Surv(R,v,delta,type = "counting") ~ .,data.frame(X),
                     m=n_controls,samplestat = sampstat)  # with truncation
  tmp = toc(quiet = T)
  ncc_samu_time[m] = tmp$toc - tmp$tic
  ncc_samu_res[m,] = coef(fit_samu_ncc)
  var_NCC_S[[m]] = fit_samu_ncc$var

  cover1 = b <= ncc_samu_res[m,] + qnorm(0.975)*sqrt(diag(var_NCC_S[[m]]))
  cover2 = b >= ncc_samu_res[m,] - qnorm(0.975)*sqrt(diag(var_NCC_S[[m]]))

  coverage[m,6] = sum(cover1 & cover2)

}
```

Now we summarize the results using RMSE with regards to the true parameters and with regards to the full-data PL.

```r
bmat <- matrix(b,nrow = N_samples,ncol = r,byrow = T)
RMSE_real_b = c(mean(sqrt(rowSums((PL_res - bmat)^2))),
               mean(sqrt(rowSums((opt_L_res - bmat)^2))),
               mean(sqrt(rowSums((opt_A_res - bmat)^2))),
               mean(sqrt(rowSums((unif_res - bmat)^2))),
               mean(sqrt(rowSums((ncc_naive_res - bmat)^2))),
              mean(sqrt(rowSums((ncc_samu_res - bmat)^2))))

RMSE_real_b_sd = c(sd(sqrt(rowSums((PL_res - bmat)^2))),
                sd(sqrt(rowSums((opt_L_res - bmat)^2))),
                sd(sqrt(rowSums((opt_A_res - bmat)^2))),
                sd(sqrt(rowSums((unif_res - bmat)^2))),
```

```
                sd(sqrt(rowSums((ncc_naive_res - bmat)^2))),
                sd(sqrt(rowSums((ncc_samu_res - bmat)^2))))

rmse_PL = sqrt(rowSums((PL_res - bmat)^2))
RR_real = c(mean((sqrt(rowSums((opt_L_res - bmat)^2)))/rmse_PL),
            mean((sqrt(rowSums((opt_A_res - bmat)^2)))/rmse_PL),
            mean((sqrt(rowSums((unif_res - bmat)^2)))/rmse_PL),
            mean((sqrt(rowSums((ncc_naive_res - bmat)^2)))/rmse_PL),
            mean((sqrt(rowSums((ncc_samu_res - bmat)^2)))/rmse_PL))

RMSE_PL_b = c(0,mean(sqrt(rowSums((opt_L_res - PL_res)^2))),
             mean(sqrt(rowSums((opt_A_res - PL_res)^2))),
             mean(sqrt(rowSums((unif_res - PL_res)^2))),
             mean(sqrt(rowSums((ncc_naive_res - PL_res)^2))),
             mean(sqrt(rowSums((ncc_samu_res - PL_res)^2))))

RMSE_PL_b_sd = c(0,sd(sqrt(rowSums((opt_L_res - PL_res)^2))),
              sd(sqrt(rowSums((opt_A_res - PL_res)^2))),
              sd(sqrt(rowSums((unif_res - PL_res)^2))),
              sd(sqrt(rowSums((ncc_naive_res - PL_res)^2))),
              sd(sqrt(rowSums((ncc_samu_res - PL_res)^2))))
```