07-16-2023

Hengde Ouyang

2023-07-16

1. Simulate Data from Cox model

We know the formula:

$$S(t) = exp(-\int_0^t h(u)\, du)$$

Then:

$$S(t|\mathbf{x}_i) = exp(-\int_0^t h(u|\mathbf{x}_i)\, du)$$

In our case:

$$b = exp(\mathbf{x}_i^T \beta),\ h(t|\mathbf{x}_i) = abt,\ S(t|\mathbf{x}_i) = exp(-\frac{abt^2}{2})$$

Then:

$$F(t|\mathbf{x}_i) = 1 - exp(-\frac{abt^2}{2})$$

We can generate:

$$T_i = \sqrt{\frac{-2 * log(1 - U_i)}{ab}}$$

```
Cox_Simulation <- function(n,a,beta,lower,upper){
  x = runif(length(beta)*n,lower,upper)
  X = matrix(x,n,length(beta))

  # the exponential part
  b = as.numeric(exp(X%*%beta))
```

```r
  U = runif(n,0,1)
  time = sqrt(-2*log(1-U)/(a*b))

  return(list(Time = time,
              X = X))
}
```

```r
Cox_Simulation(5,2,c(1,2,3),-1,1)
```

```
## $Time
## [1] 0.4771704 0.5152549 0.8028813 0.3677431 2.6974134
##
## $X
##            [,1]        [,2]        [,3]
## [1,] -0.7901221 -0.70882245  0.36835214
## [2,] -0.3901459  0.02682875  0.77667383
## [3,] -0.8738454 -0.06083261  0.67130537
## [4,]  0.9501158  0.13321456  0.03723915
## [5,] -0.0575270 -0.58725164 -0.47382597
```

```r
result = Cox_Simulation(1000,2,c(1,2,3),-1,1)
delta = rbinom(1000,size=1,prob=0.8)
```

```r
summary(coxph(Surv(result$Time,delta)~result$X))
```

```
## Call:
## coxph(formula = Surv(result$Time, delta) ~ result$X)
##
##   n= 1000, number of events= 814
##
##               coef exp(coef) se(coef)     z Pr(>|z|)
## result$X1  1.06744   2.90793  0.06921 15.42   <2e-16 ***
## result$X2  2.02301   7.56108  0.08508 23.78   <2e-16 ***
## result$X3  2.94484  19.00758  0.10548 27.92   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##           exp(coef) exp(-coef) lower .95 upper .95
## result$X1     2.908    0.34389     2.539     3.330
## result$X2     7.561    0.13226     6.400     8.933
## result$X3    19.008    0.05261    15.458    23.373
##
## Concordance= 0.838  (se = 0.006 )
## Likelihood ratio test= 1208  on 3 df,   p=<2e-16
## Wald test            = 905.5  on 3 df,   p=<2e-16
## Score (logrank) test = 1061  on 3 df,   p=<2e-16
```

Question: What if we don't specify baseline hazard?

2. Gaussian Process

To see the table, see the last page

Summary:

1. In Gaussian Process, Scaling of design matrix seems to be important. It's difficult to see the good result without scaling.

2. By proper selection of hyper-parameter, we have the potential to get good result in both cases. (gamma prior and inverse-gamma prior)

3. algorithm will crash by improper choice of hyper-parameter.

For example:
small value of lambda -> small value entries in kernel K -> too many 0 in K -> have issues when calculating the inverse of KK (n1 x n2).

```
MH_GP_Sampling1 <- function(tti,Y,Y.test,delta,delta.test,tau,
                        A,A.all,beta0,var.prop,alpha0,v0,
                        m,B,eta,K.all,
                        Wmat_option=0){


  accept_beta = 0
  accept_lambda = 0
  beta = beta0
  lambda = lambda0

  # What we want to record
  BETA = matrix(0,m,dim(A)[1])
  BETA.test = matrix(0,m,(dim(A.all)[1]-dim(A)[1]))
  LAMBDA = matrix(0,m,dim(A)[2])
  C_stat = c()
  C_stat.test = c()

  # For safety m>B
  if (B>m){
    B = 0
  }


  # 0 means we use Harrell C statistics
  # 1 means we use Uno C statistics
  if (Wmat_option==0){
    Wmat <- HarrellC_Wmat(Y, delta, tau)
    Wmat.test <- HarrellC_Wmat(Y.test, delta.test, tau)
  }else if (Wmat_option==1){
    Wmat <- UnoC_Wmat(Y, delta, tau)
    Wmat.test <- UnoC_Wmat(Y.test, delta.test, tau)
  }else{  # Other Possible C index...
```

```r
  Wmat <- HarrellC_Wmat(Y, delta, tau)
  Wmat.test <- HarrellC_Wmat(Y.test, delta.test, tau)
}


for (i in 1:m){

  # Sample beta from proposal distribution
  beta.p = t(rmvnorm(1,beta,var.prop))


  # Compute theta from current and last iteration
  theta.p = beta.p
  theta = beta

  # Get covariance matrix for training set
  K = K.all[1:tti,1:tti]
  K.test = K.all[(tti+1):n,(tti+1):n]
  KK = K.all[1:tti,(tti+1):n]

  # Compute C-statistics from current and last iteration

  HC.p = C_index(theta.p, Wmat)
  HC = C_index(theta, Wmat)

  # Record C-statistics from last iteration
  C_stat = c(C_stat,HC)



  #############################################
  # Compute log of MH ratio

  lrMH = eta*log(HC.p) +
        dmvnorm(as.numeric(beta.p),beta0,K,log=T)-
        eta*log(HC) -
        dmvnorm(as.numeric(beta),beta0,K,log=T)

    if (log(runif(1))<lrMH){
      beta = beta.p
      accept_beta = accept_beta + 1
    }
    BETA[i,] = beta
  #############################################

  # Calculate the C_index for the testing data
  interim = t(KK)%*%solve(K)

  mu = interim%*%beta
  sig = K.test - interim%*%KK
  beta.test = mu
  theta.test = beta.test
```

```
    BETA.test[i,] = beta.test

    HC.test = C_index(theta.test,Wmat.test)
    C_stat.test = c(C_stat.test,HC.test)

     ################################################################
    # Compute log of MH_lambda ratio
    lambda.p = exp(t(rnorm(dim(A)[2],log(lambda),rep(1,dim(A)[2]))))
    lambda.p = as.vector(lambda.p)
    K.p = matrix_K(A,lambda.p)

    lrMH_lambda = dmvnorm(as.numeric(beta),beta0,K.p,log=T)+
                  sum(dgamma(lambda.p,alpha0,v0,log = T))  -
                  dmvnorm(as.numeric(beta),beta0,K,log=T) -
                  sum(dgamma(lambda,alpha0,v0,log = T))

    if (log(runif(1))<lrMH_lambda){
        lambda = lambda.p
        K.all = matrix_K(A.all,lambda)
        accept_lambda = accept_lambda + 1
      }
      LAMBDA[i,] = lambda
  }
  ######################################################################




  if (B == 0){
    return(list(BETA=BETA,
                BETA_test=BETA.test,
                LAMBDA = LAMBDA,
                accept_beta=accept_beta/m,
                C_stat = C_stat,
                C_stat_test = C_stat.test))
  }else{
    return(list(BETA=BETA[-c(1:B),],
                BETA_test=BETA.test[-c(1:B),],
                LAMBDA = LAMBDA[-c(1:B),],
                accept_beta=accept_beta/m,
                C_stat = C_stat[-c(1:B)],
                C_stat_test = C_stat.test[-c(1:B)]))
  }



}
```

```
no_na_lung = na.omit(lung)
n = dim(no_na_lung)[1]

# Input data
```

```r
train_test_index = 120  # the index to separate training and testing data
tti = train_test_index  # make the name shorter

no_na_lung.train = no_na_lung[1:tti,]
no_na_lung.test = no_na_lung[(tti+1):dim(no_na_lung)[1],]


Y = no_na_lung.train$time
Y.test = no_na_lung.test$time
delta = no_na_lung.train$status - 1
delta.test = no_na_lung.test$status - 1

tau = 2500
A <- scale(model.matrix(time ~ -1+ inst+ age + sex + ph.ecog + ph.karno + pat.karno
                  +meal.cal+wt.loss,
                  data=no_na_lung.train))
A.test <- scale(model.matrix(time ~ -1+ inst+ age + sex + ph.ecog + ph.karno + pat.karno
                  +meal.cal+wt.loss,
                  data=no_na_lung.test))
A.all <- scale(model.matrix(time ~ -1+ inst+ age + sex + ph.ecog + ph.karno + pat.karno
                  +meal.cal+wt.loss,
                  data=no_na_lung))

# Gaussian Process

beta0 = rep(0,dim(A)[1])
sigma0 = rep(1,dim(A)[1])
lambda0 = rep(2,dim(A)[2])

# A relatively Small data set, can increase the iteration
m = 11000
B = 1000
eta = length(Y)
Wmat_option = 0

var.prop = diag(0.01,dim(A)[1])

K.all = as.matrix(matrix_K(A.all,lambda0))
```

```r
system.time({
  alpha0 = 1
  v0 = 1
  eta = length(Y)
  result1 = MH_GP_Sampling1(tti,Y,Y.test,delta,delta.test,tau,
                        A,A.all,beta0,var.prop,alpha0,v0,
                        m,B,eta,K.all,
                        Wmat_option=0)






  alpha0 = 3
```

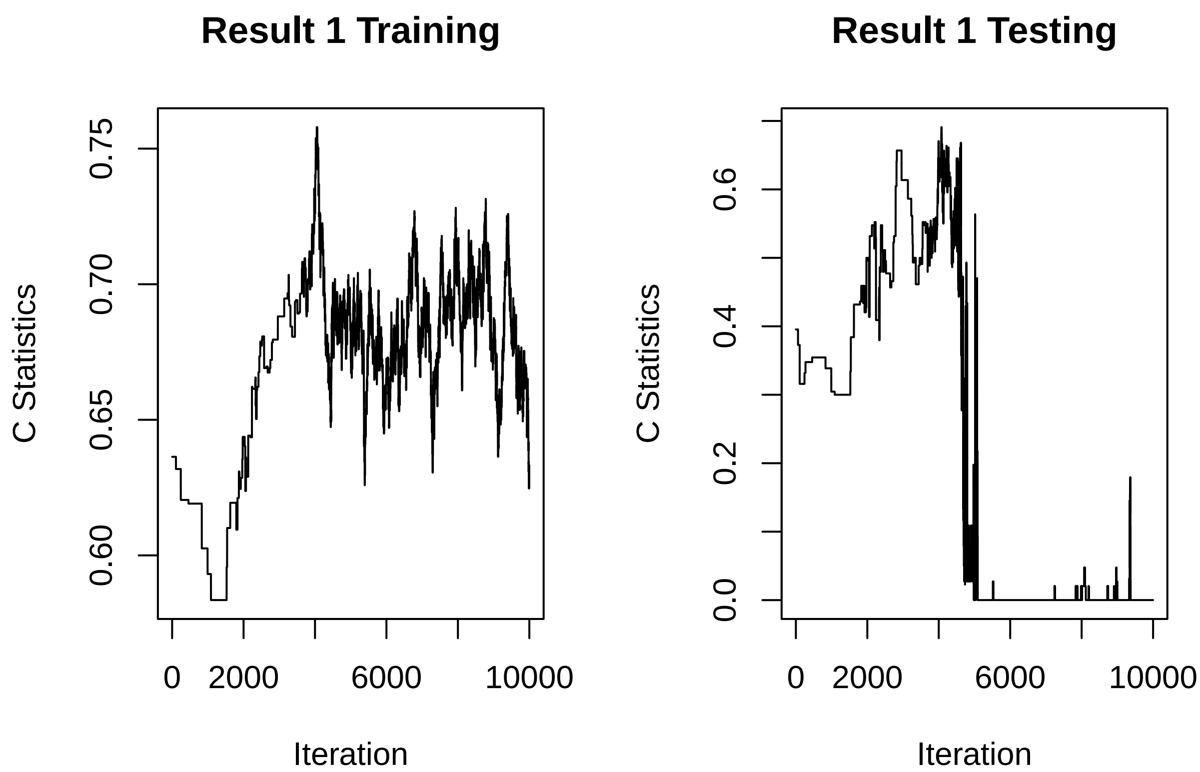
```r
  v0 = 1
  eta = length(Y)
  result2 = MH_GP_Sampling1(tti,Y,Y.test,delta,delta.test,tau,
                         A,A.all,beta0,var.prop,alpha0,v0,
                         m,B,eta,K.all,
                         Wmat_option=0)

  alpha0 = 5
  v0 = 1
  eta = length(Y)
  result3 = MH_GP_Sampling1(tti,Y,Y.test,delta,delta.test,tau,
                         A,A.all,beta0,var.prop,alpha0,v0,
                         m,B,eta,K.all,
                         Wmat_option=0)

  alpha0 = 3
  v0 = 1
  eta = 2*length(Y)
  result4 = MH_GP_Sampling1(tti,Y,Y.test,delta,delta.test,tau,
                         A,A.all,beta0,var.prop,alpha0,v0,
                         m,B,eta,K.all,
                         Wmat_option=0)

  alpha0 = 3
  v0 = 1
  eta = 0.5*length(Y)
  result5 = MH_GP_Sampling1(tti,Y,Y.test,delta,delta.test,tau,
                         A,A.all,beta0,var.prop,alpha0,v0,
                         m,B,eta,K.all,
                         Wmat_option=0)


  alpha0 = 3
  v0 = 2
  eta = length(Y)
  result6 = MH_GP_Sampling1(tti,Y,Y.test,delta,delta.test,tau,
                         A,A.all,beta0,var.prop,alpha0,v0,
                         m,B,eta,K.all,
                         Wmat_option=0)



})
```

```
##    user  system elapsed
## 1535.52   27.26 2247.22
```

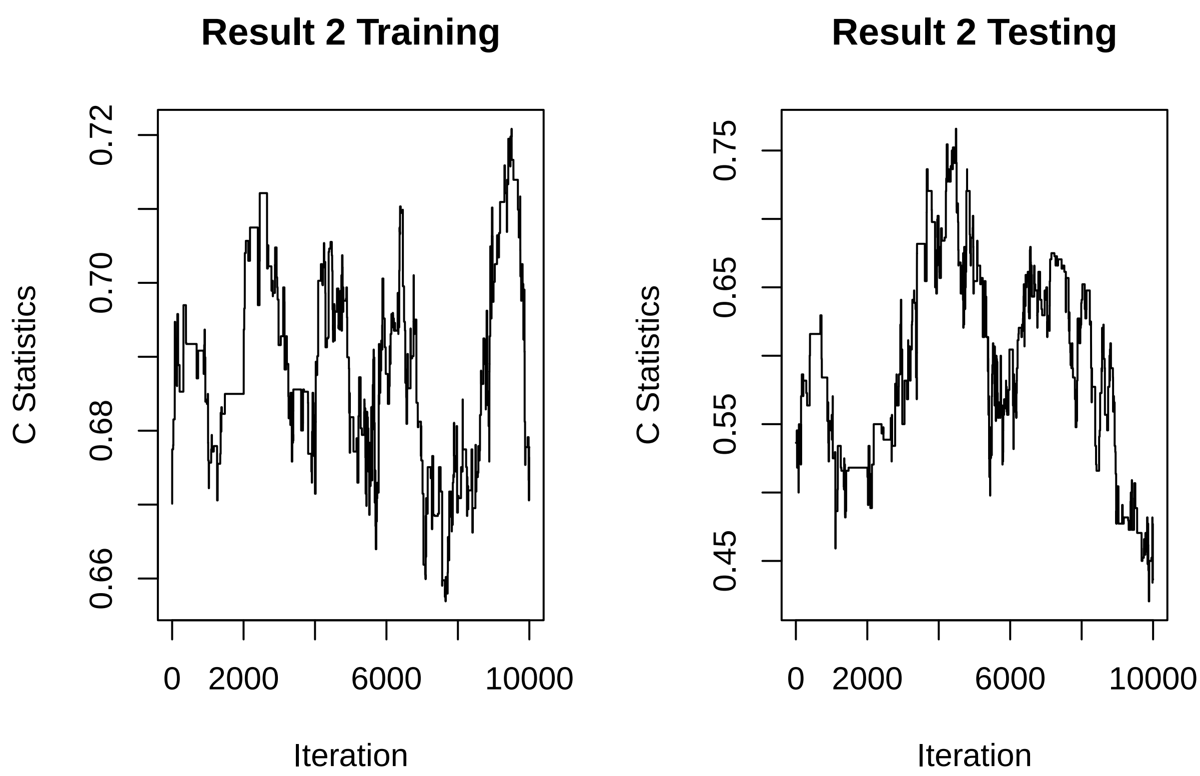
Pick some of results as an illustration

```r
par(mfrow=c(1,2))
plot(1:(m-B),result1$C_stat,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Result 1 Training")
```

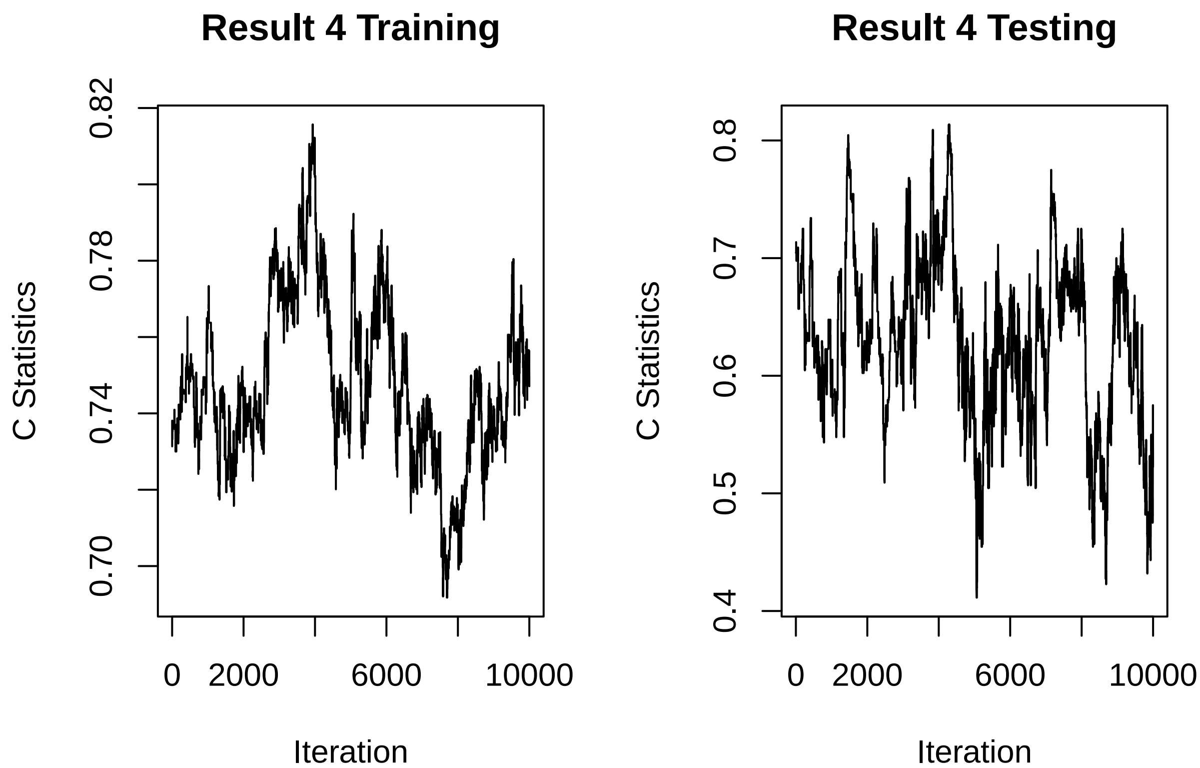
```
plot(1:(m-B),result1$C_stat_test,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Result 1 Testing")
```



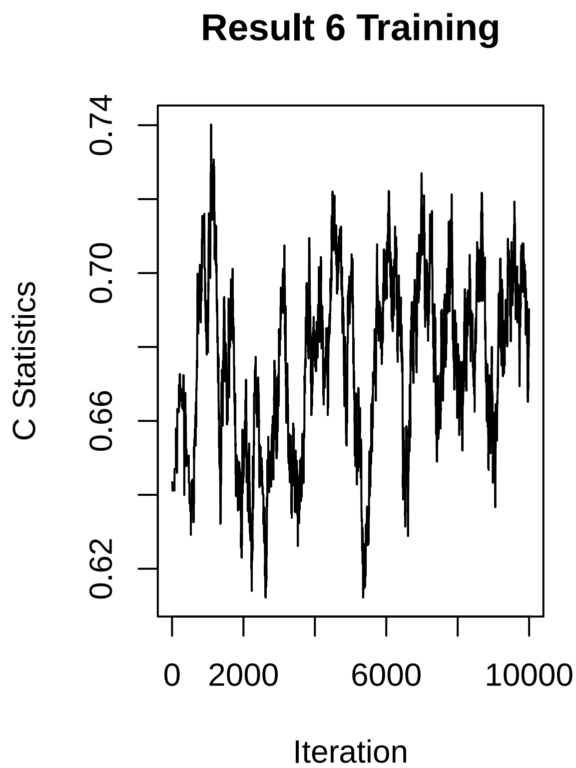

```
par(mfrow=c(1,2))
plot(1:(m-B),result2$C_stat,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Result 2 Training")
plot(1:(m-B),result2$C_stat_test,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Result 2 Testing")
```

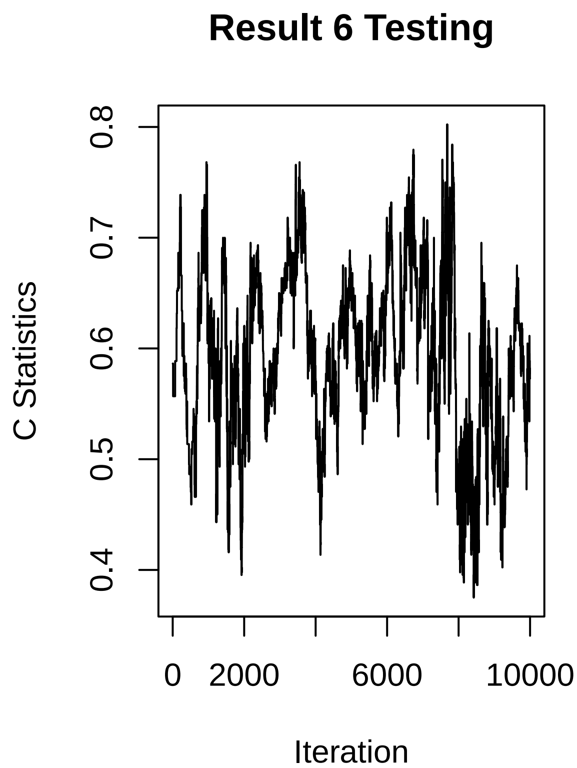



```
par(mfrow=c(1,2))
plot(1:(m-B),result4$C_stat,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Result 4 Training")
plot(1:(m-B),result4$C_stat_test,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Result 4 Testing")
```




```
par(mfrow=c(1,2))
plot(1:(m-B),result6$C_stat,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Result 6 Training")
plot(1:(m-B),result6$C_stat_test,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Result 6 Testing")
```






```
Wmat = HarrellC_Wmat(Y,delta,tau)
Wmat.test = HarrellC_Wmat(Y.test,delta.test,tau)
```

```
##   Index   C_train    C_test Alpha0 V0 Eta
## 1     1 0.8698784 0.4795455      1  1 1.0
## 2     2 0.7355546 0.6204545      3  1 1.0
## 3     3 0.6777728 0.4886364      5  1 1.0
## 4     4 0.8637250 0.7068182      3  1 2.0
## 5     5 0.5676122 0.6431818      3  1 0.5
## 6     6 0.8337085 0.6613636      3  2 1.0
```