07-09-2023

Hengde Ouyang

2023-07-09

Gaussian Process

First we need to separate the data into training and testing data.

$$\binom{f}{f^*} \; with \; sizes \binom{n \times 1}{n^* \times 1}$$

We have the kernel covariance matrix:

$$\begin{pmatrix} \Sigma_{XX} & \Sigma_{XX^*} \\ \Sigma_{X^*X} & \Sigma_{X^*X^*} \end{pmatrix} \; with \; sizes \begin{pmatrix} n \times n & n \times n^* \\ n^* \times n & n^* \times n^* \end{pmatrix}$$

Then We have the following predictive distribution:

$$f^* | X, X^*, f \sim N(\mu, \Sigma)$$

where:

$$\mu = \Sigma_{X^*X} \, \Sigma_{XX}^{-1} \, f$$

$$\Sigma = \Sigma_{X^*X^*} - \Sigma_{X^*X} \, \Sigma_{XX}^{-1} \, \Sigma_{XX^*}$$

Discussion

Before moving to the algorithm I built, I may need to talk about some points to justify my work. (I'm not sure it's reasonable, it's just an attempt to solve the issue)

1. Our old algorithm for sampling length scale parameter λ (Step 4-6) seems to prefer a very small value of λ . That results in a more "wiggly" function, indicating an over fitting issue.

After I separate training and testing data, the C statistics indeed show this issue. (Training C Stat: 0.9, Testing C Stat: 0.5)

2. To solve this, I try to fix the lambda first. (Skip Step 4-6, try modifying lambda by ourselves)

It does not work very well.

3. Then I try to scale our design matrix.

Reason: the mechanism of GP is “Similar x’s produce similar y’s”. If we don’t scale the X, most of the X’s are NOT similar, which might cause some issues.

(If you look at the kernel covariance matrix before scaling, most of the entries are close to 0, indicating dissimilarities)

It improves a little bit, but it still does not work very well.

4. Instead of getting a predictive distribution in each iteration, I directly use the μ in predictive distribution to estimate the beta in testing data.

It works! But it’s not that reasonable.

5. Other solutions?

Algorithm

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


  accept_beta = 0
  accept_lambda = 0
  beta = beta0


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

```r
    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)

  }




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



}
```

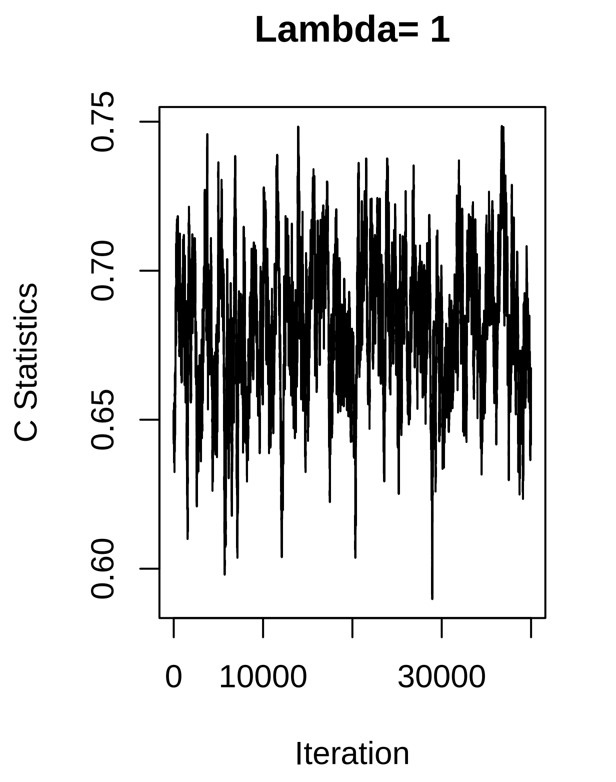
```r
no_na_lung = na.omit(lung)
n = dim(no_na_lung)[1]

# Input data
train_test_index = 100  # 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))
```

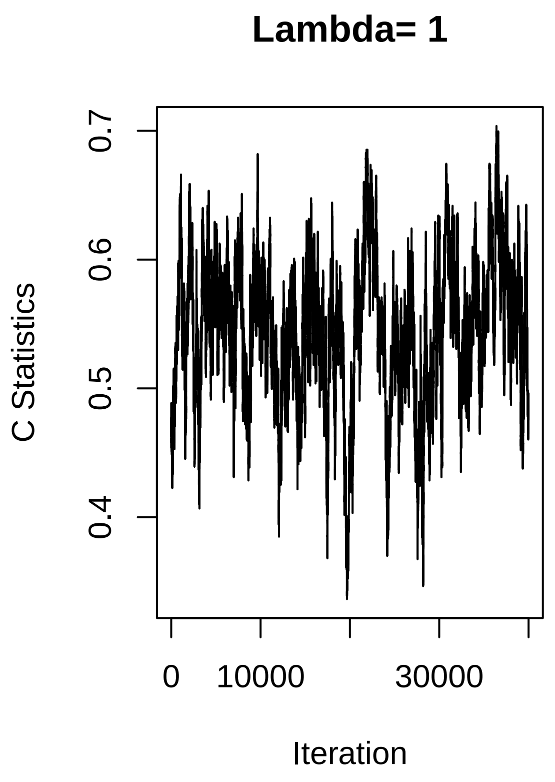
```r
A.all <- scale(model.matrix(time ~ -1+ inst+ age + sex + ph.ecog + ph.karno + pat.karno
                  +meal.cal+wt.loss,
                  data=no_na_lung))

# Gaussian Process

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

# Modify lambda here
lambda0 = rep(1,dim(A)[2])

# A relatively Small data set, can increase the iteration
m = 42000
B = 2000
eta = length(Y)
Wmat_option = 0

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

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

```r
system.time({
lambda0 = rep(1,dim(A)[2])
result1 = MH_GP_Sampling(tti,Y,Y.test,delta,delta.test,tau,
                        A,A.all,beta0,var.prop,
                        m,B,eta,K.all,
                        Wmat_option=0)
})
```

```
##    user  system elapsed
##  438.48   24.32  784.58
```

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


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

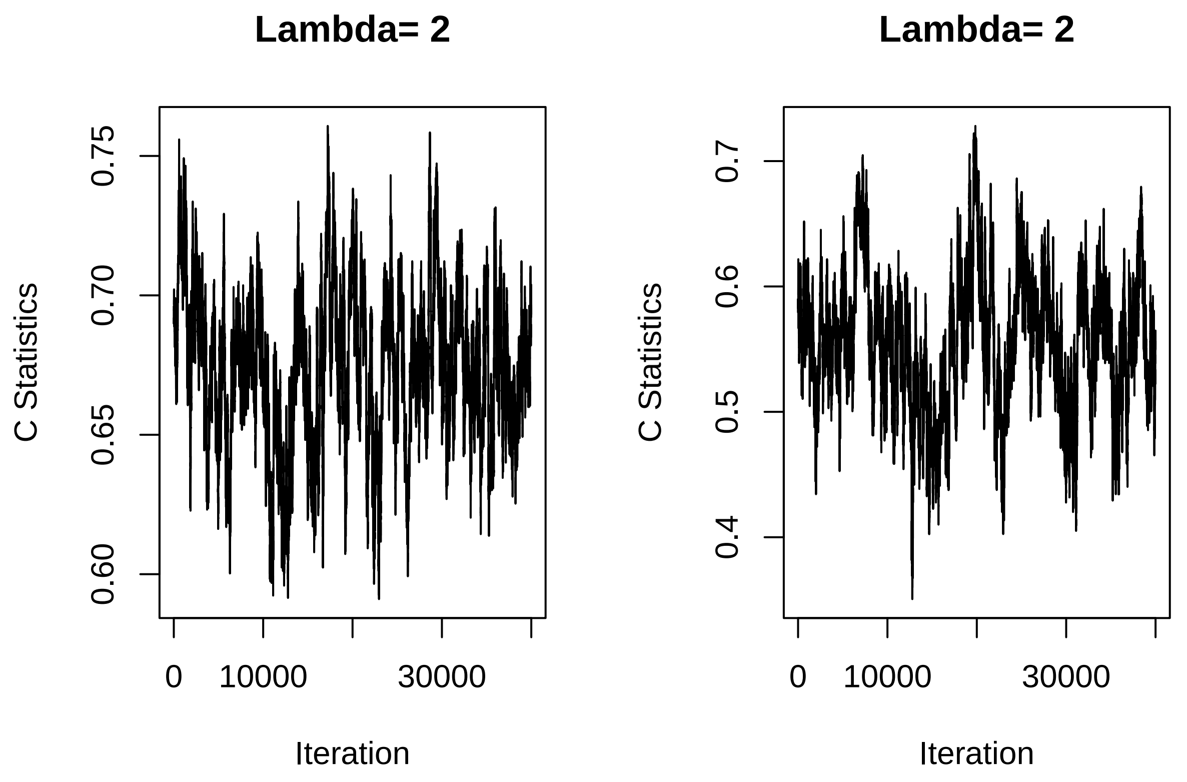




```
system.time({
lambda0 = rep(2,dim(A)[2])
result2 = MH_GP_Sampling(tti,Y,Y.test,delta,delta.test,tau,
                        A,A.all,beta0,var.prop,
                        m,B,eta,K.all,
                        Wmat_option=0)
})
```

```
##    user  system elapsed
##  356.24   17.99  621.51
```

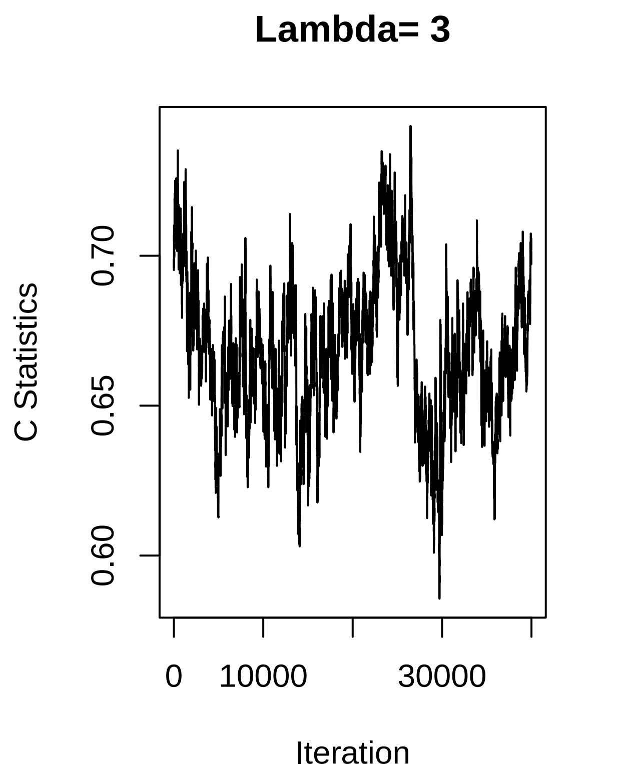
```
par(mfrow=c(1,2))
plot(1:(m-B),result2$C_stat,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Lambda= 2")
plot(1:(m-B),result2$C_stat_test,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Lambda= 2")
```

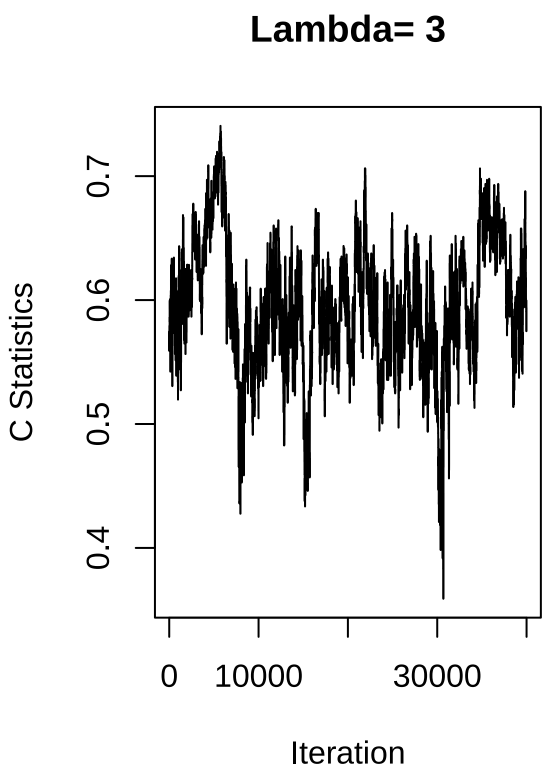


```r
system.time({
lambda0 = rep(3,dim(A)[2])
K.all = as.matrix(matrix_K(A.all,lambda0))
result3 = MH_GP_Sampling(tti,Y,Y.test,delta,delta.test,tau,
                         A,A.all,beta0,var.prop,
                         m,B,eta,K.all,
                         Wmat_option=0)
})
```

```
##    user  system elapsed
##  347.72   21.53  586.75
```

```r
par(mfrow=c(1,2))
plot(1:(m-B),result3$C_stat,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Lambda= 3")
plot(1:(m-B),result3$C_stat_test,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Lambda= 3")
```

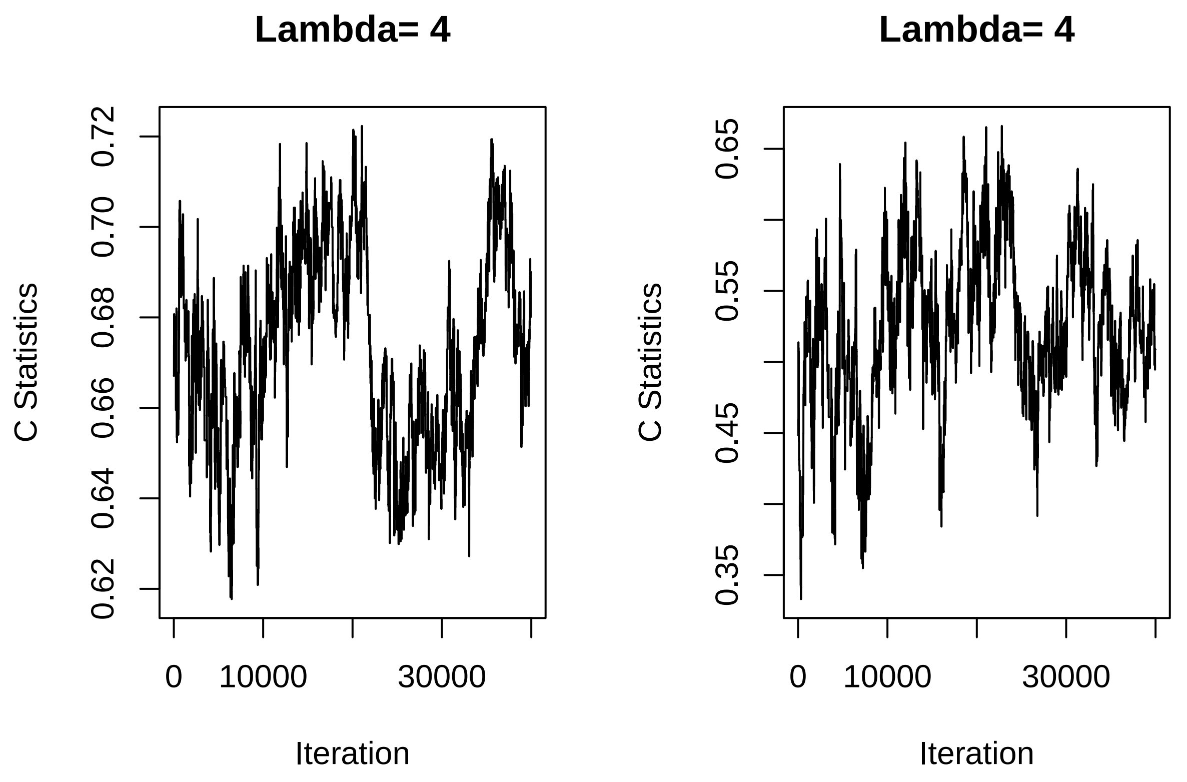




```r
system.time({
lambda0 = rep(4,dim(A)[2])
K.all = as.matrix(matrix_K(A.all,lambda0))
result4 = MH_GP_Sampling(tti,Y,Y.test,delta,delta.test,tau,
                        A,A.all,beta0,var.prop,
                        m,B,eta,K.all,
                        Wmat_option=0)
})
```

```
##    user  system elapsed
##  319.50   24.25  562.57
```

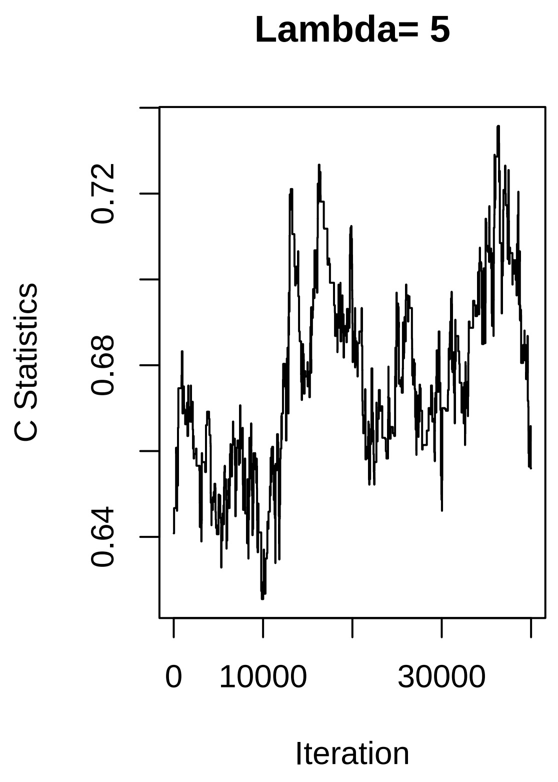
```r
par(mfrow=c(1,2))
plot(1:(m-B),result4$C_stat,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Lambda= 4")
plot(1:(m-B),result4$C_stat_test,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Lambda= 4")
```

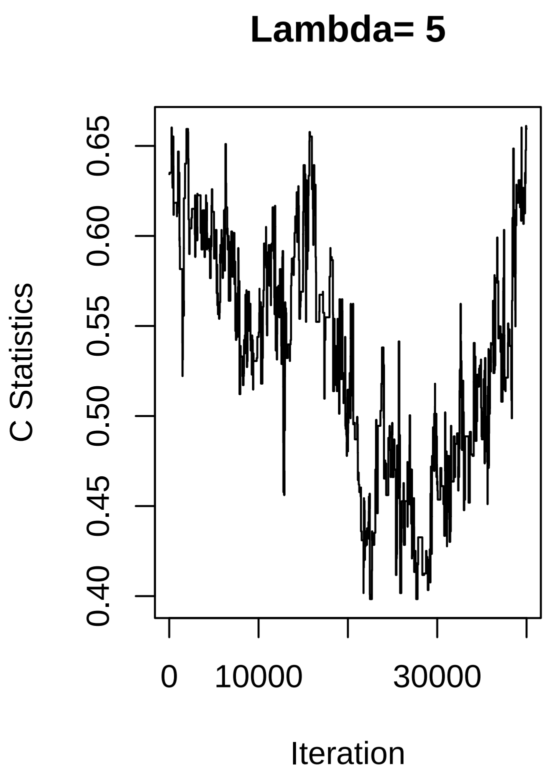


```r
system.time({
lambda0 = rep(5,dim(A)[2])
K.all = as.matrix(matrix_K(A.all,lambda0))
result5 = MH_GP_Sampling(tti,Y,Y.test,delta,delta.test,tau,
                        A,A.all,beta0,var.prop,
                        m,B,eta,K.all,
                        Wmat_option=0)
})
```

```
##    user  system elapsed
##  323.75   23.50  564.09
```

```r
par(mfrow=c(1,2))
plot(1:(m-B),result5$C_stat,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Lambda= 5")
plot(1:(m-B),result5$C_stat_test,type = "l",
     xlab = "Iteration",ylab = "C Statistics",main = "Lambda= 5")
```





```
all_c = data.frame(C_train = c(C_index(colMeans(result1$BETA),Wmat),
                               C_index(colMeans(result2$BETA),Wmat),
                               C_index(colMeans(result3$BETA),Wmat),
                               C_index(colMeans(result4$BETA),Wmat),
                               C_index(colMeans(result5$BETA),Wmat)),
                   C_test = c(C_index(colMeans(result1$BETA_test),Wmat.test),
                              C_index(colMeans(result2$BETA_test),Wmat.test),
                              C_index(colMeans(result3$BETA_test),Wmat.test),
                              C_index(colMeans(result4$BETA_test),Wmat.test),
                              C_index(colMeans(result5$BETA_test),Wmat.test)),
                   Lambda = c(1,2,3,4,5))
all_c
```

```
##     C_train    C_test Lambda
## 1 0.8662046 0.6016736      1
## 2 0.8628439 0.6184100      2
## 3 0.7601344 0.6426778      3
## 4 0.7448015 0.5481172      4
## 5 0.7256879 0.5497908      5
```