Hyper Leanring

Hengde Ouyang

2023-07-30

Code Part (Result is on Page 8)

```
library(invgamma)
Hyper_Learning <- function(tti,Y,Y.test,delta,delta.test,tau,
                        A,A.all,beta0,alpha0,v0,kappa,
                        m,B,eta,K.all,n,
                        Wmat_option=0){

  # Three chains


  accept_beta1 = 0
  accept_beta2 = 0
  accept_beta3 = 0
  accept_lambda1 = 0
  accept_lambda2 = 0
  accept_lambda3 = 0
  beta1 = beta0
  beta2 = beta0
  beta3 = beta0
  lambda1 = lambda0
  lambda2 = lambda0
  lambda3 = lambda0
  alpha1 = alpha0
  alpha2 = alpha0
  alpha3 = alpha0
  v1 = v0
  v2 = v0
  v3 = v0
  eta1 = eta
  eta2 = eta
  eta3 = eta
  K1.all = K.all
  K2.all = K.all
  K3.all = K.all


  # What we want to record
  BETA1 = matrix(0,m,dim(A)[1])
  BETA2 = matrix(0,m,dim(A)[1])
```

```r
BETA3 = matrix(0,m,dim(A)[1])
BETA1.test = matrix(0,m,(dim(A.all)[1]-dim(A)[1]))
BETA2.test = matrix(0,m,(dim(A.all)[1]-dim(A)[1]))
BETA3.test = matrix(0,m,(dim(A.all)[1]-dim(A)[1]))
LAMBDA1 = matrix(0,m,dim(A)[2])
LAMBDA2 = matrix(0,m,dim(A)[2])
LAMBDA3 = matrix(0,m,dim(A)[2])
C_stat1 = c()
C_stat2 = c()
C_stat3 = c()
C_stat1.test = c()
C_stat2.test = c()
C_stat3.test = c()
ALPHA = matrix(0,m,3)
V = matrix(0,m,3)
ETA = matrix(0,m,3)



# 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){
  # Get covariance matrix for training set
  K1 = K1.all[1:tti,1:tti]
  K1.test = K1.all[(tti+1):n,(tti+1):n]
  KK1 = K1.all[1:tti,(tti+1):n]

  K2 = K2.all[1:tti,1:tti]
  K2.test = K2.all[(tti+1):n,(tti+1):n]
  KK2 = K2.all[1:tti,(tti+1):n]

  K3 = K3.all[1:tti,1:tti]
  K3.test = K3.all[(tti+1):n,(tti+1):n]
  KK3 = K3.all[1:tti,(tti+1):n]
```

```
# Sample beta from proposal distribution
beta1.p = t(rmvnorm(1,beta1,kappa*K1))
beta2.p = t(rmvnorm(1,beta2,kappa*K2))
beta3.p = t(rmvnorm(1,beta3,kappa*K3))


# Compute theta from current and last iteration
theta1.p = beta1.p
theta1 = beta1
theta2.p = beta2.p
theta2 = beta2
theta3.p = beta3.p
theta3 = beta3



# Compute C-statistics from current and last iteration

HC1.p = C_index(theta1.p, Wmat)
HC1 = C_index(theta1, Wmat)
HC2.p = C_index(theta2.p, Wmat)
HC2 = C_index(theta2, Wmat)
HC3.p = C_index(theta3.p, Wmat)
HC3 = C_index(theta3, Wmat)

# Record C-statistics from last iteration
C_stat1 = c(C_stat1,HC1)
C_stat2 = c(C_stat2,HC2)
C_stat3 = c(C_stat3,HC3)



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

lrMH1 = eta1*log(HC1.p) +
      dmvnorm(as.numeric(beta1.p),beta0,K1,log=T)-
      eta1*log(HC1) -
      dmvnorm(as.numeric(beta1),beta0,K1,log=T)

  if (log(runif(1))<lrMH1){
    beta1 = beta1.p
    accept_beta1 = accept_beta1 + 1
  }
  BETA1[i,] = beta1

lrMH2 = eta2*log(HC2.p) +
      dmvnorm(as.numeric(beta2.p),beta0,K2,log=T)-
      eta2*log(HC2) -
      dmvnorm(as.numeric(beta2),beta0,K2,log=T)

  if (log(runif(1))<lrMH2){
    beta2 = beta2.p
```

```
    accept_beta2 = accept_beta2 + 1
  }
  BETA2[i,] = beta2

lrMH3 = eta3*log(HC3.p) +
      dmvnorm(as.numeric(beta3.p),beta0,K3,log=T)-
      eta3*log(HC3) -
      dmvnorm(as.numeric(beta3),beta0,K3,log=T)

  if (log(runif(1))<lrMH3){
    beta3 = beta3.p
    accept_beta3 = accept_beta3 + 1
  }
  BETA3[i,] = beta3
#############################################

# Calculate the C_index for the testing data
interim1 = t(KK1)%*%solve(K1)
interim2 = t(KK2)%*%solve(K2)
interim3 = t(KK3)%*%solve(K3)



mu1 = interim1%*%beta1
beta1.test = mu1
theta1.test = beta1.test
mu2 = interim2%*%beta2
beta2.test = mu2
theta2.test = beta2.test
mu3 = interim3%*%beta3
beta3.test = mu3
theta3.test = beta3.test



BETA1.test[i,] = beta1.test
BETA2.test[i,] = beta2.test
BETA3.test[i,] = beta3.test
HC1.test = C_index(theta1.test,Wmat.test)
HC2.test = C_index(theta2.test,Wmat.test)
HC3.test = C_index(theta3.test,Wmat.test)
C_stat1.test = c(C_stat1.test,HC1.test)
C_stat2.test = c(C_stat2.test,HC2.test)
C_stat3.test = c(C_stat3.test,HC3.test)

 ###############################################################
# Compute log of MH_lambda ratio
lambda1.p = exp(t(rnorm(dim(A)[2],log(lambda1),rep(1,dim(A)[2]))))
lambda1.p = as.vector(lambda1.p)
K1.p = matrix_K(A,lambda1.p)

lrMH_lambda1 = dmvnorm(as.numeric(beta1),beta0,K1.p,log=T)+
              sum(dinvgamma(lambda1.p,alpha1,v1,log = T))  -
```

```
              dmvnorm(as.numeric(beta1),beta0,K1,log=T) -
              sum(dinvgamma(lambda1,alpha1,v1,log = T))

if (log(runif(1))<lrMH_lambda1){
    lambda1 = lambda1.p
    K1.all = matrix_K(A.all,lambda1)
    accept_lambda1 = accept_lambda1 + 1
  }
  LAMBDA1[i,] = lambda1


lambda2.p = exp(t(rnorm(dim(A)[2],log(lambda2),rep(1,dim(A)[2]))))
lambda2.p = as.vector(lambda2.p)
K2.p = matrix_K(A,lambda2.p)

lrMH_lambda2 = dmvnorm(as.numeric(beta2),beta0,K2.p,log=T)+
              sum(dinvgamma(lambda2.p,alpha2,v2,log = T))  -
              dmvnorm(as.numeric(beta2),beta0,K2,log=T) -
              sum(dinvgamma(lambda2,alpha2,v2,log = T))

if (log(runif(1))<lrMH_lambda2){
    lambda2 = lambda2.p
    K2.all = matrix_K(A.all,lambda2)
    accept_lambda2 = accept_lambda2 + 1
  }
  LAMBDA2[i,] = lambda2


lambda3.p = exp(t(rnorm(dim(A)[2],log(lambda3),rep(1,dim(A)[2]))))
lambda3.p = as.vector(lambda3.p)
K3.p = matrix_K(A,lambda3.p)

lrMH_lambda3 = dmvnorm(as.numeric(beta3),beta0,K3.p,log=T)+
              sum(dinvgamma(lambda3.p,alpha3,v3,log = T))  -
              dmvnorm(as.numeric(beta3),beta0,K3,log=T) -
              sum(dinvgamma(lambda3,alpha3,v3,log = T))

if (log(runif(1))<lrMH_lambda3){
    lambda3 = lambda3.p
    K3.all = matrix_K(A.all,lambda3)
    accept_lambda3 = accept_lambda3 + 1
  }
  LAMBDA3[i,] = lambda3





  ######
  at = 1/i
  ct = 1/i^(1/3)

  # Set the hyper parameter
```

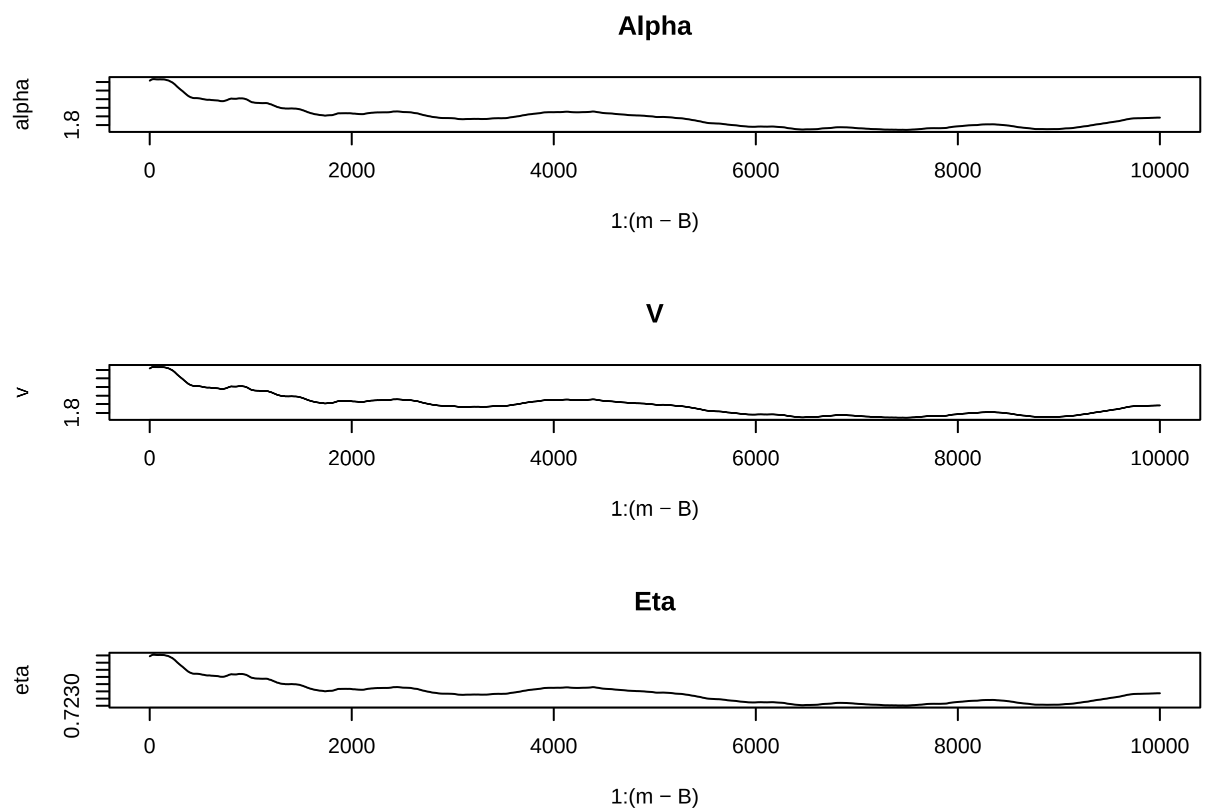
```
    add = at*((HC2.test-HC3.test)/(2*ct))
    alpha1 = alpha1 + add
    v1 = v1 + add
    eta1 = eta1 + add

    alpha2 = alpha1 + ct
    v2 = v1 + ct
    eta2 = eta1 + ct

    alpha3 = alpha1 - ct
    v3 = v1 - ct
    eta3 = eta1 - ct

    ALPHA[i,] = c(alpha1,alpha2,alpha3)
    V[i,] = c(v1,v2,v3)
    ETA[i,] = c(eta1,eta2,eta3)

}


#####################################################################




if (B == 0){
  return(list(BETA1=BETA1,
              BETA2=BETA2,
              BETA3=BETA3,
              BETA_test1=BETA1.test,
              BETA_test2=BETA2.test,
              BETA_test3=BETA3.test,
              LAMBDA1 = LAMBDA1,
              LAMBDA2 = LAMBDA2,
              LAMBDA3 = LAMBDA3,
              accept_beta1=accept_beta1/m,
              accept_beta2=accept_beta2/m,
              accept_beta3=accept_beta3/m,
              C_stat1 = C_stat1,
              C_stat2 = C_stat2,
              C_stat3 = C_stat3,
              C_stat_test1 = C_stat1.test,
              C_stat_test2 = C_stat2.test,
              C_stat_test3 = C_stat3.test,
              ALPHA = ALPHA,
              V = V,
              ETA = ETA/n))
}else{
  return(list(BETA1=BETA1[-c(1:B),],
              BETA2=BETA2[-c(1:B),],
              BETA3=BETA3[-c(1:B),],
              BETA_test1=BETA1.test[-c(1:B),],
```

```
                    BETA_test2=BETA2.test[-c(1:B),],
                    BETA_test3=BETA3.test[-c(1:B),],
                    LAMBDA1 = LAMBDA1[-c(1:B),],
                    LAMBDA2 = LAMBDA2[-c(1:B),],
                    LAMBDA3 = LAMBDA3[-c(1:B),],
                    accept_beta1=accept_beta1/m,
                    accept_beta2=accept_beta2/m,
                    accept_beta3=accept_beta3/m,
                    C_stat1 = C_stat1[-c(1:B)],
                    C_stat2 = C_stat2[-c(1:B)],
                    C_stat3 = C_stat3[-c(1:B)],
                    C_stat_test1 = C_stat1.test[-c(1:B)],
                    C_stat_test2 = C_stat2.test[-c(1:B)],
                    C_stat_test3 = C_stat3.test[-c(1:B)],
                    ALPHA = ALPHA[-c(1:B),],
                    V = V[-c(1:B),],
                    ETA = ETA[-c(1:B),]/n))
  }



}
```

```
system.time({


  alpha0 = 1
  v0 = 1
  eta = length(Y)
  result1 = Hyper_Learning(tti,Y,Y.test,delta,delta.test,tau,
                        A,A.all,beta0,alpha0,v0,kappa,
                        m,B,eta,K.all,n,
                        Wmat_option=0)
})
```

```
##    user  system elapsed
## 1002.65   19.05 1384.88
```

```
par(mfrow=c(3,1))
plot(1:(m-B),result1$ALPHA[,1],type = "l",ylab="alpha",main = "Alpha")
plot(1:(m-B),result1$V[,1],type = "l",ylab="v",main = "V")
plot(1:(m-B),result1$ETA[,1],type = "l",ylab="eta",main = "Eta")
```



Check some of the values

```
tail(result1$ALPHA)
```

```
##                [,1]      [,2]      [,3]
```

```
## [9995,] 1.885718 1.930690 1.840747
## [9996,] 1.885707 1.930677 1.840737
## [9997,] 1.885677 1.930646 1.840709
## [9998,] 1.885652 1.930619 1.840685
## [9999,] 1.885631 1.930597 1.840665
## [10000,] 1.885620 1.930584 1.840655
```

```
tail(result1$V)
```

```
##               [,1]     [,2]     [,3]
## [9995,] 1.885718 1.930690 1.840747
## [9996,] 1.885707 1.930677 1.840737
## [9997,] 1.885677 1.930646 1.840709
## [9998,] 1.885652 1.930619 1.840685
## [9999,] 1.885631 1.930597 1.840665
## [10000,] 1.885620 1.930584 1.840655
```

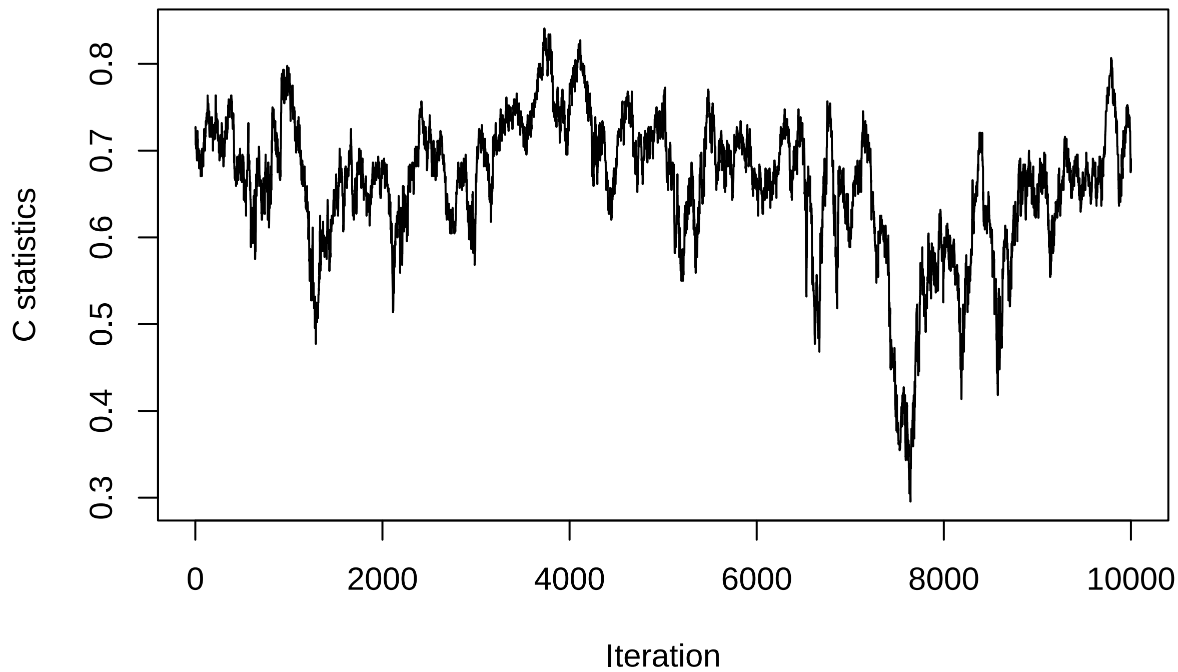
```
tail(result1$ETA)
```

```
##                [,1]      [,2]      [,3]
## [9995,] 0.7238666 0.7241359 0.7235973
## [9996,] 0.7238665 0.7241358 0.7235972
## [9997,] 0.7238663 0.7241356 0.7235971
## [9998,] 0.7238662 0.7241354 0.7235969
## [9999,] 0.7238661 0.7241353 0.7235968
## [10000,] 0.7238660 0.7241352 0.7235967
```

```
system.time({


  alpha0 = result1$ALPHA[m-B,1]
  v0 = result1$V[m-B,1]
  eta = result1$ETA[m-B,1]*length(Y)
  result2 = MH_GP_Sampling(tti,Y,Y.test,delta,delta.test,tau,
                        A,A.all,beta0,alpha0,v0,kappa,
                        m,B,eta,K.all,
                        Wmat_option=0)
})
```

```
##    user  system elapsed
##  234.34    3.25  367.19
```

```
plot(1:(m-B),result2$C_stat_test,type = "l",
     xlab="Iteration",ylab = "C statistics")
```




```
Wmat = HarrellC_Wmat(Y,delta,tau)
Wmat.test = HarrellC_Wmat(Y.test,delta.test,tau)
C_index(colMeans(result2$BETA),Wmat)
```

```
## [1] 0.6816749
```

```
C_index(colMeans(result2$BETA_test),Wmat.test)
```

```
## [1] 0.7090909
```