05-22-2023(2)

Hengde Ouyang

2023-05-22

3. C-statistic for the time-varying AUC

For the same theta, using AUC type weight matrix seems to have a higher value.
(AUC C > Harrell C > Uno C)

```r
AUCC_Wmat <- function(Y, delta, tau){


    # An index indicates whether the observation is censored
    censor = ifelse(delta==0,1,0)

    # Censoring Distribution Estimate using Kaplan-Meier Estimator
    KM_G = survfit(formula = Surv(Y ,censor) ~ 1)





    # Consider the effect of tau
    ordered_time = KM_G$time[KM_G$time<tau]
    difference = diff(ordered_time)

    # Get G(Y) for each observation
    # (Since G(Y) in KM_G is ordered we want each G(Y) to match original Y)
    G_y = KM_G$surv[match(Y,KM_G$time)]



    # Get the survival function for each Tk
    S_t = c()
    for (k in 1:length(KM_G$time)){
      invG_y = (G_y)^(-1)
      invG_y[is.infinite(invG_y)] = 0
      S_tk = sum((Y>KM_G$time[k])*delta*invG_y)/length(Y)
      S_t = c(S_t,S_tk)
    }

    # n > K
    n <- length(Y)
    K = length(ordered_time)
    Wmat <- matrix(0, n, n)
    for(i in 1:n) {
       if(delta[i] != 0) {
          VALUE = matrix(0,K,n)
          for (k in 1:K){
              if (k == 1){  # In this case, k-1 = 0, t0 = 0
                if (S_t[k] ==1){
                  # S(t_k) = 1, 1-S(t_k) = 0, denominator = 0
                  # but we define the value = 0
                  value = rep(0,n)
                }else{
                   value = (Y[i]<ordered_time[k])*(ordered_time[k]<Y)*
                  ordered_time[k]/(1-S_t[k])
                }

                VALUE[k,] = value
              }else{  # k > 1
                if (S_t[k] ==1 | S_t[k-1]==0){
                  # S(t_k) = 1, 1-S(t_k) = 0, denominator = 0
                  # but we define the value = 0
                  value = rep(0,n)
                }else{
                  value = (Y[i]<ordered_time[k])*(ordered_time[k]<Y)*
                  difference[k-1]/(S_t[k-1]*(1-S_t[k])*KM_G$surv[k-1])
                }

                VALUE[k,] = value
              }
          }

          Wmat[i,] <- delta[i]*colSums(VALUE)/(G_y[i]*n^2)
       }
    }
    Wmat <- Wmat/sum(Wmat)
    return(Wmat)
}
```

```
theta_test = colMeans(result_test$THETA)
Wmat1 = HarrellC_Wmat(Y,delta,tau)
Wmat2 = UnoC_Wmat(Y,delta,tau)
Wmat3 = AUCC_Wmat(Y,delta,tau)
data.frame(HarrellC= HarrellC(theta_test,Wmat1),
           UnoC = HarrellC(theta_test,Wmat2),
           AUCC = HarrellC(theta_test,Wmat3))
```

```
##    HarrellC      UnoC      AUCC
## 1 0.6321131 0.6197375 0.6825279
```

4. Gaussian processes MCMC implementation

1. Based on what I have implemented before, I add the procedure of updating the lambda.

2. The function matrix K is used to produce the covariance matrix.

Each element (i,j) represents the kernel for input xi and xj.

3. My GP_MH Sampling algorithm is even more time consuming! Needs improvement.

```r
matrix_K <- function(X,lambda){
  n = dim(X)[1]
  p = dim(X)[2]
  cov_K = matrix(0,n,n)
  for (i in 1:n){
    cov_K[i,] = exp(-0.5*colSums((t(X)[,i]-t(X))^2/lambda))
  }
  return (cov_K)
}
```

```r
GPMH_Sampling <- function(Y,delta,tau,
                        A,beta0,var.prop,
                        m,eta){

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


  # What we want to record
  BETA = matrix(0,m,dim(A)[1])   # The dimension is different from LR
  LAMBDA = matrix(0,m,dim(A)[2])
  C_stat = c()
  # Dimension beta: nx1
  # Dimension lambda: px1

  for (i in 1:m){


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


    # Compute C-statistics from current and last iteration
    Wmat <- HarrellC_Wmat(Y, delta, tau)
    HC.p = HarrellC(beta.p, Wmat)  # Since in Gaussian Process, theta = beta
    HC = HarrellC(beta, Wmat)

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


    # Compute log of MH ratio

    lrMH = eta*log(HC.p) +
          # Note: I modify this part
          dmvnorm(as.numeric(beta.p),beta0,var.prop,log=T)-
          eta*log(HC) -
          dmvnorm(as.numeric(beta.p),beta0,var.prop,log=T)

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




    # Sample lamda from proposal distribution
    lambda.p = exp(t(rnorm(dim(A)[2],log(lambda),rep(1,dim(A)[2]))))

    # Compute log of MH_lambda ratio
    var.prop.p = matrix_K(A,as.numeric(lambda.p))

    lrMH_lambda = dmvnorm(as.numeric(beta),beta0,var.prop.p,log=T)-
                  dmvnorm(as.numeric(beta),beta0,var.prop,log=T)

    if (log(runif(1))<lrMH_lambda){
        lambda = lambda.p
        var.prop = var.prop.p
        accept_lambda = accept_lambda + 1
      }
      LAMBDA[i,] = lambda
  }




  return(list(BETA=BETA,
              LAMBDA = LAMBDA,
              accept_beta=accept_beta/m,
              accept_lambda=accept_lambda/m,
              C_stat = C_stat))

}
```

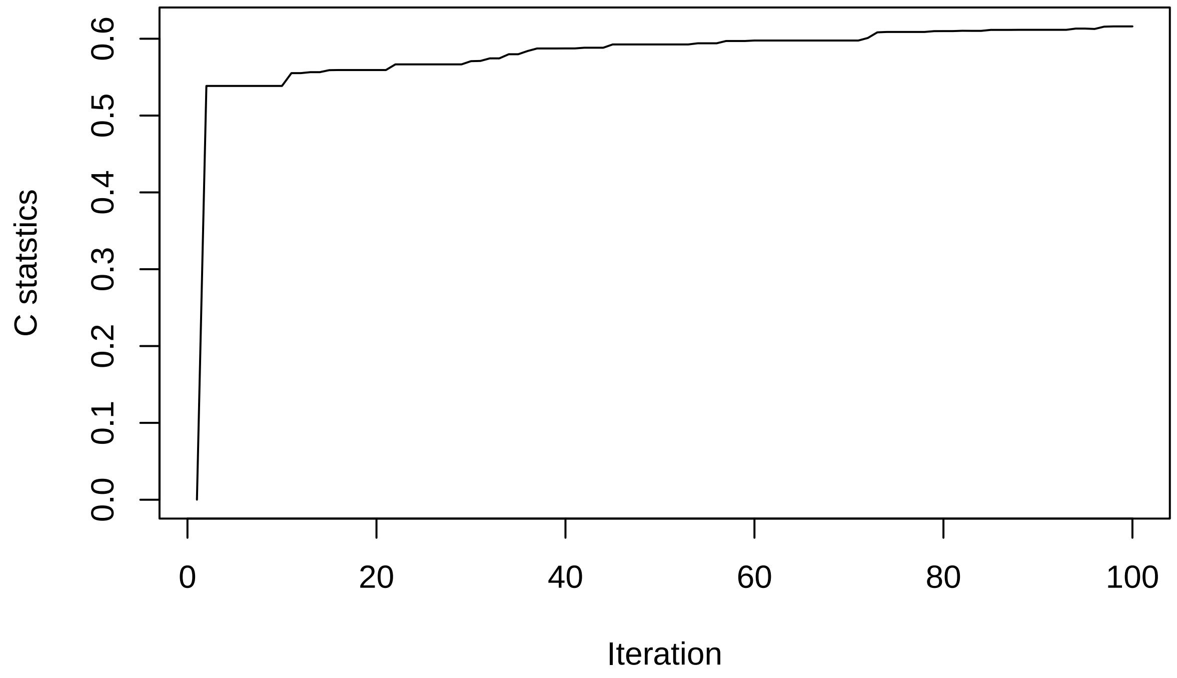
```
beta0 = rep(0,dim(A)[1])
lambda0 = rep(1,dim(A)[2])
var.prop = matrix_K(A,lambda0)
```

```
m = 100

system.time({
  result_GP = GPMH_Sampling(Y,delta,tau,
                        A,beta0,var.prop,
                        m,eta)
})
```

```
##    user  system elapsed
##  378.13    9.06  607.60
```

```
plot(1:m,result_GP$C_stat,xlab = "Iteration",
     ylab = "C statstics",type = "l")
```



Analysis of Why it's so time consuming

It takes a much longer time to sample multivariate normal sample for n = 686

```
# Gaussian Process Case
beta0 = rep(0,dim(A)[1])  # nx1 vector
lambda0 = rep(1,dim(A)[2])
var.prop = matrix_K(A,lambda0)
system.time({ beta.p = t(rmvnorm(1,beta0,var.prop))})
```

```
##    user  system elapsed
##    3.02    0.06    3.96
```

```
# Linear Regression Case
beta0 = rep(0,dim(A)[2])  #
kappa = 10
var.prop = kappa*solve(t(A)%*%A)
system.time({ beta.p = t(rmvnorm(1,beta0,var.prop))})
```

```
##    user  system elapsed
##       0       0       0
```