haoharryfeng / **ISLET**

Type / to search

Code Issues Pull requests Actions Projects Security Insights

ISLET / R / islet.test.R

haoharryfeng 0.99.6 8b8e137 · 2 years ago History

```r
#########################
#########################
#########################
##functions for LRT
#########################
#########################
#########################
#(1) data prep for LRT
#function to change input format wrt to each cell type, to get ready for LRT
changeinput<-function(dc, iK){
  K <- dc@K
  X.tmp1 <- dc@X
  X.tmp2 <- X.tmp1[, -(K+iK)]
  dc@X <- X.tmp2
  return(dc)
}

changeinput_slope<-function(dc, iK){
    K <- dc@K
    X.tmp1 <- dc@X
    ## For slope test, the parameters being tested are: B_t[(3*K+1):(4*K)]
    X.tmp2 <- X.tmp1[, -(3*K+iK)]
    dc@X <- X.tmp2
    return(dc)
}


#(2) LRT function in Unix and Windows
###function to implement EM algorithm in ISLET algorithm


###function to implement EM algorithm in ISLET algorithm
#function here for windows only, using lapply, no parallel computing
islet.lrt.block<-function(Y, datuse, ktest){
  #exp_case = as.matrix(datuse@exp_case)
  #exp_ctrl = as.matrix(datuse@exp_ctrl)
    X <- datuse@X
    A <- datuse@A
    K <- datuse@K
    NU <- datuse@NU
    NS <- datuse@NS
#    para <- datuse@para

    #initialization of parameters parameter estimation storage
    B_est <- NULL
    Sig0_est <- NULL
    SigU_est <- NULL
    E_U_est <- NULL
    llk <- NULL
    ##
    Y <- t(Y)
    G <- ncol(Y)
    #  Y=log2(Y+1)

    ####1. Initialization of parameters
    #1.1 cell type profiles AND csDE B parameters
    #B_0 = solve(X, Y)
    B_0 <- Matrix::tcrossprod( Matrix::tcrossprod(solve( Matrix::crossprod(X)), X), t(Y))
```

```r

    #1.2 error terms
    # sig <- mean((Y-X%*%B_0)^2)
    sig <- colMeans((Y-X%*%B_0)^2)
    #sig <- 20

    #1.3 missing values
    U_0 <- rep(0, NU*K)

    B_t <- B_0
    #sig_t = rep(sig, 7)
    U_t <- U_0
    #sig0_t <- rep(sig, G)
    #sigK_t <- rep(sig, K)
    sig0_t <- sig #rep(sig, G)
    sigK_t <- matrix(rep(sig, each=K), nrow=K)

    iem <- 1
    diff1 <- 100
    diff2 <- 100
    pp <- 1
    norm <- mean(colMeans(Y))

    #Sig_U = diag(rep(sigK_t, each = NU))
    Sig_p<-lapply(seq_len(G), function(x, A, sig0_t, sigK_t, NU, Y, X, B_t){
        #invSig_U<-Matrix::bdiag(diag(rep(1/sigK_t, each=NU)))
        invSig_U<-Matrix::bdiag(diag(rep(1/sigK_t[,x], each=NU)))
        Sig<-solve( Matrix::crossprod(A)/sig0_t[x]+invSig_U)
        hftmp1 <- Matrix::tcrossprod(Sig, A)
        hftmp2 <- BiocGenerics::t(Y[, x] - Matrix::tcrossprod(X, BiocGenerics::t(B_t[, x])))
        U<- Matrix::tcrossprod(hftmp1, hftmp2)/sig0_t[x]
        return(list(Sig=Sig, U=U))
    }, A, sig0_t, sigK_t, NU, Y, X, B_t)
    E_Up<-do.call(cbind, lapply(Sig_p, function(x)x$U))


    while(iem<15){
#         cat("iteration=", iem, "\n")
        iem <- iem + 1
        ####2. E-step
        #observed data COV(Y) = V

        #V = A%*%Sig_U%*%t(A) + diag(rep(sig0_t, 5*600))

        #2.1 E[U|Y]: missing data [U|Y] given observed data
        #invV = solve(V)
        # E_U = mu_p = t(Sig_U) %*% t(A) %*% invV %*% (Y - X %*% B_t)
        # Sig_p = Sig_U - crossprod(Sig_U,t(A)) %*% invV %*% A %*% Sig_U

        # Estimate from last iteration

        E_U <- E_Up
        mu_p <- E_Up
        E_U_frame <- as.data.frame(as.matrix(E_U))
        #2.2 E[t(S)S|Y]
        E_StS <- lapply(seq_len(G), function(x, A, Sig_p, mu_p, X, B_t, Y){
            sum( Matrix::diag(Matrix::tcrossprod( Matrix::tcrossprod(A, Sig_p[[x]]$Sig), A))) +
                sum(( Matrix::tcrossprod(A, BiocGenerics::t(mu_p[, x])) +
                      Matrix::tcrossprod(X, BiocGenerics::t(B_t[, x])) - Y[, x])^2)},
            A, Sig_p, mu_p, X, B_t, Y)
        E_StS <-unlist(E_StS)

        #2.3 E[U_k^T U_k|Y]
        mutra_split <- lapply(Sig_p, function(x){
            sig_p<-split(diag(x$Sig), ceiling(seq_len(NU*K)/NU))
            tra<-unlist(lapply(sig_p, sum))
            return(tra)
        })
        mu_split <- split(E_U_frame, ceiling(seq_along(E_U_frame[, 1])/NU))

        E_UkTUk <- do.call('cbind', mutra_split) + do.call('rbind', lapply(mu_split, colss))
```

$u \sim N(0,$

每种cell type 是1个 σ_i^2

相当于每个人有一个 J×K 的 vector

indicate individual variance

$u \in (JK)\times 1.$

Given that $A \in N\times(JK)$

$\Sigma_u \in (0,+\infty)^{(JK)\times(JK)}$

$\Rightarrow Au \in \mathbb{R}^{JK\times 1}$

每个人自己的

$$\Rightarrow u = \begin{bmatrix} u_{11} \\ u_{21} \\ \vdots \\ u_{J1} \\ u_{12} \\ u_{22} \\ \vdots \\ u_{JK} \end{bmatrix}$$

$$\Sigma_u = \begin{bmatrix} \sigma_1^2 & & & & & & \\ & \sigma_1^2 & & & & & \\ & & \ddots & & & & \\ & & & \sigma_1^2 & & & \\ & & & & \sigma_2^2 & & \\ & & & & & \sigma_2^2 & \\ & & & & & & \ddots \\ & & & & & & & \ddots \\ & & & & & & & & \sigma_K^2 \end{bmatrix}_{(JK)\times(JK)}$$

param of interest: $\beta, \sigma_0, u.$

fixed effect

$y|u \sim N(X\beta + Au, \sigma_0^2 I)$

```
        ####3. M-step
        #3.1 B
        B_tp <- Matrix::tcrossprod( Matrix::tcrossprod(solve( Matrix::crossprod(X)), X),
                                  BiocGenerics::t(Y- Matrix::tcrossprod(A, BiocGenerics::t(E_U))) )
```

迭代 regression algorithm

```

        #make correction in case B[1:K]<0 or B_tp[(K+1):2K]<0
        #important to bound the estimation to positive values

        # B_tp[1:K,]=ifelse(B_tp[1:K,]<0,0,B_tp[1:K,])
        # B_tp[-(1:K),]=ifelse(B_tp[1:K,]+B_tp[-(1:K),]<0,-B_tp[1:K,],B_tp[-(1:K),])


        #3.2 sigma_0^2
        sig0_tp <- E_StS/(NS)

        #3.3 sigma_k^2
        sigK_tp <- E_UkTUk/(NU)

        ####4. Stopping criteria
        diff1 <- sum(abs(B_tp - B_t)) + abs(sig0_tp - sig0_t) + sum(abs(sigK_tp - sigK_t))
```

$$\text{diff1} = \text{sum}(|\beta^{(t+1)} - \beta^{(t)}| + |\sigma_0^{(t+1)} - \sigma_0^{(t)}| +$$

```

        n1 <- sum(abs(B_tp - B_t))/length(B_tp)
        n2 <- sum(abs(B_tp))/length(B_tp)
        pp <- n1/n2
#        cat("B_sum_val=", n2, "\n")
#        cat("B_change_val=", n1, "\n")
#        cat("B_change_prop=", pp*100, "% \n")

        ####5. Update params
        B_t<-B_tp
        sig0_t <- sig0_tp
        sigK_t <- sigK_tp

        Sig_p<-lapply(seq_len(G), function(x, A, sig0_t, sigK_t, NU, Y, X, B_t){
            Sig_U<- Matrix::bdiag(diag(rep(sigK_t[, x], each=NU)))
            invSig_U<-Matrix::bdiag(diag(rep(1/sigK_t[, x], each=NU)))
```

$$\text{diff2} = \underbrace{\text{sum}(|\mathbb{E}u^{(t+1)} - \mathbb{E}u^{(t)}|}_{}) / (JK \cdot \bar{y}^2)$$

random-effect.

average 的 $\mathbb{E}u$ 的变化量.

given that $JK \cdot \bar{y}$ 已知, 只考虑 diff2 的分子, < 1e-5

```
            invSig_U<- Matrix::bdiag(diag(rep(1/sigK_t[, x], each=NU)))
            Sig<-solve( Matrix::crossprod(A)/sig0_t[x]+invSig_U)
            U<- Matrix::tcrossprod(Matrix::tcrossprod(Sig, A),
                                   t(as.matrix(Y[, x] -
                                Matrix::tcrossprod(X, BiocGenerics::t(B_t[, x])))) )/sig0_t[x]
            return(list(Sig_U=Sig_U, Sig=Sig, U=U))
        }, A, sig0_t, sigK_t, NU, Y, X, B_t)
        Sig_p_all<-NULL
        # Sig_p_all=do.call(rbind, Sig_p)
        # E_Up_all=Sig_p_all%*% t(A)%*% (Y - X %*% B_t)/sig0_t
        # E_Up_diag=split(as.data.frame(as.matrix(E_Up_all)), rep(1:G, each=ncol(A)))
        E_Up<-do.call(cbind, lapply(Sig_p, function(x)x$U))

        diff2 <- sum(abs(E_Up - E_U))/(length(E_U)*mean(colMeans(Y))^2)
#        cat("Random effect diff2=", diff2, "\n")

    }
    # Estimate of fixed effect
    B_est<-cbind(B_est, B_t)
    # Estimate of random effect
    E_U_est<-cbind(E_U_est, E_Up)
    # Estimate of variance Sigma_U, Sigma_0
    Sig0_est <- cbind(Sig0_est, sig0_t)
    SigU_est <- cbind(SigU_est, sigK_t)

    #calculate LLK
    llk<-lapply(seq_len(G), function(x){
        Sig<- Matrix::tcrossprod( Matrix::tcrossprod(A, Sig_p[[x]]$Sig_U), A)+
            Matrix::bdiag(diag(sig0_t[x], nrow = nrow(A)))
        l<- Matrix::determinant(Sig)$modulus+
            Matrix::tcrossprod(Matrix::crossprod(Y[, x]- Matrix::tcrossprod(X, BiocGenerics::t(B_t[, x])),
                                                 solve(Sig)),
                               BiocGenerics::t(Y[, x]- Matrix::tcrossprod(X,
                                                                          BiocGenerics::t(B_t[, x]))) )
        return(-as.numeric(l))
    })
```

```r
    llk<-unlist(llk)

  #compile return list

  LLK <- llk
    return(LLK)

  #cat("Complete: LRT calculation for one cell type.")
}


###Wrap function to run ISLET LRT, using parallel computing
#ipc is the index of parallel computing for

isletTest<-function(input, BPPARAM=bpparam() ){
    G <- nrow(input@exp_case)
    type<-input@type
    Yall<-as.matrix(cbind(input@exp_case, input@exp_ctrl))
    aval.nworkers<-BPPARAM$workers
    block.size<-max(ceiling(G/aval.nworkers), 5)
    Yall.list <- split(as.data.frame(Yall), ceiling(seq_len(G)/block.size))

#  if(.Platform$OS.type == "unix") {
    ## do some parallel computation under Unix
#      multicoreParam <- MulticoreParam(workers = ncores)
    mf <- bplapply(X=Yall.list, islet.solve.block, datuse=input, BPPARAM=BPPARAM)
    #use islet.lrt.unix

    test.fun<-function(iTest){
        cat("csDE testing on cell type",iTest, "\n")
        if(type == 'intercept'){
            inputnew <- changeinput(dc=input, iK=iTest)
        }else{inputnew <- changeinput_slope(dc=input, iK=iTest)}

        tmp1 <- bplapply(X=Yall.list, islet.lrt.block, datuse=inputnew,
                         ktest=iTest, BPPARAM=BPPARAM)
                         Ktest=Ttest, BPPARAM=BPPARAM)
        tmp2 <- unlist(tmp1)
        tmp3 <- unlist(lapply(mf, '[[' , 7))
        ###obtain the p-values from each cell type
        tmp4 <- LRT(tmp3, tmp2, df=1)
        return(tmp4)
    }
    test.list<-lapply(seq_len(input@K), FUN=test.fun)
    test.res<-do.call(cbind,test.list)
    colnames(test.res) <- colnames(input@X[, seq_len(input@K)])
    cat("csDE testing on", input@K,"cell types finished", "\n")
#    }else {
    ## This will be windows
    ## Use serial param or do not use any parallel functions, just use ‘lapply’
    ## result should be of the same “type” from both the if and else statements.

#    nworkers<-ncores
#    cl <- makeCluster(nworkers)

    ## Remove clusterExport(), clusterEvalQ() if use devtools::install() to build package
#    clusterExport(cl,list('colss'))
#    clusterEvalQ(cl,{
#        library(Matrix)
#        library(BiocGenerics)})

#    mf <- parLapply(cl, X=Yall.list, islet.solve.block, datuse = input)


#    test.list<-lapply(seq_len(input@K),FUN=test.fun)
#    test.res<-do.call(cbind,test.list)
 #   colnames(test.res) <- colnames(input@X[, seq_len(input@K)])
 #   cat("csDE testing on", input@K,"cell types finished", "\n")
#    stopCluster(cl)
#  }
  return(test.res)
}
```