Non conveixity issue

Chanwoo Lee

Dec 03, 2019

1 Non convexity in the intercept and the slope

Let us assume $\Theta = \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3 = g(\mathcal{C}, A_1, A_2, A_3)$. For CP decomposition model, the log-likelihood is

$$\mathcal{L}_{\mathcal{Y}}(\mathcal{C}, A_1, A_2, A_3, \boldsymbol{\omega}) = \sum_{l\in[K]} \sum_{y_{ijk}=l} \log(f_l(g(\mathcal{C}, A_1, A_2, A_3)).$$

where $f_l(\theta) = \phi(w_l + \theta) - \phi(w_{l-1} + \theta)$, and ϕ is logistic function such that $\phi(x) = \frac{1}{1+e^{-x}}$. By the diccussion in [1] and [2], this log-likelihood function is concave as long as a function g of parameters is linear. Unfortunately, this is not the case because of the structure of CP-decomposition structure, which makes g non linear function with respect to $(\mathcal{C}, A_1, A_2, A_3)$. However, we can prove that the log-likelihood with respect to $(\boldsymbol{\omega}, \mathcal{C})$ is concave for fixed A_1, A_2, A_3. It is based on the vectorization of Θ.

$$\text{vec}(\Theta) = (A_3 \otimes A_2 \otimes A_1)\,\text{vec}(\mathcal{C}) = A\,\text{vec}(\mathcal{C}).$$

where $A = A_3 \otimes A_2 \otimes A_1$. Let us define a function Index : $\mathbb{R}^3 \to \mathbb{R}$ such that $\Theta_{ijk} = \text{vec}(\Theta)_{\text{Index}(i,j,k)}$ From the above formulas, we can write the log-likelihood function for fixed A_1, A_2, A_3 as

$$\mathcal{L}_{\mathcal{Y}}(\mathcal{C}, \boldsymbol{\omega}|A_1, A_2, A_3) = \sum_{l\in[K]} \sum_{y_{ijk}=l} \log(f_l(g(\mathcal{C}|A_1, A_2, A_3)) = \sum_{l\in[K]} \sum_{y_{ijk}=l} \log(f_l(a^T_{\text{Index}(i,j,k)}\,\text{vec}(\mathcal{C}))).$$

where a_i is i-th row of A. Therefore, we have linear function g with respect to $\mathcal{C}$.

2 $\mathcal{C}$ and ω together

To update $\mathcal{C}$ and $\boldsymbol{\omega}$ together, I changed several part of function.R code.

1. I used "optim" function instead of "nlminb" function. There are the main two reasons for this change. Firstly, calculating Hessian with respect to $(\boldsymbol{\omega}, \mathcal{C})$ is not an easy work. Secondly, $\boldsymbol{\omega}$ has a constraint that $\omega_1 < \omega_2 \cdots < \omega_{K-1}$. To perform constrained optimization, "optim" function is better.

2. In the previous condes, $\boldsymbol{\omega}$ is only updated once. For this reason, I changed the structure of the codes to make sure that $\boldsymbol{\omega}$ is updated for each iteration when $\boldsymbol{\omega}$ is unknown.

3. I changed order of updates when the threshold α is given. In the previous codes, order of updates remains the same as our first codes. To get α constrained optimizer, I put $(\boldsymbol{\omega}, \mathcal{C})$ in front.

3 Code

```
library(MASS)
library(rTensor)
library(pracma)
library(ggplot2)
library(ggthemes)
library(gridExtra)
library(Matrix)


likelihood = function(ttnsr,thet,alpha){
  p1 = logistic(c(thet) + alpha[1])
  p2 = logistic(c(thet) + alpha[2])
  p = cbind(p1,p2-p1,1-p2)
  return(-sum(log(c(p[which(c(ttnsr)==1),1],p[which(c(ttnsr)==2),2],p[
    which(c(ttnsr)==3),3]))))
}

logistic = function(x){
  return(1/(1+exp(-x)))
```

```
}

########### simulate ordinal tensors based on logistic model
# realization = function(tnsr,alpha){
#   tnsr=as.tensor(tnsr)
#   thet <- k_unfold(tnsr,1)@data
#   theta1 <- thet + alpha[1]
#   theta2 <- thet + alpha[2]
#   result <- k_unfold(tnsr,1)@data
#   p1 <- logistic(theta1)
#   p2 <- logistic(theta2)-logistic(theta1)
#   p3 <- matrix(1,nrow = nrow(thet),ncol = ncol(thet))-logistic(theta2)
#   for (i in 1:nrow(thet)) {
#     for(j in 1:ncol(thet)){
#       result[i,j] <- sample(c(1,2,3),1,prob= c(p1[i,j],p2[i,j],p3[i,j]))
#     }
#   }
#   return(k_fold(result,1,modes = tnsr@modes))
# }


# ########### Hessian w.r.t. A_1 ###########
# Hessi = function(A_1,W1,ttnsr,omega){
#   thet =W1%*%c(A_1)
#   p1 = logistic(thet + omega[1])
#   p2 = logistic(thet + omega[2])
#   q1 = p1*(1-p1)
#   q2 = p2*(1-p2)+p1*(1-p1)
#   q3 = p2*(1-p2)
#   H = t(rbind(W1[which(c(ttnsr)==1),])*q1[which(c(ttnsr)==1)])%*%rbind(
    W1[which(c(ttnsr)==1),])+t(rbind(W1[which(c(ttnsr)==2),])*q2[which(c(
    ttnsr)==2)])%*%rbind(W1[which(c(ttnsr)==2),])+t(rbind(W1[which(c(ttnsr)
    ==3),])*q3[which(c(ttnsr)==3)])%*%rbind(W1[which(c(ttnsr)==3),])
#   return(H)
# }
#
# ########### cost function ###########
# h1 = function(A_1,W1,ttnsr,omega){
#   thet =W1%*%c(A_1)
```

```
#   p1 = logistic(thet + omega[1])
#   p2 = logistic(thet + omega[2])
#   p = cbind(p1,p2-p1,1-p2)
#   return(-sum(log(c(p[which(c(ttnsr)==1),1],p[which(c(ttnsr)==2),2],p[which(c(ttnsr)==3),3]))))
# }
#
# ########### gradient ###########
# g1 = function(A_1,W1,ttnsr,omega){
#   thet =W1%*%c(A_1)
#   p1 = logistic(thet + omega[1])
#   p2 = logistic(thet + omega[2])
#   q1 <- p1-1
#   q2 <- (p2*(1-p2)-p1*(1-p1))/(p1-p2)
#   q3 <- p2
#   gd = apply(rbind(W1[which(c(ttnsr)==1),])*q1[which(c(ttnsr)==1)],2,sum)+apply(rbind(W1[which(c(ttnsr)==2),])*q2[which(c(ttnsr)==2)],2,sum)+apply(rbind(W1[which(c(ttnsr)==3),])*q3[which(c(ttnsr)==3)],2,sum)
#
#   return(gd)
# }

########### Hessian w.r.t. A_1 ###########
Hessi = function(A_1,W1,ttnsr,omega){
  k = length(omega)
  thet =W1%*%c(A_1)
  p = matrix(nrow = length(thet),ncol = k)
  for (i in 1:k) {
    p[,i] = logistic(thet + omega[i])
  }
  q = matrix(nrow = length(thet),ncol = k+1)
  q[,1] = p[,1]*(1-p[,1])
  for (i in 2:k) {
    q[,i] =  p[,i]*(1-p[,i])+p[,i-1]*(1-p[,i-1])
  }
  q[,k+1] = p[,k]*(1-p[,k])
  l= lapply(1:(k+1),function(i) t(rbind(W1[which(c(ttnsr)==i),])*q[which(c(ttnsr)==i),i])%*%rbind(W1[which(c(ttnsr)==i),]))
  return(Reduce("+", l))
```

```
}


########### cost function ###########
h1 = function(A_1,W1,ttnsr,omega){
  k = length(omega)
  thet =W1%*%c(A_1)
  p = matrix(nrow = length(thet),ncol = k)
  for (i in 1:k) {
    p[,i] = logistic(thet + omega[i])
  }
  p =  cbind(p,rep(1,length(thet)))-cbind(rep(0,length(thet)),p)
  l = lapply(1:(k+1),function(i) -log(p[which(c(ttnsr)==i),i]))
  return(sum(unlist(l)))
}

########### gradient ###########
g1 = function(A_1,W1,ttnsr,omega){
  k = length(omega)
  thet =W1%*%c(A_1)
  p = matrix(nrow = length(thet),ncol = k)
  for (i in 1:k) {
    p[,i] = logistic(thet + omega[i])
  }
  q = matrix(nrow = length(thet),ncol = k+1)
  q[,1] <- p[,1]-1
  for (i in 2:k) {
    q[,i] <-  (p[,i]*(1-p[,i])-p[,i-1]*(1-p[,i-1]))/(p[,i-1]-p[,i])
  }
  q[,k+1] <- p[,k]
  l <- lapply(1:(k+1),function(i) apply(rbind(W1[which(c(ttnsr)==i),])*q[
    which(c(ttnsr)==i),i],2,sum))
  return(Reduce("+", l))
}

g2 = function(core,W4,ttnsr,omega){
  k = length(omega)
  thet =W4%*%c(core)
  p = matrix(nrow = length(thet),ncol = k)
```

```
  for (i in 1:k) {
    p[,i] = logistic(thet + omega[i])
  }
  #derivative for C
  q = matrix(nrow = length(thet),ncol = k+1)
  q[,1] <- p[,1]-1
  for (i in 2:k) {
    q[,i] <-  (p[,i]*(1-p[,i])-p[,i-1]*(1-p[,i-1]))/(p[,i-1]-p[,i])
  }
  q[,k+1] <- p[,k]

  #deriavtive for omega
  r = list()
  r[[1]] <- cbind(p[,1]-1,p[,1]*(1-p[,1])/(p[,2]-p[,1]))
  for (i in 2:(k-1)) {
    r[[i]] <-  cbind(-p[,i]*(1-p[,i])/(p[,i]-p[,i-1]),p[,i]*(1-p[,i])/(p[,
     i+1]-p[,i]))
  }
  r[[k]] <- cbind(-p[,k]*(1-p[,k])/(p[,k]-p[,k-1]),p[,k])


  l1 <- vector(length = k)
  for (i in 1:k) {
    l1[i] <- sum(r[[i]][which(c(ttnsr)==i),1])+sum(r[[i]][which(c(ttnsr)==
     i+1),2])
  }
  l2 <- lapply(1:(k+1),function(i) apply(rbind(W4[which(c(ttnsr)==i),])*q[
     which(c(ttnsr)==i),i],2,sum))
  return(c(l1,Reduce("+", l2)))
}


######### realization ############

realization = function(theta,omega){
  k = length(omega)
  theta=as.tensor(theta)
  thet <- c(theta@data)
  p = matrix(nrow = length(thet),ncol = k)
```

```r
  for (i in 1:k) {
    p[,i] = logistic(thet + omega[i])
  }
  p =  cbind(p,rep(1,length(thet)))-cbind(rep(0,length(thet)),p)
  for (j in 1:length(thet)) {
    thet[j] <-  sample(1:(k+1),1,prob = p[j,])
  }
  as.tensor(array(thet,dim =theta@modes))
  return(as.tensor(array(thet,dim =theta@modes)))
}



####### update a factor matrix at one time while holding others fixed
    ###########
comb = function(A,W,ttnsr,k,omega,alph=TRUE){
  nA = A
  tnsr1 <- k_unfold(as.tensor(ttnsr),k)@data
  if (alph==TRUE) {
    l <- lapply(1:nrow(A),function(i){optim(A[i,],function(x) h1(x,W,tnsr1
    [i,],omega),function(x) g1(x,W,tnsr1[i,],omega),method = "BFGS")$par})

    nA <- matrix(unlist(l),nrow = nrow(A),byrow = T)
  }else{
    l <- lapply(1:nrow(A),function(i){constrOptim(A[i,],
                                                 function(x) h1(x,W,tnsr1
    [i,],omega),function(x) g1(x,W,tnsr1[i,],omega),
                                                 ui = as.matrix(rbind(W,-
    W)),ci = rep(-alph,2*nrow(W)),method = "BFGS")$par})
    nA <- matrix(unlist(l),nrow = nrow(A),byrow = T)
  }
  return(nA)
}

####### update core tensor #######
corecomb = function(C,W,ttnsr,omega,alph=TRUE){
  Cvec <- c(C@data)
  h <- function(x) h1(x,W,ttnsr,omega)
  g <- function(x) g1(x,W,ttnsr,omega)
```

```
  H <- function(x) Hessi(x,W,ttnsr,omega)
  d <- nlminb(Cvec,h,g,H)
  ##d <- optim(Cvec,h,g,method="BFGS")  ## seems BFGS is faster??
  C <- new("Tensor",C@num_modes,C@modes,data =d$par)
  return(C)
}
### updating core tensor and omega
coreomgcomb = function(C,W,ttnsr,omega,alph=TRUE){
  omgc = list()
  k = length(omega)
  coeff <- c(omega,C@data)
  u = cbind(-diag(k-1),matrix(0,nrow = k-1,ncol = length(coeff)-k+1))+
    cbind(0,diag(k-1),matrix(0,nrow = k-1,ncol = length(coeff)-k))
  h <- function(x) h1(x[-(1:k)],W,ttnsr,x[1:k])
  g <- function(x) g2(x[-(1:k)],W,ttnsr,x[1:k])
  d <- constrOptim(coeff,h,g,method="BFGS",ui = u, ci = rep(0,k-1))  ## seems BFGS is faster??
  omgc$C <- new("Tensor",C@num_modes,C@modes,data =d$par[-(1:k)])
  omgc$omega = omega = d$par[1:k]
  return(omgc)
}
####### update core tensor #######
prevcorecomb = function(C,W,ttnsr,omega,alph=TRUE){
  Cvec <- c(C@data)
  h <- function(x) h1(x,W,ttnsr,omega)
  g <- function(x) g1(x,W,ttnsr,omega)
  H <- function(x) Hessi(x,W,ttnsr,omega)


  if (alph==TRUE) {
    d <- nlminb(Cvec,h,g,H)
    C <- new("Tensor",C@num_modes,C@modes,data =d$par)
  }else{
    d <- constrOptim(Cvec,h,g,ui = rbind(W,-W),ci = rep(-alph,2*nrow(W)), method = "BFGS")
    C <- new("Tensor",C@num_modes,C@modes,data =d$par)
  }
  return(C)
}
```

```
########ordinal tensor decomposition based on Tucker structure #######
fit_ordinal = function(ttnsr,C,A_1,A_2,A_3,omega=TRUE,alph = TRUE){
  alphbound <- alph+10^-4
  result = list()
  error<- 3
  iter = 0
  cost=NULL
  omg = omega
  if(sum(omg)==TRUE) omega <- polr(as.factor(c(ttnsr[ttnsr>0]))~offset(-c(
   prevtheta[ttnsr>0])))$zeta
  if (alph == TRUE) {
    while ((error > 10^-4)&(iter<50) ) {
      iter = iter +1

      prevtheta <- ttl(C,list(A_1,A_2,A_3),ms=1:3)@data
      prev <- likelihood(ttnsr[ttnsr>0],prevtheta[ttnsr>0],omega)

      if(sum(omg)==TRUE) {
        # update C and omega together
        W4 <- kronecker(kronecker(A_3,A_2),A_1)
        coreomg <- coreomgcomb(C,W4,c(ttnsr),omega)
        C <- coreomg$C
        omega <- coreomg$omega
      }else{
        # update C
        W4 <- kronecker(kronecker(A_3,A_2),A_1)
        C <- corecomb(C,W4,c(ttnsr),omega)
      }


      #update A_1
      W1 = kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
      A_1 <- comb(A_1,W1,ttnsr,1,omega)
      #orthognalize A_1
      qr_res=qr(A_1)
      A_1=qr.Q(qr_res)
      C=ttm(C,qr.R(qr_res),1)
```

```
    # update A_2
    W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
    A_2 <- comb(A_2,W2,ttnsr,2,omega)
    #orthognalize A_2
    qr_res=qr(A_2)
    A_2=qr.Q(qr_res)
    C=ttm(C,qr.R(qr_res),2)

    # update A_3
    W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
    A_3 <- comb(A_3,W3,ttnsr,3,omega)
    #orthognalize A_3
    qr_res=qr(A_3)
    A_3=qr.Q(qr_res)
    C=ttm(C,qr.R(qr_res),3)

    theta <- ttl(C,list(A_1,A_2,A_3),ms=1:3)@data
    new <- likelihood(ttnsr[ttnsr>0],theta[ttnsr>0],omega)
    cost = c(cost,new)
    (error <- abs((new-prev)/prev))
  }
}else{
  while ((error > 10^-4)&(iter<50) ) {
    iter = iter +1

    prevtheta <- ttl(C,list(A_1,A_2,A_3),ms=1:3)@data
    prev <- likelihood(ttnsr[ttnsr>0],prevtheta[ttnsr>0],omega)

    if(sum(omega)==TRUE) {
      # update C and omega together
      W4 <- kronecker(kronecker(A_3,A_2),A_1)
      coreomg <- coreomgcomb(C,W4,c(ttnsr),omega)
      C <- coreomg$C
      omega <- coreomg$omega
    }else{
      # update C
      W4 <- kronecker(kronecker(A_3,A_2),A_1)
      C <- corecomb(C,W4,c(ttnsr),omega)
```

```
      }

      #update A_1
      W1 =kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
      A_1 <- comb(A_1,W1,ttnsr,1,omega,alphbound)
      if(max(abs(ttl(C,list(A_1,A_2,A_3),ms=1:3)@data))>=alph) break
      #orthognalize A_1
      qr_res=qr(A_1)
      A_1=qr.Q(qr_res)
      C=ttm(C,qr.R(qr_res),1)


      # update A_2
      W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
      A_2 <- comb(A_2,W2,ttnsr,2,omega,alphbound)
      if(max(abs(ttl(C,list(A_1,A_2,A_3),ms=1:3)@data))>=alph) break
      #orthognalize A_2
      qr_res=qr(A_2)
      A_2=qr.Q(qr_res)
      C=ttm(C,qr.R(qr_res),2)


      # update A_3
      W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
      A_3 <- comb(A_3,W3,ttnsr,3,omega,alphbound)
      if(max(abs(ttl(C,list(A_1,A_2,A_3),ms=1:3)@data))>=alph) break
      #orthognalize A_3
      qr_res=qr(A_3)
      A_3=qr.Q(qr_res)
      C=ttm(C,qr.R(qr_res),3)

      theta <- ttl(C,list(A_1,A_2,A_3),ms=1:3)@data
      new <- likelihood(ttnsr[ttnsr>0],theta[ttnsr>0],omega)
      cost = c(cost,new)
      error <- abs((new-prev)/prev)
      if(max(abs(ttl(C,list(A_1,A_2,A_3),ms=1:3)@data))>=alph) break
    }
  }

```

```
  result$C <- C; result$A_1 <- A_1; result$A_2 <- A_2; result$A_3 <- A_3
  result$iteration <- iter
  result$cost = cost; result$omega=omega
  return(result)
}


####### ordinal tensor decomposition based on CP structure #######
fit_ordinal_cp=function(ttnsr,A_1,A_2,A_3,omega=TRUE,alph = TRUE){
  alphbound <- alph+10^-4
  result = list()
  error<- 3
  iter = 0
  cost=NULL
  if (alph == TRUE) {
    while ((error > 10^-4)&(iter<50) ) {
      iter = iter +1

      prevtheta <- tensorize(A_1,A_2,A_3)
      #update omega
      if(sum(omega)==TRUE) omega <- polr(as.factor(c(ttnsr[ttnsr>0]))~
    offset(-c(prevtheta[ttnsr>0])))$zeta
      prev <- likelihood(ttnsr[ttnsr>0],prevtheta[ttnsr>0],omega)

      #update A_1
      W1 = KhatriRao(A_3,A_2)
      A_1 <- comb(A_1,W1,ttnsr,1,omega)
      A_1 = apply(A_1,2,function(x){x/norm(x,"2")})


      # update A_2
      W2 <- KhatriRao(A_3,A_1)
      A_2 <- comb(A_2,W2,ttnsr,2,omega)
      A_2 = apply(A_2,2,function(x){x/norm(x,"2")})

      # update A_3
      W3 <-  KhatriRao(A_2,A_1)
      A_3 <- comb(A_3,W3,ttnsr,3,omega)

```

```r

    theta <- tensorize(A_1,A_2,A_3)
    new <- likelihood(ttnsr[ttnsr>0],theta[ttnsr>0],omega)
    cost = c(cost,new)
    (error <- abs((new-prev)/prev))
  }
}else{
  while ((error > 10^-4)&(iter<50) ) {
    iter = iter +1

    prevtheta <- tensorize(A_1,A_2,A_3)
    #update omega
    if(sum(omega)==TRUE) omega <- polr(as.factor(c(ttnsr[ttnsr>0]))~
 offset(-c(prevtheta[ttnsr>0])))$zeta
    prev <- likelihood(ttnsr[ttnsr>0],prevtheta[ttnsr>0],omega)

    #update A_1
    W1 = KhatriRao(A_3,A_2)
    A_1 <- comb(A_1,W1,ttnsr,1,omega,alphbound)
    if(max(abs(tensorize(A_1,A_2,A_3)))>=alph) break


    # update A_2
    W2 <- KhatriRao(A_3,A_1)
    A_2 <- comb(A_2,W2,ttnsr,2,omega,alphbound)
    if(max(abs(tensorize(A_1,A_2,A_3)))>=alph) break

    # update A_3
    W3 <-  KhatriRao(A_2,A_1)
    A_3 <- comb(A_3,W3,ttnsr,3,omega,alphbound)
    if(max(abs(tensorize(A_1,A_2,A_3)))>=alph) break

    pre=rescale(A_1,A_2,A_3)
    A_1=pre$A_1
    A_2=pre$A_2
    A_3=pre$A_3


    theta <- tensorize(A_1,A_2,A_3)
```

```
      new <- likelihood(ttnsr[ttnsr>0],theta[ttnsr>0],omega)
      cost = c(cost,new)
      error <- abs((new-prev)/prev)
      if(max(abs(tensorize(A_1,A_2,A_3)))>=alph) break
    }
  }

  result$A_1 <- A_1; result$A_2 <- A_2; result$A_3 <- A_3
  result$iteration <- iter
  result$cost = cost; result$omega=omega
  return(result)
}


##### construct tensor from CP factors
tensorize=function(A_1,A_2,A_3){
  r=ncol(A_1)
  tensor=0
  for(i in 1:r){
    tensor=tensor+A_1[,i]%o%A_2[,i]%o%A_3[,i]
  }
  return(tensor)
}


## BIC: Inputs d and r are vectors.
bic = function(ttnsr,theta,omega,d,r){
  return(2*likelihood(ttnsr,theta,omega)+(prod(r)+sum(r*(d-r)))*log(prod(d
    )))
}



rescale=function(A_1,A_2,A_3){
  r=ncol(A_1)
  for(i in 1:r){
    A_3[,i]=A_3[,i]*sqrt(sum(A_1[,i]^2))*sqrt(sum(A_2[,i]^2))
    A_2[,i]=A_2[,i]/sqrt(sum(A_2[,i]^2))
    A_1[,i]=A_1[,i]/sqrt(sum(A_1[,i]^2))
```

```
  }
  return(list("A_1"=A_1,"A_2"=A_2,"A_3"=A_3))
}
```

References

[1] Peter McCullagh (1980). *Regression Models for Ordinal Data* Journal of the Royal Statistical Society. Series B (Methodological) Vol. 42, No. 2 , pp. 109-142 (34 pages)

[2] BURRIDGE, J. (1980). *A note on maximum likelihood estimation for regression models using grouped data.* J. R. Statist. Soc. B, 42, in press.