swCAM User Manual

Chiung-Ting Wu

3/6/2021

Contents

1 Introduction

Sample-wise Convex Analysis of Mixtures (swCAM) is a deconvolution method that can estimate subtype proportions and subtype-specific expressions in individual samples from bulk tissue transcriptomes. We extend our previous CAM framework (https://github.com/Lululuella/debCAM) to include a new term accounting for between-sample variations and formulate swCAM as a nuclear-norm and $\mathbf{l_{2,1}}$-norm regularized low-rank matrix factorization problem. We determine hyperparameter value using a cross-validation scheme with random entry exclusion and obtain swCAM solution using an efficient alternating direction method of multipliers. The swCAM is implemented in open-source R scripts.

2 Quick Start

After download the files, please source the needed function.

```
source("sCAMfastNonNeg.R")
```

For the matrix for the sample-wise deconvolution ($\mathbf{X}$), please prepare the proportion matrix ($\mathbf{A}$) and source matrix (($\mathbf{S}$)) with our debCAM package (https://github.com/Lululuella/debCAM) or any other method.

Then, we can use function `sCAMfastNonNeg()` for sample-wise deconvolution.

```
lambda <- 5 # just an example

rsCAM <- sCAMfastNonNeg(X, A, S, lambda = lambda)
Swest <- rsCAM$S # sample-wise expression
```

`rsCAM$S` contains the results of sample-wise deconvolution for $\mathbf{X}$.

For the details to estimate $\mathbf{A}$ and $\mathbf{S}$ with $\mathbf{X}$, please refer to https://github.com/Lululuella/debCAM.

3 Sample-wise Deconvolution with Low-rank regularization

To start, please download all the files on https://github.com/Lululuella/swCAM first.

3.1 Simulation Data Generation

The simulation dataset is generated from a benchmark real gene expression dataset (GSE19380). The ideal simulation assumes an independent relationship between variance and mean for genes, and the realistic simulation entertains a variance-mean relationship for genes close to that observed in real gene expression data. In this manual, we use the realistic simulation for demonstration.

```
load("../.RData")
rm(list=ls())
library(gtools)
library(nnls)
library(MASS)
library(truncnorm)
library(ggplot2)
library(scales)
library(gplots)
mat <- as.matrix(read.table("../GSE19380/plier-mm.matrix.txt",header=T,
                            row.names=1))
```

The simulation involves three subtypes, twelve function modules (four unique functional modules in each subtype), 300 genes, and 50 samples. The baseline subtype-specific expression profiles are sampled from the real gene expression of the purified subtypes.

```
Sraw <- mat[,c(1:4, 1:4+8, 1:4+12)]
label <- rep(1:3, each=4)
Smean0 <- sapply(1:3, function(x) rowMeans(Sraw[,label == x]))
L <- n <- 300

set.seed(111)
sampleIdx <- sample(1:nrow(Sraw),n)
Smean <- Smean0[sampleIdx,]

K <- 3
M <- m <- 50 #sample size
r<- 4 # 4 modules in each of 3 types
q<-3
p<- c(0.06,0.06, 0.04,0.04)*n # number of genes involved in 4 modules
```

$\mathbf{A}$ is drawn randomly from a flat Dirichlet distribution, and the sparse between-sample variation matrix is from normal distribution. The variance of subtype expressions and variance of overall noise are set to be proportional to the means of subtype and mixed expressions, respectively. The coefficient of variation is drawn from the uniform distributions.

```
SnoiseR <- matrix(rnorm(r*q*m,0,1), r*q, m)
SnoiseR <- SnoiseR - rowMeans(SnoiseR) #center the noise signal
SnoiseL <- matrix(0, K*n, r*q)

ggco <- sample(1:n,q*sum(p)) #co-expressed gene index
A <- rdirichlet(m,c(1,1,1))
start <- 0
#1st cell type
for(i in 1:r){
    coeffi <- runif(p[i],0.15,0.3)
    sign <- rep(c(1,-1), p[i]/2)
    SnoiseL[ggco[start + 1:p[i]], i] <- sign*coeffi
    start <- start + p[i]
}
#2nd cell type
```

```
for(i in 1:r){
    coeffi <- runif(p[i],0.15,0.3)
    sign <- rep(c(1,-1), p[i]/2)
    SnoiseL[ggco[start + 1:p[i]] + n, i+r] <- sign*coeffi
    start <- start + p[i]
}
#3rd cell type
for(i in 1:r){
    coeffi <- runif(p[i],0.15,0.3)
    sign <- rep(c(1,-1), p[i]/2)
    SnoiseL[ggco[start + 1:p[i]]  + 2 * n, i+2*r] <- sign*coeffi
    start <- start + p[i]
}

SnoiseVec <-  SnoiseL %*% SnoiseR
Snoise <- t(matrix(c(SnoiseVec),nrow=n))
S.sample <- t(Smean)[rep(1:K,m),] * (1+ Snoise)

X<- c()
for(i in 1:m){
    X <- rbind(X, A[i,,drop=F]%*%S.sample[K*(i-1) + 1:K,])
}
dX <- X - A%*%t(Smean)[1:K,]

Xsd <- runif(m*n, 0.02, 0.05)
Xnoise <- matrix(rnorm(m*n,0,1), m, n)
Xn <- X*(1+Xnoise*Xsd) #end of data simulation
```

In this example, we assume $\mathbf{A}$ is known, so we can obtain $\mathbf{S}$ with non-negative least square.

```
Sest <- apply(Xn,2, function(x) coef(nnls(A,x)))
```

In the real application, if $\mathbf{A}$ is unknown, please refer to our debCAM package (https://github.com/Lululuella/debCAM) or other methods to estimate $\mathbf{A}$ first.

3.2 swCAM Workflow

3.2.1 Hyperparameter estimation

Before sample-wise deconvolution, first we need to decide the value of λ by k-fold cross-validation. Here we set k as 10.

```
Kfold <- 10

set.seed(111)


nall <- ncol(Xn) * nrow(Xn)
tmpseq <- rep(1:Kfold, nall %/% Kfold)
if (nall %% Kfold > 0) {
```

```
    tmpseq <- c(tmpseq, 1:(nall %% Kfold))
}
sample_exp <- matrix(sample(tmpseq), nrow(Xn))
```

The total computation time may be extremly long, so here we use parallel computing.

```
source('../sCAMfastNonNegNA.R')


Sest1 <- Sest
Aest1 <- A

library(doSNOW)
cl <- makeCluster(Kfold, type = "SOCK")
registerDoSNOW(cl)
```

The values in `lambdaAll` are only examples. The user can test some of them and decide to try more or not for faster speed. This step may take about 2 hours.

```
lambdaAll <- c(10000, 5000, 2000, 1500, 1000, 800, 500, 400, 300, 200, 150, 100,
               80, 50, 40, 30, 20, 10 , 8, 5 , 2, 1, 0.5, 0.2, 0.1, 0.05, 0.01,
               0.001, 0.0001, 0.00001)

ptm <- proc.time()
res <- foreach (ifold = 1:Kfold, .combine=rbind) %dopar% {
    warmstart <- matrix(0, L * K, M)
    Xtrain <- Xn
    Xtrain[sample_exp == ifold] <- NA
    errlambda <- c()

    for (lambda in lambdaAll) {
        rsCAMdtrain <- sCAMfastNonNegNA(Xtrain, Aest1, Sest1, iter = 1,  r = 1,
                                        lambda = lambda, iteradmm=10000,
                                        silent = T, eps = 1e-10,
                                        warm.start=warmstart)

        warmstart <- rsCAMdtrain$W[1:(L*K),]

        Xest <- c()
        for(i in 1:M){
            tmps1 <- Sest1 + t(matrix(rsCAMdtrain$W[1:(K*L),i],nrow = ncol(Xn)))
            tmps2 <- t(matrix(rsCAMdtrain$W[1:(K*L) + K*L,i], nrow = ncol(Xn)))
            tmps <- (tmps1 + tmps2) /2
            Xest <- rbind(Xest, Aest1[i,,drop=F]%*%tmps)
        }
        errlambda <- c(errlambda, sum((Xest[sample_exp == ifold] -
                                           Xn[sample_exp == ifold])^2),
                       rsCAMdtrain$epPri, rsCAMdtrain$epDual, rsCAMdtrain$iterrun)
    }
    errlambda
}
```

```
proc.time() - ptm


stopCluster(cl)
```

The RMSE obtained by 10-fold cross-validation is relatively small when $\lambda = 1$~50 and reaches the minimum at $\lambda = 5$.

```
rmse <- res[,seq(1,ncol(res),4)]
rmse <- sqrt(rmse/ (M  * L /Kfold))

self_fun <- function(x) {sqrt(sum(x^2 *(M  * L /Kfold) ) / (M*L))}

df.cv <- data.frame(x=factor(rep(lambdaAll, each=nrow(rmse))),y=c(rmse))
ggplot(subset(df.cv, x %in% lambdaAll),
       aes(x=x, y=y)) + geom_boxplot() + theme_bw(base_size = 16) +
       stat_summary(fun=self_fun, geom="line", aes(group=1),  colour="blue") +
       theme(axis.text.x=element_text(angle=90, hjust=1, vjust=0.5, size=10))+
    xlab("lambda") + ylab("RMSE")
```

3.2.2 Sample-wise Deconvolution

From the last section, we obtain $\lambda = 5$, so we can perform the sample-wise deconvolution on the whole $\mathbf{X}$ with function `sCAMfastNonNeg()`. The deconvolution result is in `rsCAMdtrain$S`.

```
Sest1 <- Sest
Aest1 <- A

source('../sCAMfastNonNeg.R')

rsCAMdtrain <- sCAMfastNonNeg(Xn, Aest1, Sest1, iter = 1,  r = 1, lambda = 5,
                              iteradmm=10000, silent = T, eps = 1e-10)
Swest <- rsCAMdtrain$S # sample-wise expression
```