Exact Procedure: With Z stats

Sarah Urbut

September 28, 2015

First, I load the 16,069 x 44 matrix of maximum z statistics statistics into memory. I had previously computed the SFA decomposition on this matrix using 5 factors on the **PPS Cluster**. The steps that follow were run on the **Midway Cluster**.

```
z.stat <- read.table("maxz.txt")
lambda.mat=as.matrix(read.table("zsfa_lambda.out"))
factor.mat=as.matrix(read.table("zsfa_F.out"))
```

1 Deconvolution Step

For a given ω_l, we specify 4 'types' of RxR prior covariance matrices $U_{k,l}$.

1. $U_{k=1,l} = \omega_l \mathbf{I}_R$
2. $U_{k=2,l} = \omega_l \mathrm{X}_z$ The (naively) estimated tissue covariance matrix as estimated from the column-centered J $\times R$ matrix of Z statistics, Z_{center}: $\frac{1}{J}\ {Z_{center}}^t\ Z_{center}$
3. $U_{k=3,l} = \omega_l\ \frac{1}{J}\ V_{1\ldots p}\ d^2_{1\ldots p}\ V^t_{1..p}$ is the rank p eigenvector approximation of the tissue covariance matrices, i.e., the sum of the first p eigenvector approximations, where $_{1\ldots p}$ represent the eigenvectors of the covariance matrix of tissues and $_{1\ldots p}$ are the first p eigenvalues.
4. $U_{k=4:4+Q,l} = \frac{1}{J}((\Lambda\mathbf{F})^t\Lambda\mathbf{F})_q$ corresponding to the q_{th} sparse factor representation of the tissue covariance matrix
5. $U_{k=4+Q+1,l} = \frac{1}{J}\ (\Lambda\mathbf{F})^t\Lambda\mathbf{F}$ is the sparse factor representation of the tissue covariance matrix, estimated using all q factors.

To retrieve a 'denoised' or 'deconvoluted' estimate of the non-single rank dimensional reduction matrices, I then perform deconvolution.em which initializes the EM algorithm with the matrices specified in (2), (3) and (5). The final results of this iterative procedure preserves the rank of the initialization matrix, and allows us to use the 'true' effect component as missing data in deconvoluting the prior covariance matrices.

In brief, this is how the deconvolution.em algorithm works.

1. Produce a 2 element list of initialization parameters containing the initial covariance matrices and a vector of their initial weights, π. Critically, this vector π will need to be recomputed when we add these deconvoluted estimates to the full set of covariance matrices.

2. Return a list with the denoised covariance matrix and corresponding mixture weights.

From extensive investigation with testing and training data, I found that using a rank 3 SVD approximation for the matrix in (5) as well as the rank 5 SFA approximation and the empirical covariance matrix maximized the likelihood of a test data set.

After loading the correct package (i.e., $library('ExtremeDeconvolution')$)

```
max.step=deconvolution.em.with.bovy(t.stat=t.stat,factor.mat=factor.mat,lambda.mat=lambda.mat,K=3,P=3
```

2 Generation of List of Covariance Matrices

I then use these three non single-rank covariance matrix in place of our original choice of the empirical covariance matrix, SFA and SVD approximations to create a KxL list of covariance matrices. The function **compute.hm.covmat** chooses an 'L' element grid according to the range of effect sizes present in the initial 16,069 x 44 matrix of strong Z statistics. Here, I also used the Identity (K=1), 5 single-rank SFA factors (K=4-9), and the 44+1 eqtlbma.lite configurations (K=10:54). This is 54 matrices, and in this data set, **autoselect.mixgrid** chose a grid with 22 omegas for a total of 1188 covariance matrices.

```
covmat=compute.hm.covmat.all.max.step(z.stat,v.j,Q=5,lambda.mat,
A="filename",factor.mat,max.step=max.step)
```

covmat is thus a list of 1188 covariance matrices.

3 Mixture Weights

We now need to compute the mixture weights hierarchically. I use a randomly chosen set of 20000 gene snp pairs from the matrix QTL output to estimate these mixture proportions. This set does not contain the strongest gene-snp pairs, and thus will allow for substantial shrinkage, as a majority of these gene-snp pairs will have their likelihood maximized at low ω components.

Here is a visualization of the training data, which you can see is a mixture of mostly small $\hat{\beta}$ and accordingly small Z statistics.

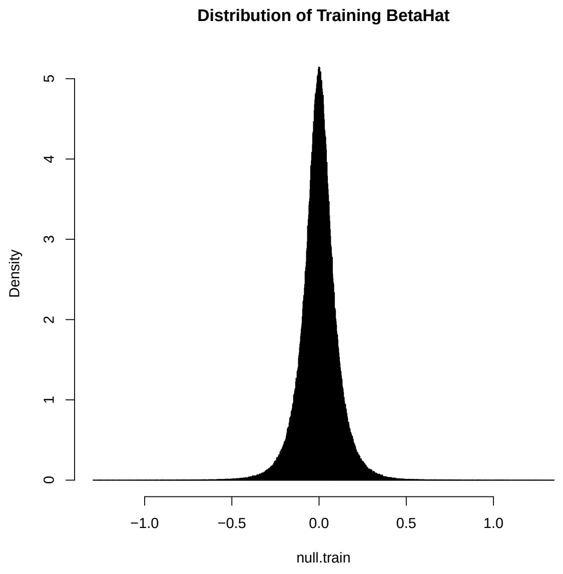
```
compute.hm.train(train.b = train.z,se.train = train.s,covmat = covmat,A="jul3")
```

compute.hm.train produces an rds object with prior weights, a likelihood matrix, and a pdf of the barplot of these weights.

4 Posterior Quantities

Now that I have the estimated mixture weights stored in the vector **pis**. I proceed to the inference step, where I compute the posterior weights and corresponding posterior quantities across all original 16069 gene snp pairs. In brief, the posterior mean, post covariance matrix and tissue specific tail probabilities are computed across all K components for each gene snp pair, and then weighted according to the posterior weights. This is performed in the **weightedquants** step.

Figure 1: Distribution of MLEs



```
weightedquants=lapply(seq(1:nrow(z.stat)),function(j){total.quant.per.snp(j,covmat,b.gp.hat=z.stat,
se.gp.hat = s.j,pis,A,checkpoint = FALSE)})
```

This will return a set of 6 files containing the JxR matrix of posterior means, marginal variances, tail probabilities, local false sign rates, and the JxK matrix of posterior weights. Checkpoint = FALSE means that the files will be created (rather than simply outputting an object array which contains the posterior quantities.

5 Testing and Training

In order to determine the optimal number and rank of the covariance matrices, we divide our data set into a training and test data set, each containing 8000 genes.

In the training set, we proceed as above: choosing the top SNP for each of the 8000 genes, creating a list of covariance matrices through deconvolution and grid selection of these top 'training gene-snp' pairs.

Then, within the training data, we similarly choose a random set of gene-snp pairs (restricting our analysis to genes contained in the training set. Again, we choose 20,000 random-gene snp pairs and use the EM algorithm to learn the mixture proportions π from this data set.

We then use the KxL vector of π from the training set to estimate the log likelihood of each data point in the test data set. If our model is 'overfit' to the training data set, than a larger number of covariance matrices may actually decrease the test log-likelihood.

I found that the K=1188 set of covariance matrices containing the Identity, the denoised empirical covariance matrix, rank 5 SFA approximation and rank 10 SVD approximation as well as 5 single-rank SFA factors and the 45 eqtl.bma.lite configurations maximized this likelihood.

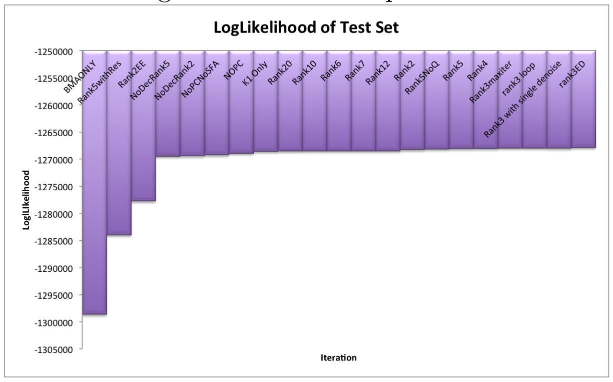
```
mdat <- GetSS("max", "/project/mstephens/gtex/analysis/april2015/query/MatrixEQTLSumStats.h5")

ndat <- GetSS("null", "/project/mstephens/gtex/analysis/april2015/query/MatrixEQTLSumStats.h5")

N1 <- 8000
N2 <- 16069
strong.train <- SubsetMatLists(mdat, seq(1, N1))
strong.test <- SubsetMatLists(mdat, seq(N1 + 1, N2))

strong.train.genes <- as.character(lapply(strsplit(rownames(strong.train$beta), "_"), function(x) x[1

strong.test.genes <- as.character(lapply(strsplit(rownames(strong.test$beta), "_"), function(x) x[1])

null.genes <- as.character(lapply(strsplit(rownames(ndat$beta), "_"), function(x) x[1]))


null.train <- SubsetMatLists(ndat, which(null.genes %in% strong.train.genes))$z[1:20000,]

null.test <- SubsetMatLists(ndat, which(null.genes %in% strong.test.genes))$z[1:20000,]

big.train=strong.train$z
big.test=strong.test$z


max.stepk1=readRDS("maxstep1_train.rds")
max.steprank5=readRDS("maxsteprank5pcfastjul20_train.rds")
max.steprank2=readRDS("maxsteprank2pcfastjul20_train.rds")
#max.steprank10=readRDS("maxsteprank10pc_trainjul20.rds")

max.steprank10=readRDS("trainmaxsteprank10pc.rds")

factor.mat=as.matrix(read.table("../jul3/zsfa_F.out"))
lambda.mat=as.matrix(read.table("../jul3/zsfa_lambda.out"))


z.stat=big.train
rownames(z.stat)=NULL
colnames(z.stat)=NULL
v.j=matrix(rep(1,ncol(z.stat)*nrow(z.stat)),ncol=ncol(z.stat),nrow=nrow(z.stat))



train.z=null.train[1:20000,]
rownames(train.z)=NULL
colnames(train.z)=NULL

train.v=matrix(rep(1,ncol(train.z)*nrow(train.z)),ncol=ncol(train.z),nrow=nrow(train.z))

test.z=null.test
```

```
rownames(test.z)=NULL
colnames(test.z)=NULL

##withmaxstepk1"
##compute covarinace matrices on strongest training gene-snp pairs

covmat=compute.hm.covmat(z.stat,v.j,Q=5,lambda.mat,P=2,A="jul21k1",factor.mat,max.step=max.stepk1)

##compute weights on random training set

tim=proc.time()
compute.hm.train(train.b = train.z,se.train = train.v,covmat = covmat,A="jul21k1") ##compute the HM w
proc.time()-tim

A="jul21k1"
pis=readRDS(paste0("pis",A,".rds"))$pihat

##use training weights to compute test set likelihood
compute.lik.test(test.z,J=nrow(null.test),train.v,covmat,A="jul21k1",pis)
rm(covmat)
rm(pis)
```

Here I show the results:

```
Run Likelihood
BMAONLY -1298672
Rank5withRes -1283993
Rank2EE -1277777
NoDecRank5 -1269504
NoDecRank2 -1269442
NoPCNoSFA -1269277
NOPC -1268966
K1 Only -1268622
Rank20 -1268552
Rank10 -1268525
Rank6 -1268520
Rank7 -1268519
Rank12 -1268516
Rank2 -1268322
Rank5NoQ -1268191
Rank5 -1268102
Rank4 -1268065.997
Rank3maxiter -1267998.4
rank3 loop  -1267998.484
Rank3 with single denoise -1268049.69
rank3ED -1267997.5
```

Figure 2: Model Comparisons



6 Simulation

I have also simulated, for each gene snp pair, 100 draws from the multivariate normal distribution characterized by the posterior mean and covariance produced at a component chosen according to its posterior weight (responsibility). For each gene-snp pair, I count the number of simulations in which at least two signs differed.

```
sim.array.generation = function(j,b.j.hat,se.j.hat,covmat,pis,sim){
K=length(covmat)
b.mle=as.vector(t(b.j.hat[j,]))##turn i into a R x 1 vector
V.j.hat=diag(se.j.hat[j,]^2)
lik=lik.func(b.mle,V.j.hat,covmat)
post.weights=lik*pis/sum(lik*pis)
component=apply(rmultinom(sim,1,prob = post.weights),2,function(x){which(x==1)})##choose a component
tinv=lapply(seq(1:sim),function(sim){k=component[sim];solve(covmat[[k]]+V.j.hat)})##generate a list o
b.j.=lapply(seq(1:sim),function(sim){ k=component[sim];post.b.jk.ed.mean(b.mle,tinv=tinv[[sim]],covma
B.j.=lapply(seq(1:sim),function(sim){ k=component[sim];post.b.jk.ed.cov(tinv=tinv[[sim]],covmat[[k]])
simulations=sapply(seq(1:sim),function(x){
  dat=mvrnorm(1,mu = b.j.[[x]],Sigma = B.j.[[x]])##for each simulation, generate a multivariate norma
   pos=sum(dat>0);neg=sum(dat<0);pos*neg!=0})##for each simulation, ask if they are all one direction

return(sum(simulations)/length(simulations)
)}
```

7 Testing with Residuals

Here, I use the covariance matrix of the residuals from the beta$_{h}ats, normalizedsuchthatthediagonalis1(becausethevarianc$

```
se.mat.train=cor(read.table("null.train.se.txt"))
se.mat.test=cor(read.table("null.test.se.txt"))



A="rank5withres"
```

```
covmat=compute.hm.covmat.all.max.step(z.stat,v.j,Q=5,lambda.mat,A,factor.mat,max.step=max.steprank5)

tim=proc.time()
compute.hm.train.semat(train.z,se.mat.train,covmat,A)
proc.time()-tim


A="rank5withres"
pis=readRDS(paste0("pis",A,".rds"))$pihat

compute.lik.test.semat(test.z,J=nrow(null.test),se.mat.test,covmat,A,pis)

rm(covmat)
rm(pis)
```