HEAT Example Application

2024-03-15

In this document, we demonstrate how to use the provided software to implement HEAT. Note that this version of our software only allows for $J = 2$: a generalization of this software is forthcoming.

Let us begin by generating some data. Following the manuscript, there are two populations labeled AA and EA. Below, we generate data exactly as in the simulation studies, except the predictors are generated from a normal distribution.

```r
set.seed(5)
p <- 200
n <- 500
SigmaEA <- matrix(0, nrow=p, ncol=p)
SigmaAA <- matrix(0, nrow=p, ncol=p)
for (jj in 1:p) {
  for (kk in 1:p) {
    SigmaEA[jj,kk] <- 0.9^abs(jj-kk)
    SigmaAA[jj,kk] <- 0.5^abs(jj-kk)
  }
}
eoEA <- eigen(SigmaEA)
SigmaEAsqrt <- eoEA$vec%*%diag(eoEA$val^0.5)%*%t(eoEA$vec)
eoAA <- eigen(SigmaAA)
SigmaAAsqrt <- eoAA$vec%*%diag(eoAA$val^0.5)%*%t(eoAA$vec)
SNP.AA <- matrix(rnorm(n*p), nrow=n, ncol=p)%*%SigmaAAsqrt
SNP.EA <- matrix(rnorm(n*p), nrow=n, ncol=p)%*%SigmaEAsqrt

R2 <- 0.5
Prop_pQTLsShared <- 0.95
sigmaDiff <- 3
s <- 10
aa.sd <- apply(SNP.AA, 2, sd)
ea.sd <- apply(SNP.EA, 2, sd)
beta.AA <- rep(0, p)
preds <- sample(1:p, s)
beta.AA[preds] <- aa.sd[preds]
beta.EA <- rep(0, p)
preds2 <- sample(preds, s*Prop_pQTLsShared)
beta.EA[preds2] <- ea.sd[preds2]
if(Prop_pQTLsShared < 1){
  preds3 <- sample(c(1:p)[-preds2], s - s*Prop_pQTLsShared)
  beta.EA[preds3] <- ea.sd[preds3]
}
beta.EA[which(ea.sd == 0)] <- 0
sign.vec.AA <- sample(c(1, -1), p, replace=TRUE, prob = c(0.8, 0.2))
sign.vec.EA <- sample(c(1, -1), p, replace=TRUE, prob = c(0.8, 0.2))
beta.AA <- beta.AA*sign.vec.AA
beta.EA <- beta.EA*sign.vec.EA
```

```
sigma <- sd(SNP.EA%*%beta.EA)
b0.AA <- -mean(SNP.AA%*%beta.AA)
prot.AA <- b0.AA + SNP.AA%*%beta.AA + rnorm(n, sd = sigma*sigmaDiff)
b0.EA <- -mean(SNP.EA%*%beta.EA)
prot.EA <- b0.EA + SNP.EA%*%beta.EA + rnorm(n, sd = sigma)
```

In order to run the HEAT algorithm we need to source the R functions.

```
source("~/HEATsims/Functions/penMLE_cvL.R")
source("~/HEATsims/Functions/penMLE_fixedRhos_cvL.R")
```

Exact version of HEAT

Now, we run the full version of HEAT using the function `penMLE_CV` based on five-fold cross-validation. This will take a few minutes, so please exercise patience.

```
# define folds for reproducibility
nfolds <- 5
fold.id.AA <- sample(rep(nfolds:1, length=length(prot.AA)))
fold.id.EA <- sample(rep(nfolds:1, length=length(prot.EA)))

# fit the model and perform cross-validation
t0 <- penMLE_CV(prot.AA = prot.AA, # protein expression for AA
                  prot.EA = prot.EA, # protein expression for EA
                  X.AA = SNP.AA, # design matrix for AA
                  X.EA = SNP.EA, # design matrix for EA
                  nlambda = 10, # number of candidate lambda
                  delta = 0.05, # ratio of largest to smallest lambda
                  ngamma = 10, # number of candidate gamma
                  nfolds = nfolds,
                  fold.id.AA = fold.id.AA,
                  fold.id.EA = fold.id.EA)
```

```
## Through fold  1
## Through fold  2
## Through fold  3
## Through fold  4
## Through fold  5
```

Now, we can look at the output.

```
str(t0)
```

```
## List of 9
##  $ errs.AA   : num [1:10, 1:10, 1:5] 12971 12971 12971 12971 12971 ...
##  $ errs.EA   : num [1:10, 1:10, 1:5] 2880 2897 2897 2877 2897 ...
##  $ L.errs.AA : num [1:10, 1:10, 1:5] 587 587 587 587 587 ...
##  $ L.errs.EA : num [1:10, 1:10, 1:5] 436 437 437 436 437 ...
##  $ full.fit  :List of 10
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:10] 0.087 0.087 0.087 0.087 0.087 ...
##   .. ..$ rhoE: num [1:10] 0.183 0.184 0.185 0.185 0.185 ...
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:10] 0.087 0.0872 0.0876 0.0878 0.0878 ...
##   .. ..$ rhoE: num [1:10] 0.183 0.19 0.195 0.197 0.198 ...
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:10] 0.087 0.0873 0.0884 0.0889 0.089 ...
##   .. ..$ rhoE: num [1:10] 0.183 0.196 0.209 0.214 0.215 ...
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:10] 0.087 0.0878 0.0893 0.09 0.0902 ...
##   .. ..$ rhoE: num [1:10] 0.183 0.207 0.224 0.229 0.23 ...
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:10] 0.087 0.0875 0.0897 0.0911 0.0916 ...
##   .. ..$ rhoE: num [1:10] 0.183 0.208 0.232 0.24 0.243 ...
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:10] 0.087 0.0877 0.0906 0.0927 0.0936 ...
##   .. ..$ rhoE: num [1:10] 0.183 0.216 0.241 0.25 0.253 ...
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:9] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:9] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:9] 0.087 0.0879 0.0917 0.0949 0.0962 ...
##   .. ..$ rhoE: num [1:9] 0.183 0.222 0.249 0.259 0.262 ...
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:8] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:8] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:8] 0.087 0.088 0.093 0.0974 0.0992 ...
##   .. ..$ rhoE: num [1:8] 0.183 0.226 0.254 0.266 0.27 ...
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:8] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:8] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:8] 0.087 0.0881 0.0944 0.0999 0.1023 ...
##   .. ..$ rhoE: num [1:8] 0.183 0.229 0.259 0.272 0.278 ...
##   ..$ :List of 4
##   .. ..$ phiA: num [1:200, 1:8] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ phiE: num [1:200, 1:8] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ rhoA: num [1:8] 0.087 0.0882 0.0956 0.1023 0.1055 ...
##   .. ..$ rhoE: num [1:8] 0.183 0.232 0.262 0.278 0.286 ...
##  $ fold.id.AA: int [1:500] 1 2 4 1 3 5 2 1 1 3 ...
##  $ fold.id.EA: int [1:500] 5 3 5 4 4 2 5 5 5 2 ...
##  $ lambda    : num [1:10] 0.373 0.27 0.195 0.141 0.102 ...
##  $ gamma.mat : num [1:10, 1:10] 0.042292 0.011768 0.003275 0.000911 0.000254 ...
```

In the above output `phiA` and `phiE` are the estimated $\theta_{(\mathrm{AA})}$ and $\theta_{(\mathrm{EA})}$, respectively at different tuning parameter pairs. However, note that these are on scale of the standardized predictors: see, e.g., lines 101 and 103 in "penMLE_cvL.R".

Since we just performed cross-validation, we can now determine which tuning parameter pair minimized the cross-validation error. What metric to use for cross-validation is context dependent. Here, we look at the total sum of squared residuals over both populations.

```
errs <- apply(t0$errs.AA + t0$errs.EA, c(1,2), sum)
```

Now, let us extract the estimated $\beta_{(\mathrm{AA})}$ and $\beta_{(\mathrm{EA})}$. To do so, we get coefficients back to the original scale.

```
inds <- which(errs == min(errs), arr.ind=TRUE)
getInd <- inds[1,2]
full.fit <- t0$full.fit[[inds[1,1]]]
full.sd <- apply(rbind(SNP.AA, SNP.EA), 2, sd)

hat.beta.AA <- (full.fit$phiA[,getInd]/full.sd)/full.fit$rhoA[getInd]
hat.beta.EA <- (full.fit$phiE[,getInd]/full.sd)/full.fit$rhoE[getInd]
```

Next, we can check the estimation accuracy.

```
sum((hat.beta.AA - beta.AA)^2)
```

```
## [1] 5.512513
```

```
sum((hat.beta.EA - beta.EA)^2)
```

```
## [1] 4.222049
```

Now, let's compare this to using the lasso estimator for each population on its own, then both populations aggregated.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
fit.AA <- cv.glmnet(y = prot.AA, x = SNP.AA)
hat.beta.AA.lasso <- coef(fit.AA, s = fit.AA$lambda.min)[-1]
fit.EA <- cv.glmnet(y = prot.EA, x = SNP.EA)
hat.beta.EA.lasso <- coef(fit.EA, s = fit.EA$lambda.min)[-1]
sum((hat.beta.AA.lasso - beta.AA)^2)
```

```
## [1] 7.414414
```

```
sum((hat.beta.EA.lasso - beta.EA)^2)
```

```
## [1] 6.104603
```

```
fit.full <- cv.glmnet(y = rbind(prot.EA, prot.AA), x = rbind(cbind(1, SNP.EA), cbind(0, SNP.AA)),  pena
hat.beta.lasso <- coef(fit.full, s = fit.full$lambda.min)[-c(1,2)]
sum((hat.beta.lasso - beta.AA)^2)
```

```
## [1] 8.144751
```

```
sum((hat.beta.lasso - beta.EA)^2)
```

```
## [1] 7.133988
```

We could alternatively use the validation likelihood error. This is denoted by `L.errs.AA` and `L.errs.EA` in the software.

```
errs <- apply(t0$L.errs.AA + t0$L.errs.EA, c(1,2), sum)
inds <- which(errs == min(errs), arr.ind=TRUE)
getInd <- inds[1,2]
full.fit <- t0$full.fit[[inds[1,1]]]
hat.beta.AA <- (full.fit$phiA[,getInd]/full.sd)/full.fit$rhoA[getInd]
```

```r
hat.beta.EA <- (full.fit$phiE[,getInd]/full.sd)/full.fit$rhoE[getInd]
sum((hat.beta.AA - beta.AA)^2)
```

```
## [1] 6.768338
```

```r
sum((hat.beta.EA - beta.EA)^2)
```

```
## [1] 4.247376
```

One could also use error isolated to each population (AA or EA), though we do not recommend this. In our experience, this can lead to strange behavior.

Approximate version of HEAT

In this section, we show how to implement the approximation version of HEAT. This will require using the `natural` package in R.

```r
library(natural)
sigma.AA.est <- nlasso_cv(x = SNP.AA, y = prot.AA, nfold=5)$sig_obj
sigma.EA.est <- nlasso_cv(x = SNP.EA, y = prot.EA, nfold=5)$sig_obj

t1 <- penMLE_fixedrhos_CV(prot.AA = prot.AA,
                              prot.EA = prot.EA,
                              X.AA = SNP.AA,
                              X.EA = SNP.EA,
                              nlambda = 10,
                              delta = 0.05,
                              ngamma = 10,
                              nfolds = nfolds,
                              fold.id.AA = fold.id.AA,
                              fold.id.EA = fold.id.EA,
                              rhoA = 1/sigma.AA.est,
                              rhoE = 1/sigma.EA.est)
```

```
## Through fold  1
## Through fold  2
## Through fold  3
## Through fold  4
## Through fold  5
```

```r
errs <- apply(t1$errs.AA + t1$errs.EA, c(1,2), sum)
getInd <- inds[1,2]
full.fit <- t1$full.fit[[inds[1,1]]]
full.sd <- apply(rbind(SNP.AA, SNP.EA), 2, sd)
hat.beta.AA.app <- (full.fit$phiA[,getInd]/full.sd)*sigma.AA.est
hat.beta.EA.app <- (full.fit$phiE[,getInd]/full.sd)*sigma.EA.est
sum((hat.beta.AA.app - beta.AA)^2)
```

```
## [1] 5.424754
```

```r
sum((hat.beta.EA.app - beta.EA)^2)
```

```
## [1] 4.289603
```

We see this performs similar to the full version while requiring less computing time.