R Notebook

Code

Part 0: Data Read-in

Hide

```
rawgenos <- read.table(file="Consolidated Genotypes.csv", header = TRUE, sep=",", row
.names = 1)
snpinfo <- read.table(file="SNPinfo.csv", header = TRUE, sep=",")
pheno <- read.table(file="pheno.csv", header = TRUE, sep=",",row.names = 1)
```

Part 1: mini-GWAS and Basic Bonferroni Correction

Hide

```
######Package install######
#Answer y when asked whether to install from a source that needs compilation
install.packages("SNPassoc")
library(SNPassoc)

######format data for SNPassoc (similar to PLINK format)######
geno_pheno<-cbind(pheno,rawgenos)
org_geno<-setupSNP(geno_pheno, 4:ncol(geno_pheno), sort = TRUE, snpinfo, sep = "")

#Check that setup worked (wouldn't recommend doing this w/o row specification)
summary(org_geno[,4:8])
plot(org_geno,which=13)

######Run the mini-GWAS######
suborg_geno<-org_geno[,1:1000]

start_time <- Sys.time()
#This next line of code is all you really need, start and end times are just for refe
rence.
#You purposely leave the SNP out of the equation as part of how this function works.
miniGWAS<-WGassociation(birthweight_kg~1, suborg_geno, model = "codominant", genotypi
ngRate = 80)
end_time <- Sys.time()
end_time - start_time

Bonferroni.sig(miniGWAS, model = "codominant", alpha = 0.05)
plot(miniGWAS)

#BONUS: If you wanted to include covariates, you would simply replace the 1 in the mi
nigwas
#with the name of the covariate. If you want to try you can add the covariate 'trauma
' to
#the model, which I've included in column 3 of suborg_geno if you examine it.
```

Part 2: Testing Using Simple M

Hide

```
#Useful citations for this:

#Gao X, Starmer J, and Martin ER. (2008). A multiple testing correction method for ge
netic
#association studies using correlated single nucleotide polymorphisms. Genetic Epidem
iology,
#32, 361-369.

#Johnson RC, Nelson GW, Troyer JL, Lautenberger JA, Kessing BD, Winkler CA, and O'Bri
en SJ.
#(2010). Accounting for multiple comparisons in a genome-wide association study (GWAS
).
#BMC Genomics, 11, 724-724.

######Convert the genotypes to numeric values######
mgenos<-rawgenos
for(n in 1:ncol(mgenos)){
  mgenos[,n]<-as.numeric(mgenos[,n])-1
}


######Compute the Composite Linkage Disequilibrium Score######

#The commented out line will likely not work with a larger dataset due to lack of RAM
.
#However, I am including it for illustration purposes as its the first thing many thi
nk of
#and even the papers above recommend using this function, but don't adress this diffi
culty.

#compld<-cor(mgenos)

#In light of this, you usually need to break the data up into ~5,000 SNP chunks.
#Ideally, you would do this based on haplotype blocks with haploview, but since our
#data is small, we'll just separate it by chromosome. We'll do chromosome 10 as an ex
ample.

n<-10
chromsnps<-snpinfo$Chromosome == n
#If you want to know how many SNPs that is use sum(chromsnps ==TRUE)
compld<-cor(mgenos[,chromsnps], use= "complete.obs")

#Always a good idea to inspect the matrix
compld[1:10,1:10]

#There are snps without enough variance so they'll need to be removed. The fact they
have
#so little variance means they couldn't be tested anyway meaning they don't contribut
e to
#the multiple testing burden.
```

```
newcompld<-compld[is.na(compld[,1]) == FALSE, is.na(compld[,1]) == FALSE]

#Calculate the Eigenvalues
eigns<-eigen(newcompld, only.values = TRUE)

#We now need to add the eigenvalues together until we reach 99.5% of the variance and
count
#how many that takes. The total variance in this case is the sum of the number of var
iables.
#Keep in mind this number is going to be very low because we have so few samples. Wit
h a
#normal genomics study you would want many more and you'd need to worry about things
like
#population structure, which we ignored in our miniGWAS, but are likely reflected her
e.
thresholdvar<-ncol(newcompld)*.995
eigentotal<-0
counter<-0
for (v in 1:length(eigns$values)){
  if(eigentotal<thresholdvar){
    counter<-counter + 1
    eigentotal<-eigentotal + eigns$values[v]
  }
}

print(counter)

#This would then be repeated for each chromosome and you would add the values togethe
r
#to get the total number of tests to apply Bonferroni correction. Try it with chromos
ome 1
#and be sure to inspect the corrlation matrix before removing bad SNPs as there is a
trap!
```

Part 3: Basic FDR Correction

Hide

```
#Extract the p-values from our earlier mini-GWAS
pvaladd<-codominant(miniGWAS)

#Check if FDR is appropriate
hist(pvaladd, breaks = 20)

#Since the histogram is relatively uniform, by strictest standards we shouldn't be us
ing
#FDR here, but well keep going for demonstration purposes

qvals<-p.adjust(pvaladd, "fdr")

#Since we only did 1000 SNPs, this is small enough to look at directly.
hist(qvals, xlim = 0:1)

#Clearly there is nothing significant in this case, which is to be expected based on
the
#histogram and sample size. Just to confirm:
sum(qvals < 0.05, na.rm = TRUE)
```

Part 4: Empirical P-values using Reshuffling (Permutation Testing)

Hide

```
#We're going to use a method called max(T) permutation where we take the lowest p-val
ue from
#testing each reshuffle against the simulated phenotype. Not necessarily the most eff
icient,
#but certainly valid under the majority of circumstances.

######Simulate p-values and generate empirical ones######
n_sims<-40
simoutputGWAS<-NULL

#system timer so we can see how long it takes
start_time_sim <- Sys.time()

for (n in 1:n_sims){
  #Create a new object to simulate the dataset (might need to modify original if larg
er)
  sim_suborg_geno<-suborg_geno

  #Shuffle the phenotype row order using the sample function
  newrowindex<-sample(nrow(sim_suborg_geno), nrow(sim_suborg_geno))

  #Use the new row order to assign phenotypes
  sim_suborg_geno[,1]<-as.data.frame(as.matrix(sim_suborg_geno[newrowindex,1]))

  #Run the mini-GWAS
  sim_miniGWAS<-WGassociation(birthweight_kg~1, sim_suborg_geno, model = "codominant"
, genotypingRate = 80)

  #Extract the pvalues
  sim_pvals<-codominant(sim_miniGWAS)

  #Take the smallest p-value and save it
  simoutputGWAS[n]<-min(sim_pvals, na.rm = TRUE)

}

#Generate empirical p-values by comparing to the simulated ones
permutedpvals<-NULL
for (c in 1:length(pvaladd)){
  n_nonsig<-sum(pvaladd[c]>=simoutputGWAS, na.rm = TRUE)
  permutedpvals[c]<-n_nonsig/n_sims
}


#Check total time it took
end_time_sim <- Sys.time()
end_time_sim - start_time_sim

######See how many SNPs are 'significant'######
```

```
#See how many SNPs are significant
sum(permutedpvals < 0.05, na.rm = TRUE)

#Get SNP IDs and info
sig_snps<-which(permutedpvals < 0.05)

sig_snpIDs<-colnames(suborg_geno)[sig_snps]

snpinfo[snpinfo$dbSNP.RS.ID %in% sig_snpIDs,]
#NOTE: %in% compares something to a vector like == compares to a value

#Permutation gets much more complicated with covariates and you need a lot more plann
ing.
#If interested in this, look up the 'BiasedUrn' package and literature related to it.
```

Part 5: Basic Parallel Computing

Hide

```
#Detecting the number of cores and setting them up for use
library(doParallel)
library(foreach)
ncore<-detectCores()
cl<-makeCluster(as.numeric(ncore))
registerDoParallel(cl)

######Define the permutation reshuffles we did as its own function######

permshuffle<-function(genodata, n_sims) {

  simoutputGWAS<-NULL

  for (n in 1:n_sims){

    #Create a new object to simulate the dataset (might need to modify original if la
rger)
    sim_suborg_geno<-genodata

    #Shuffle the phenotype row order using the sample function
    newrowindex<-sample(nrow(sim_suborg_geno), nrow(sim_suborg_geno))

    #Use the new row order to assign phenotypes
    sim_suborg_geno[,1]<-as.data.frame(as.matrix(sim_suborg_geno[newrowindex,1]))

    #Run the mini-GWAS
    sim_miniGWAS<-WGassociation(birthweight_kg~1, sim_suborg_geno, model = "codominan
t", genotypingRate = 80)

    #Extract the pvalues
    sim_pvals<-codominant(sim_miniGWAS)

    #Take the smallest p-value and save it
    simoutputGWAS[n]<-min(sim_pvals, na.rm = TRUE)

  }
  simoutputGWAS

}


######Run the function across multiple cores######

#Start time
start_time_parasim <- Sys.time()

#Split the simulations among each core and combine the results together
#(I changed n_sims because the times 4 means it will end up being 40 and
#that splits it evenly across 4 cores).
parallelsimoutput<-foreach(times(4), .combine = "c", .packages="SNPassoc") %dopar% pe
```

```
rmshuffle(suborg_geno, n_sims=10)

#Generate empirical p-values by comparing to the simulated ones
parallelpermutedpvals<-NULL
for (c in 1:length(pvaladd)){
  n_nonsig<-sum(pvaladd[c]>=parallelsimoutput, na.rm = TRUE)
  parallelpermutedpvals[c]<-n_nonsig/40
}

#Check total time it took
end_time_parasim <- Sys.time()
end_time_parasim - start_time_parasim


######See how many SNPs are 'significant'######

#See how many SNPs are significant
sum(parallelpermutedpvals < 0.05, na.rm = TRUE)

#Get SNP IDs and info
sig_snps<-which(parallelpermutedpvals < 0.05)

sig_snpIDs<-colnames(suborg_geno)[sig_snps]

snpinfo[snpinfo$dbSNP.RS.ID %in% sig_snpIDs,]


#This version of the code is designed mainly for someone with 4 cores, but depending
on
#how many your setup may have available, other configurations might be better. Feel f
ree
#to place with the times() and n_sims options in the foreach line to see how it affec
ts
#computing time.
```