BTRY4830_Lab9

Olivia Lang

3/28/2019

0. Review of Recent Labs

Lab6

-Reading in genetic data

-Converting genetic data into Xa and Xd codings

-Filtering the data by minor allele frequency

Lab 7

-Completing a regression on each variant (F-Statistic)

-Make a Manhattan Plot

Lab 8

-Check model with QQ plot

-Multiple test corrections (Bonferroni)

-Principal Components Analysis

Today (Lab 9)

1. Linear regression using lm()
2. Mini eQTL Analysis
3. Covariates
4. Exercise–PC1 as a covariate

1. Linear regression using lm()

Now that we have coded up linear regressions from scratch multiple times, we can move on to simpler ways. The most common way to do linear regressions in R is through the function **lm()**, which stands for linear model.

Let's use the mini Hapmap dataset that can be found in the zip file downloaded from CMS. The './HapMap_phenotypes.tsv' file has records of 10 phenotypes for 107 individuals where each row records the phenotypes of an individual and each column records the values for a phenotype. The './HapMap_genotypes.tsv' file is formatted such that each row records the genotypes of an individual and each column records the genotypes for the individuals at a given polymorphic site. Note that the genotype file is already in the X_a encodings so you just need to get the X_d encodings.

```
phenotypes <- read.table("./HapMap_phenotypes.tsv", header = T)
genotypes <- read.table("./HapMap_genotypes.tsv", header = T)
Xa.all <- as.matrix(genotypes)
Xd.all <- 1 - 2 * abs(Xa.all)
```

There are multiple ways to call **lm()** with a specific model and we will look at two ways of doing that.

(1) The first way is to generate a dataframe and using the structure of the dataframe to specify the model. In our case, we are interested in regressing the phenotypes against the dummy variables X_a and X_d and the expression inside lm is going to be something like this:

$$lm(Y \ Xa + Xd) \tag{1}$$

This is saying that Y should be the dependent variable, and X_a and X_d should be the indepenedent variables. So to call this for a *single genotype* and *single phenotype* we would call **lm()** like this:

```
# Regression on the first genotype and first phenotypein the
# hapmap data
regression.df <- data.frame(Y = phenotypes[, 1], Xa = Xa.all[,
    1], Xd = Xd.all[, 1])
lm(Y ~ Xa + Xd, data = regression.df)
```

```
##
## Call:
## lm(formula = Y ~ Xa + Xd, data = regression.df)
##
## Coefficients:
## (Intercept)           Xa           Xd
##     6.94741     -0.01778      0.01442
```

The data = regression.df part is telling **lm()** to look for Y, X_a and X_d values in the specified dataframe.

(2) Another way of using lm is to use the objects in the workspace as they are (which is a less favorable way since the code gets a bit messier).

```
lm(phenotypes[, 1] ~ Xa.all[, 1] + Xd.all[, 1])
```

```
##
## Call:
## lm(formula = phenotypes[, 1] ~ Xa.all[, 1] + Xd.all[, 1])
##
## Coefficients:
## (Intercept)  Xa.all[, 1]  Xd.all[, 1]
##     6.94741     -0.01778      0.01442
```

You can see that the outputs are cleaner in the former call, so let's just stick with the first way of calling **lm()**. Let us do a sanity check and see if the values are identical from those which we will get by calculating them using the equations that we leanred in class.

```
# MLE beta from Lab 7
library(MASS)
X.mx <- cbind(1, Xa.all[, 1], Xd.all[, 1])
Y <- as.matrix(phenotypes[, 1])
MLE_beta <- ginv(t(X.mx) %*% X.mx) %*% t(X.mx) %*% Y
MLE_beta
```

```
##             [,1]
## [1,]  6.94740631
```

```
## [2,] -0.01777821
## [3,]  0.01441792
```

It looks like they are almost identical (probably so because the lm function prints out the values rounded up to the 5th decimal point) !

So far, we have only gathered estimated values for each parameter ($\beta_m u$ = intercept, β_a = Xa, β_d = Xd). How can we get to the p-value of the whole model?

To answer this, look what happens when we save the **lm()** call into a variable and use the values from that object and then call **summary()** on the output from **lm()**

```
linear.fit <- lm(Y ~ Xa + Xd, data = regression.df)
lm.summary <- summary(linear.fit)
print(lm.summary)
```

```
##
## Call:
## lm(formula = Y ~ Xa + Xd, data = regression.df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.16871 -0.08484  0.00025  0.07644  0.20903
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.94741    0.01372 506.413   <2e-16 ***
## Xa          -0.01778    0.02295  -0.775    0.440
## Xd           0.01442    0.01372   1.051    0.296
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.09856 on 104 degrees of freedom
## Multiple R-squared:  0.05185,    Adjusted R-squared:  0.03362
## F-statistic: 2.844 on 2 and 104 DF,  p-value: 0.06275
```

This is giving us way more information than the **lm()** output. The call is showing us which model we have used, the residuals field is showing us the summary of the residuals, the coefficients field is showing us the parameter estimates and p values for each of the parameters, and at the end we can see the f-statistic degrees of freedoms and the p-value for the whole model. Let see how we can access these values. If you save the summary output to a variable you can see that it also has a lot of information stored inside. The values that we are interested can be found in the fstatistic list and coefficients list.

To get the p-value for the model we can simply do this:

```
fstat <- lm.summary$fstatistic
pvalue <- pf(fstat[1], fstat[2], fstat[3], lower.tail = FALSE)
print(pvalue)
```

```
##      value
## 0.06274908
```

2. Mini eQTL Analysis

We are going to run a mini-eQTL analysis with some real data downloaded from the HapMap Project. Since we would probably need something more powerful than a standard laptop to analyze the complete dataset, I

have downloaded the genotype and phenotype data for a single population (YRI) and scaled down the data to 400 genotypes and 10 phenotypes for 107 individuals.

You will find 4 files posted with this lab note:

- HapMap_phenotypes.tsv = 10 phenotypes (gene expression levels) for 107 individuals
- HapMap_genoytpes.tsv = 400 genotypes (coded as -1,0,1) for 107 individuals
- HapMap_gene_info.tsv = Partial gene information (entrez gene id, gene symbol, position)
- HapMap_snp_info.tsv = Partial SNP information (chromosome, position)

All the information in the files are tab separated (as you can guess from the extension .tsv).

The best place to always start is reading in the data:

```
hapmap.pheno.mx <- read.table("./HapMap_phenotypes.tsv", sep = "\t")
hapmap.geno.mx <- read.table("./HapMap_genotypes.tsv", sep = "\t")
hapmap.gene.info.df <- read.table("./HapMap_gene_info.tsv", sep = "\t")
hapmap.snp.info.df <- read.table("./HapMap_snp_info.tsv", sep = "\t")
```

Let's check to make sure the dimensions I gave are correct

```
dim(hapmap.pheno.mx)
```

```
## [1] 107  10
```

```
dim(hapmap.geno.mx)
```

```
## [1] 107 400
```

```
dim(hapmap.gene.info.df)
```

```
## [1] 1000    6
```

```
dim(hapmap.snp.info.df)
```

```
## [1] 1001    3
```

Nice! Everything checks out.

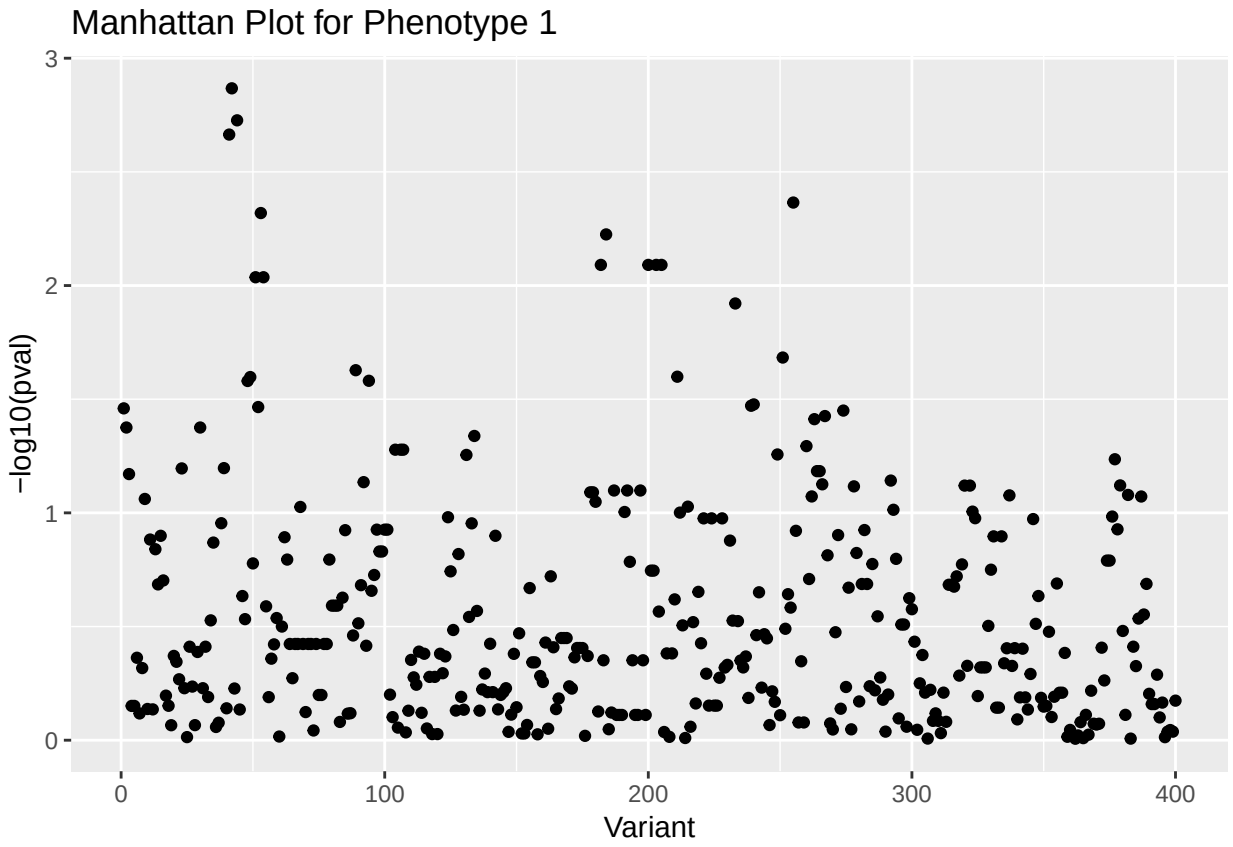
Since the data is already in -1,0,1 format we do not have to convert any letters into Xa and Xd codings. Rather we can jump right into the regression framework that we have been doing for the last couple weeks.

```
# Function to return pval from phenotype and genotype inputs
get_pval <- function(phenotypes, genotypes) {
    N.samples <- length(phenotypes)
    X.mx <- cbind(1, genotypes)
    MLE.beta <- solve(t(X.mx) %*% X.mx) %*% t(X.mx) %*% phenotypes
    y.hat <- X.mx %*% MLE.beta
    SSM <- sum((y.hat - mean(phenotypes))^2)
    SSE <- sum((phenotypes - y.hat)^2)
    df.M <- 1
    df.E <- N.samples - 2
    MSM <- SSM/df.M
    MSE <- SSE/df.E
    Fstatistic <- MSM/MSE
    return(pf(Fstatistic, df.M, df.E, lower.tail = FALSE))
}

# Initialize data structure to store pvals
pval.mx <- matrix(NA, nrow = ncol(hapmap.pheno.mx), ncol = ncol(hapmap.geno.mx))
```

```
# Loop through each phenotype and polymorphic site pair
for (p in 1:dim(hapmap.pheno.mx)[2]) {
    for (g in 1:dim(hapmap.geno.mx)[2]) {
        pval.mx[p, g] <- get_pval(hapmap.pheno.mx[, p], hapmap.geno.mx[,
            g])
    }
}
```

The p-values are currently held within a matrix with 10 rows for each phenotype and 400 columns for each variant. Let's reorganize this and plot a manhattan plot.

```
pval.mx <- data.frame(t(pval.mx))

# Using ggplot...
ggplot(pval.mx, aes(1:nrow(pval.mx), -log10(X1))) + geom_point() +
    labs(x = "Variant", y = "-log10(pval)", title = "Manhattan Plot for Phenotype 1")
```




```
# Compare with Basic R plot: plot(seq(nrow(pval.mx)),
# -log10(pval.mx[,1]), xlab='Variant', ylab='-log10(pval)',
# main='Manhattan Plot for Phenotype 1')
```

This plot looks good, but we now have much more information to actually spruce the plot up and provide more data. First we have to find the SNPs that we actually investigated, then include that data with our plotting data frame.

```
# Initialize new columns in pval.mx to store geno and pheno
# info
pval.mx$chrom = rep("unknown", nrow(pval.mx))
```

```
pval.mx$pos = rep("unknown", nrow(pval.mx))

hapmap.snp.info.df$chrom <- as.character(hapmap.snp.info.df$chrom)
hapmap.snp.info.df$id <- as.character(hapmap.snp.info.df$id)

#
i = 1
for (rs in colnames(hapmap.geno.mx)) {
    if (any(hapmap.snp.info.df$id == rs)) {
        pval.mx$chrom[i] <- hapmap.snp.info.df$chrom[hapmap.snp.info.df$id ==
            rs]
        pval.mx$pos[i] <- hapmap.snp.info.df$pos[hapmap.snp.info.df$id ==
            rs]
    }
    i = i + 1
}
```

```
head(pval.mx)
```

```
##           X1         X2        X3         X4        X5        X6        X7
## 1 0.03469745 0.55898794 0.8290260 0.05434251 0.5311641 0.1537401 0.9771972
## 2 0.04212491 0.51833765 0.8113998 0.03653108 0.6604735 0.1497931 0.9447695
## 3 0.06750895 0.81724955 0.2403243 0.12177296 0.8762431 0.1904715 0.6421696
## 4 0.70657580 0.37119041 0.5728266 0.27150125 0.8655386 0.1029649 0.8081247
## 5 0.70657580 0.37119041 0.5728266 0.27150125 0.8655386 0.1029649 0.8081247
## 6 0.43382537 0.09586456 0.5919667 0.93939698 0.3443391 0.7110445 0.3484010
##          X8        X9       X10 chrom       pos
## 1 0.6792351 0.6427457 0.5484219  chr7 128232781
## 2 0.7827312 0.8325633 0.5701952  chr7 128232938
## 3 0.8794031 0.6482962 0.8606116  chr7 128240559
## 4 0.5810495 0.3782971 0.9772663  chr7 128243641
## 5 0.5810495 0.3782971 0.9772663  chr7 128244335
## 6 0.1814152 0.6282845 0.6158774  chr7 128248635
```

This isn't a great example, since all of the SNPs are from chromosome 7, but ina typical GWAS you can (and probably should) make sure really nice alternating coloring that quickly allows for SNP location and chromosome identification.

The last thing we might want to do with the data is determine which variant is the most significant

```
phenotypeIndex <- 1
minpval.row <- which.min(pval.mx[, phenotypeIndex])

pheno_name <- colnames(hapmap.pheno.mx)[phenotypeIndex]
geno_name <- colnames(hapmap.geno.mx)[minpval.row]

gene.info.vec <- hapmap.gene.info.df[which(hapmap.gene.info.df$probe ==
    pheno_name), ]
genotype.info.vec <- hapmap.snp.info.df[which(hapmap.snp.info.df$id ==
    geno_name), ]

cat("Gene = ", pheno_name, "| ", paste0("chr", gene.info.vec$chromosome),
    "Start =", gene.info.vec$start, "End =", gene.info.vec$end,
    "\n")
```

```
## Gene =  ILMN_1757379 |  chr7 Start = 128772488 End = 128775789
```

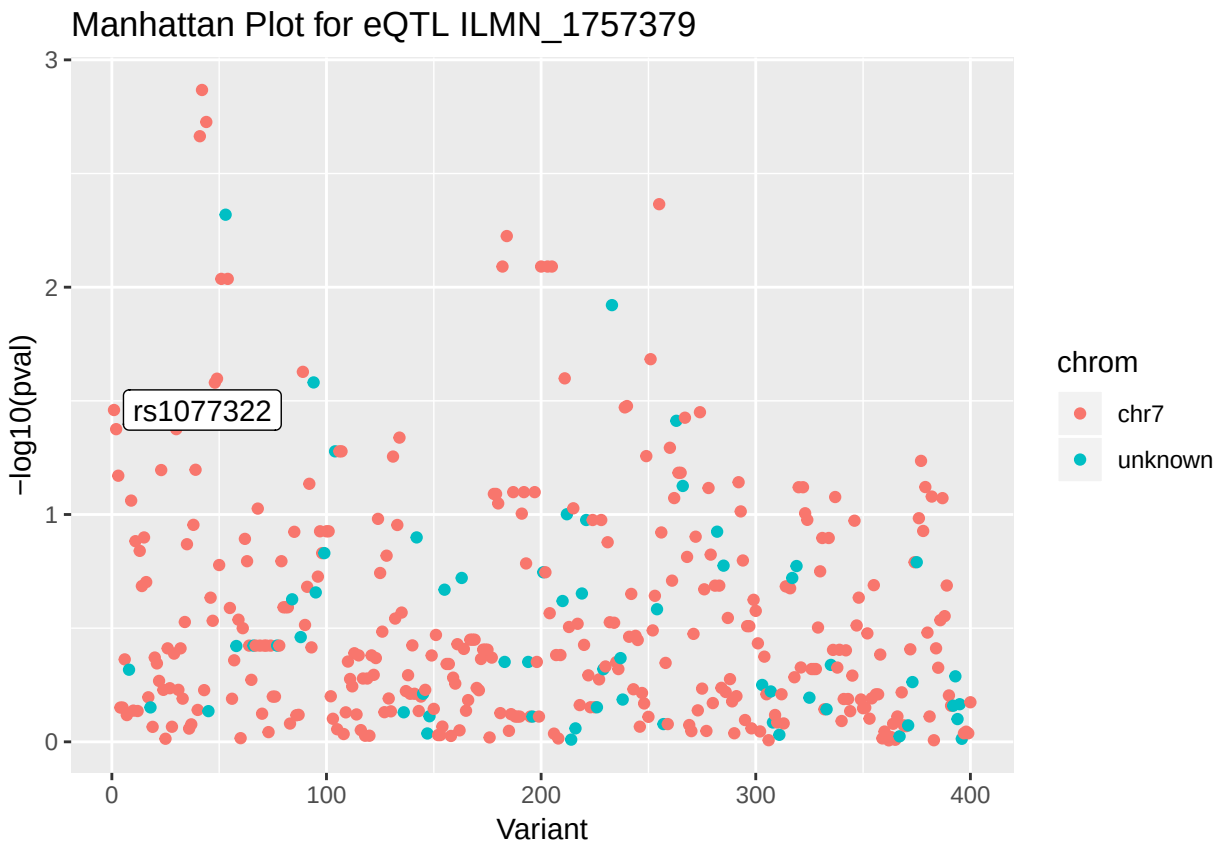
```
cat("SNP = ", geno_name, "| ", toString(genotype.info.vec$chrom),
    "Position =", genotype.info.vec$position, "\n")
```

```
## SNP =  rs1077322 |  chr7 Position = 128383467
```

Lastly, to again spruce up the GWAS and show the power of ggplot2, let's add the rs information and gene ID to the Manhattan Plot.

```
# install.packages('ggrepel')
library(ggrepel)

pval.mx$sigRsid <- NA
pval.mx$sigRsid[phenotypeIndex] <- colnames(hapmap.geno.mx)[minpval.row]

ggplot(pval.mx, aes(1:nrow(pval.mx), -log10(X1), label = sigRsid)) +
    geom_point(aes(color = chrom)) + labs(x = "Variant", y = "-log10(pval)",
    title = paste("Manhattan Plot for eQTL", colnames(hapmap.pheno.mx)[phenotypeIndex])) +
    geom_label_repel()
```

```
## Warning: Removed 399 rows containing missing values (geom_label_repel).
```



3. Covariates

The lm() framework also allows us to easily include the covariates within our analysis. All we need to do is add another term to the design equation. Going back to our basic example:

```
df <- data.frame(y = rnorm(100), x = rnorm(100, mean = 2), covar = rnorm(100,
    mean = 5))
```

```
model <- lm(data = df, y ~ x + covar)
model
```

```
##
## Call:
## lm(formula = y ~ x + covar, data = df)
##
## Coefficients:
## (Intercept)            x        covar
##    -0.26539     -0.03109      0.02957
```

By including different covariates in our model we can piece out where each component that makes up the phenotype comes from. In the below very basic example, the linear regression is super accurately able to determine that two times the z value makes up the y value compared to only one of the x value.

```
df <- data.frame(x = rnorm(100, mean = -2), z = rnorm(100, mean = 5))
df$y <- df$x + 2 * df$z
model <- lm(data = df, y ~ x + z)
model
```

```
##
## Call:
## lm(formula = y ~ x + z, data = df)
##
## Coefficients:
## (Intercept)            x            z
##   2.132e-15    1.000e+00    2.000e+00
```

This is an important feature if there was some odd confounding factor in the genetic data, for example population structure. To remove population structure we would make the covariate the principal components we disccussed in last lab. Currently the data is all from one population, so we don't have the population structure. To make a problem we will add some signal to certain people with the 1 genotype.

```
peopleToChange <- which(hapmap.geno.mx[, 2] == 1)[1:20]
alter.hapmap.pheno.mx <- data.frame(hapmap.pheno.mx)
alter.hapmap.pheno.mx[peopleToChange, 2] <- alter.hapmap.pheno.mx[peopleToChange,
    2] + 1
changeStatus <- rep(0, nrow(hapmap.geno.mx))
changeStatus[peopleToChange] <- 1
changeStatus <- as.factor(changeStatus)
```

We record which people were changed in a factor vector called changeStatus, so that we can later use it as a covariate. Now let's run out the regression for each scenario:

```
pvalOrig <- summary(lm(hapmap.pheno.mx[, 2] ~ hapmap.geno.mx[,
    1]))$coefficients[2, 4]
pvalOrig
```

```
## [1] 0.5589879
```

```
pvalAlter <- summary(lm(alter.hapmap.pheno.mx[, 2] ~ hapmap.geno.mx[,
    1]))$coefficients[2, 4]
pvalAlter
```

```
## [1] 0.0003544341
```

```
pvalCovar <- summary(lm(alter.hapmap.pheno.mx[, 2] ~ hapmap.geno.mx[,
    1] + changeStatus))$coefficients[2, 4]
```

```
pvalCovar
```

```
## [1] 0.8095094
```

We see that the original pvalue is not significant, but whe we altered the phenotype there is a very significant association. We can fix this statistical damage done by still using the altered phenotype but including the change status covariate. The outcome is a pvalue that is even less statistically significant than the original. This is a good result, as we no longer have a false positive although it should be noted that we are not able to perfectly recover our results.

4. Exercise–PC1 as a covariate

Calculate the principal components of the genotype matrix. Use the first principal component as a covariate within a linear regression test between all of the variants and the fourth phenotype. Plot a QQ-Plot of the p-values with the covariate regression and without the covariate regression. Are there any significant hits in either of your regression models? Submit your code as an R or Rmd file to CMS.