BTRY4830_Lab10

Olivia Lang

4/11/2019

1. Review: Reading in Data
2. Calculate p-value with covariates

   Compare F-statitstic calculation with and without covariates

   Simulate some phenotype data with
3. Logistic Regression

   GWAS model for catgorical phenotypes (binary, take values 0 or 1)
4. Iterative Re-weighted Least Squares (IRLS) Algorithm

   How to get the MLE betas for this new model
5. Exercise–Implementing the IRLS Algorithm to conduct a Logistic Regression GWAS
6. Solution

1. Review: Reading in Data

We have been reading in data from genotype and phenotype files for the past few labs, but now let's slow down and actualy discuss the different arguments that can help with this process. The basic arguments are header which is a boolean argument deciding whether the first row should become the rownames, and row.names which is a numeric or F

```
geno <- read.table("example.tsv", header = T, row.names = 1)
head(geno)
```

```
##         rs791607 rs791608 rs4236625 rs4731427 rs10244329
## NA18486        0        0         1         1         -1
## NA18487       -1       -1         0         0          0
## NA18488        0        0         1         1          0
## NA18489        1        1         0         0          0
## NA18498        1        1         0         0          1
## NA18499        1        1         0         0          1
```

In order to remove these row names set the argument to null. If the length of the header line was the same as the first row we could set header = F to also remove the column names, but because the header line is shorter we cannot. You can test this out.

```
geno <- read.table("example.tsv", header = T, row.names = NULL)
head(geno)
```

```
##   row.names rs791607 rs791608 rs4236625 rs4731427 rs10244329
## 1   NA18486        0        0         1         1         -1
## 2   NA18487       -1       -1         0         0          0
## 3   NA18488        0        0         1         1          0
## 4   NA18489        1        1         0         0          0
## 5   NA18498        1        1         0         0          1
## 6   NA18499        1        1         0         0          1
```

However this process would not work if you tried to read in a comma separated file:

```
geno <- read.table("example.csv", header = T, row.names = 1)
head(geno)
```

```
## data frame with 0 columns and 6 rows
```

We have to specify the delimiter or switch to using the read.csv() function:

```
geno <- read.table("example.csv", header = T, row.names = 1, sep=",")   # specify delimiter
head(geno)
```

```
##         rs791607 rs791608 rs4236625 rs4731427 rs10244329
## NA18486       TT       TT        AA        AA         CC
## NA18487       CC       CC        TT        TT         TT
## NA18488       TT       TT        AA        AA         TT
## NA18489       AA       AA        TT        TT         TT
## NA18498       AA       AA        TT        TT         AA
## NA18499       AA       AA        TT        TT         AA
```

```
geno <- read.csv("example.csv", header = T, row.names = 1)              # switch to read.csv()
head(geno)
```

```
##         rs791607 rs791608 rs4236625 rs4731427 rs10244329
## NA18486       TT       TT        AA        AA         CC
## NA18487       CC       CC        TT        TT         TT
## NA18488       TT       TT        AA        AA         TT
## NA18489       AA       AA        TT        TT         TT
## NA18498       AA       AA        TT        TT         AA
## NA18499       AA       AA        TT        TT         AA
```

Look at the data types of each column, they are factors. Factors are a data type, just like numeric, character, or integers. There are two parts of a factor, the data and the levels. To a computer the data is a basic integer 1,2,3, ... and the levels are the translation 1="a", 2="b", 3="c", ...

```
str(geno[,1])
```

```
##  Factor w/ 3 levels "AA","CC","TT": 3 2 3 1 1 1 3 3 1 3 ...
```

Factors can be great for two reasons: you have catagorical data or you have lots of data.

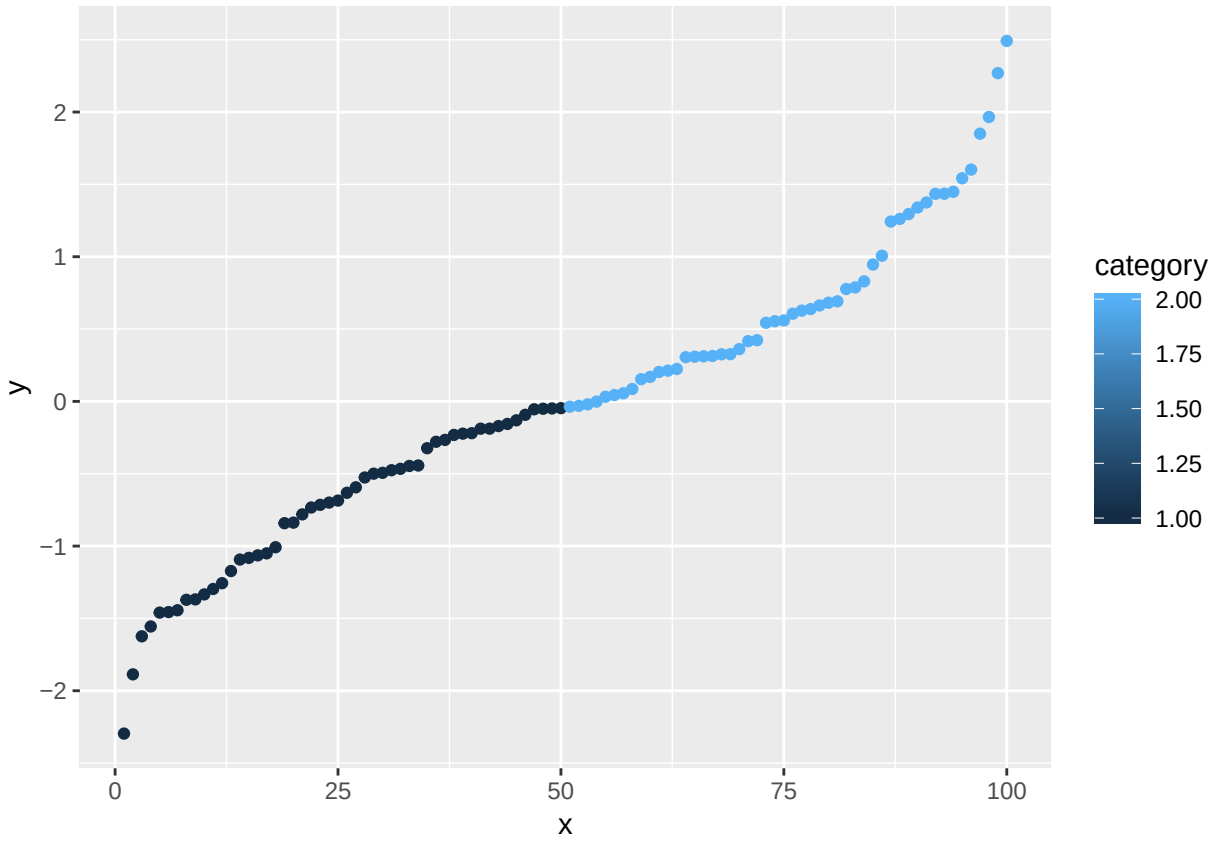
```
x <- sample(c("AA","CC"),10000,replace = T)
object.size(x)
```

```
## 80160 bytes
```

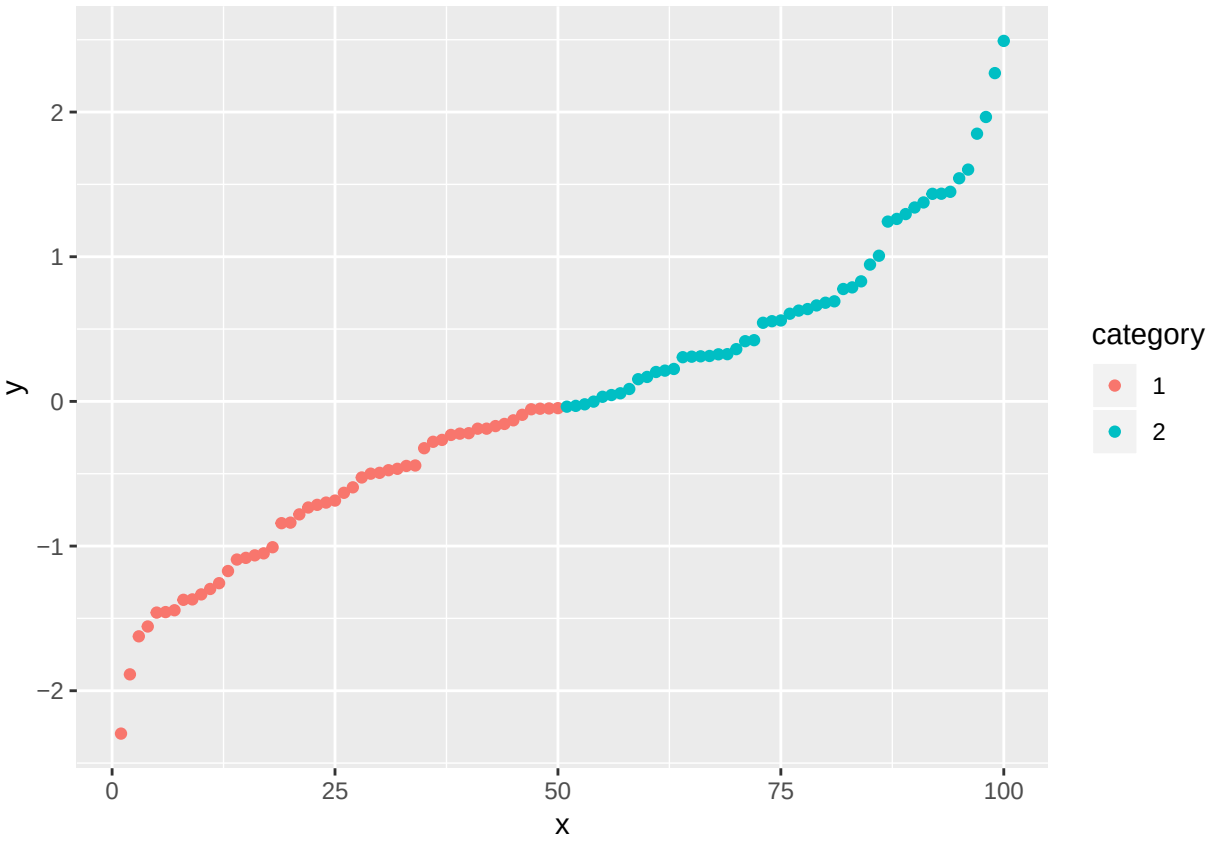
```
y <- as.factor(x)
object.size(y)
```

```
## 40560 bytes
```

```
df <- data.frame(x=1:100,y=sort(rnorm(100)),
                 category=c(rep(1,50),rep(2,50)))

ggplot(df,aes(x,y))+geom_point(aes(color=category))
```




```
df$category <- as.factor(df$category)
ggplot(df,aes(x,y))+geom_point(aes(color=category))
```



Factors can also come with plenty of problems, as you cannot do any calculations with factors and they do not sort as you would expect:

```
x <- as.factor(1:10)
mean(x)
```

```
## Warning in mean.default(x): argument is not numeric or logical: returning
## NA
```

```
## [1] NA
```

Therefore, it is likely a good idea to just steer clear of factors unless we know we want them for a particular purpose. To do this we simply set stringsAsFactors=F.

```
geno <- read.csv("example.csv", header = T, row.names = 1, stringsAsFactors = F)
head(geno)
```

```
##         rs791607 rs791608 rs4236625 rs4731427 rs10244329
## NA18486       TT       TT        AA        AA         CC
## NA18487       CC       CC        TT        TT         TT
## NA18488       TT       TT        AA        AA         TT
## NA18489       AA       AA        TT        TT         TT
## NA18498       AA       AA        TT        TT         AA
## NA18499       AA       AA        TT        TT         AA
```

There are plenty of other useful read.table arguments, and even other functions altogether. However what I would consider to be the 4 major arguments of read.table have just been covered. If messing with these arguments is still not getting the data into R, try to open the data file in vi, emacs, NotePad, or textEdit. Check out the delimiter, does it change throughout the data, or those spaces or are there tabs, do some lines have more elements than others? Try to tidy up the data so each row has the same number of elements

accoring to one consistent delimiter and the data should be able to be read into R. Further command line tools such as head, tail, cut, and tr can also be very helpful.

2. Calculate p-value with covariates

Recall in Lab 7 that the p-value calculator used an F-statistic ratio that was calculated using the variance between the estimates and the mean along with the variance between the estimates and the real phenotype values:

$$SSM = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 \qquad SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad F_{[2,n-3]} = \frac{\frac{SSM}{df(M)}}{\frac{SSE}{df(E)}} = \frac{\frac{SSM}{2}}{\frac{SSE}{n-3}} \tag{1}$$

```
# Lab 7 code calculating Fstatistic according to Lecture 12 equations

# Function to calculate the pval given a set of individuals' phenotype, and genotype encodings.
pval_calculator_lab7 <- function(pheno_input, xa_input, xd_input){
    n_samples <- length(xa_input)

    X_mx <- cbind(1,xa_input,xd_input)

    MLE_beta <- ginv(t(X_mx) %*% X_mx) %*% t(X_mx) %*% pheno_input
    y_hat <- X_mx %*% MLE_beta

    SSM <- sum((y_hat - mean(pheno_input))^2)
    SSE <- sum((pheno_input - y_hat)^2)

    df_M <- 2
    df_E <- n_samples - 3

    MSM <- SSM / df_M
    MSE <- SSE / df_E

    Fstatistic <- MSM / MSE

    pval <- pf(Fstatistic, df_M, df_E,lower.tail = FALSE)

    return(pval)
}
```

Below is a function for calculating the F-statistic based on the equations from Lecture 15. It is important to remember that this pvalue is controlling for covariates by including the covariates in the null hypothesis.

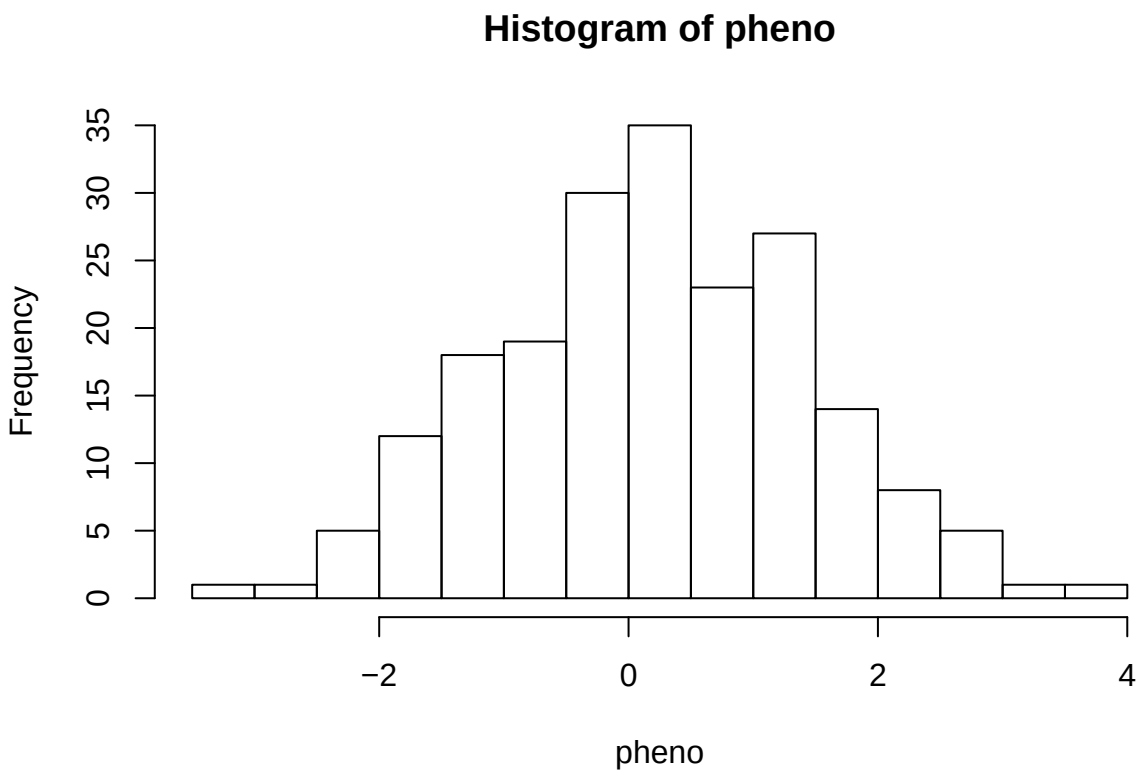
$$SSE(\hat{\theta_0}) = \sum_{i=1}^{n}(y_i - \hat{y}_{i,\hat{\theta_0}})^2 \qquad SSE(\hat{\theta_1}) = \sum_{i=1}^{n}(y_i - \hat{y}_{i,\hat{\theta_1}})^2 \qquad F_{[2,n-3]} = \frac{\frac{SSE(\hat{\theta_0}) - SSE(\hat{\theta_1})}{2}}{\frac{SSE(\hat{\theta_1})}{n-3}} \tag{2}$$

```
# New function to calculate the pval given a set of individuals' phenotype, and genotype encodings, adj
pval_calculator_lab10 <- function(pheno_input, xa_input, xd_input, z_input){
    n_samples <- length(xa_input)

    # Set up random variables for null (Z_mx) and with genotypes (XZ_mx)
    Z_mx <- cbind(1,z_input)                                                    # H0 (w/ cova
```

```
    XZ_mx <- cbind(1,xa_input,xd_input,z_input)                                  # w/ genotype

    # Calculate MLE betas for both null model and model with genotypes and covariates
    MLE_beta_theta0 <- ginv(t(Z_mx)  %*% Z_mx)  %*% t(Z_mx)  %*% pheno_input    # H0 (w/ cova
    MLE_beta_theta1 <- ginv(t(XZ_mx) %*% XZ_mx) %*% t(XZ_mx) %*% pheno_input    # w/ genotype

    # Get Y estimates using the betas calculated above to give each hypothesis its best chance
    y_hat_theta0 <- Z_mx  %*% MLE_beta_theta0                                   # H0 (w/ cova
    y_hat_theta1 <- XZ_mx %*% MLE_beta_theta1                                   # w/ genotype

    # Get the variance between the true phenotype values and our estimates under each hypothesis
    SSE_theta0 <- sum((pheno_input - y_hat_theta0)^2)                           # H0 (w/ cova
    SSE_theta1 <- sum((pheno_input - y_hat_theta1)^2)                           # w/ genotype

    # Set degrees of freedom
    df_M <- 2
    df_E <- n_samples - 3

    # Put together calculated terms to get Fstatistic
    Fstatistic <- ((SSE_theta0-SSE_theta1)/df_M) / (SSE_theta1/df_E)

    # Determine pval of the Fstatistic
    pval <- pf(Fstatistic, df_M, df_E,lower.tail = FALSE)
    return(pval)
}
```

Below is some simulated data where we generate a bunch of genotypes as Xa and Xd encodings. We also simulated covariates and then simulated phenotypes such that they are dependent on the genotypes and covariates

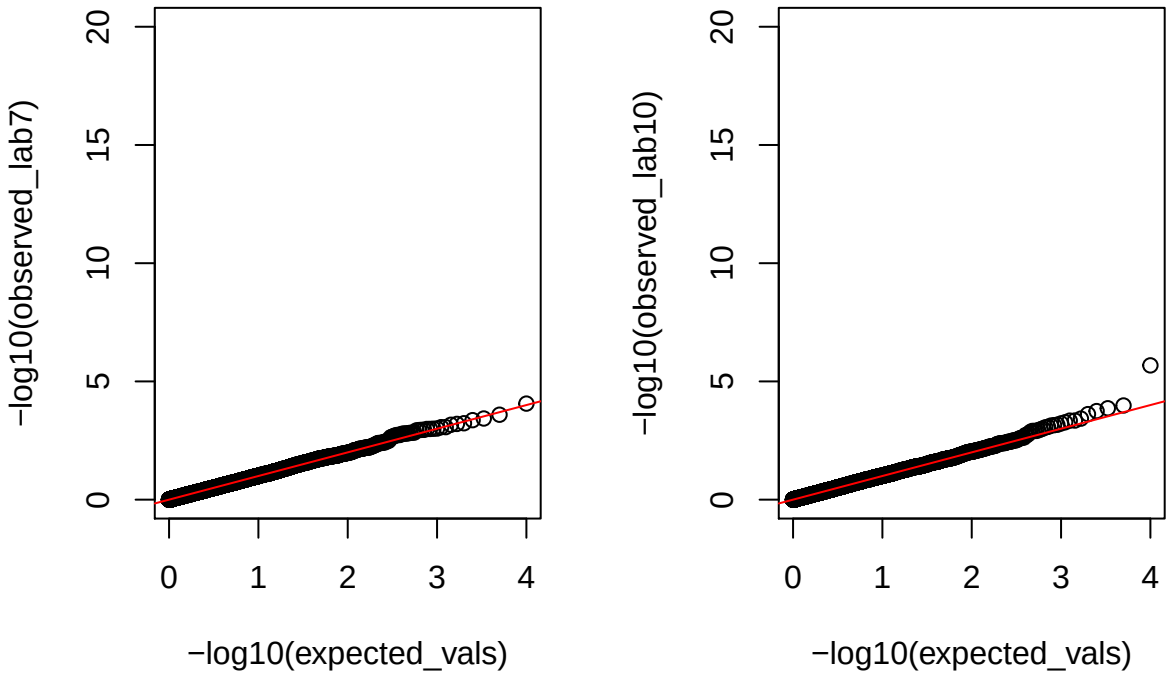
```
set.seed(2019)

# Set the dimensions of the data (Number of individuals and polymorphic sites to obtain genotypes for)
n_individuals = 200
n_polymorphic_sites = 10000
# simulate genotypes as Xa and Xd encodings using HW frequencies with each allele freq=0.5
xa_sim <- matrix( sample(c(1,0,-1),
                         n_individuals*n_polymorphic_sites,
                         replace=T,
                         prob=c(0.25, 0.5, 0.25)),
                  nrow = n_individuals,
                  ncol = n_polymorphic_sites)
xd_sim <- 2*abs(xa_sim) -1
# simulate two covariates
z_sim <- cbind( sample(c(-1, 1), n_individuals, replace=T, prob = c(0.5,0.5)),
                sample(c(-1, 1), n_individuals, replace=T, prob = c(0.5,0.5)) )
# simulate phenotypes based on the following true parameters
causal_site <- 150
true_betas <- c(  0.3,  0.5, -0.2,  0.3,  0.6 )  # Bmu, Ba, Bd, Bz1, Bz2, ...
epsion_sigmasq <- 1                  # sigma_sq in \epsilon=N(0,sigma_sq)
pheno <- cbind( 1, xa_sim[,causal_site], xd_sim[,causal_site], z_sim ) %*% true_betas + rnorm(n_individu
hist(pheno, breaks=20)
```



Below we used the simulated data to calculate pvalues for each polymorphic site using the calculators from lab7 and the new one we are introducing in this lab now. When we compare the QQ-plots side-by-side, you can see that controlling for covariates using the new pvalue calculator tightens the data around the y=x red line where out pvalues ideally would reside.

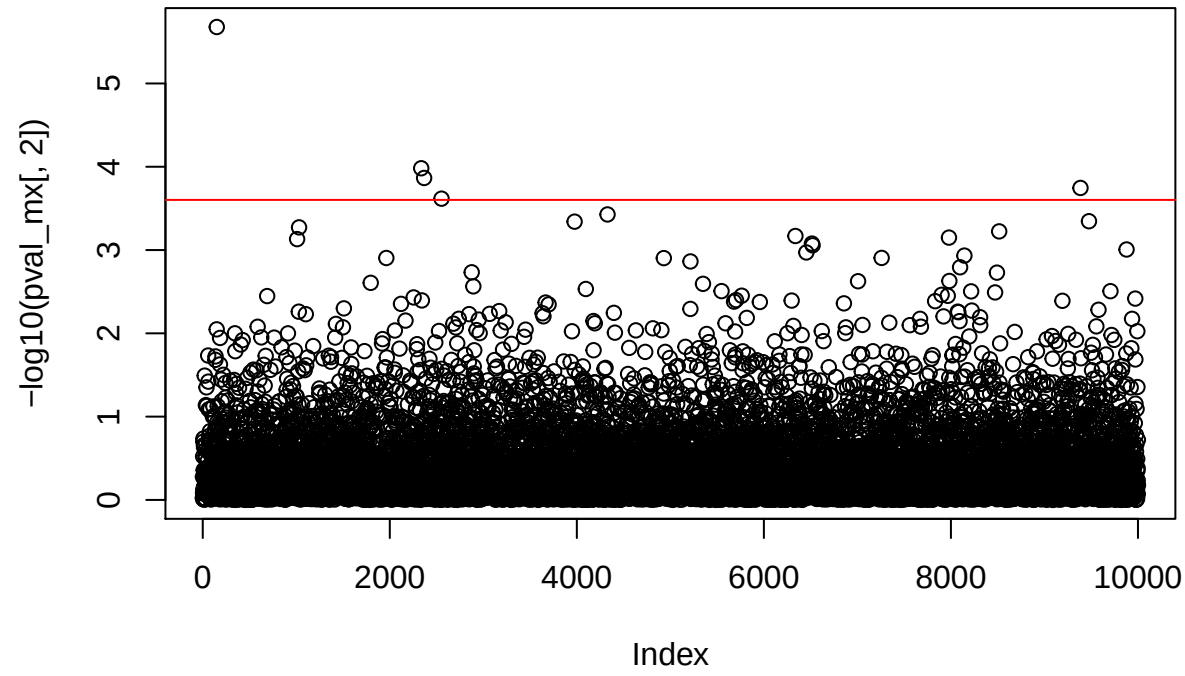
```
# Initialize some variables and constants
pval_mx <- matrix(NA, nrow = n_polymorphic_sites, ncol=2)

# Calculate and save pvals for each phenotype-genotype pair
for (i in 1 : n_polymorphic_sites){
  pval_mx[i,1] <- pval_calculator_lab7(pheno_input = pheno,
                                       xa_input = xa_sim[,i],
                                       xd_input = xd_sim[,i])
  pval_mx[i,2] <- pval_calculator_lab10(pheno_input = pheno,
                                        xa_input = xa_sim[,i],
                                        xd_input = xd_sim[,i],
                                        z_input = z_sim)
}

par(mfrow=c(1,2))

# Compare QQ plots
expected_vals <- seq(1/n_polymorphic_sites, 1, by=1/n_polymorphic_sites)
observed_lab7 <- sort(pval_mx[,1])
observed_lab10<- sort(pval_mx[,2])

plot(-log10(expected_vals), -log10(observed_lab7),
     ylim=c(0,20))
abline(a=0,b=1,col='red')
```

```
plot(-log10(expected_vals), -log10(observed_lab10),
     ylim=c(0,20))
abline(a=0,b=1,col='red')
```



And finally, from the manhattan plot of the new pvalue calculator, we can identify the correct causal site.

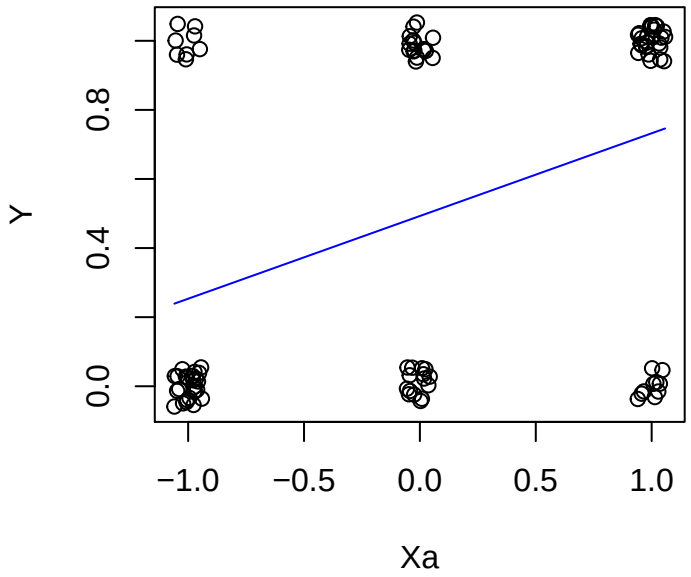
```
par(mfrow=c(1,1))
plot(-log10(pval_mx[,2]) )
abline(h=-log10(0.05/n_individuals),
       col='red')
```



3. Logistic Regression

Today we are going to be talking about performing a **logistic regression.** In a logistic regression, the dependent variable y (in our case phenotype) is **categorical** instead of continuous. Specifically, we are going to use logistic regression to deal with binary phenotypes coded as 0 and 1. For example, in genome-wide association studies (GWAS) a healthy or normal control phenotype would be 0 and a disease phenotype (ex. diabetes, alzheimers, etc . . .) would be 1, and the goal is to identify genomic variations that increase the probability of belonging to the disease category.

You might be wondering why we need this in the first place. So let's try to use linear regression for a binary phenotype and see what happens.



-What is the predicted value of Y for $X_a = -1$ in this case?

-If you plot out the residuals, how would it look like?

It becomes quite clear that we need a better alternative to linear regression for binary phenotypes. Simply put, logistic regression transforms the linear model that we use to "fit" the binary phenotypes. The logistic function takes the form as follows:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

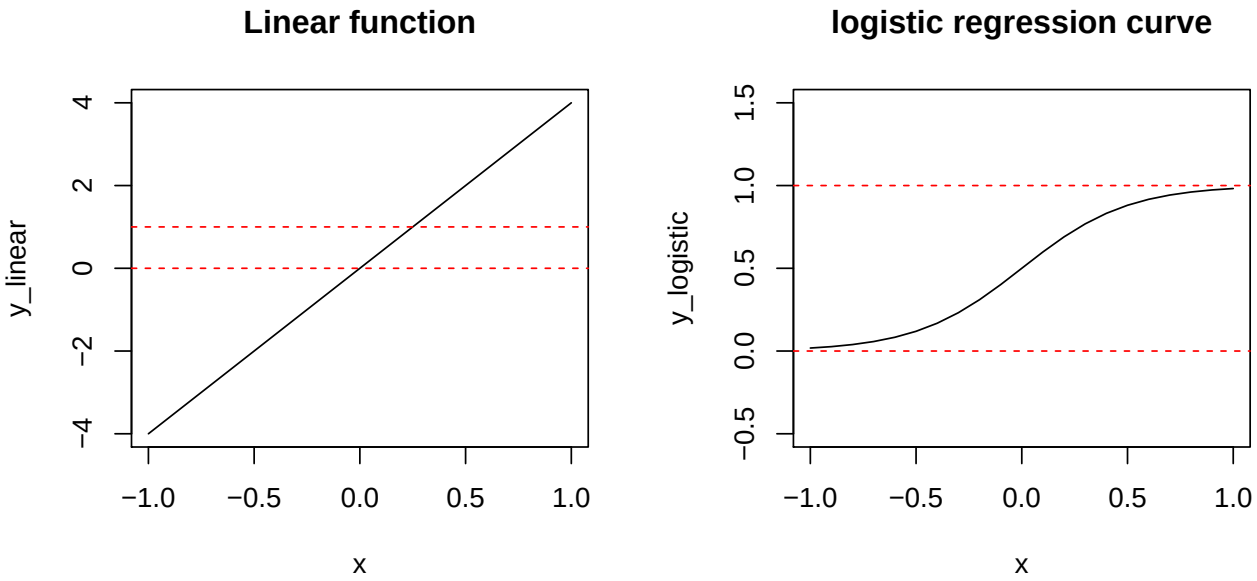
If we substitute t with the linear function that depends on x and beta values, the logistic function that we use becomes

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

A simple visualization in R might give us a better idea of how the data is transformed. Basically, logistic regression confines the original dependent values within the range of 0 and 1.

```
x <- seq(-1,1,by = 0.1)
y_linear <- x * 4
y_logistic <- 1 / ( 1 + exp(-y_linear))

par(mfrow = c(1,2))
plot(x, y_linear, main = "Linear function", type = "l")
```

```r
abline(h = 0, col = "red", lty = 2)
abline(h = 1, col = "red", lty = 2)
plot(x,y_logistic, main = "logistic regression curve", type ="l", ylim = c(-0.5,1.5))
abline(h = 0, col = "red", lty = 2)
abline(h = 1, col = "red", lty = 2)
```



Let's project the settings in our problem onto the above equation and clarify our goal. We have a given phenotype that takes either a value of 1 or 0, and two matrices for genotypes in the form of Xa(-1,0,1) coding and Xd(-1,1) coding. Just like in linear regression, the goal is to find the values for β_μ, β_a and β_d for each genotype that best explain the relationship between the genotype and phenotype.

If the relationship was error free and the genotype value directly predicts the phenotype, we would not need logistic regression (For example, if A2A2 indicates phenotype = 1 with 100% certainty). However, that is more than often not the case in real world genetics/genomics so we would have to "soft" model the relationship between genotypes and phenotypes by using probabilities, (In other words, A2A2 has a higher chance of having phenotype=1 than phenotype=0) and that is what the transformation given in the above equation is doing.

4. Iterative Re-weighted Least Squares (IRLS) Algorithm

However, unlike the simple matrix calculation in linear regression for the MLE(beta) values, we don't have that closed form estimate here (recall: $(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$.The solution to this problem is an "iterative" approach where the algorithm starts at a given point and keeps looking for a better solution in following steps until the better solution is almost identical to the solution from the previous step.

Algorithms similar to this have a general outline as follows.

- An **objective (or cost) function:**. This is a function which represents how well the model fits. For example, in linear regression a lower sum of squared errors (deviation of the predicted phenotypes from the actual phenotypes) represents a better model fit. So the goal of these algorithms would be to minimize (or maximize depending on the situation) the given objective function.
- **Optimization function:** The core of the algorithm which finds the parameter values that minimize the objective function. Most of the algorithms will use methods based on gradients (derivatives) of the objective function to find the direction to update the parameters.

Imagine that you are on a mountain in complete darkness and that you only know the angle of the ground and the current altitude (objective function) which you can check every 5 minutes. The goal for you is to get to the highest point (find the maxiumum) that you can reach and shoot up a flare to call for help. The optimal strategy for you will likely be to pick a different direction to walk for 5 minutes (step size) based on the angle of the ground (derivative of objective function) you are standing on, and check your altitude after 5 minutes to confirm that you actually went uphill not downhill. When you are close to (or on) the top the altitude might not change much after walking for 5 minutes and that might be your best place for shooting a flare. This is kind of what is going on in the algorithm that we are implementing.

4.1 Useful equations for the exercise

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} 1 & x_{1,a} & x_{1,d} \\ 1 & x_{2,a} & x_{2,d} \\ \vdots & \vdots & \vdots \\ 1 & x_{n,a} & x_{n,d} \end{bmatrix} \qquad \beta^{[0]} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \beta^{[t]} = \begin{bmatrix} \beta_\mu^{[t]} \\ \beta_a^{[t]} \\ \beta_d^{[t]} \end{bmatrix} \tag{3}$$

$$\beta^{[t+1]} = \beta^{[t]} + [\mathbf{x}^T\mathbf{W}\mathbf{x}]^{-1}\mathbf{x}^T(\mathbf{y} - \gamma^{-1}(\mathbf{x}\beta^{[t]})) \tag{4}$$

$$\gamma^{-1}(\beta_\mu^{[t]} + x_{i,a}\beta_a^{[t]} + x_{i,d}\beta_d^{[t]}) = \frac{e^{\beta_\mu^{[t]} + x_{i,a}\beta_a^{[t]} + x_{i,d}\beta_d^{[t]}}}{1 + e^{\beta_\mu^{[t]} + x_{i,a}\beta_a^{[t]} + x_{i,d}\beta_d^{[t]}}} \tag{5}$$

$$\gamma^{-1}(\mathbf{x}\beta^{[t]}) = \frac{e^{\mathbf{x}\beta^{[t]}}}{1 + e^{\mathbf{x}\beta^{[t]}}} \tag{6}$$

$$W_{ii} = \gamma^{-1}(\mathbf{x}\beta^{[t]})(1 - \gamma^{-1}(\mathbf{x}\beta^{[t]})) \qquad W_{ij} = 0 \; for \; i \neq j \tag{7}$$

$$\Delta D = |D[t+1] - D[t]| \qquad \Delta D < 10^{-6} \tag{8}$$

$$D = 2\sum_{i=1}^{n}[y_i ln(\frac{y_i}{\gamma^{-1}(\mathbf{x}\beta^{[t]})}) + y_i ln(\frac{1 - y_i}{1 - \gamma^{-1}(\mathbf{x}\beta^{[t]})})] \tag{9}$$

$$LRT = -2ln\Lambda = 2l(\hat{\theta}_1|\mathbf{y}) - 2l(\hat{\theta}_0|\mathbf{y}) \tag{10}$$

$$\hat{\beta}_{\mu,0} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{11}$$

$$l(\hat{\theta}_0|\mathbf{y}) = \sum_{i=1}^{n}[y_i ln(\gamma^{-1}(\hat{\beta}_{\mu,0} + x_{i,a} * 0 + x_{i,d} * 0)) + (1 - y_i)ln(1 - \gamma^{-1}(\hat{\beta}_{\mu,0} + x_{i,a} * 0 + x_{i,d} * 0))] \tag{12}$$

$$l(\hat{\theta}_1|\mathbf{y}) = \sum_{i=1}^{n}[y_i ln(\gamma^{-1}(\hat{\beta}_\mu + x_{i,a}\hat{\beta}_a + x_{i,d}\hat{\beta}_d)) + (1 - y_i)ln(1 - \gamma^{-1}(\hat{\beta}_\mu + x_{i,a}\hat{\beta}_a + x_{i,d}\hat{\beta}_d))] \tag{13}$$

5. Exercise–Implementing the IRLS Algorithm to conduct a Logistic Regression GWAS

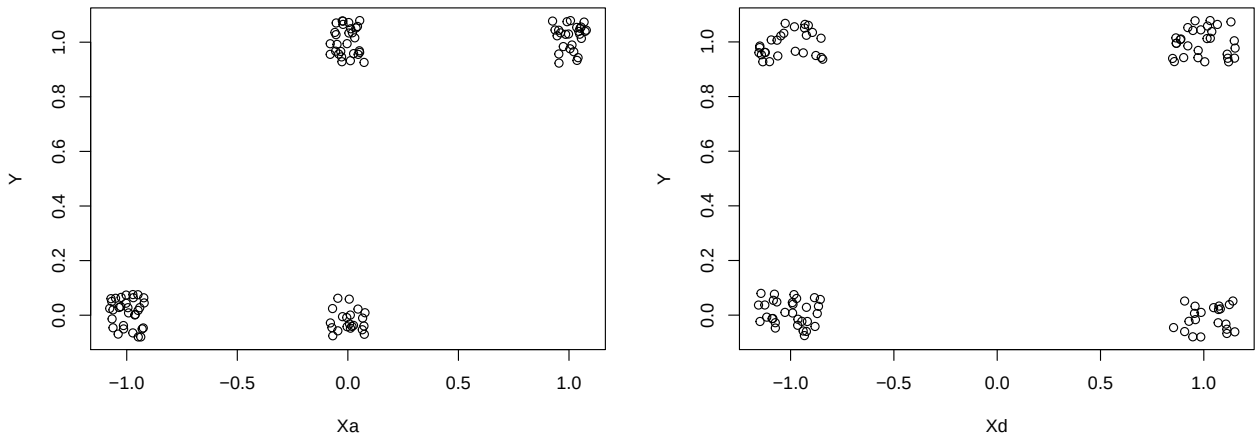
Directions below, please submit the code to generate your Manhattan plot to CMS as a .R or .Rmd file when you are done.

1) Download the phenotype and genotype files from the github site and read them in. You should have 292 genotypes and 1 phenotype for 107 samples.
2) Note that the genotypes are already in Xa codings, and you only have to create the Xd matrix from it.
3) Use the template given below and try to fill in the code to make it a functional algorithm.
4) Plot a manhattan plot for the phenotype and look for significant peaks.
5) Your code should look something like this :

```
W_calc <- function(gamma_inv){

    return(W)
}

beta_update <- function(X_mx, W, Y, gamma_inv, beta){

    return(beta_up)
}

gamma_inv_calc <- function(X_mx, beta_t){

    return(gamma_inv)
}

dev_calc <- function(Y, gamma_inv){

    return(deviance)
}

loglik_calc <- function(Y, gamma_inv){

    return(loglik)
}


logistic.IRLS<- function(Xa,Xd,Y = Y, beta.initial.vec = c(0,0,0), d.stop.th = 1e-6, it.max = 100) {

    # Create Initial values


    # Start of optimization loop
    for(i in 1:it.max) {

        # calculate W

        # update beta

        # update gamma_inv
```

```
        # calculate deviation

        # check if deviation is smaller than threshold
        if( ) {
            cat("Convergence at iteration:", i, "at threshold:", d.stop.th, "\n")
            logl<-# Log likelihood goes here
            return(list(beta_t,logl)) # return a list that has beta.t and logl saved
        }
    }

    # In case the algorithm did not coverge
    cat("Convergence not reached after iteration:", i, "at threshold:", d.stop.th, "\n")
    return(list(beta_t= c(NA,NA,NA),logl=NA)) # return NA values
}


G <- dim(Xa)[2]
logl <- vector(length = G)

for(j in 1:G){

  result.list <- call our function
  logl<- # How do we extract an element from a list? might want to use [[]]

}

# Calculate the log likelihood for the NULL using IRLS
logl_H0 <- logistic.IRLS(Y=Y, Xa= NULL, Xd=NULL, beta.initial.vec = c(0))[[2]]

LRT<-2*logl-2*logl_H0 #likelihood ratio test statistic

pval <- # chi squared test with the following parameters (LRT, 2, lower.tail = F)

# Plot manhattan plot with cut off line
plot(-log10(pval))
abline(-log10(0.05/300),0,col="red")
```

6) You can also visualize the individual genotype effect by using the jitter() function with plot()



Your code should look something like this:

```
library(MASS)
library(ggplot2)

W_calc <- function(gamma_inv){
    N <- length(gamma_inv)
        W<-diag(as.vector(gamma_inv * (1- gamma_inv)))

    return(W)
}

beta_update <- function(X_mx, W, Y, gamma_inv, beta){
  beta_up <- beta + ginv(t(X_mx)%*%W%*%X_mx)%*%t(X_mx)%*%(Y-gamma_inv)
    return(beta_up)
}

gamma_inv_calc <- function(X_mx, beta_t){
    #initialize gamma
    # K is the part which goes into the exponent
    K <- X_mx %*% beta_t
    gamma_inv <- exp(K)/(1+exp(K))
    return(gamma_inv)
}

dev_calc <- function(Y, gamma_inv){
    deviance <- 2*( sum(Y[Y==1]*log(Y[Y==1]/gamma_inv[Y==1])) + sum((1-Y[Y==0])*log((1-Y[Y==0])/(1-gamma
    return(deviance)
}

loglik_calc <- function(Y, gamma_inv){
    loglik <- sum(Y*log(gamma_inv)+(1-Y)*log(1-gamma_inv))
    return(loglik)
}

logistic.IRLS<- function(Xa,Xd,Y =Y, beta.initial.vec = c(0,0,0), d.stop.th = 1e-6, it.max = 100) {

  #Create the X matrix
  X_mx <- cbind(rep(1,nrow(Y)), Xa, Xd)

  #check this matrix:
    #initialize the beta parameter vector at t=0
    beta_t <- beta.initial.vec

  # initialize deviance at d[t]
    dt <- 0

    #initialize gamma
  # K is the part which goes into the exponent
  gamma_inv <- gamma_inv_calc(X_mx, beta_t)

    for(i in 1:it.max) {
        dpt1 <- dt #store previous deviance
```

```
    # create empty matrix W
        W <- W_calc(gamma_inv)

        beta_t <- beta_update(X_mx, W, Y, gamma_inv, beta_t)

        #update gamma since it's a function of beta

        gamma_inv <- gamma_inv_calc(X_mx, beta_t)

        #calculate new deviance
        dt <- dev_calc(Y, gamma_inv)

        absD <- abs(dt - dpt1)

        if(absD < d.stop.th) {
            #cat("Convergence at iteration:", i, "at threshold:", d.stop.th, "\n")
            logl <- loglik_calc(Y, gamma_inv)
            return(list(beta_t,logl))
        }
    }
    #cat("Convergence not reached after iteration:", i, "at threshold:", d.stop.th, "\n")
    return(list(beta_t= c(NA,NA,NA),logl=NA))
}

Y <- read.table("./phenotypes4lab10.tsv", header = F,stringsAsFactors = F)
geno <- read.table("./genotypes4lab10.tsv", header = T)

Y <- as.matrix(Y)
colnames(Y) <- NULL
xa_matrix <- as.matrix(geno)
xd_matrix <- 1 - 2*abs(xa_matrix)

beta<-NULL
logl<-NULL
for(j in 1:dim(xa_matrix)[2]){
    myList<-logistic.IRLS(xa_matrix[,j],xd_matrix[,j],Y=Y)
    beta<-cbind(beta,myList[[1]])
    logl<-c(logl,myList[[2]])
#   cat("Locus ",j,"'s beta values: ",myList[[1]],"\n")
}

# log likelihood for NULL hypothesis
logl_H0 <- logistic.IRLS(Y=Y, Xa= NULL, Xd=NULL, beta.initial.vec = c(0))[[2]]

# alternative approach that sets Xa and Xd to zero
logl_H0 <- logistic.IRLS(Y=Y, Xa= rep(0,nrow(Y)), Xd=rep(0,nrow(Y)), beta.initial.vec = c(0,0,0))[[2]]


LRT<-2*logl-2*logl_H0 #likelihood ratio test statistic

#likelihood ratio test statistic for every genotype
pval <- pchisq(LRT, 2, lower.tail = F)
```

```r
plot(-log10(pval), main = "P-values / Bonferroni cut-off")
abline(-log10(0.05/300),0,col="red")
```