BTRY4830_Lab12

Olivia Lang

4/25/2019

1. Linear Mixed Models
2. Motivation
3. EM Algorithm–Theory
4. EM Algorithm - Practice
5. Exercise

1. Linear Mixed Models

Linear mixed models are named so because unlike linear models with only fixed effects, they use a "mix" of fixed effects and random effects. To directly compare the difference between the two models, let's review the linear model in its simplest form we have used to model genotype effects on the phenotype.

$$\vec{y} = \mathbf{X}\vec{\beta} + \vec{\epsilon}$$

$$\vec{y_i} \sim N(\vec{0}, \sigma_e^2\mathbf{I})$$

$\vec{y_i}$ is a vector with n measurements of the phenotype, $\mathbf{X}$ is the $n \times j$ matrix with $j-1$ independent variables and one column with 1s for the mean, $\vec{\beta}$ is a j dimensional vector of the beta values, and $\vec{\epsilon}$ is the normally distributed error with a variance of σ_e^2. We can also represent the model as a multivariate normal distribution as:

$$\vec{y_i} \sim N(\mathbf{X}\vec{\beta}, \sigma_e^2\mathbf{I})$$

where the mean of the phenotypes depend on $\mathbf{X}\vec{\beta}$ and the error has no structure. In this case, we had to only worry about the likelihood of $\vec{\beta}$ and σ_e^2 given the data. This had a nice closed form solution to caluclate the maximum likelihood estimators for the β values which were the center of attention and the variance σ_e^2 which we did not focus on that much.

In comparison, linear mixed models have additional random effect terms in the model. The most commonly used form in genomics will look something like this:

$$\vec{y} = \mathbf{X}\vec{\beta} + \vec{a} + \vec{\epsilon}$$

$$\vec{a} \sim N(\vec{0}, \sigma_a^2\mathbf{A})$$

$$\vec{\epsilon} \sim N(\vec{0}, \sigma_e^2\mathbf{I})$$

where $\vec{a}$ is an n dimensional vector representing the random effect drawn from a multivariate normal distribution with a variance structure defined by the $n \times n$ covariance matrix $\mathbf{A}$. In a multivariate normal distribution form, the model can be shown as:

$$\vec{y} \sim N(\mathbf{X}\vec{\beta}, \sigma_a^2\mathbf{A} + \sigma_e^2\mathbf{I})$$

The key difference between the linear model and the linear mixed model lies in the modeling of the variance. Intuitively, **A** represents the pairwise similarity between the n samples, which has an affect on the residual variance. Simply put, the random effect is included to account for samples that are not independent. Now that we have three parameters to estimate ($\vec{\beta}$,σ_a^2,σ_e^2) there is no closed form solution anymore. This week we will learn how to get the maximum likelihood estimators for those parameters using an EM algorithm.

2. Motivation

From first comleting the simple or univariate regression in Lab 7 we have learned that there are certain factors that may confound the relationship between X and y. The two largest are population structure and linkage disequilirbium. Explanations for how they may cause confounding and how mixed models alleviate their problems are as follows:

We have dealt with population structure by including principal components as covariates. While this method works well, we are implicitly throwing away any knowledge about complex population interactions. For instance if person A has variant 1 and is very tall we may say that variant 1 causes height. By adding a covariate we may see that people in person's A population structure tend to be slightly taller, but there may be other variants acting upon this phenotype. However, by looking at person B who is related to person A we find that variant -1 also causes height. These contradictory results would lead to the conclusion of no association. By using person B, we gain more specific population information and variant-fine clarity of how population structure might be confounding.

Linkage disequilibrium has been an after though in linear regression. When looking at Manhattan Plots we condense significant variants into peaks or loci rather than single points of importance. While linkage disequilibrium is not explicitly considered within mixed models, we are implicitly taking it into account since multiple variants are being regressed simultaneously against the same phenotype. For example, we can consider 3 variants which are highly correlated within all of the samples. When estimating the association between each variant and the phenotypes the iterative mixed model process would first find the first variant to be highly significant. But on the next iteration the second variant is the significant one, and the significance of the fist becomes slightly diminished. This process continues, shrinking the association between variaints in high LD.

3. EM Algorithm–Theory

Roughly speaking, the process of the EM algorithm can be outlined in two steps. During the first step it fixes the parameters (betas, and sigmas) to calculate the best estimate for *alpha*, V in the code, which are used in the next step to update the betas and sigmas until the likelihood does not change much.

The technical aspect of the EM algorithm is beyond the scope of this course, so in this exercise we are going to focus on understanding teh concept, inputs and outputs. (If you are interested, I've outlined the steps below)

1. At step $[t]$ for $t = 0$, assign values to the parameters: $\beta^{[0]} = [\beta_\mu^{[0]}, \beta_a^{[0]}, \beta_d^{[0]}], \sigma_a^{2\,[0]}, \sigma_e^{2\,[0]}$. These need to be selected such that they are possible values of the parameters (e.g. no negative values for the variance parameters).
2. Calculate the expectation step for $[t]$:

$$\mathbf{a}^{[t]} = \left(\mathbf{Z}^T\mathbf{Z} + \mathbf{A}^{-1}\frac{\sigma_\epsilon^{2\,[t-1]}}{\sigma_a^{2\,[t-1]}}\right)^{-1}\mathbf{Z}^T(\mathbf{y} - \mathbf{x}\beta^{[t-1]})$$

$$V_a^{[t]} = \left(\mathbf{Z}^T\mathbf{Z} + \mathbf{A}^{-1} \frac{\sigma_\epsilon^{2[t-1]}}{\sigma_a^{2[t-1]}} \right)^{-1} \sigma_\epsilon^{2[t-1]}$$

3. Calculate the maximization step for $[t]$:

$$\beta^{[t]} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T(\mathbf{y} - \mathbf{Z}\mathbf{a}^{[t]})$$

$$\sigma_a^{2[t]} = \frac{1}{n}\left[\mathbf{a}^{[t]}\mathbf{A}^{-1}\mathbf{a}^{[t]} + tr \right]$$

4. EM Algorithm - Practice

Following the slides, the EM algorithm can be laid out in two steps. The first step is to calculate a and V, the second step is to calculate beta, sigma of a and sigma of epsilon. The V variable is a sum of both epsilon and a, in this way we are first expecting values that are implicity considered in the model and then secondly we are using these values to maximize the variables that are directly used to form the effects.

The technical aspect of the EM algorithm is beyond the scope of this course, so in this exercise we are going to focus on the important skill of converting the equations in the notes into working code. We begin by laying out the skeleton the code will fit within:

```
EM_algorithm = function(Y, X_j, A, max.iter = 100) {
  #initiate values

  while (iter < max.iter) {
    #Expect

    #Maximize

    if (log_L[iter] - log_L[iter - 1] < 1e-05) { break }
    iter = iter + 1
  }
  return(list(beta = beta, sigma_sq_a = sigma_sq_a, sigma_sq_e = sigma_sq_e, log_L = log_L[iter - 1]))
}
```

Beginning from the top we can break down "initiate values" into three different parts. First we need to declare our guesses to the values that will later be maximized, as they are needed for the inital step of expectation. Secondly, we can write down some "shortcuts", or name variables that we will use often. This practice of storing calculated values that are used many times later is a common practice in coding as it speeds up execution times. Thirdly we need to compute the likelihood value, which will be used after the first iteration has been completed to determine whether the algorithm is done running.

```
EM_algorithm = function(Y, X_j, A, max.iter = 100) {
  #Initiate values
  #These values are "shortcuts" for future calculations
  solve_A = ginv(A)
  n = length(Y)
  Z = diag(1, n)
  log_L = c()

  #These are our random guesses of the maximized values
  sigma_sq_a = 70
```

```
  sigma_sq_e = 10
  beta = as.vector(rep(0, ncol(X_j)))

  #Calculate initial likelihood
  C = A * sigma_sq_a + Z * sigma_sq_e
  log_L[1] = -1/2 * determinant(C)$modulus - 1/2 * t(Y - X_j %*% beta) %*% ginv(C) %*% (Y - X_j %*% beta
  while (iter < max.iter) {
    #Expect

    #Maximize

    if (log_L[iter] - log_L[iter - 1] < 1e-05) { break }
    iter = iter + 1
  }
  return(list(beta = beta, sigma_sq_a = sigma_sq_a, sigma_sq_e = sigma_sq_e, log_L = log_L[iter - 1]))
}
```

To fill in the expectation and maximization steps we can look at just one slide from lecture:

```
knitr::include_graphics("lectureSlide.png")
```

Mixed models: inference III

1. At step $[t]$ for $t = 0$, assign values to the parameters: $\beta^{[0]} = \left[\beta_\mu^{[0]}, \beta_a^{[0]}, \beta_d^{[0]}\right], \sigma_a^{2,[0]}, \sigma_\epsilon^{2,[0]}$. These need to be selected such that they are possible values of the parameters (e.g. no negative values for the variance parameters).
2. Calculate the expectation step for $[t]$:

$$\mathbf{a}^{[t]} = \left(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \mathbf{A}^{-1}\frac{\sigma_\epsilon^{2,[t-1]}}{\sigma_a^{2,[t-1]}}\right)^{-1}\mathbf{Z}^{\mathrm{T}}(\mathbf{y} - \mathbf{x}\beta^{[t-1]})$$

$$V_{\mathbf{a}}^{[t]} = \left(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \mathbf{A}^{-1}\frac{\sigma_\epsilon^{2,[t-1]}}{\sigma_a^{2,[t-1]}}\right)^{-1}\sigma_\epsilon^{2,[t-1]}$$

3. Calculate the maximization step for $[t]$:

$$\beta^{[t]} = (\mathbf{x}^{\mathrm{T}}\mathbf{x})^{-1}\mathbf{x}^{\mathrm{T}}(\mathbf{y} - \mathbf{Z}\mathbf{a}^{[t]})$$

$$\sigma_a^{2,[t]} = \frac{1}{n}\left[\mathbf{a}^{[t]}\mathbf{A}^{-1}\mathbf{a}^{[t]} + tr(\mathbf{A}^{-1}V_{\mathbf{a}}^{[t]})\right]$$

$$\sigma_\epsilon^{2,[t]} = -\frac{1}{n}\left[\mathbf{y} - \mathbf{x}\beta^{[t]} - \mathbf{Z}\mathbf{a}^{[t]}\right]^{\mathrm{T}}\left[\mathbf{y} - \mathbf{x}\beta^{[t]} - \mathbf{Z}\mathbf{a}^{[t]}\right] + tr(\mathbf{Z}^{\mathrm{T}}\mathbf{Z}V_{\mathbf{a}}^{[t]})$$

where tr is a trace function, which is equal to the sum of the diagonal elements of a matrix.

4. Iterate steps 2, 3 until $(\beta^{[t]}, \sigma_a^{2,[t]}, \sigma_\epsilon^{2,[t]}) \approx (\beta^{[t+1]}, \sigma_a^{2,[t+1]}, \sigma_\epsilon^{2,[t+1]})$ (or alternatively $lnL^{[t]} \approx lnL^{[t+1]}$).

The two values we need to expect, a and Va, look very similar. Therefore to make the code more efficient we can again make a base value that is then shared by the other two:

```
    S = ginv(t(Z) %*% Z + solve_A * sigma_sq_e/sigma_sq_a)
    a = S %*% t(Z) %*% (Y - X_j %*% beta)
    V = S * sigma_sq_e
```

Next we write out the maximization equations:

```
    beta = ginv(t(X_j) %*% X_j) %*% t(X_j) %*% (Y - Z %*% a)
    sigma_sq_a = as.numeric(1/n * (t(a) %*% solve_A %*% a + sum(diag(solve_A %*% V))))
    sigma_sq_e = as.numeric(1/n * (t(Y - X_j %*% beta - Z %*% a) %*% (Y - X_j %*% beta - Z %*% a) + sum
```

We also must calculate the likelihood again and make a comparison. If the comparison satisfies our exit threshold we use the break statement to exit the loop and return the result. With all of the equations now written out we can fill in the function and finish the EM algorithm:

```
EM_algorithm = function(Y, X_j, A, max.iter = 100) {
  #Initiate values
  #These values are "shortcuts" for future calculations
  solve_A = ginv(A)
  n = length(Y)
  Z = diag(1, n)
  log_L = c()

  #These are our random guesses of the maximized values
  sigma_sq_a = 70
  sigma_sq_e = 10
  beta = as.vector(rep(0, ncol(X_j)))
  iter = 2

  #Calculate initial likelihood
  C = A * sigma_sq_a + Z * sigma_sq_e #really means V
  log_L[1] = -1/2 * determinant(C)$modulus - 1/2 * t(Y - X_j %*% beta) %*% ginv(C) %*% (Y - X_j %*% beta

  while (iter < max.iter) {
    #Expect
    S = ginv(t(Z) %*% Z + solve_A * sigma_sq_e/sigma_sq_a)
    a = S %*% t(Z) %*% (Y - X_j %*% beta)
    V = S * sigma_sq_e

    #Maximize
    beta = ginv(t(X_j) %*% X_j) %*% t(X_j) %*% (Y - Z %*% a)
    sigma_sq_a = as.numeric(1/n * (t(a) %*% solve_A %*% a + sum(diag(solve_A %*% V))))
    sigma_sq_e = as.numeric(1/n * (t(Y - X_j %*% beta - Z %*% a) %*% (Y - X_j %*% beta - Z %*% a) + sum

    #Recalcuate log-likelihood then compare
    C = A * sigma_sq_a + Z * sigma_sq_e #really means V
    log_L[iter] = -1/2 * determinant(C)$modulus - 1/2 * t(Y - X_j %*% beta) %*% ginv(C) %*% (Y - X_j %*
    if (log_L[iter] - log_L[iter - 1] < 1e-05) { break }
    iter = iter + 1
  }
  return(list(beta = beta, sigma_sq_a = sigma_sq_a, sigma_sq_e = sigma_sq_e, log_L = log_L[iter - 1]))
}
```

The algorithm looks great to me! However, there may be an error waiting somewhere within. To test it out we will ready in some data and produce a Manhattan plot. Note that in this lab we do not only need X and Y, or genotypes and phenotypes, but also the A values.

```
library(MASS) # load MASS package to use the ginv() function
X = as.matrix(read.table("QG18_Lab12_EM_X.txt"))
Y = as.matrix(read.table("QG18_Lab12_EM_Y.txt"))
A = as.matrix(read.table("QG18_Lab12_EM_A.txt"))
```

We convert the output betas, likelihoods, and sigmas into a p-value by conducting a likelihood ratio test. This process is nearly identical to what we did last week within the IRLS algorithm. As a quick review, the null hypothesis value is calculated out front and then for each variant the EM algorithm is run. The likelihood of the null and variant are then formed into a likelihood test statistic, which is then coverted to a pvalue through the chi-square distribution.

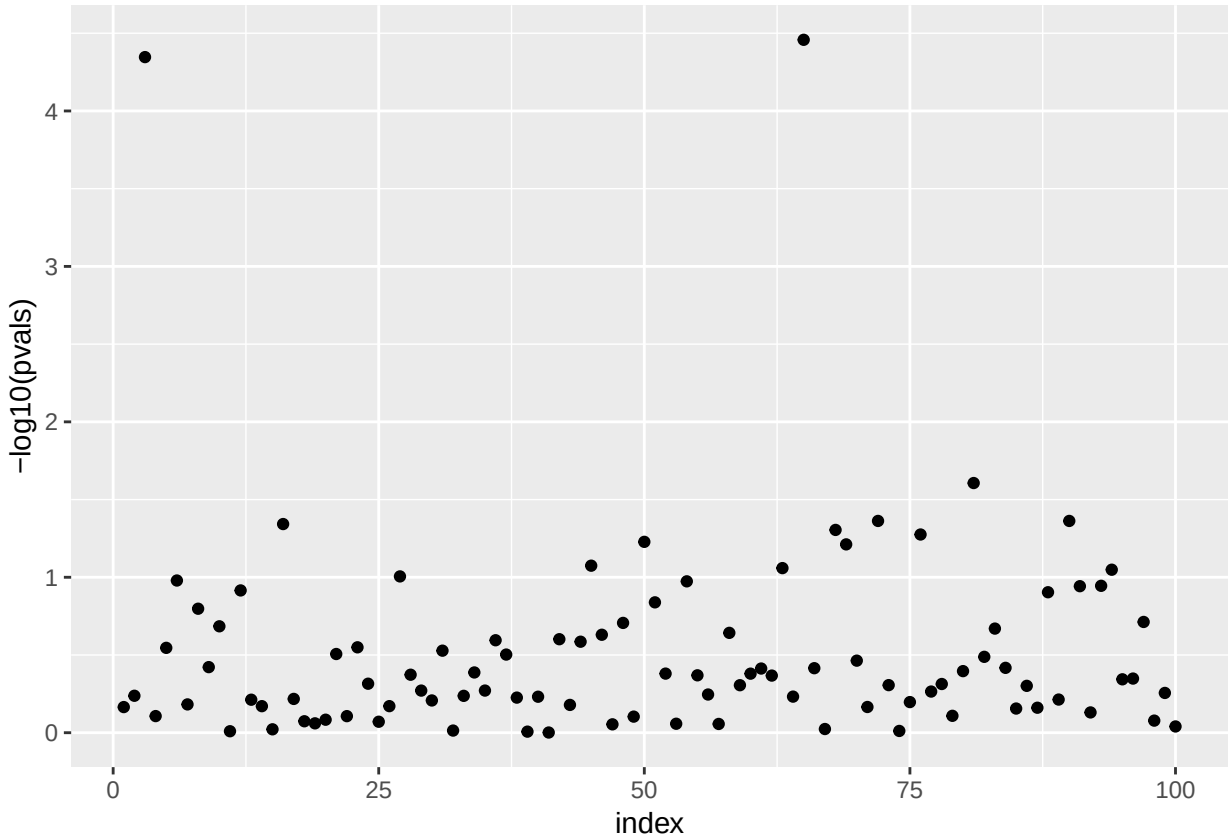
```
# Null model
n_indivs = length(Y)
One = as.matrix(rep(1, n_indivs))
log_L_null = EM_algorithm(Y, One, A)$log_L

# Full model
p_values_EM = c()
for (j in 1:ncol(X)) {
  X_j = cbind(1, X[, j])
  fit = EM_algorithm(Y, X_j, A)
  p_values_EM[j] = pchisq(-2 * (log_L_null - fit$log_L), 1, lower.tail = FALSE)
  cat(".")
}
```

```
## ..................................................................................................
```

Finally we plot the pvalues:

```
library(ggplot2)
pvalDf <- data.frame(index=1:length(p_values_EM),pvals=p_values_EM)
ggplot(pvalDf,aes(index,-log10(pvals)))+geom_point()
```



The pvalues were calculated and the Manhattan plot looks reasonable, with two significant variants.

To highlight the importance of predeclaring variables, or by making an intermediate variable that is then used multiple times later on, we can re-write the algorithm and take all of these variables out.

```
EM_algorithm_slow = function(Y, X_j, A, max.iter = 100) {
  #Initiate values
  #These values are "shortcuts" for future calculations
  n = length(Y)
  Z = diag(1, n)
  log_L = c()

  #These are our random guesses of the maximized values
  sigma_sq_a = 70
  sigma_sq_e = 10
  beta = as.vector(rep(0, ncol(X_j)))
  iter = 2

  #Calculate initial likelihood
  log_L[1] = -1/2 * determinant((A * sigma_sq_a + Z * sigma_sq_e))$modulus -
    1/2 * t(Y - X_j %*% beta) %*% ginv((A * sigma_sq_a + Z * sigma_sq_e)) %*% (Y - X_j %*% beta)

  while (iter < max.iter) {
    #Expect
    a = (ginv(t(Z) %*% Z + ginv(A) * sigma_sq_e/sigma_sq_a)) %*% t(Z) %*% (Y - X_j %*% beta)
    V = (ginv(t(Z) %*% Z + ginv(A) * sigma_sq_e/sigma_sq_a)) * sigma_sq_e

    #Maximize
    beta = ginv(t(X_j) %*% X_j) %*% t(X_j) %*% (Y - Z %*% a)
    sigma_sq_a = as.numeric(1/n * (t(a) %*% ginv(A) %*% a + sum(diag(ginv(A) %*% V))))
    sigma_sq_e = as.numeric(1/n * (t(Y - X_j %*% beta - Z %*% a) %*% (Y - X_j %*% beta - Z %*% a) + sum

    #Recalcuate log-likelihood then compare
    log_L[iter] = -1/2 * determinant((A * sigma_sq_a + Z * sigma_sq_e))$modulus -
      1/2 * t(Y - X_j %*% beta) %*% ginv((A * sigma_sq_a + Z * sigma_sq_e)) %*% (Y - X_j %*% beta)
    if (log_L[iter] - log_L[iter - 1] < 1e-05) { break }
    iter = iter + 1
  }
  return(list(beta = beta, sigma_sq_a = sigma_sq_a, sigma_sq_e = sigma_sq_e, log_L = log_L[iter - 1]))
}
```

Using our timing function from the previous lab we can now use both EM algorithms and check whether our intermediate variable strategy is actually the way to go.

```
# Null model
n_indivs = length(Y)
One = as.matrix(rep(1, n_indivs))
log_L_null = EM_algorithm(Y, One, A)$log_L

# Full model fast
start_time_fast <- Sys.time()
p_values_EM = c()
for (j in 1:ncol(X)) {
  X_j = cbind(1, X[, j])
  fit = EM_algorithm(Y, X_j, A)
  p_values_EM[j] = pchisq(-2 * (log_L_null - fit$log_L), 1, lower.tail = FALSE)
  cat(".")
```

```
}
```

```
## ..................................................................................................
```

```
end_time_fast <- Sys.time()
cat("\n")
```

```
# Full model slow
start_time_slow <- Sys.time()
p_values_EM_slow = c()
for (j in 1:ncol(X)) {
  X_j = cbind(1, X[, j])
  fit = EM_algorithm_slow(Y, X_j, A)
  p_values_EM_slow[j] = pchisq(-2 * (log_L_null - fit$log_L), 1, lower.tail = FALSE)
  cat("*")
}
```

```
## **************************************************************************************************
```

```
end_time_slow <- Sys.time()

cat("The fast time is:", end_time_fast - start_time_fast,"\n")
```

```
## The fast time is: 11.41751
```

```
cat("The end time is:", end_time_slow - start_time_slow)
```

```
## The end time is: 26.89799
```

It is! Using intermediate variables, especially in computationally expensive steps such as taking the inverse of a matrix, is a good coding practice for anything else you write.

5. Exercise

How does the mixed model association framework compare to the univariate association that we have been doing? To answer this question, please produce a Manhattan plot with the X and Y matricies alone using the lab7_pval_calculator, provided below. Assume that the X matrix provided is the Xa matrix.

```
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)
}
```