

Logit Models

Bill

2019-12-30

Contents

1	Basics	5
1.1	Logit	5
1.2	Probit	7
2	MLE	11
2.1	Basic idea of MLE	11
2.2	Coin flip example, probit, and logit	12
2.3	Further on logit	14
2.4	Fisher information	15
2.5	References	15
3	Linear Mixed Models	17
3.1	Calculate mean	17
3.2	Test the treatment effect	19
3.3	Another example	20
3.4	Full LMM model	22
3.5	Serial correlations in time and space	24
4	Basic Concepts	27
4.1	Score	27
4.2	Gradient, Jacobian, and Hessian	28
4.3	Canonical link function	28
4.4	Ordinary Least Squares (OLS)	29
4.5	Fisher Scoring	30

5	Generalized Linear Mixed Models	33
5.1	Computing techniques	33
5.2	Basics of GLMM	42
5.3	Some References	43
6	Twitter Example	45
6.1	Model	45
6.2	Simulating Data of Senators on Twitter	46
6.3	Simulating Data of Conservative Users on Twitter and Model Testing	47
6.4	Simulating Data of Liberal Users on Twitter and Model Testing .	49

Chapter 1

Basics

1.1 Logit

$$f(x) = \log\left(\frac{p(y=1)}{1-p(y=1)}\right)$$

The basic idea of logistic regression:

$$p(y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)}} = \frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}}$$

Thus, $e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}$ can be from $-\infty$ to $+\infty$, and $p(y=1)$ will be always within the range of $(0, 1)$.

```
f<-function(x){exp(x)/(1+exp(x))}  
data<-seq(-10,10,1)  
plot(data,f(data),type = "b")
```



We can also write the function into another format as follows:

$$\log \frac{p(y=1)}{1-p(y=1)} = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n$$

Thus, we know that the regression coefficients of β_i actually change the “log-odds” of the event. Of course, note that the magnitude of β_i is dependent upon the units of x_i .

The following is an example testing whether that home teams are more likely to win in NFL games. The results show that the odd of winning is the same for both home and away teams.

```
mydata = read.csv(url('https://raw.githubusercontent.com/nfl-football-ops/Big-Data-Bow
mydata$result_new<-ifelse(mydata$HomeScore>mydata$VisitorScore,1,0)
summary(mydata$result_new)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.0000 0.0000 0.0000 0.4945 1.0000 1.0000
```

```
mylogit1 = glm(result_new~1, family=binomial, data=mydata)
summary(mylogit1)
```

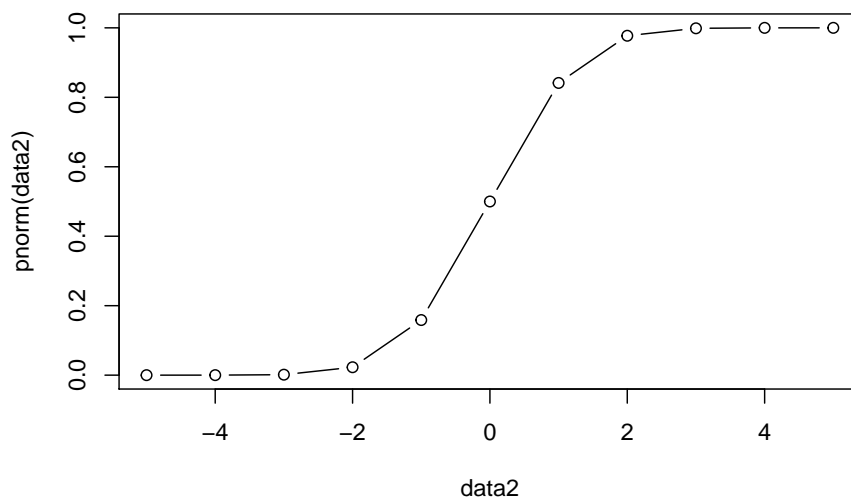
```
##
## Call:
```

```
## glm(formula = result_new ~ 1, family = binomial, data = mydata)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.168  -1.168  -1.168   1.187   1.187
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.02198    0.20967  -0.105   0.917
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 126.14  on 90  degrees of freedom
## Residual deviance: 126.14  on 90  degrees of freedom
## AIC: 128.14
##
## Number of Fisher Scoring iterations: 3
```

1.2 Probit

As noted above, logit $f(x) = \log\left(\frac{p(y=1)}{1-p(y=1)}\right)$ provides the resulting range of $(0, 1)$. Another way to provide the same range is through the cdf of normal distribution. The following R code is used to illustrate this process.

```
data2<-seq(-5,5,1)
plot(data2,pnorm(data2),type = "b")
```



Thus, the cdf of normal distribution can be used to indicate the probability of $p(y = 1)$.

$$\Phi(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n) = p(y = 1)$$

Similar to logit model, we can also write the inverse function of the cdf to get the function that can be from $-\infty$ to $+\infty$.

$$\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n = \Phi^{-1}(p(y = 1))$$

Thus, for example, if $X\beta = -2$, based on $\Phi(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n) = p(y = 1)$ we can get that the $p(y = 1) = 0.023$.

In contrast, if $X\beta = 3$, the $p(y = 1) = 0.999$.

```
pnorm(-2)
```

```
## [1] 0.02275013
```

```
pnorm(3)
```

```
## [1] 0.9986501
```


Let's assume that there is a latent variable called Y^* such that

$$Y^* = X\beta + \epsilon, \epsilon \sim N(0, \sigma^2)$$

You could think of Y^* as a kind of “proxy” between $X\beta + \epsilon$ and the observed $Y(1 \text{ or } 0)$. Thus, we can get the following. Note that, it does not have to be zero, and can be any constant.

$$Y^* = \begin{cases} 0 & \text{if } y_i^* \leq 0 \\ 1 & \text{if } y_i^* > 0 \end{cases}$$

Thus,

$$y_i^* > 0 \Rightarrow \beta' X_i + \epsilon_i > 0 \Rightarrow \epsilon_i > -\beta' X_i$$

Thus, we can write it as follows. Note that $\frac{\epsilon_i}{\sigma} \sim N(0, 1)$

$$p(y = 1|x_i) = p(y_i^* > 0|x_i) = p(\epsilon_i > -\beta' X_i) = p\left(\frac{\epsilon_i}{\sigma} > \frac{-\beta' X_i}{\sigma}\right) = \Phi\left(\frac{\beta' X_i}{\sigma}\right)$$

We thus can get:

$$p(y = 0|x_i) = 1 - \Phi\left(\frac{\beta' X_i}{\sigma}\right)$$

For $p(y = 1|x_i) = \Phi\left(\frac{\beta' X_i}{\sigma}\right)$, we can not really estimate both β and σ as they are in a ratio. We can assume $\sigma = 1$, then $\epsilon \sim N(0, 1)$. We know y_i and x_i since we observe them. Thus, we can write it as follows.

$$p(y = 1|x_i) = \Phi(\beta' X_i)$$

Chapter 2

MLE

2.1 Basic idea of MLE

Suppose that we flip a coin, $y_i = 0$ for tails and $y_i = 1$ for heads. If we get p heads from n trials, we can get the proportion of heads is p/n , which is the sample mean. If we do not do any further calculation, this is our best guess.

Suppose that the true probability is ρ , then we can get:

$$\mathbf{L}(y_i) = \begin{cases} \rho & y_i = 1 \\ 1 - \rho & y_i = 0 \end{cases}$$

Thus, we can also write it as follows.

$$\mathbf{L}(y_i) = \rho^{y_i} (1 - \rho)^{1-y_i}$$

Thus, we can get:

$$\prod \mathbf{L}(y_i|\rho) = \rho^{\sum y_i} (1 - \rho)^{\sum (1-y_i)}$$

Further, we can get a log-transformed format.

$$\log(\prod \mathbf{L}(y_i|\rho)) = \sum y_i \log \rho + \sum (1 - y_i) \log(1 - \rho)$$

To maximize the log-function above, we can calculate the derivative with respect to ρ .

$$\frac{\partial \log(\prod \mathbf{L}(y_i|\rho))}{\partial \rho} = \sum y_i \frac{1}{\rho} - \sum (1 - y_i) \frac{1}{1 - \rho}$$

Set the derivative to zero and solve for ρ , we can get

$$\begin{aligned}
& \sum y_i \frac{1}{\rho} - \sum (1 - y_i) \frac{1}{1 - \rho} = 0 \\
& \Rightarrow (1 - \rho) \sum y_i - \rho \sum (1 - y_i) = 0 \\
& \Rightarrow \sum y_i - \rho \sum y_i - n\rho + \rho \sum y_i = 0 \\
& \Rightarrow \sum y_i - n\rho = 0 \\
& \Rightarrow \rho = \frac{\sum y_i}{n} = \frac{p}{n}
\end{aligned}$$

Thus, we can see that the ρ maximizing the likelihood function is equal to the sample mean.

2.2 Coin flip example, probit, and logit

In the example above, we are not really trying to estimate a lot of regression coefficients. What we are doing actually is to calculate the sample mean, or intercept in the regression sense. What does it mean? Let's use some data to explain it.

Suppose that we flip a coin 20 times and observe 8 heads. We can use the R's `glm` function to estimate the ρ . If the result is consistent with what we did above, we should observe that the *cdf* of the estimate of β_0 (i.e., intercept) should be equal to $8/20 = 0.4$.

```
coins<-c(rep(1,times=8),rep(0,times=12))
table(coins)
```

```
## coins
##  0  1
## 12  8
```

```
coins<-as.data.frame(coins)
```

2.2.1 Probit

```
probitresults <- glm(coins ~ 1, family = binomial(link = "probit"), data = coins)
probitresults
```

```
##
## Call:  glm(formula = coins ~ 1, family = binomial(link = "probit"),
##       data = coins)
##
## Coefficients:
## (Intercept)
##      -0.2533
##
## Degrees of Freedom: 19 Total (i.e. Null);  19 Residual
## Null Deviance:      26.92
## Residual Deviance: 26.92    AIC: 28.92
```

```
pnorm(probitresults$coefficients)
```

```
## (Intercept)
##           0.4
```

As we can see the intercept is -0.2533 , and thus $\Phi(-0.2533471) = 0.4$

2.2.2 Logit

We can also use logit link to calculate the intercept as well. Recall that

$$p(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)}} = \frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}}$$

Thus,

$$p(y = 1) = \frac{e^{\beta_0}}{1 + e^{\beta_0}}$$

```
logitresults <- glm(coins ~ 1, family = binomial(link = "logit"), data = coins)
logitresults$coefficients
```

```
## (Intercept)
##      -0.4054651
```

```
exp(logitresults$coefficients)/(1+exp(logitresults$coefficients))
```

```
## (Intercept)
##           0.4
```

Note that, the default link for the binomial in the glm function is logit.

2.3 Further on logit

The probability of $y = 1$ is as follows:

$$p = p(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)}} = \frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}}$$

Thus, the likelihood function is as follows:

$$\begin{aligned} L &= \prod p^{y_i} (1-p)^{1-y_i} = \prod \left(\frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)}} \right)^{y_i} \left(\frac{1}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}} \right)^{1-y_i} \\ &= \prod (1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)})^{-y_i} (1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n})^{-(1-y_i)} \end{aligned}$$

Thus, the log-likelihood is as follows:

$$\log L = \sum (-y_i \cdot \log(1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)}) - (1 - y_i) \cdot \log(1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}))$$

Typically, optimisers minimize a function, so we use negative log-likelihood as minimising that is equivalent to maximising the log-likelihood or the likelihood itself.

#Source of R code: <https://www.r-bloggers.com/logistic-regression/>

```
mle.logreg = function(fmla, data)
{
  # Define the negative log likelihood function
  logl <- function(theta,x,y){
    y <- y
    x <- as.matrix(x)
    beta <- theta[1:ncol(x)]

    # Use the log-likelihood of the Bernoulli distribution, where p is
    # defined as the logistic transformation of a linear combination
    # of predictors, according to logit(p)=(x%*%beta)
    loglik <- sum(-y*log(1 + exp(-(x%*%beta))) - (1-y)*log(1 + exp(x%*%beta)))
    return(-loglik)
  }

  # Prepare the data
  outcome = rownames(attr(terms(fmla),"factors"))[1]
  dfrTmp = model.frame(data)
  x = as.matrix(model.matrix(fmla, data=dfrTmp))
}
```

```

y = as.numeric(as.matrix(data[,match(outcome,colnames(data))]))

# Define initial values for the parameters
theta.start = rep(0,(dim(x)[2]))
names(theta.start) = colnames(x)

# Calculate the maximum likelihood
mle = optim(theta.start,logl,x=x,y=y, method = 'BFGS', hessian=T)
out = list(beta=mle$par,vcov=solve(mle$hessian),ll=2*mle$value)
}

mydata = read.csv(url('https://stats.idre.ucla.edu/stat/data/binary.csv'))
mylogit1 = glm(admit~gre+gpa+as.factor(rank), family=binomial, data=mydata)

mydata$rank = factor(mydata$rank) #Treat rank as a categorical variable
fmla = as.formula("admit~gre+gpa+rank") #Create model formula
mylogit2 = mle.logreg(fmla, mydata) #Estimate coefficients

print(cbind(coef(mylogit1), mylogit2$beta))

```

```

##                [,1]      [,2]
## (Intercept)    -3.989979073 -3.772676422
## gre            0.002264426  0.001375522
## gpa            0.804037549  0.898201239
## as.factor(rank)2 -0.675442928 -0.675543009
## as.factor(rank)3 -1.340203916 -1.356554831
## as.factor(rank)4 -1.551463677 -1.563396035

```

2.4 Fisher information

https://people.missouristate.edu/songfengzheng/Teaching/MTH541/Lecture%20notes/Fisher_info.pdf

2.5 References

http://www.columbia.edu/~so33/SusDev/Lecture_9.pdf

Chapter 3

Linear Mixed Models

The following is a shortened version of Jonathan Rosenblatt's LMM tutorial.
<http://www.john-ros.com/Rcourse/lme.html>.

In addition, another reference is from Douglas Bates's R package document.
https://cran.r-project.org/web/packages/lme4/vignettes/lmer.pdf?fbclid=IwAR1nmmRP9A0BrhKdgBibNjM5acR_spTpXV8QlQGdmTWyQz3ZtV3LYn6kCbQ

Assume that y is a function of x and u , where x is the fixed effect and u is the random effect. Thus, we can get,

$$y|x, u = x'\beta + z'u + \epsilon$$

For random effect, one example can be that you want to test the treatment effect, and sample 8 observations from 4 groups. You measure before and after the treatment. In this case, x represents the treatment effect, whereas z represents the group effect (i.e., random effect). Note that, in this case, it reminds the paired t-test. Remember in SPSS, why do we do paired t-test? Typically, it is the case when we measure a subject (or, participant) twice. In this case, we can consider each participant as an unit of random effect (rather than as group in the last example.)

3.1 Calculate mean

The following code generates 4 numbers ($N(0, 10)$) for 4 groups. Then, replicate it within each group. That is, in the end, there are 8 observations.

Note that, in the following code, there are no “independent variables”. Both the linear model and mixed model are actually just trying to calculate the mean. Note that `lmer(y~1+1|groups)` and `lmer(y~1|groups)` will generate the same results.

```

set.seed(123)
n.groups <- 4 # number of groups
n.repeats <- 2 # samples per group
#Generating index for observations belong to the same group
groups <- as.factor(rep(1:n.groups, each=n.repeats))
n <- length(groups)
#Generating 4 random numbers, assuming normal distribution
z0 <- rnorm(n.groups, 0, 10)
z <- z0[as.numeric(groups)] # generate and inspect random group effects
z

```

```

## [1] -5.6047565 -5.6047565 -2.3017749 -2.3017749 15.5870831 15.5870831 0.7050839
## [8] 0.7050839

```

```

epsilon <- rnorm(n,0,1) # generate measurement error
beta0 <- 2 # this is the actual parameter of interest! The global mean.
y <- beta0 + z + epsilon # sample from an LMM

# fit a linear model assuming independence
# i.e., assume that there is no "group things".
lm.5 <- lm(y~1)

# fit a mixed-model that deals with the group dependence
#install.packages("lme4")
library(lme4)
lme.5.a <- lmer(y~1+1|groups)
lme.5.b <- lmer(y~1|groups)
lm.5

```

```

##
## Call:
## lm(formula = y ~ 1)
##
## Coefficients:
## (Intercept)
##          4.283

```

```
lme.5.a
```

```

## Linear mixed model fit by REML ['lmerMod']
## Formula: y ~ 1 + 1 | groups
## REML criterion at convergence: 36.1666
## Random effects:
## Groups      Name          Std.Dev.

```

```
## groups (Intercept) 8.8521
## Residual          0.8873
## Number of obs: 8, groups: groups, 4
## Fixed Effects:
## (Intercept)
##          4.283
```

```
lme.5.b
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: y ~ 1 | groups
## REML criterion at convergence: 36.1666
## Random effects:
## Groups Name Std.Dev.
## groups (Intercept) 8.8521
## Residual          0.8873
## Number of obs: 8, groups: groups, 4
## Fixed Effects:
## (Intercept)
##          4.283
```

3.2 Test the treatment effect

As we can see that, LLM and paired t-test generate the same t-value.

```
times<-rep(c(1,2),4) # first time and second time
times
```

```
## [1] 1 2 1 2 1 2 1 2
```

```
data_combined<-cbind(y,groups,times)
data_combined
```

```
##           y groups times
## [1,] -3.4754687      1      1
## [2,] -1.8896915      1      2
## [3,]  0.1591413      2      1
## [4,] -1.5668361      2      2
## [5,] 16.9002303      3      1
## [6,] 17.1414212      3      2
## [7,]  3.9291657      4      1
## [8,]  3.0648977      4      2
```

```

lme_diff_times<- lmer(y~times+(1|groups))

t_results<-t.test(y~times, paired=TRUE)

lme_diff_times

## Linear mixed model fit by REML ['lmerMod']
## Formula: y ~ times + (1 | groups)
## REML criterion at convergence: 35.0539
## Random effects:
##   Groups   Name                Std.Dev.
##   groups   (Intercept)  8.845
##   Residual                                1.013
## Number of obs: 8, groups:  groups, 4
## Fixed Effects:
##   (Intercept)            times
##           4.5691           -0.1908

print("The following results are from paired t-test")

## [1] "The following results are from paired t-test"

t_results$statistic

##           t
## 0.2664793

```

3.3 Another example

```

data(Dyestuff, package='lme4')
attach(Dyestuff)

## The following objects are masked from Dyestuff (pos = 6):
##
##   Batch, Yield

Dyestuff

```

```
##      Batch Yield
## 1      A  1545
## 2      A  1440
## 3      A  1440
## 4      A  1520
## 5      A  1580
## 6      B  1540
## 7      B  1555
## 8      B  1490
## 9      B  1560
## 10     B  1495
## 11     C  1595
## 12     C  1550
## 13     C  1605
## 14     C  1510
## 15     C  1560
## 16     D  1445
## 17     D  1440
## 18     D  1595
## 19     D  1465
## 20     D  1545
## 21     E  1595
## 22     E  1630
## 23     E  1515
## 24     E  1635
## 25     E  1625
## 26     F  1520
## 27     F  1455
## 28     F  1450
## 29     F  1480
## 30     F  1445
```

```
lme_batch<- lmer( Yield ~ 1 + (1|Batch) , Dyestuff )
summary(lme_batch)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: Yield ~ 1 + (1 | Batch)
##      Data: Dyestuff
##
## REML criterion at convergence: 319.7
##
## Scaled residuals:
##      Min      1Q  Median      3Q      Max
## -1.4117 -0.7634  0.1418  0.7792  1.8296
##
```

```
## Random effects:
## Groups   Name                Variance Std.Dev.
## Batch    (Intercept) 1764      42.00
## Residual                2451      49.51
## Number of obs: 30, groups:  Batch, 6
##
## Fixed effects:
##              Estimate Std. Error t value
## (Intercept) 1527.50      19.38      78.8
```

3.4 Full LMM model

In the following, I used the data from the package of lme4. For Days + (1 | Subject), it only has random intercept; in contrast, Days + (Days| Subject) has both random intercept and random slope for Days. Note that, random effects do not generate specific slopes for each level of Days, but rather just a variance of all the slopes.

Therefore, we can see that “Days + (Days| Subject)” and “Days + (1+Days| Subject)” generate the same results. For more discussion, you can refer to the following link: <https://www.jaredknowles.com/journal/2013/11/25/getting-started-with-mixed-effect-models-in-r>

```
data(sleepstudy, package='lme4')
attach(sleepstudy)
```

```
## The following objects are masked from sleepstudy (pos = 6):
##
##      Days, Reaction, Subject
```

```
fm1 <- lmer(Reaction ~ Days + (1 | Subject), sleepstudy)
summary(fm1)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: Reaction ~ Days + (1 | Subject)
##      Data: sleepstudy
##
## REML criterion at convergence: 1786.5
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -3.2257 -0.5529  0.0109  0.5188  4.2506
##
```

```
## Random effects:
##   Groups   Name                Variance Std.Dev.
##   Subject (Intercept) 1378.2    37.12
##   Residual                960.5    30.99
## Number of obs: 180, groups:  Subject, 18
##
## Fixed effects:
##               Estimate Std. Error t value
## (Intercept) 251.4051    9.7467    25.79
## Days        10.4673     0.8042    13.02
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.371
```

```
fm2<-lmer ( Reaction ~ Days + ( Days | Subject ) , data= sleepstudy )
summary(fm2)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: Reaction ~ Days + (Days | Subject)
##   Data: sleepstudy
##
## REML criterion at convergence: 1743.6
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -3.9536 -0.4634  0.0231  0.4633  5.1793
##
## Random effects:
##   Groups   Name                Variance Std.Dev. Corr
##   Subject (Intercept) 611.90    24.737
##           Days        35.08     5.923   0.07
##   Residual                654.94    25.592
## Number of obs: 180, groups:  Subject, 18
##
## Fixed effects:
##               Estimate Std. Error t value
## (Intercept) 251.405    6.824    36.843
## Days        10.467     1.546     6.771
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.138
```

```
fm3<-lmer ( Reaction ~ Days + (1+Days| Subject ) , data= sleepstudy )
summary(fm3)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: Reaction ~ Days + (1 + Days | Subject)
## Data: sleepstudy
##
## REML criterion at convergence: 1743.6
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -3.9536 -0.4634  0.0231  0.4633  5.1793
##
## Random effects:
## Groups Name Variance Std.Dev. Corr
## Subject (Intercept) 611.90 24.737
## Days 35.08 5.923 0.07
## Residual 654.94 25.592
## Number of obs: 180, groups: Subject, 18
##
## Fixed effects:
## Estimate Std. Error t value
## (Intercept) 251.405 6.824 36.843
## Days 10.467 1.546 6.771
##
## Correlation of Fixed Effects:
## (Intr)
## Days -0.138
```

3.5 Serial correlations in time and space

The hierarchical model of $y|x, u = x'\beta + z'u + \epsilon$ can work well for correlations within blocks, but not for correlations in time as the correlations decay in time. The following uses nlme package to calculate time serial data.

```
library(nlme)
head(nlme::Ovary,n=50)
```

```
## Grouped Data: follicles ~ Time | Mare
## Mare Time follicles
## 1 1 -0.13636360 20
## 2 1 -0.09090910 15
## 3 1 -0.04545455 19
```


## 4	1	0.00000000	16
## 5	1	0.04545455	13
## 6	1	0.09090910	10
## 7	1	0.13636360	12
## 8	1	0.18181820	14
## 9	1	0.22727270	13
## 10	1	0.27272730	20
## 11	1	0.31818180	22
## 12	1	0.36363640	15
## 13	1	0.40909090	18
## 14	1	0.45454550	17
## 15	1	0.50000000	14
## 16	1	0.54545450	18
## 17	1	0.59090910	14
## 18	1	0.63636360	16
## 19	1	0.68181820	17
## 20	1	0.72727270	18
## 21	1	0.77272730	18
## 22	1	0.81818180	17
## 23	1	0.86363640	14
## 24	1	0.90909090	12
## 25	1	0.95454550	12
## 26	1	1.00000000	14
## 27	1	1.04545500	10
## 28	1	1.09090900	11
## 29	1	1.13636400	16
## 30	2	-0.15000000	6
## 31	2	-0.10000000	6
## 32	2	-0.05000000	8
## 33	2	0.00000000	7
## 34	2	0.05000000	16
## 35	2	0.10000000	10
## 36	2	0.15000000	13
## 37	2	0.20000000	9
## 38	2	0.25000000	7
## 39	2	0.30000000	6
## 40	2	0.35000000	8
## 41	2	0.40000000	8
## 42	2	0.45000000	6
## 43	2	0.50000000	8
## 44	2	0.55000000	7
## 45	2	0.60000000	9
## 46	2	0.65000000	6
## 47	2	0.70000000	4
## 48	2	0.75000000	5
## 49	2	0.80000000	8

```
## 50      2  0.85000000      11
```

```
fm10var.lme <- nlme::lme(fixed=follicles ~ sin(2*pi*Time) + cos(2*pi*Time),
                        data = Ovary,
                        random = pdDiag(~sin(2*pi*Time)),
                        correlation=corAR1() )
summary(fm10var.lme)
```

```
## Linear mixed-effects model fit by REML
## Data: Ovary
##      AIC      BIC    logLik
##  1563.448 1589.49 -774.724
##
## Random effects:
## Formula: ~sin(2 * pi * Time) | Mare
## Structure: Diagonal
##      (Intercept) sin(2 * pi * Time) Residual
## StdDev:      2.858385      1.257977 3.507053
##
## Correlation Structure: AR(1)
## Formula: ~1 | Mare
## Parameter estimate(s):
##      Phi
## 0.5721866
## Fixed effects: follicles ~ sin(2 * pi * Time) + cos(2 * pi * Time)
##      Value Std.Error DF   t-value p-value
## (Intercept)  12.188089 0.9436602 295 12.915760 0.0000
## sin(2 * pi * Time) -2.985297 0.6055968 295 -4.929513 0.0000
## cos(2 * pi * Time) -0.877762 0.4777821 295 -1.837159 0.0672
## Correlation:
##      (Intr) s(*p*T
## sin(2 * pi * Time) 0.000
## cos(2 * pi * Time) -0.123 0.000
##
## Standardized Within-Group Residuals:
##      Min      Q1      Med      Q3      Max
## -2.34910093 -0.58969626 -0.04577893 0.52931186 3.37167486
##
## Number of Observations: 308
## Number of Groups: 11
```

Chapter 4

Basic Concepts

4.1 Score

The score is the gradient (the vector of partial derivatives) of $\log L(\theta)$, with respect to an m -dimensional parameter vector θ .

$$S(\theta) = \frac{\partial \ell}{\partial \theta}$$

Typically, they use ∇ to note the partial derivative.

$$\nabla \ell$$

Such differentiation will generate a $m \times 1$ row vector, which indicates the sensitivity of the likelihood.

Quote from Steffen Lauritzen's slides: "Generally the solution to this equation must be calculated by iterative methods. One of the most common methods is the Newton–Raphson method and this is based on successive approximations to the solution, using Taylor's theorem to approximate the equation."

For instance, using logit link, we can get the first derivative of log likelihood logistic regression as follows. We can not really find β easily to make the equation to be 0.

$$\begin{aligned} \frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^n x_i^T \left[y_i - \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \right] \\ &= \sum_{i=1}^n x_i^T [y_i - \hat{y}_i] \end{aligned}$$

4.2 Gradient, Jacobian, and Hessian

4.3 Canonical link function

Inspired by a Stack Exchange post, I created the following figure:

$$\frac{\text{Parameter}}{\theta} \rightarrow \gamma'(\theta) = \mu \rightarrow \frac{\text{Mean}}{\mu} \rightarrow g(\mu) = \eta \rightarrow \frac{\text{Linear predictor}}{\eta}$$

For the case of n time Bernoulli (i.e., Binomial), its canonical link function is logit. Specifically,

$$\frac{\text{Parameter}}{\theta = \beta^T x_i} \rightarrow \gamma'(\theta) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \rightarrow \frac{\text{Mean}}{\mu = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}}} \rightarrow g(\mu) = \log \frac{\frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}}}{1 - \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}}} \rightarrow \frac{\text{Linear predictor}}{\eta = \beta^T x_i}$$

Thus, we can see that,

$$\theta \equiv \eta$$

The link function $g(\mu)$ relates the linear predictor $\eta = \beta^T x_i$ to the mean μ .

Remarks:

- (1) Parameter is $\theta = \beta^T x_i$ (Not μ !).
- (2) $\mu = p(y = 1) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}}$ (NOT logit!).
- (3) Link function (i.e., $g(\mu)$) = logit = logarithm of odds = $\log \frac{\text{Event-Happened}}{\text{Event-Not-Happened}}$.
- (4) $g(\mu) = \log \frac{\mu}{1-\mu} = \beta^T x_i$. Thus, link function = linear predictor = log odds!
- (5) Quote from the Stack Exchange post “Newton Method and Fisher scoring for finding the ML estimator coincide, these links simplify the derivation of the MLE.”

(Recall, we know that μ or $p(y = 1)$ is the mean function. Recall that, n trials of coin flips, and get p heads. Thus $\mu = \frac{p}{n}$.)

4.4 Ordinary Least Squares (OLS)

Suppose we have n observation, and m variables.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ \dots & & & & \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix}$$

Thus, we can write it as the following n equations.

$$y_1 = \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \dots + \beta_m x_{1m}$$

$$y_2 = \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_m x_{2m}$$

$$y_3 = \beta_0 + \beta_1 x_{31} + \beta_2 x_{32} + \dots + \beta_m x_{3m}$$

...

$$y_n = \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + \dots + \beta_m x_{nm}$$

We can combine all the n equations as the following one:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_m x_{im} (i \in [1, n])$$

We can further rewrite it as a matrix format as follows.

$$y = X\beta$$

Where,

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \dots \\ y_n \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ 1 & x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ \dots & & & & & \\ 1 & x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix}$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \dots \\ \beta_m \end{bmatrix}$$

Since later we need the inverse of X , we need to make it into a square matrix.

$$X^T y = X^T X \hat{\beta} \Rightarrow \hat{\beta} = (X^T X)^{-1} X^T y$$

We can use R to implement this calculation. As we can see, there is no need to do any iterations at all, but rather just pure matrix calculation.

```
X<-matrix(rnorm(1000),ncol=2) # we define a 2 column matrix, with 500 rows
X<-cbind(1,X) # add a 1 constant
beta_true<-c(2,1,2) # True regression coefficients
beta_true<-as.matrix(beta_true)
y=X%%beta_true+rnorm(500)

transposed_X<-t(X)
beta_hat<-solve(transposed_X%%X)%%transposed_X%%y
beta_hat
```

```
##           [,1]
## [1,] 2.017690
## [2,] 1.054682
## [3,] 2.037671
```

Side Notes The function of `as.matrix` will automatically make `c(2,1,2)` become the dimension of 3×1 , you do not need to transpose the β .

4.5 Fisher Scoring

<https://stats.stackexchange.com/questions/176351/implement-fisher-scoring-for-linear-regression>

References:

1. Steffen Lauritzen's slides:

<http://www.stats.ox.ac.uk/~steffen/teaching/bs2HT9/scoring.pdf>

2. The Stack Exchange post:

<https://stats.stackexchange.com/questions/40876/what-is-the-difference-between-a-link-function-and-a-canonical-link-function>

3. Wikipedia for OLS

https://en.wikipedia.org/wiki/Ordinary_least_squares

Chapter 5

Generalized Linear Mixed Models

5.1 Computing techniques

Since GLMM can use EM algorithm in its maximum likelihood calculation (see McCulloch, 1994), it is practically useful to rehearse EM and other computing techniques.

5.1.1 Monte carlo approximation

Example: calculate the integral of $p(z > 2)$ when $z \sim N(0, 1)$. To use Monte Carlo approximation, we can have an indicator function, which will determine whether the sample from $N(0, 1)$ will be included into the calculation of the integral.

```
Nsim=10^4

indicator=function(x){
  y=ifelse((x>2),1,0)
  return(y)}

newdata<-rnorm(Nsim, 0,1 )

mc=c(); v=c(); upper=c(); lower=c()

for (j in 1:Nsim)
{
```

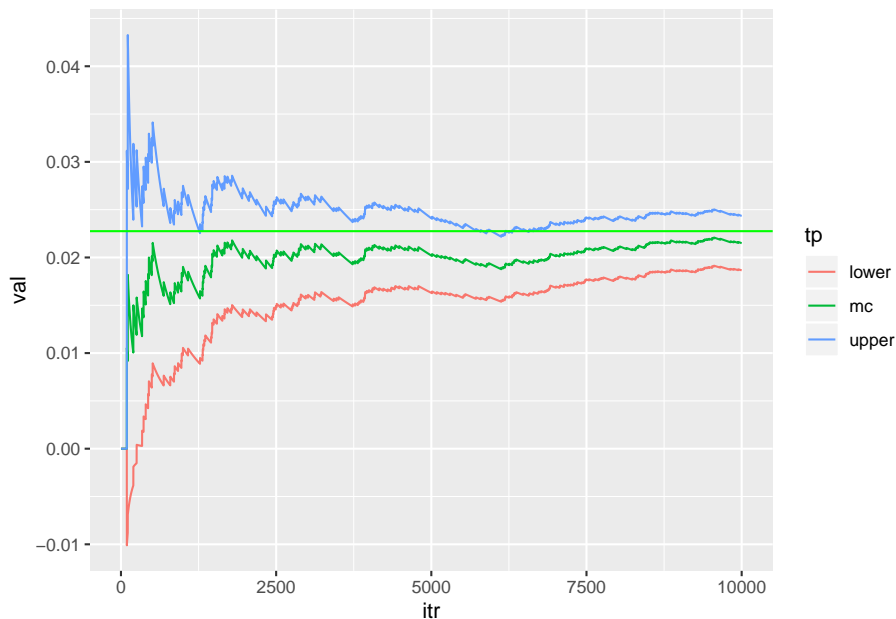
```

mc[j]=mean(indicator(newdata[1:j]))
v[j]=(j^{-1})*var(indicator(newdata[1:j]))
upper[j]=mc[j]+1.96*sqrt(v[j])
lower[j]=mc[j]-1.96*sqrt(v[j])
}

library(ggplot2)
values=c(mc,upper,lower)
type=c(rep("mc",Nsim),rep("upper",Nsim),rep("lower",Nsim))
itr=rep(seq(1:Nsim),3)
data=data.frame(val=values, tp=type, itr=itr)
Rcode<-ggplot(data,aes(itr,val,col=tp))+geom_line(size=0.5)
Rcode+geom_hline(yintercept=1-pnorm(2),color="green",size=0.5)

```

Warning: Removed 2 rows containing missing values (geom_path).



5.1.2 Importance sampling

Importance sampling has samples generated from a different distribution than the distribution of interest. Specifically, assume that we want to calculate the expected value of $h(x)$, and $x \sim f(x)$.

$$E(h(x)) = \int h(x)f(x)dx = \int h(x)\frac{f(x)}{g(x)}g(x)dx$$

We can sample x_i from $g(x)$ and then calculate the mean of $h(x_i)\frac{f(x_i)}{g(x_i)}$.

Using the same explain above, we can use a shifted exponential distribution to help calculate the integral for normal distribution. Specifically,

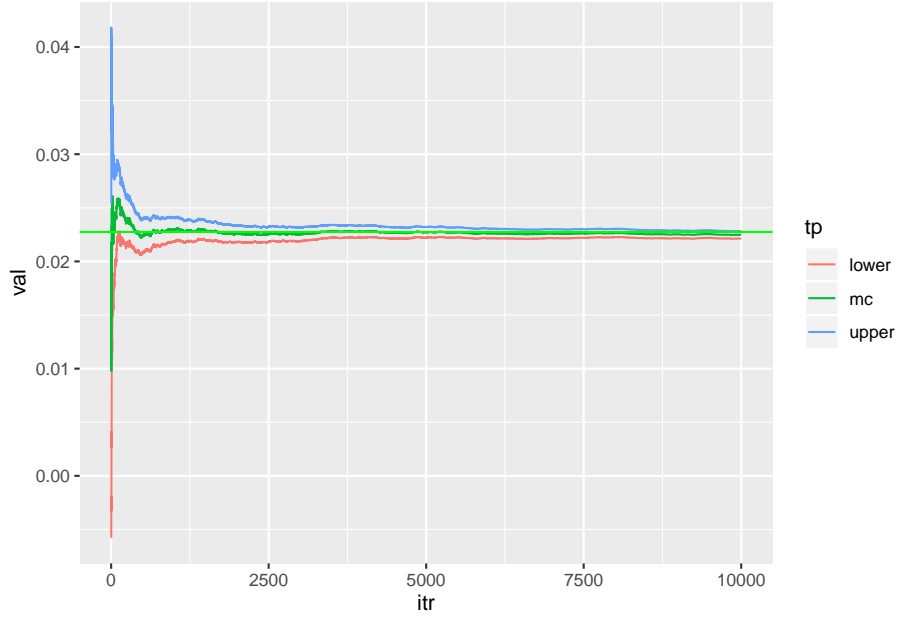
$$\int_2^{\infty} \frac{1}{2\pi} e^{-\frac{1}{2}x^2} dx = \int_2^{\infty} \frac{\frac{1}{2\pi} e^{-\frac{1}{2}x^2}}{e^{-(x-2)}} e^{-(x-2)} dx$$

The idea is that, we can generate x_i from exponential distribution of $e^{-(x-2)}$, and then insert them into the targeted “expected (value) function” of $\frac{\frac{1}{2\pi} e^{-\frac{1}{2}x^2}}{e^{-(x-2)}}$. Thus, as you can see, importance sampling is based on the law of large numbers (i.e., If the same experiment or study is repeated independently a large number of times, the average of the results of the trials must be close to the expected value). We can use it to calculate integral based on link of the definition of expected value.

```
Nsim=10^4
normal_density=function(x)
{y=(1/sqrt(2*pi))*exp(-0.5*(x^2))
return(y)}
x=2-log(runif(Nsim))
ImpS=c(); v=c(); upper=c(); lower=c()
for (j in 1:Nsim)
{
ImpS[j]=mean(normal_density(x[1:j])/exp(-(x[1:j]-2)))
v[j]=(j^-1)*var(normal_density(x[1:j])/exp(-(x[1:j]-2)))
upper[j]=ImpS[j]+1.96*sqrt(v[j])
lower[j]=ImpS[j]-1.96*sqrt(v[j])
}

library(ggplot2)
values=c(ImpS,upper,lower)
type=c(rep("mc",Nsim),rep("upper",Nsim),rep("lower",Nsim))
iter=rep(seq(1:Nsim),3)
data=data.frame(val=values, tp=type, itr=iter)
ggplot(data,aes(itr,val,col=tp))+geom_line(size=0.5)+
geom_hline(yintercept=1-pnorm(2),color="green",size=0.5)
```

```
## Warning: Removed 2 rows containing missing values (geom_path).
```



5.1.3 Newton Raphson algorithm

For nonlinear functions, it is sometimes difficult to calculate MLEs. Newton Raphson algorithm is an iterative procedure to calculate MLEs.

The basic idea of Newton Raphson is to find a approximate function that can be easily maximized analytically. We need some theoretical background from Taylor's Theorem.

If f has $k + 1$ times differentiable on an open interval I . For any x and $x + h$ in I , there is a point of w between x and $x + h$ where we can get the following:

$$f(x + h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \dots + \frac{1}{k!}f^{[k]}(x)h^k + \frac{1}{(k+1)!}f^{[k+1]}(w)h^{k+1}$$

If h goes to be close to 0, the higher order terms will go to 0 as well. Thus, we can get:

$$f(x + h) \approx f(x) + f'(x)h$$

This is the first order Taylor approximation of f at x . In a similar vein, we also have the second order Taylor approximation of f at x as follows.

$$f(x + h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2$$

For the first order, we can rewrite it as follows.

$$f(x+h) \approx f(x) + f'(x)h = a + bh$$

where,

$$a = f(x), b = f'(x)$$

Similarly,

$$f(x+h) \approx f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 = a + bh + \frac{1}{2}ch^2$$

We can calculate the derivative with respect to h , we can get:

$$f'(x+h) \approx b + ch$$

We can then set it to zero, and get:

$$0 = b + c\hat{h}$$

Thus, we can get,

$$\hat{h} = -\frac{b}{c} = -\frac{f'(x)}{f''(x)}$$

Thus, we can get that the following can maximize f at x :

$$x + \hat{h} = x - \frac{f'(x)}{f''(x)}$$

Thus, the basic idea of Newton Raphson algorithm is as follows. - set a tolerance (typically a very small number) - Check if $|f'(x)| < thetolerance$. If not, $i \leftarrow i + 1; x_i \leftarrow x_{i-1} - \frac{f'(x_{i-1})}{f''(x_{i-1})}$

Practice:

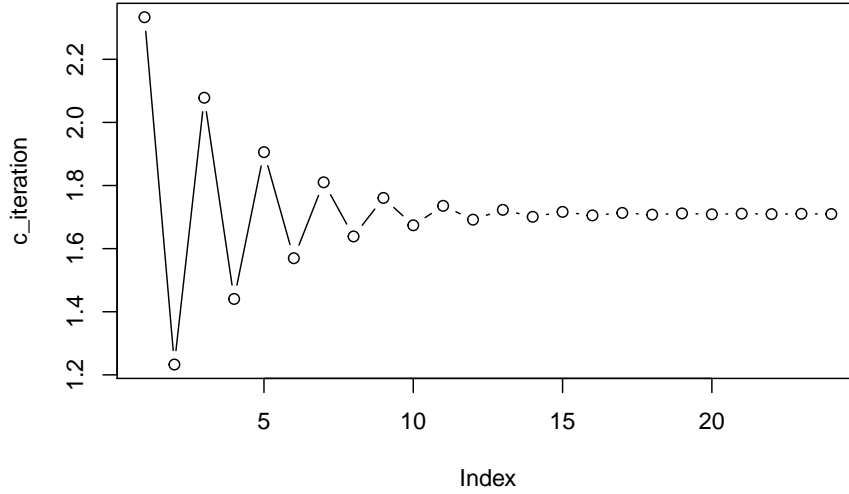
$$(1) \text{ Solve the } x^3 - 5 = 0$$

Note that, this is obviously not a maximization problem. In contrast, it involves a function with zero. As we can see, we can think it as the first order of Taylor approximation. That is, $f'(x) = x^3 - 5 = 0$. As we can see the following plot, it converts very quickly. [I will revisit this part later to double check whether such narrative is correct. But, the code is correct, for sure.]

```

f_firstorder=function(x){x^3-5}
f_secondorder=function(x){3*x}
x_old=1;tolerance=1e-3
max_its=2000;iteration=1;difference=2
c_iteration<-c() ## to collect numbers generated in the iteration process
while(difference>tolerance & iteration<max_its){
  x_updated=x_old-(f_firstorder(x_old)/f_secondorder(x_old))
  difference=abs(x_updated-x_old);
  iteration=iteration+1;
  x_old=x_updated
  c_iteration<-c(c_iteration,x_updated)}
plot(c_iteration,type="b")

```



2. Use Newton Raphson to calculate the logistic regression

Suppose we have n observation, and m variables.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ \dots & & & & \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix}$$

Typically, we add a vector of 1 being used to estimate the constant.

$$\begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ 1 & x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ \dots & & & & & \\ 1 & x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix}$$

And, we have observe a vector of n y_i as well, which is a binary variable:

$$Y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ \dots \\ 1 \end{bmatrix}$$

Using the content from the MLE chapter, we can get:

$$\mathbf{L} = \prod_{i=1}^n p_i^{y_i} (1 - p_i)^{(1-y_i)}$$

Further, we can get a log-transformed format.

$$\log(\mathbf{L}) = \sum_{i=1}^n [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Given that $p_i = \frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n}} = \frac{e^{\beta^T x}}{1 + e^{\beta^T x}}$, we can rewrite it as follows:

$$\log(\mathbf{L}) = \ell = \sum_{i=1}^n [y_i \log\left(\frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}}\right) + (1 - y_i) \log\left(1 - \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}}\right)]$$

Before doing the derivative, we set.

$$\frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} = p(\beta^T x_i)$$

$$\log(\mathbf{L}) = \ell = \sum_{i=1}^n [y_i \log(p(\beta^T x_i)) + (1 - y_i) \log(1 - p(\beta^T x_i))]$$

Note that, $\frac{\partial p(\beta^T x_i)}{\partial (\beta^T x_i)} = p(\beta^T x_i)(1 - p(\beta^T x_i))$. We will use it later.

$$\begin{aligned}
\frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^n \left[y_i \frac{1}{p(\beta^T x_i)} \frac{\partial p(\beta^T x_i)}{\partial(\beta^T x_i)} \frac{\partial(\beta^T x_i)}{\partial \beta} + (1 - y_i) \frac{1}{1 - p(\beta^T x_i)} (-1) \frac{\partial p(\beta^T x_i)}{\partial(\beta^T x_i)} \frac{\partial(\beta^T x_i)}{\partial \beta} \right] \\
&= \sum_{i=1}^n x_i^T \left[y_i \frac{1}{p(\beta^T x_i)} p(\beta^T x_i) (1 - p(\beta^T x_i)) + (1 - y_i) \frac{1}{1 - p(\beta^T x_i)} (-1) p(\beta^T x_i) (1 - p(\beta^T x_i)) \right] \\
&= \sum_{i=1}^n x_i^T \left[y_i \frac{1}{p(\beta^T x_i)} p(\beta^T x_i) (1 - p(\beta^T x_i)) - (1 - y_i) \frac{1}{1 - p(\beta^T x_i)} p(\beta^T x_i) (1 - p(\beta^T x_i)) \right] \\
&= \sum_{i=1}^n x_i^T [y_i (1 - p(\beta^T x_i)) - (1 - y_i) p(\beta^T x_i)] \\
&= \sum_{i=1}^n x_i^T [y_i - y_i p(\beta^T x_i) - p(\beta^T x_i) + y_i p(\beta^T x_i)] \\
&= \sum_{i=1}^n x_i^T [y_i - p(\beta^T x_i)] \\
&= \sum_{i=1}^n x_i^T \left[y_i - \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \right]
\end{aligned}$$

As noted, the Newton Raphson algorithm needs the second order.

$$\begin{aligned}
\text{Secondorder} &= \frac{\partial \sum_{i=1}^n x_i^T [y_i - p(\beta^T x_i)]}{\partial \beta} \\
&= - \sum_{i=1}^n x_i^T \frac{\partial p(\beta^T x_i)}{\partial \beta} \\
&= - \sum_{i=1}^n x_i^T \frac{\partial p(\beta^T x_i)}{\partial(\beta^T x_i)} \frac{\partial(\beta^T x_i)}{\partial \beta} \\
&= - \sum_{i=1}^n x_i^T p(\beta^T x_i) (1 - p(\beta^T x_i)) x_i
\end{aligned}$$

The following are the data simulation (3 IVs and 1 DV) and Newton Raphson analysis.

```

# Data generation
set.seed(123)
n=500
x1_norm<-rnorm(n)
x2_norm<-rnorm(n,3,4)
x3_norm<-rnorm(n,4,6)
x_combined<-cbind(1,x1_norm,x2_norm,x3_norm) # dimension: n*4

```



```

coefficients_new<-c(1,2,3,4) #true regression coefficient
inv_logit<-function(x,b){exp(x**b)/(1+exp(x**b))}
prob_generated<-inv_logit(x_combined,coefficients_new)
y<-c()
for (i in 1:n) {y[i]<-rbinom(1,1,prob_generated[i])}

# Newton Raphson

#We need to set random starting values.
beta_old<-c(1,1,1,1)
tolerance=1e-3
max_its=2000;iteration=1;difference=2
W<-matrix(0,n,n)

while(difference>tolerance & iteration<max_its)
{
  # The first order
  f_firstorder<-t(x_combined)**(y-inv_logit(x_combined,beta_old))
  # The second order
  diag(W) = inv_logit(x_combined,beta_old)*(1-inv_logit(x_combined,beta_old))
  f_secondorder<-t(x_combined)**W**x_combined
  # Calculate the beta_updated
  beta_updated=beta_old-(solve(f_secondorder)**f_firstorder)
  difference=max(abs(beta_updated-beta_old));
  iteration=iteration+1;
  beta_old=beta_updated}

beta_old

##           [,1]
##      0.9590207
## x1_norm 1.7974165
## x2_norm 3.0072303
## x3_norm 3.9578107

```

$$\begin{aligned}
\frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^n \left[y_i \frac{1}{p(\beta^T x_i)} \frac{\partial p(\beta^T x_i)}{\partial (\beta^T x_i)} \frac{\partial (\beta^T x_i)}{\partial \beta} + (1-y_i) \frac{1}{1-p(\beta^T x_i)} (-1) \frac{\partial p(\beta^T x_i)}{\partial (\beta^T x_i)} \frac{\partial (\beta^T x_i)}{\partial \beta} \right] \\
&= \sum_{i=1}^n \left[y_i \frac{1}{p(\beta^T x_i)} \phi(\beta^T x_i) - (1-y_i) \frac{1}{1-p(\beta^T x_i)} \phi(\beta^T x_i) \right] x_i
\end{aligned}$$

$$\Phi(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3) = p(y = 1)$$

```

# Data generation
n=500
x1_norm<-rnorm(n)
x2_norm<-rnorm(n)
x3_norm<-rnorm(n)
x_combined<-cbind(1,x1_norm,x2_norm,x3_norm)
coefficients_new<-c(2,2,3,3) #true regression coefficient
inv_norm<-function(x,b){pnorm(x*%b)}
prob_generated<-inv_norm(x_combined,coefficients_new)
y<-c()
for (i in 1:n) {y[i]<-rbinom(1,1,prob_generated[i])}

# Newton Raphson

#We need to set random starting values.
x_old<-c(1,1,1,1)
tolerance=1e-3
max_its=2000;iteration=1;difference=2

while(difference>tolerance & iteration<max_its){
  x_updated=x_old-(f_firstorder(x_old)/f_secondorder(x_old))
  difference=abs(x_updated-x_old);
  iteration=iteration+1;
  x_old=x_updated
  c_iteration<-c(c_iteration,x_updated)}

plot(c_iteration,type="b")

```

Some links related to this topic (canonical link function): <http://www.win-vector.com/blog/2011/09/the-simpler-derivation-of-logistic-regression/> https://en.wikipedia.org/wiki/Generalized_linear_model#Link_function <https://stats.stackexchange.com/questions/40876/what-is-the-difference-between-a-link-function-and-a-canonical-link-function> <https://stats.stackexchange.com/questions/344309/why-using-newtons-method-for-logistic-regression-optimization-is-called-iterati> <https://www.stat.washington.edu/adobra/classes/536/Files/week1/newtonfull.pdf> <https://tomroth.com.au/logistic/> <https://www.stat.cmu.edu/~cshalizi/350/lectures/26/lecture-26.pdf> <http://seankross.com/2019/10/27/R-as-a-First-Programming-Language.html>

5.2 Basics of GLMM

Recall the formula in the probit model:

$$Y^* = X\beta + \epsilon, \epsilon \sim N(0, \sigma^2) = N(0, I)$$

Similar to LMM, binary model with random effect can be written as follows.

$$Y^* = X\beta + Zu + \epsilon$$

where,

$$\epsilon \sim N(0, I)$$

$$u \sim N(0, D)$$

We also assume ϵ and u are independent. Thus, we know that D represents the variances of the random effects. If we make $u = 1$, the model becomes the usual probit model. McCulloch (1994) states that there are a few advantages to use probit, rather than logit models.

The following is the note from Charle E. McCulloch's "Maximum likelihood algorithms for Generalized Linear Mixed Models"

5.3 Some References

<http://www.biostat.umn.edu/~baolin/teaching/linmods/glmm.html>

http://www.biostat.umn.edu/~baolin/teaching/probmods/GLMM_mcmc.html

<https://bbolker.github.io/mixedmodels-misc/glmmFAQ.html>

Chapter 6

Twitter Example

The following is part of my course project for Stat 536. It aims to replicate part of the findings from Barbera (2015) Birds of the Same Feather Tweet Together: Bayesian Ideal Point Estimation Using Twitter Data. Political Analysis 23 (1). Note that, the following model is much simpler than that in the original paper.

6.1 Model

Suppose that a Twitter user is presented with a choice between following or not following another target $j \in \{1, \dots, m\}$. Let $y_j = 1$ if the user decides to follow j , and $y_j = 0$ otherwise.

$$y_j = \begin{cases} 1 & \text{Following} \\ 0 & \text{NotFollowing} \end{cases}$$

$$p(y_j = 1|\theta) = \frac{\exp(-\theta_0|\theta_1 - x_j|^2)}{1 + \exp(-\theta_0|\theta_1 - x_j|^2)}$$

We additionally know the priors of θ .

$$\theta_i \sim N(0, 10^2)(i = 0, 1)$$

The likelihood function is as follows.

$$L(Y|\theta) = \prod_{j=1}^m \left(\frac{\exp(-\theta_0|\theta_1 - x_j|^2)}{1 + \exp(-\theta_0|\theta_1 - x_j|^2)} \right)^{y_j} \left(1 - \frac{\exp(-\theta_0|\theta_1 - x_j|^2)}{1 + \exp(-\theta_0|\theta_1 - x_j|^2)} \right)^{(1-y_j)}$$

Thus, the posterior is as follows.

$$L(Y|\theta) \cdot N(\theta_0|0, 10) \cdot N(\theta_1|0, 10)$$

$$\propto \prod_{j=1}^m \left(\frac{\exp(-\theta_0|\theta_1 - x_j|^2)}{1 + \exp(-\theta_0|\theta_1 - x_j|^2)} \right)^{y_j} \left(1 - \frac{\exp(-\theta_0|\theta_1 - x_j|^2)}{1 + \exp(-\theta_0|\theta_1 - x_j|^2)} \right)^{(1-y_j)} \cdot \exp\left(-\frac{1}{2}\left(\frac{\theta_0}{10}\right)^2\right) \cdot \exp\left(-\frac{1}{2}\left(\frac{\theta_1}{10}\right)^2\right)$$

6.2 Simulating Data of Senators on Twitter

Assume that we have 100 senators, 50 Democrats and 50 Republicans, who we know their ideology. Assume that Democrats have negative ideology scores to indicate that they are more liberal, whereas Republicans have positive scores to indicate that they are more conservative. The following is data simulation for senators.

```
# Republicans are more conservative, and they have positive numbers.
Republicans<-c()
Republicans<-rnorm(50,1,0.5)
No_Republicans<-rep(1:50,1)
Part_1<-cbind(No_Republicans,Republicans)

# Democrats are more liberal, and they have negative numbers.
Democrats<-c()
Democrats<-rnorm(50,-1,0.5)
No_Democrats<-rep(51:100,1)
Part_2<-cbind(No_Democrats,Democrats)
Data_Elites<-rbind(Part_1,Part_2)
Data_Elites<-as.data.frame(Data_Elites)
colnames(Data_Elites) <- c("Elite_No","Elite_ideology")

head(Data_Elites)
```

```
##   Elite_No Elite_ideology
## 1      1      1.2848895
## 2      2      0.1733722
## 3      3      0.6666728
## 4      4      0.7758829
## 5      5      1.5219457
## 6      6      1.5140870
```

6.3 Simulating Data of Conservative Users on Twitter and Model Testing

Assume that we observe one Twitter user, who is more conservative. To simulate Twitter following data for this user, I assign this user to follow more Republican senators. Thus, if the Metropolis Hastings algorithm works as intended, we would expect to see a positive estimated value for their ideology. Importantly, as we can see in the histogram below, the estimated value indeed is positive, providing preliminary evidence for the statistical model and the algorithm. In addition, for the acceptance rate, we can see that the constant has a lower number than ideology, since we only accept a constant when it is positive.

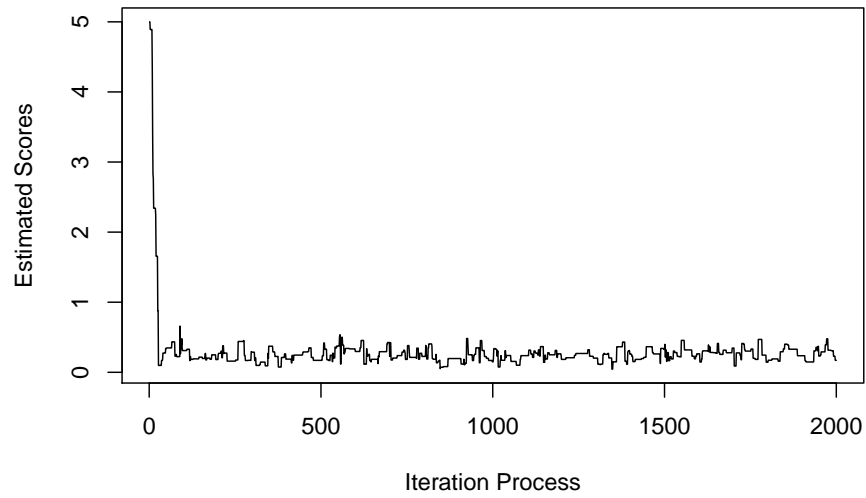
```
#This user approximately follows 45 Republican Senators and 10 Democrat Senators.
Data_user<-as.data.frame(matrix(c(ifelse(runif(50)<.1,0,1),ifelse(runif(50)<.8,0,1))), 100, 1)
colnames(Data_user)<-c("R_User")
Data_combined<-cbind(Data_Elites,Data_user)

X_data<-Data_combined$Elite_ideology
Y_data<-Data_combined$R_User

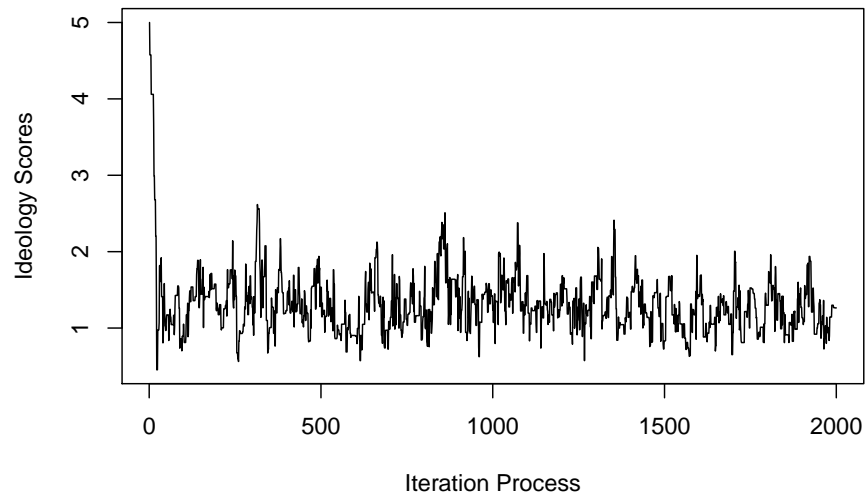
fit_C<-Bayes_logit(Y_data,X_data)
fit_C$acceptance_rate
```

```
## [1] 0.1590795 0.4817409
```

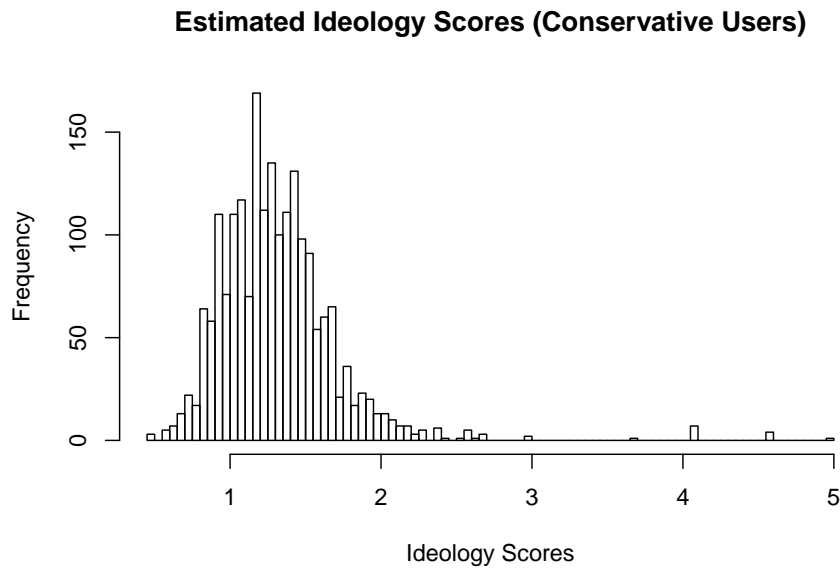
```
plot(fit_C$theta[,1],main="Constant (Conservative Users)",
     xlab="Iteration Process",ylab="Estimated Scores",type="l")
```

Constant (Conservative Users)

```
plot(fit_C$theta[,2],main="Estimated Ideology Scores (Conservative Users)",  
     xlab="Iteration Process",ylab="Ideology Scores",type="l")
```

Estimated Ideology Scores (Conservative Users)


```
hist(fit_C$theta[,2],main="Estimated Ideology Scores (Conservative Users)",
     xlab="Ideology Scores",breaks = 100)
```



6.4 Simulating Data of Liberal Users on Twitter and Model Testing

To further verify the Metropolis Hastings algorithm, I plan to test the opposite estimate. Specifically, assume that we observe another user, who is more liberal. To simulate Twitter following data for this user, I assign this user to follow more Democrat senators. In this case, we would expect to see a negative value for their estimated ideology. As we can see in the histogram shown below, as expected, the estimated value is negative, providing convergent evidence for the model and the algorithm.

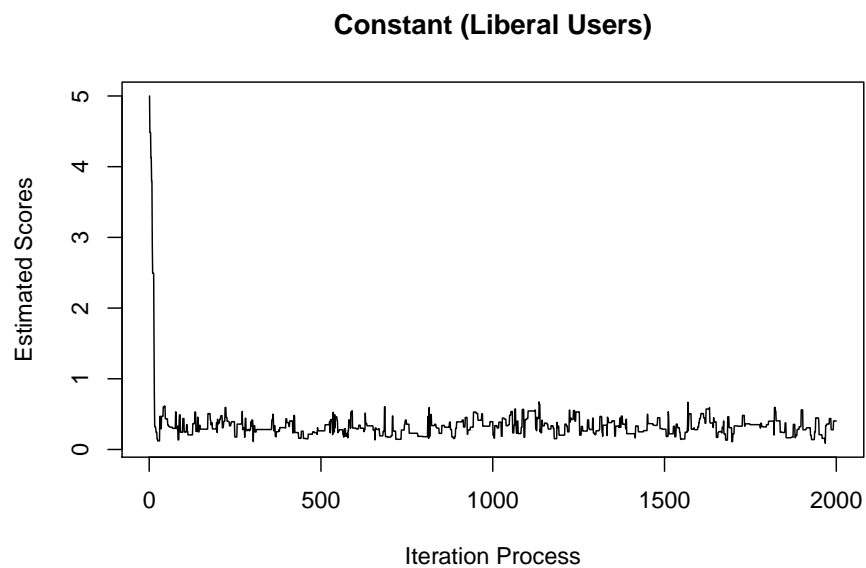
```
#This user approximately follows 10 Republican Senators and 45 Democrat Senators.
Data_user<-as.data.frame(matrix(c(ifelse(runif(50)< .8,0,1),ifelse(runif(50)< .1,0,1))), 100, 1)
colnames(Data_user)<-c("L_User")
Data_combined<-cbind(Data_Elites,Data_user)

X_data<-Data_combined$Elite_ideology
Y_data<-Data_combined$L_User
```

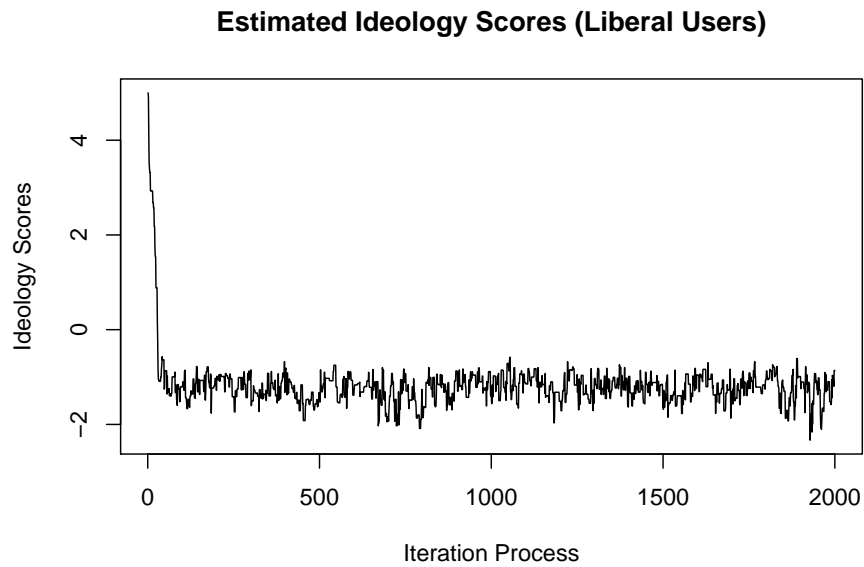
```
fit_L<-Bayes_logit(Y_data,X_data)
fit_L$acceptance_rate
```

```
## [1] 0.2081041 0.4337169
```

```
plot(fit_L$theta[,1],main="Constant (Liberal Users)",
      xlab="Iteration Process",ylab="Estimated Scores",type="l")
```



```
plot(fit_L$theta[,2],main="Estimated Ideology Scores (Liberal Users)",
      xlab="Iteration Process",ylab="Ideology Scores",type="l")
```



```
hist(fit_L$theta[,2],main="Estimated Ideology Scores (Liberal Users)",  
     xlab="Ideology Scores",breaks = 100)
```

