Logistic Regression Solvers

Lei Kang

September 12, 2016

1. Gradient Descent

Score function of logistic regression is:

$$U(\beta) = X^T(Y - P) \tag{1}$$

where,

$$P_i = \frac{exp(\beta^T x_i)}{1 + exp(\beta^T x_i)} \tag{2}$$

Then the update rule of gradient descent is:

$$\beta^{t+1} = \beta^t + \alpha U(\beta) = \beta^t + \alpha X^T(Y - P) \tag{3}$$

I use "plus" because my score function is not computed with respect to negative LL. Since the largest eigenvalue of Hessian is bounded from above by 0.25 times the largest eigenvalue of $X^T X$, I will specify alpha using that upper bound.

2. Newton-Rhapson

Score function is the same as before, but we need to compute Hessian.

$$H_{jk} = \sum_i^n \frac{x_{ij}x_{ik}}{var(y_i)}(\frac{d\mu_i}{d\eta_i})^2 = X^T W X \tag{4}$$

where,

$$W_{ii} = \frac{1}{var(y_i)}(\frac{d\mu_i}{deta_i})^2 = \frac{1}{p_i(1-p_i)}(p_i(1-p_i))^2 = p_i(1-p_i) \tag{5}$$

Then the update rule of Newton-Rhapson is:

$$\beta^{t+1} = \beta^t + H^{-1}U(\beta) = \beta^t + (X^T W X)^{-1}X^T(Y - P) \tag{6}$$

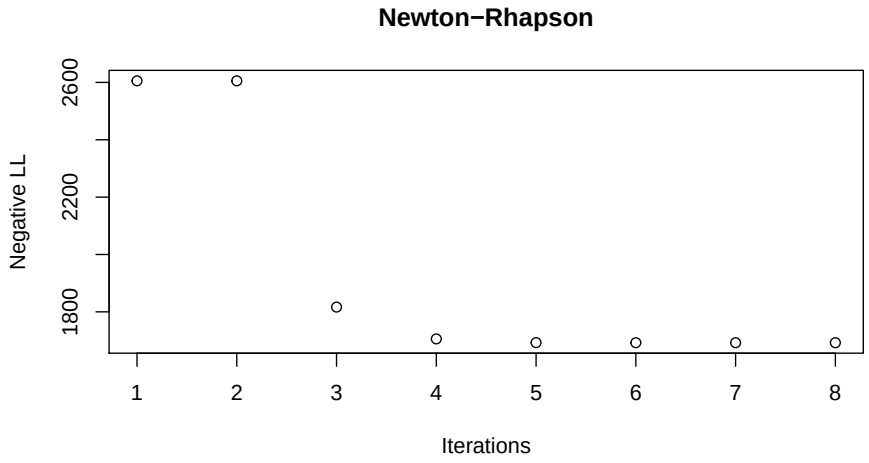
3. Comparison

For Abalone data, it suggests NR method converge much faster than GD and the results of NR results are identical to R glm function results. GD's results are slightly different from the other two.

```
###use R glm function
g1 <- glm(old~length+diameter+height+whole.weight+
            shucked.weight+viscera.weight+
            shell.weight,family="binomial",data=ab.tr)

print(g1$coefficients)
```

```
##    (Intercept)         length       diameter         height   whole.weight
##       3.564877       5.477840      -7.312210      -6.164202     -10.106507
## shucked.weight viscera.weight   shell.weight
##      18.321258       5.689058      -8.572537
```

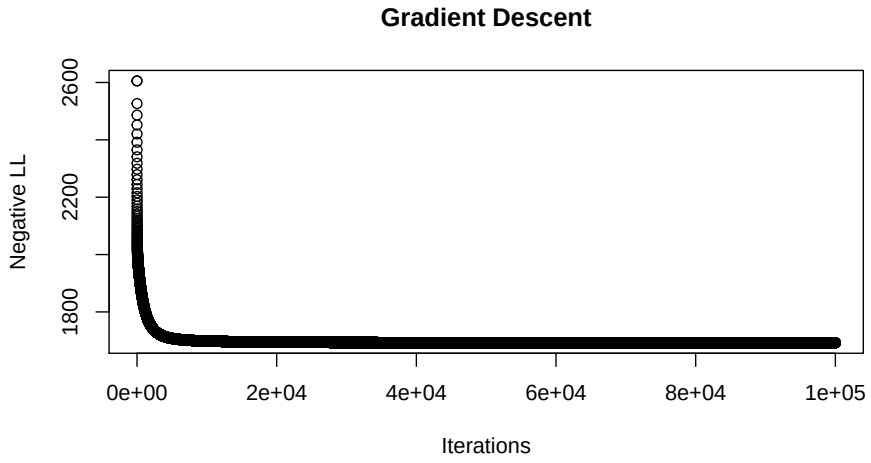
```
###use Newton method
fit_NR1<-fitModel_NR(tr.old, ab.tr2, max.iter=100, eps=1e-10)
plot(fit_NR1[[2]],ylab="Negative LL",xlab="Iterations",
     main="Newton-Rhapson") ##plot -LL for each iteration
```




```
print(fit_NR1[[1]][,dim(fit_NR1[[1]])[2]]) ##obtain the last update
```

```
## [1]   3.564877   5.477840  -7.312210  -6.164202 -10.106508  18.321259
## [7]   5.689059  -8.572536
```

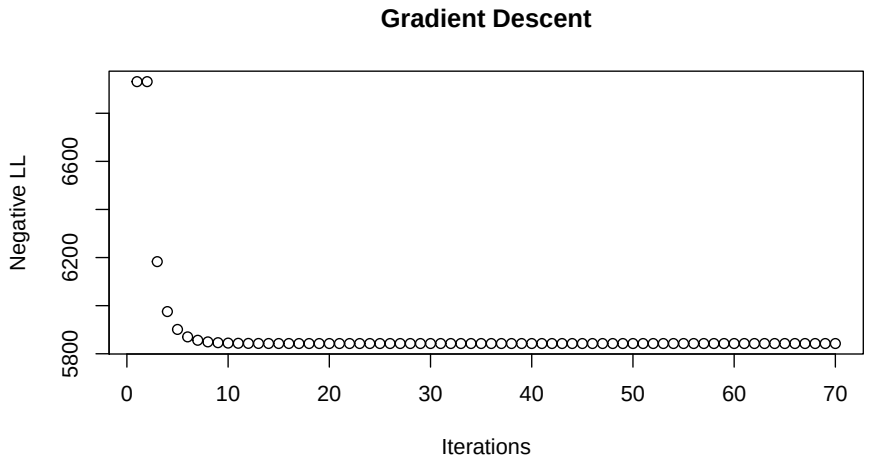
```
###use Gradient method
fit_GD1<-fitModel_GD(tr.old, ab.tr2, max.iter=100000, eps=1e-10)
plot(fit_GD1[[2]],ylab="Negative LL",xlab="Iterations",
     main="Gradient Descent") ##plot -LL for each iteration
```



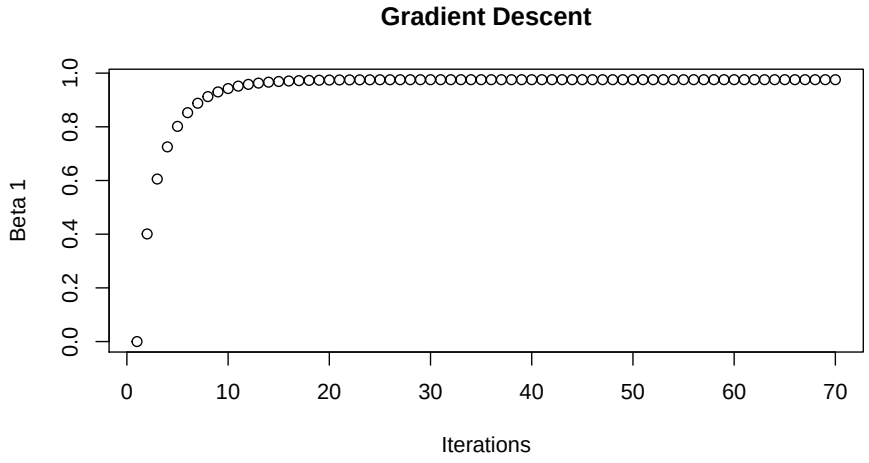

```
print(fit_GD1[[1]][,dim(fit_GD1[[1]])[2]]) ##obtain the last update

## [1]   3.567905   5.151263  -6.889560  -6.160882 -10.063053  18.284867
## [7]   5.654712  -8.662517
```

For this simulated dataset, GD and NR end up with identical results, but still GD requires larger number of iterations to converge. Again, their results are consistent with glm results and very close to true values.

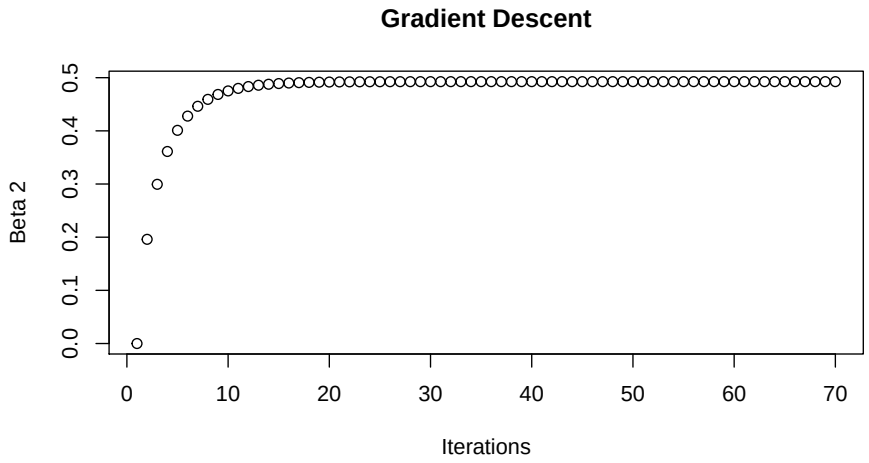
```
###use Gradient method
fit_GD2<-fitModel_GD(sim.y, sim.x, max.iter=100000, eps=1e-10)
plot(fit_GD2[[2]],ylab="Negative LL",xlab="Iterations",
     main="Gradient Descent") ##plot -LL for each iteration
```




```
plot(fit_GD2[[1]][2,],ylab="Beta 1",xlab="Iterations",
     main="Gradient Descent") ##plot beta 1
```

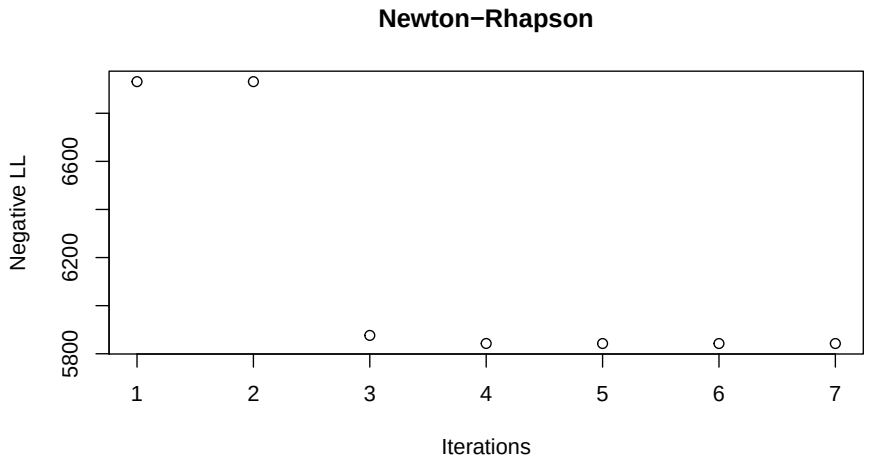



```
plot(fit_GD2[[1]][3,],ylab="Beta 2",xlab="Iterations",
     main="Gradient Descent") ##plot beta 2
```

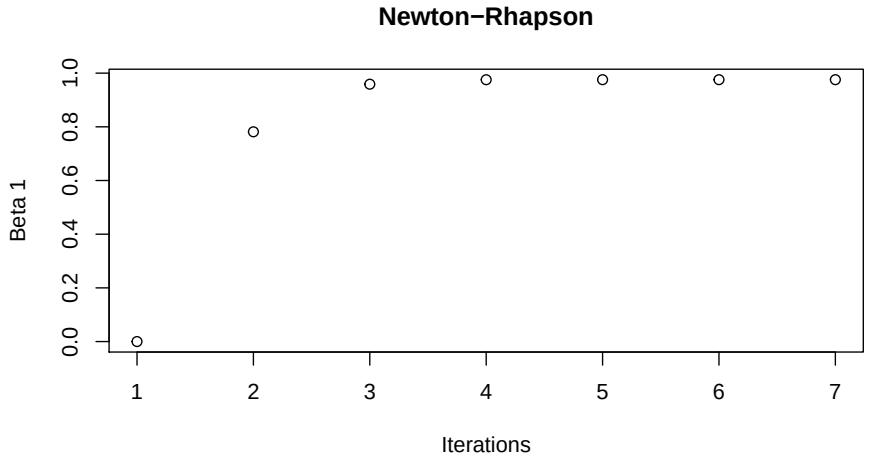



```
print(fit_GD2[[1]][,dim(fit_GD2[[1]])[2]]) ##obtain the last update

## [1] 0.03775448 0.97561962 0.49254801

print(beta.true)  ##true value

## [1] 0.0 1.0 0.5
```

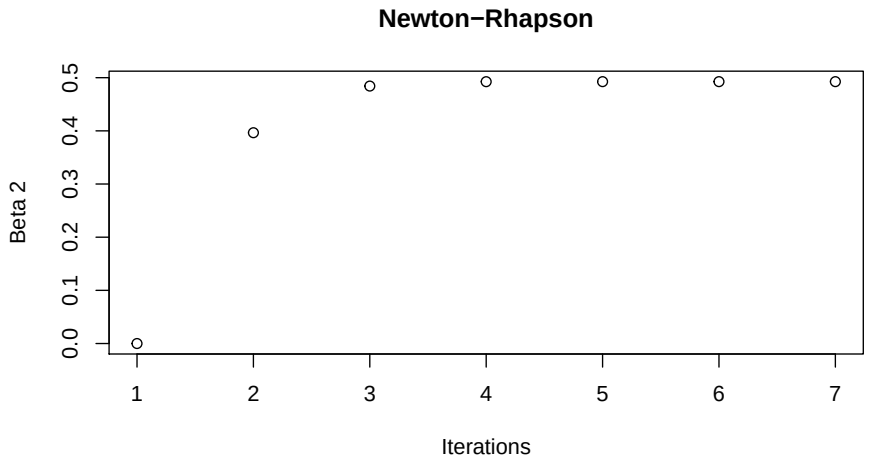
```
###use Newton method
fit_NR2<-fitModel_NR(sim.y , sim.x, max.iter=100, eps=1e-10)

plot(fit_NR2[[2]],ylab="Negative LL",xlab="Iterations",
     main="Newton-Rhapson") ##plot -LL for each iteration
```




```
plot(fit_NR2[[1]][2,],ylab="Beta 1",xlab="Iterations",
     main="Newton-Rhapson") ##plot beta 1
```




```
plot(fit_NR2[[1]][3,],ylab="Beta 2",xlab="Iterations",
     main="Newton-Rhapson") ##plot beta 2
```




```
print(fit_NR2[[1]][,dim(fit_NR2[[1]])[2]]) ##obtain the last update

## [1] 0.03775448 0.97561962 0.49254801

print(beta.true)  ##true value

## [1] 0.0 1.0 0.5
```

```
###use glm function
g2<-glm(sim.y~x0+x1+x2-1, data=data.frame(cbind(sim.y,sim.x)),family="binomial")

print(g2$coefficients)

##          x0          x1          x2
## 0.03775448 0.97561962 0.49254801
```

4. Appendix-function code

```
cost <- function(X, y, beta) { ##return negative LL
  z<-X %*% beta
  LL<-t(y)%*%z-sum(log(1+exp(z)))
  return (-LL)
}

gradient <- function(X, y, beta) {
  p<-exp(X %*% beta)/(1+exp(X %*% beta))
```

```r
  score<-t(X) %*% (y - p)
  return (score)
}

hessian <- function(X, beta) {
  p<-exp(X %*% beta)/(1+exp(X %*% beta))
  w <- diag(c(p*(1-p)))
  J<-t(X) %*% w %*% X
  return (J)
}
#######Newton
fitModel_NR <- function(y, X, max.iter=100, eps=1e-10) {
  n.beta <- ncol(X)
  B <-  matrix(NA,ncol = max.iter,nrow = n.beta)
  NLL<-rep(0,max.iter) ###negative LL
  B[,1] <- rep(0,ncol(X)) ##starting value
  NLL[1]<- cost(X,y,B[,1])
  for (i in 2:max.iter) {
    B[,i] <- B[,i-1]+solve(hessian(X,B[,i-1])) %*% gradient(X,y,B[,i-1])
    NLL[i] <-cost(X,y,B[,i-1])
    if (all(abs(B[,i]-B[,i-1]) < eps)) break;
  }
  B <- B[ , !apply(is.na(B), 2, all)] ##remove NA columns if there is any
  NLL<-NLL[NLL>0]
  return (list(B,NLL))
}
####gradient descent
fitModel_GD <- function(y, X, max.iter=70000, eps=1e-10) {
  n.beta <- ncol(X)
  B <-  matrix(NA,ncol = max.iter,nrow=n.beta)
  NLL<-rep(0,max.iter) ###negative LL
  B[,1] <- rep(0,ncol(X)) ##starting value
  NLL[1]<- cost(X,y,B[,1])
  alpha = 4/(svd(cbind(1, X))$d[1]^2)

  for (i in 2:max.iter) {
    B[,i] <- B[,i-1]+alpha*gradient(X,y,B[,i-1])
        NLL[i] <-cost(X,y,B[,i-1])
        if (all(abs(B[,i]-B[,i-1]) < eps)) break;
  }
  B <- B[ , !apply(is.na(B), 2, all)] ##remove NA columns if there is any
  NLL<-NLL[NLL>0]
  return (list(B,NLL))
}
```