SDS 385: Exercise 02

September 8, 2016

Jennifer Starling

Linear Regression

Part A

In Exercise 01, Generalized Linear Regression, Part A, the derivation of the gradient of the negative log-likelihood in matrix form is:

$\nabla l(\beta) = -\sum_{i=1}^{N}(y_i - m_i w_i)x_i$

In matrix form: $\nabla l(\beta) = -X'(y - mw) = X'(mw - y) = X'(\hat{y} - y)$

with $w_i = w_i(\beta) = \frac{1}{1+exp(-x_i{}'\beta)}$

Part B

Suppose you draw a single point at random from the sample of data, giving the pair $\{y_i, x_i\}$, where y_i is a single response from row i and x_i is the i^{th} row of design matrix X.

Then $E(ng_i(\beta)) = nE(g_i(\beta))$.

i is the only random value here, with $P(i = j) = 1/n$ for $j \in \{i = 1, ..., n\}; 0$ otherwise.

So $nE(g_i(\beta)) = n(\frac{1}{n})\sum_{i=1}^{n} g_i(\beta) = \sum_{i=1}^{n} g_i(\beta) = \nabla(l(\beta))$.

Therefore $E(ng_i(\beta)) = \nabla(l(\beta))$.

Part C

I implemented the running average of $l(\beta)$ to track convergence.

Part D

For a decaying step size, $C = 40, \alpha = 0.5, t_0 = 2$ gave good approximations of the betas within a million iterations.

Part E

The Polyak-Ruppert Averaging did not improve my results. My results without the averaging were actually closer to the GLM-generated βs.

Results

My results for the stochastic gradient descent were as follows, using the following:
- $C = 40, \alpha = 0.5, t_0 = 2$
- One million iterations
- No specific convergence criteria; just ran for full iterations.

```
> beta #GLM estimates
          X         XV3         XV4         XV5         XV6         XV7
 0.48701675 -7.22185053  1.65475615 -1.73763027 14.00484560  1.07495329
        XV8         XV9        XV10        XV11        XV12
-0.07723455  0.67512313  2.59287426  0.44625631 -0.48248420
> betas[i,] #Stochastic estimates.
 [1]  0.4931931 -3.9841799  1.6160841 -4.0661091 12.7862893  1.0945374
 [7]  0.1505122  0.8018911  2.6743230  0.5257934 -0.4182426
> beta_pr  #Estimates with Polyak-Ruppert Averaging
 [1]  0.30690366 -3.63914437  1.66252165 -4.01274222 12.33884238  1.05591877
 [7]  0.05485383  0.70729371  2.61896729  0.45418036 -0.50787328
```

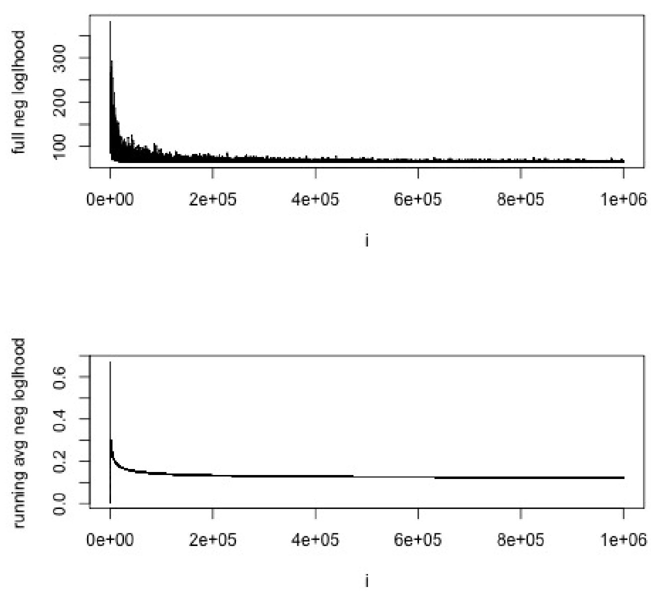

Figure 1: Negative Log-Likelihood Plots

Appendix: R Code

```
### SDS 385 - Exercises 02 - Part C
#This code implements stochastic gradient descent to estimate the
#beta coefficients for binomial logistic regression.

#Jennifer Starling
#30 August 2016

rm(list=ls())   #Cleans workspace.
library(microbenchmark)
library(permute)
library(zoo)    #For rolla pply

#PART C:

#Read in code.
wdbc = read.csv('/Users/jennstarling/UTAustin/2016_Fall_SDS 385_Stats Models for
    Big Data/Course Data/wdbc.csv', header=FALSE)
y = wdbc[,2]

#Convert y values to 1/0's.
Y = rep(0,length(y)); Y[y=='M']=1
X = as.matrix(wdbc[,-c(1,2)])

#Select features to keep, and scale features.
scrub = which(1:ncol(X) %% 3 == 0)
scrub = 11:30
X = X[,-scrub]
X <- scale(X) #Normalize design matrix features.
X = cbind(rep(1,nrow(X)),X)

#Set up vector of sample sizes.  (All 1 for wdbc data.)
m <- rep(1,nrow(X))

#------------------------------------------------------------------
#Binomial Negative Loglikelihood function.
    #Inputs: Design matrix X, vector of 1/0 vals Y,
    #   coefficient matrix beta, sample size vector m.
    #Output: Returns value of negative log-likelihood
    #   function for binomial logistic regression.
logl <- function(X,Y,beta,m){
    w <- 1 / (1 + exp(-X %*% beta)) #Calculate probabilities vector w_i.
    logl <- - sum(Y*log(w+.01) + (m-Y)*log(1-w+.01)) #Calculate log-likelihood.
        #Adding .01 to resolve issues with probabilities near 0 or 1.
    return(logl)
}

#------------------------------------------------------------------
#Stochastic Gradient Function:
    #Inputs: Vector X (One row of design matrix), vector of 1/0 vals Y,
    #   coefficient matrix beta, sample size vector m.
    #Output: Returns value of gradient function for binomial
    #   logistic regression.

gradient <- function(X,Y,beta,m){
    w <- 1 / (1 + exp(-X %*% beta)) #Calculate probabilities vector w_i.

```

```
    gradient <- array(NA,dim=length(beta))  #Initialize the gradient.
    gradient <- apply(X*as.numeric(m*w-Y),2,sum) #Calculate the gradient.

    return(gradient)
}

#------------------------------------------------------------------
#Robbins-Monro Step Size Function:
#   Inputs: C>0, a constant.  a in [.5,1], a constant.
#       t, the current iteration number.  t0, the prior number of steps.
#       (Try smallish t0, 1 to 2.)
#   Outputs: step, the step size.

rm_step <- function(C,a,t,t0){
    step <- C*(t+t0)^(-a)
    return(step)
}

#Playing with step sizes:
t <- 1:50
#sp <- rm_step(C=5,a=.75,t=t,t0=2)
p#lot(t,sp)

#Varying C:
cl <- rainbow(5)
#plot(t,rm_step(C,a[1],t,t0[2]),col=cl,lwd=1,pch=20,cex=.5)

#Varying a:
#plot(t,rm_step(C[1],a,t,t0[2]),col=cl,lwd=1,pch=20,cex=.5)

#Varying t:
cl2 <- rainbow(2)
#plot(t,rm_step(C[2],a[5],t,t0),col=cl2,lwd=1,pch=20,cex=.5)

#Play with ideal step size curve shape:
C=10;  t0=1;  a=.75;
plot(t,rm_step(C,a,t,t0),type='l',col='blue')

#------------------------------------------------------------------
#Stochastic Gradient Descent Algorithm:

#1. Fit glm model for comparison. (No intercept: already added to X.)
glm1 = glm(y~X-1, family='binomial') #Fits model, obtains beta values.
beta <- glm1$coefficients

maxiter <- 1000000  #Specify max iterations allowed.

#Initialize matrix to hold gradients for each iteration.
grad <- matrix(0,nrow=maxiter,ncol=ncol(X))

#Initialize matrix to hold beta vector for each iteration.
betas <- matrix(0,nrow=maxiter+1,ncol=ncol(X))

#Initialize vector to hold full loglikelihood fctn for each iter.
loglik <- rep(0,maxiter)
#Initialize vector to hold loglikelihood for each indiv t obs.
loglik_t <- rep(0,maxiter)
#Initialize vector to hold running avg for logl for t's.
loglik_ra <- rep(0,maxiter)
```

```
conv <- 1E-10   #Set convergence level.

#Set up random iterations through data, up to maxiter.
npermutes <- ceiling(maxiter/nrow(X))
obs_order <- as.vector(t(shuffleSet(1:nrow(X),nset=npermutes)))

#Initialize values:
i=1
t  <- obs_order[i]
Xnew <- matrix(X[t,,drop=F],nrow=1,byrow=T)
loglik_t[i] <- logl(Xnew,Y[t],betas[i,],m[t])
loglik_ra[i] <- loglik_t[i]
grad[1,] <- gradient(Xnew,Y[t],betas[i,],m[t])
betas[1,] <- 0

#2. Perform stoachstic gradient descent.
for (i in 2:maxiter){

        #Select one random obs per iter.
        t  <- obs_order[i]
        Xnew <- matrix(X[t,,drop=F],nrow=1,byrow=T)

        #Calculate Robbins-Monro step size.
        step <- rm_step(C=40,a=.5,t=i,t0=2)

        #Set new beta equal to beta - a*gradient(beta).
        betas[i,] <- betas[i-1,] - step * grad[i-1,]

        #Calculate fullloglikelihood for each iteration.
        loglik[i] <- logl(X,Y,betas[i,],m)

        #Calculate loglikelihood of individual observation t.
        loglik_t[i] <- logl(Xnew,Y[t],betas[i,],m[t])

        #Calculate running average of loglikelihood for individual t's.
        loglik_ra[i] <- (loglik_ra[i-1]*(i-1) + loglik_t[i])/i

        #Calculate stochastic gradient for beta, using only obs t.
        grad[i,] <- gradient(Xnew,Y[t],betas[i,],m[t])

        print(i)

        #Check if convergence met:  If yes, exit loop.
        #Note: Not using norm(gradient) like with regular gradient descent.
        #Gradient is too variable in stochastic case.
        #Can run for set iterations, but here, checking for convergence based
        #on iter over iter change in running avg of log-likelihoods.

        #Check if convergence met: If yes, exit loop.
        if (abs(loglik_ra[i]-loglik_ra[i-1])/abs(loglik_ra[i-1]+1E-3) < conv ){
            converged=1;
            break;
        }

} #End gradient descent iterations.

#-------------------------------------------------------------------
#Perform Polyak-Ruppert averaging to obtain final beta result:
```

```
#Calculate burn-in period to discard: 1/2 of the total iterations.
t <- floor(i*.5):i
beta_pr <- colMeans(betas[t,])

#------------------------------------------------------------------
#OUTPUT DATA RESULTS:

beta #GLM estimates
betas[i,] #Stochastic estimates.
beta_pr #Estimates with Polyak-Ruppert Averaging

#abs(loglik_ra[i]-loglik_ra[i-1])

#Plot full log-likelihood function for convergence, and running average for log-
    likelihoods.
par(mfrow=c(2,1))
plot(2:i,loglik[2:i],type='l',xlab='i',ylab='full neg loglhood')
plot(2:i,loglik_ra[2:i],type='l',xlab='i',ylab='running avg neg loglhood')


#OUTPUT PLOTS:
jpeg(file='/Users/jennstarling/UTAustin/2016_Fall_SDS 385_Stats Models for Big
    Data/Exercise 02 LaTeX Files/Ex02_loglik_and_ravg.jpeg')
par(mfrow=c(2,1))
plot(2:i,loglik[2:i],type='l',xlab='i',ylab='full neg loglhood')
plot(2:i,loglik_ra[2:i],type='l',xlab='i',ylab='running avg neg loglhood')
dev.off()
```