SDS 385: Exercises 1 - Preliminaries

August 23, 2016

Professor James Scott

Spencer Woody

Problem 1

(A)

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{N} \frac{w_i}{2} \left(y_i - x_i^T \beta\right)^2 \tag{1}$$
$$= \arg\min_{\beta \in \mathbb{R}^p} \frac{1}{2}(Y - X\beta)^T W (Y - X\beta) \tag{2}$$

$$\frac{1}{2}(Y - X\beta)^T W (Y - X\beta) = \frac{1}{2}(Y^T - \beta^T X^T) W (Y - X\beta) \tag{3}$$
$$= \frac{1}{2}(Y^T W - \beta^T X^T W)(Y - X\beta) \tag{4}$$
$$= \frac{1}{2}(Y^T W Y - \beta^T X^T W Y - Y^T W X\beta + \beta^T X^T W X \beta) \tag{5}$$
$$= \frac{1}{2}(Y^T W Y - 2(X\beta)^T W Y + \beta^T X^T W X \beta) \tag{6}$$
$$= \frac{1}{2} Y^T W Y - (X\beta)^T W Y + \frac{1}{2}\beta^T X^T W X \beta, \tag{7}$$

because

$$\beta^T X^T W Y = (X\beta)^T W Y, \tag{8}$$

and

$$Y^T W X \beta = (Y^T W X \beta)^T \because Y^T W X \beta \in \mathbb{R}^1 \tag{9}$$
$$(Y^T W X \beta)^T = (W X \beta)^T Y = (X\beta)^T W^T Y = (X\beta)^T W Y. \tag{10}$$

We want to minimize the objective function from Eqn. (7), so we take the gradient with respect to β and set it equal to zero. For each of the three terms, their are respective gradients with respect to β are

(i)

$$\frac{\partial}{\partial \beta} \frac{1}{2} Y^T W Y = 0 \tag{11}$$

(ii)

$$\frac{\partial}{\partial \beta} - (X\beta)^T W Y = -X^T W Y \tag{12}$$

(iii)

$$\frac{\partial}{\partial \beta} \frac{1}{2} \beta^T X^T W X \beta = \frac{1}{2} \beta^T (X^T W X + (X^T W X)^T) \tag{13}$$
$$= X^T W X \beta. \tag{14}$$

Summing these terms and equaling them to zero yields

$$X^T W X \beta - X^T W Y = 0 \therefore \tag{15}$$
$$(X^T W X)\hat{\beta} = X^T W Y \tag{16}$$

(B) The brute force method of solving Eqn. (16) is the *inversion method*, i.e.

$$\hat{\beta} = (X^T W X)^{-1} X^T W y. \tag{17}$$

However, this method is computationally expensive. Therefore I propose an alternative methods to solving this matrix equation using the Cholesky decomposition. **Cholesky Decomposition**
Let

$$C = X^T W X, \ \ D = X^T W y \tag{18}$$

so

$$C\hat{\beta} = D. \tag{19}$$

We decompose matrix C into a product of a lower-triangular matrix and an upper-triangular matrix, such that $U = L^T$ so

$$C = LU = LL^T \therefore \tag{20}$$

$$LL^T \hat{\beta} = D. \tag{21}$$

Furthermore we define matrix $A = L^T \hat{\beta}$. Thus we are left with two matrix equations to solve.

$$LA = D \tag{22}$$

$$L^T \hat{\beta} = A \tag{23}$$

This method will be much less computationally intensive than the inversion method because of the fact that the two left-matrices L and $U = L^T$ are triangular. We still must invert L and L^T but this is simpler than taking an inverse of a more complicated matrix $X^T W X$. This is similar to an LU decomposition, with the exception that we necessarily have two triangular matrices that are transposes of one another. Therefore, this method gains a computational advantage over LU decomposition from symmetric exploitation.

(C) Code for implementing this method is shown in the appendix to this paper.

(D) The Matrix package within R is suited to handle sparse matrices with the `sparse` argument within the `Matrix` command.

Problem 2

(A) We have $y_i \sim \text{Binomial}(m_i, w_i)$, where

$$w_i = \frac{1}{1+\exp(-x_i^T\beta)}, \quad 1 - w_i = \frac{\exp(-x_i^T\beta)}{1+\exp(-x_i^T\beta)}, \tag{24}$$

so the negative log likelihood is

$$\ell(\beta) = -\log\left\{\prod_{i=1}^N p(y_i|\beta)\right\} \tag{25}$$

$$= -\log\left\{\prod_{i=1}^N \binom{m_i}{y_i}(w_i)^{y_i}(1-w_i)^{m_i-y_i}\right\} \tag{26}$$

$$= -\left\{\sum_{i=1}^N \left(\log\binom{m_i}{y_i} + y_i\log(w_i) + (m_i - y_i)\log(1-w_i)\right)\right\} \tag{27}$$

$$= -\left\{\sum_{i=1}^N \left(\log\binom{m_i}{y_i} + y_i\log\left(\frac{1}{1+\exp(-x_i^T\beta)}\right) + (m_i-y_i)\log\left(\frac{\exp(-x_i^T\beta)}{1+\exp(-x_i^T\beta)}\right)\right)\right\} \tag{28}$$

$$= -\left\{\sum_{i=1}^N \left(\log\binom{m_i}{y_i} - y_i\log(1+\exp(-x_i^T\beta)) - (m_i-y_i)x_i^T\beta - m_i\log(1+\exp(-x_i^T\beta)) + y_i\log(1+\exp(-x_i^T\beta))\right)\right. \tag{29}$$

$$= -\left\{\sum_{i=1}^N \left(\log\binom{m_i}{y_i} - (m_i-y_i)x_i^T\beta - m_i\log(1+\exp(-x_i^T\beta))\right)\right\} \tag{30}$$

$$= \sum_{i=1}^N \left((m_i-y_i)x_i^T\beta + m_i\log(1+\exp(-x_i^T\beta)) - \log\binom{m_i}{y_i}\right) \tag{31}$$

$$\tag{32}$$

The gradient for this expression is,

$$\nabla\ell(\beta) = \sum_{i=1}^N \left((m_i-y_i)x_i - m_i\frac{\exp(-x_i^T\beta)}{1+\exp(-x_i^T\beta)}x_i\right) \tag{33}$$

$$= \sum_{i=1}^N ((m_i-y_i)x_i - m_i(1-w_i)x_i) \tag{34}$$

$$= \sum_{i=1}^N (m_iw_i - y_i)x_i \tag{35}$$

$$= -X^T(y - mw) \tag{36}$$

where y is the $n \times 1$ vector of responses and mw is the element-wise product of the two $n \times 1$ vectors m and w.

(B) Code for implementing the gradient descent method is shown in the appendix. Note that we normalize the values in the X matrix and add a column of 1's to make an intercept term. We start by having an initial arbitrary guess for β, which we define as β_0. Then we use an iterative process to converge upon the true value of β based on the calculated gradient of the log likelihood at $\hat{\beta}_t$ and an arbitrary step size, α as follows

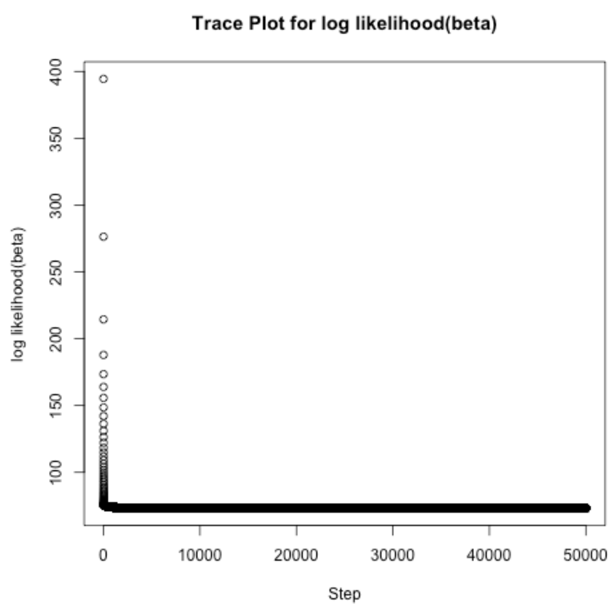
$$\hat{\beta}_{t+1} = \hat{\beta}_t - \alpha \times \nabla\ell(\hat{\beta}_t) \tag{37}$$

Problem 2 continued on next page...

	Grad descent	R: `glm`
$\hat{\beta}_1$	0.48553	0.48702
$\hat{\beta}_2$	-7.14618	-7.22185
$\hat{\beta}_3$	1.65481	1.65476
$\hat{\beta}_4$	-1.80713	-1.73763
$\hat{\beta}_5$	13.99290	14.00485
$\hat{\beta}_6$	1.07426	1.07495
$\hat{\beta}_7$	-0.07319	-0.07723
$\hat{\beta}_8$	0.67573	0.67512
$\hat{\beta}_9$	2.59383	2.59287
$\hat{\beta}_{10}$	0.44615	0.44626
$\hat{\beta}_{11}$	-0.48276	-0.48248

Table 1: Comparison of results from gradient descent and `glm`

We use an intial guess of $\beta_0 = 0$, a step size of $\alpha = 0.025$, and 50,000 iterations and reach convergence in optimizing the log likelihood, as shown in the trace plot below. Our final estimations of β are reported below along with estimations from R's native `glm` function. The two sets of estimates are in close agreement with one another.



(C) We need to calculate the Hessian matrix of the log likelihood function, $\nabla^2(\ell(\beta))$. The Hessian will be a $P \times P$ matrix, with the element in row i and column j being[1]

$$\frac{\partial^2}{\partial\beta_i\partial\beta_j}\ell(\beta) = \frac{\partial}{\partial\beta_i}\left(\frac{\partial}{\partial\beta_j}\ell(\beta)\right) \tag{38}$$

$$= \frac{\partial}{\partial\beta_i}\left(\frac{\partial}{\partial\beta_j}\sum_{k=1}^{N}(\ldots)\right) \tag{39}$$

$$= \frac{\partial}{\partial\beta_i}\left(\sum_{k=1}^{N}(m_k w_k - y_k)x_{kj}\right) \tag{40}$$

$$= \sum_{k=1}^{N} x_{ki}x_{kj}m_k w_k(1-w_k) \tag{41}$$

Note:

$$\frac{\partial}{\partial\beta_i}w_k = x_{ki}\frac{\exp(-x_k^T\beta)}{(1+\exp(-x_k^T\beta))^2} \tag{42}$$

$$= x_{ki}w_k(1-w_k) \tag{43}$$

This matrix is equivalent to X^TWX where $W = \text{diag}(m_1w_1(1-w_1), \ldots, m_Nw_N(1-w_N))$

(D) Now we use Newton's to estimate β. This is also an iterative process, though now we need far fewer iterations to achieve convergence because we are taking the curvature of our objective function ($\ell(\beta)$) into account. In fact, we only use 10 iterations and achieve estimates $\hat{\beta}$ which are *exactly* in line with estimates from `glm`.
Newton's Method:

$$\hat{\beta}_{t+1} = \hat{\beta}_t - (\nabla^2\ell(\hat{\beta}_t))^{-1}\nabla\ell(\hat{\beta}_t) \tag{44}$$

	N.'s method	R: `glm`
$\hat{\beta}_1$	0.48702	0.48702
$\hat{\beta}_2$	-7.22185	-7.22185
$\hat{\beta}_3$	1.65476	1.65476
$\hat{\beta}_4$	-1.73763	-1.73763
$\hat{\beta}_5$	14.00485	14.00485
$\hat{\beta}_6$	1.07495	1.07495
$\hat{\beta}_7$	-0.07723	-0.07723
$\hat{\beta}_8$	0.67512	0.67512
$\hat{\beta}_9$	2.59287	2.59287
$\hat{\beta}_{10}$	0.44626	0.44626
$\hat{\beta}_{11}$	-0.48248	-0.48248

Table 2: Comparison of results from Newton's method and `glm`

(E) Gradient descent requires many iterations while Newton's method must invert a matrix, which may either be impossible and is computationally intensive for large matrices.

[1] Notice the reindexing shown below for summations.

```
##########################################################
######### Created by Spencer Woody on 24 Aug 2016 #########
##########################################################

library(Matrix)
library(microbenchmark)

### No. 1 pt C

# Set N, P, X, W, and y

N <- 2000
P <-  500

X <- matrix(rnorm(N * P), nrow = N)
y <- matrix(rnorm(N), nrow = N)
W <- diag(rep(1, N))

# Inversion method

Inv.method <- function(X.Inv, W.Inv, y.Inv) {
    XtWX <- (t(X.Inv)*diag(W.Inv)) %*% X.Inv
    XtWY <- (t(X.Inv)*diag(W.Inv)) %*% y.Inv
    bhat.Inv <- solve(XtWX) %*% XtWY
    return(bhat.Inv)
}

Cho.decomp <- function(X.Cho, W.Cho, y.Cho) {
    D.Cho <- (t(X.Cho)*diag(W.Cho)) %*% y.Cho
    C.Cho <- (t(X.Cho)*diag(W.Cho)) %*% X.Cho

    U.Cho <- chol(C.Cho)
    L.Cho <- t(U.Cho)

    u <- forwardsolve(L.Cho, D.Cho)
    bhat.Cho <- backsolve(U.Cho, u)

    return(bhat.Cho)
}

microbenchmark(
    Inv.method(X, W, y),
    Cho.decomp(X, W, y),
    times=5)


### No. 1 pt D

N <- 2000
P <-  500

X <- matrix(rnorm(N * P), nrow = N)
mask <- matrix(rbinom(N * P, 1, 0.05), nrow = N)
```

```r
X <- mask * X

Inv.methodSPARSE <- function(X.Inv, W.Inv, y.Inv) {
    X <- Matrix(X, sparse = TRUE)
    XtWX <- (t(X.Inv)*diag(W.Inv)) %*% X.Inv
    XtWY <- (t(X.Inv)*diag(W.Inv)) %*% y.Inv
    bhat.Inv = Matrix::solve(XtWX, XtWY, sparse = TRUE)
    return(bhat.Inv)
}

Inv.methodSPARSE2 <- function(X.Inv, W.Inv, y.Inv) {
    XtWX <- (t(X.Inv)*diag(W.Inv)) %*% X.Inv
    XtWY <- (t(X.Inv)*diag(W.Inv)) %*% y.Inv
    bhat.Inv = solve(XtWX, XtWY)
    return(bhat.Inv)
}

microbenchmark(
    Inv.methodSPARSE(X, W, y),
    Inv.methodSPARSE2(X, W, y),
    Cho.decomp(X, W, y),
    times=5)


microbenchmark(
    solve(XtWX, XtWY),
    Matrix::solve(XtWX, XtWY, sparse = TRUE),
    solve(XtWX) %*% XtWY,
    times = 5
)



# END
```

```r
############################################################
######### Created by Spencer Woody on 24 Aug 2016 #########
############################################################

# Read in data file, standardize X

wdbc <- read.csv("wdbc.csv", header = FALSE)

X <- as.matrix(wdbc[, 3:12])
X <- scale(X)
X <- cbind(rep(1, nrow(X)), X)

y <- wdbc[, 2]
y <- y == "M"
beta <- as.matrix(rep(0, ncol(X)))
mi <- 1

# Function for computing w.i

comp.wi <- function (X, beta) {
    wi <- 1 / (1 + exp(-X %*% beta))
    return(wi)
}

# Function for computing likelihood

loglik <- function(beta, y, X, mi) {
    loglik <- apply((mi - y) * (X %*% beta)+ mi*log(1 + exp(-X %*% beta)), 2, sum)
    return(loglik)
}
# Function for computing gradient for likelihood

grad.loglik <- function(beta, y, X, mi){
  grad <- array(NA, dim = length(beta))
  wi <- comp.wi(X, beta)
  grad <- apply(X*as.numeric(mi * wi - y), 2, sum)
  return(grad)
}

###
### Gradient descent
###

stepfactor <- 0.025
n.steps <- 50000
log.lik <- NULL

for (step in 1:n.steps) {
    log.lik[step] <- loglik(beta, y, X, mi)
    beta <- beta - stepfactor * grad.loglik(beta, y, X, mi)
}

# Create trace plot of likelihood, check for convergence
```

```
png("beta_trace1.png")
plot(log.lik,
     main = "Trace Plot for log likelihood(beta)",
     xlab = "Step",
     ylab = "log likelihood(beta)")
dev.off()

# Compare results to R's glm function

mymodel <- glm(y ~ X[, c(-1)], family = "binomial")
summary(mymodel)

print(beta)

###
### Newton's method
###

beta.N <- as.matrix(rep(0, ncol(X)))

n.steps <- 10
log.lik2 <- NULL

for (step in 1:n.steps) {
    log.lik2[step] <- loglik(beta, y, X, mi)
    w.i <- as.numeric(comp.wi(X, beta.N))
    W <- diag(w.i*(1-w.i))
    Hessian <- t(X) %*% W %*% X
    beta.N <- beta.N - solve(Hessian) %*% grad.loglik(beta.N, y, X, mi)
}

# Show estimates from Newton's method

round(as.matrix(coef(mymodel)) - beta.N, 8)
```