SDS 385: Exercises 2: Online Learning

September 13, 2016

Professor Scott

Spencer Woody

Problem 1

(A) From the previous excercise, we have the gradient of β

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

We can think of $m_i w_i$ as the fitted value of y_i, or $\hat{y}_i$, when given β, so the gradient becomes

$$\begin{aligned}\nabla \ell(\beta) &= \sum_{i=1}^{N} (\hat{y}_i - y_i) x_i &(2)\\ &= \sum_{i=1}^{N} g_i(\beta) &(3)\\ g_i(\beta) &= (\hat{y}_i - y_i) x_i. &(4)\end{aligned}$$

(B)

$$\mathrm{E}(n g_i(\beta)) = n\mathrm{E}(g_i(\beta)) \tag{5}$$

In this expectation, the only random variable is i because the data X and y and the coefficients β are all fixed. The variable i is a random draw so it follows a discrete uniform distribution such that

$$P(i = j) = \begin{cases} \frac{1}{n} & j \in \{1, 2, \dots, n\} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Then we can compute the expectation.

$$\begin{aligned}\mathrm{E}(g_i(\beta)) &= \sum_{j=1}^{n} g_j(\beta) P(i = j) &(7)\\ &= \sum_{j=1}^{n} g_j(\beta) \frac{1}{n} &(8)\\ &= \frac{1}{n} \sum_{j=1}^{n} g_j(\beta) &(9)\\ &= \frac{1}{n} \nabla \ell(\beta), &(10)\\ \Rightarrow \mathrm{E}(n g_i(\beta)) = n\mathrm{E}(g_i(\beta)) &= n \frac{1}{n} \nabla \ell(\beta) = \nabla \ell(\beta) &(11)\end{aligned}$$

(C)

(D)

(E)

```
################################################################
######### Created by Spencer Woody on 13 Sep 2016 #########
################################################################

library(TTR)

# Read in data file, scale X (y = 1 represents a malignant tumor)

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

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

n <- nrow(X)

y <- wdbc[, 2]
y <- y == "M"
m.i <- 1

# # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # # # # # # # # # #

# Function for computing w.i (logit transform of Xtbeta)

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

# Function for computing likelihood

loglik <- function(beta, y, X, m.i) {
    loglik <- apply((m.i - y) * (X %*% beta)+ m.i*log(1 + exp(-X %*% beta)), 2, sum)
    return(loglik)
}

# Function for computing gradient of 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)
}



# # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # # # # # # # # # #
```

```
# # # # # # # # # # # # # # # # # # # # # # # # #

microbenchmark(
solve(Hessian, grad.loglik(as.matrix(rep(0, ncol(X))), y, X, m.i)),
qr.solve(Hessian, grad.loglik(as.matrix(rep(0, ncol(X))), y, X, m.i))
)
### NEWTON'S METHOD (results are referred to as beta.N)

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

newton.steps <- 10

for (step in 1:newton.steps) {
    Newton.wi <- as.numeric(comp.wi(X, beta.N))
    Hessian   <- t(X) %*% diag(Newton.wi*(1-Newton.wi)) %*% X
    beta.N    <- beta.N - solve(Hessian, grad.loglik(beta.N, y, X, m.i))
}

# Function for making traceplots of all betas

graph.betatrace <- function(beta, beta.N, graphname) {
    num.vars <- nrow(beta)
    n.graphrows <- floor(sqrt(num.vars))
    n.graphcols <- ceiling(num.vars / n.graphrows)
    pdf(graphname, width = n.graphcols * 2, height = n.graphrows * 2.5)
    par(mfrow = c(n.graphrows, n.graphcols), oma=c(0,0,2,0))
    for (j in 1:num.vars) {
        plot(beta[j, ],
            xlab = "iteration",
            ylab = paste("beta", sprintf("%i", j)),
            type = "l",
            ylim = c(min(beta.N[j], min(beta[j, ])) - 0.5,
            max(beta.N[j], max(beta[j, ])) + 0.5),
            log = "x"
            )
        abline(h = beta.N[j], col = "red")
    }
    title("Trace plot for all betas", outer = TRUE)
    dev.off()
}

# Stochastic gradient

SGD <- function(n.iterSGD,  beta.init, step.size, X, y, m.i) {
    #' Perform stochastic gradient descent for a binomial logistic regression
    #'
    #' @param n.iterSGD  Number of iterations to loop through
    #' @param beta.init  Initial guess for coefficients (betas)
    #' @param step.size  Constant step size
    #' @param X  N by P matrix of covariate data
    #' @param y  P-vector of responses
    #' @param m.i  n-parameter of binomial (1 for case of binary logistic)
    #'
```

```
    #' @return A matrix, each column is an iteration of computed betas.
    #
    #
        # Create zero matrix to store iterative values of beta; initialize beta
        betaSGD <- matrix(rep(0, ncol(X) * (n.iterSGD+ 1)), nrow = ncol(X))
        betaSGD[, 1] <- beta.init
        #
        for (step in 1:n.iterSGD) {

            # Draw random sample of single row of data (with replacement)
            i <- sample(nrow(X), 1)

            # Compute w.i, fitted value of p-parameter of binomial
            w.i <- 1 / (1 + exp(-crossprod(X[i, ], betaSGD[, step])))

            # Compute gradient, the descent direction
            grad <- nrow(X) * (m.i * w.i - y[i]) * X[i, ]

            # Next set of betas
            betaSGD[, step + 1] <- betaSGD[, step] - step.size * grad
    }
        return(betaSGD)
}


beta1 <- beta.N + (-1) ^ rbinom(ncol(X), 1, 0.5) * (10 + rexp(ncol(X), rate = 1))


sgd1 <- SGD(4e5, beta1, 0.0005, X, y, m.i)
graph.betatrace(sgd1, beta.N, "test.pdf")


trace <- loglik(sgd1, y, X, m.i)

# Plot exponential moving average of likelihood

plot(EMA(trace1, n = 100), type = "l", log = "xy")

# Decaying steps


SGD.decay <- function(n.iterSGD,  beta.init, C, t.0, alpha, X, y, m.i) {
    #' Perform stochastic gradient descent for a binomial logistic regression
    #'
    #' @param n.iterSGD  Number of iterations to loop through
    #' @param beta.init  Initial guess for coefficients
    #' @param C  Constant C in Robbins-Monro rule
    #' @param t.0  Constant t.0 in Robbins-Monro rule
    #' @param alpha Constant alpha in Robbins-Monro rule
    #' @param X  N by P matrix of covariate data
    #' @param y  P-vector of responses
    #' @param m.i  n-parameter of binomial (1 for case of binary logistic)
    #'
```

```
    #' @return A matrix, each column is an iteration of computed betas.
    #
    #
        # Create NULL matrix to store iterative values of beta; initialize beta
        betaSGD.decay <- matrix(rep(0, ncol(X) * (n.iterSGD+ 1)), nrow = ncol(X))
        betaSGD.decay[, 1] <- beta.init
        #
        for (step.decay in 1:n.iterSGD) {

            # Draw random sample of single row of data (with replacement)
            i <- sample(nrow(X), 1)

            # Compute w.i, fitted value of p—parameter of binomial
            w.i <- 1 / (1 + exp(-crossprod(X[i, ], betaSGD.decay[, step.decay])))

            # Compute gradient, the descent direction
            grad <- nrow(X) * (m.i * w.i - y[i]) * X[i, ]

            # Compute next step size
            stepsize.decay <- C * (step.decay + t.0) ^ -alpha

            # Next set of betas
            betaSGD.decay[, step.decay + 1] <- betaSGD.decay[, step] - stepsize.decay * grad
    }
        return(betaSGD.decay)
}

beta1 <- beta.N + (-1) ^ rbinom(ncol(X), 1, 0.5) * (3 + rexp(ncol(X), rate = 1))

decay1 <- SGD.decay(1e5, beta1, C = 10, t.0 = 1, alpha = 0.75, X, y, m.i)
graph.betatrace(decay1[, 10000:1e5], beta.N, "decay1.pdf")

# Running average of beta with decaying steps

burn.in <- 3e5
beta.burn <- sgd1[, burn.in:4e5]
beta.burnbar <- apply(beta.burn, 1, mean)

while (TRUE) {
    if (A & A | j > 1000) {
        break
    }
}
```