SDS 385: Exercises 3 - Better Online Learning (Preliminaries)

September 21, 2016

Professor Scott

Spencer Woody

Problem 1

(A) Let's frame our iterative solution to the minimization problem.

$$\beta_{k+1} = \beta_k + \alpha_k p_k, \tag{1}$$

where α_k is the *step length* and p_k is the *descent direction* such that

$$p_k = -B_k^{-1} \nabla \ell(\beta_k). \tag{2}$$

B_k is some symmetric, nonsingular matrix. In the case of gradient descent it is simply the identity matrix, and in the case of (exact) Newton's method, it is the Hessian matrix.

To make reasonable progress in reducing likelihood, our step length must meet *Wolfe's conditions*, of which there are two:

Sufficient decrease, (a.k.a. Armijo's condition)

$$\ell(\beta_k + \alpha_k p_k) \leq \ell(\beta_k) + c_1 \alpha_k \nabla \ell(\beta_k)^T p_k, \text{ and} \tag{3}$$

Curvature condition

$$\nabla \ell(\beta_k + \alpha_k p_k)^T p_k \geq c_2 \nabla \ell_k^T p_k. \tag{4}$$

Here, $c_1 \in (0, 1)$, $c_2 \in (0, 1)$, and $c_1 < c_2$. c_1 should be quite small. The Armijo condition ensures that we are indeed reducing the likelihood function. However, any fittingly small step size will meet this condition, so we also impose the curvature condition, which guarantees that we reduce the likelihood by moving the step length in a steep enough direction. In practice, we can start with a high value of α, and then reduce it by some factor until it meets the Armijo condition. Then we do not need the curvature condition.

```
Choose $\alpha_{max} > 0$, $\rho \in (0, 1)$, $c \in (0, 1)$ ;
Result: Return optimal $\alpha_k$
$\alpha_k \leftarrow \alpha_{max}$ ;
armijo.condition $\leftarrow$ ( $\ell(\beta_k + \alpha_k p_k) \leq \ell(\beta_k) + c\alpha_k \nabla \ell(\beta_k)^T p_k$ ) # Boolean value ;
while NOT armijo.condition do
    $\alpha_k \leftarrow \rho\alpha_k$;
    armijo.condition $\leftarrow$ ( $\ell(\beta_k + \alpha_k p_k) \leq \ell(\beta_k) + c\alpha_k \nabla \ell(\beta_k)^T p_k$ );
end
Return $\alpha_k$
```

Algorithm 1: Backtracking line search

(B) See Problem 2, part (B) for a discussion of the performance of backtracking line search as applied to the same data set from the previous two problem sets.

Problem 2

(A) The quasi-Newton method is used to approximate the Hessian matrix so as to perform an analogue to the Newton method. From Taylor's Theorem, we have it that

$$\nabla^2 \ell(\beta_{k+1} - \beta) \approx \nabla \ell(\beta_{k+1}) - \nabla \ell(\beta_k). \tag{5}$$

From the equation above, to approximate the Hessian matrix we impose the *secant condition*, i.e.,

$$B_{k+1} s_k = y_k \tag{6}$$

where $s_k = \beta_{k+1} - \beta_k$ and $y_k = \nabla \ell_{k+1} - \nabla \ell_k$. This ensures that our approximate Hessian matrix mimics the Taylor approximation characteristic of the true Hessian. Any approximation of the Hessian matrix should meet this criteria, as well as preserve symmetry. The Broyden-Fletcher-Goldfard-Shanno (BFGS) method for finding the Hessian fits both these critera. In this exercise, we use the BFGS method for finding the approximate *inverse* Hessian, H_{k+1}, so that we do not need to invert it later on to apply the analogue to Newton's method. Then we can find the direction using Eqn. (2) and using $B_k^{-1} = H_k$. The formula for approximating the inverse Hessian, taken from Nocedal & Wright page 25, is

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \; \rho_k = \frac{1}{y_k^T s_k}. \tag{7}$$

```
Choose line search parameters, initial $\beta$ values, number of iterations ($num.iter$) ;
Result: BFGS quasi-Newton estimates of $\beta$
Initialize $H_1 = I_p$ ;
Initialize $\nabla \ell_1$ from initial $\beta_1$ ;
for $k = 1$ to $num.iter$ do
    Store $\ell_k$ ;
    $p_k \leftarrow -H_k \nabla \ell_k$ ;
    Obtain $\alpha_k$ from line search function ;
    $\beta_{k+1} = \beta_k + \alpha_k p_k$ ;
    $\nabla \ell_{k+1} = \nabla \ell(\beta_{k+1})$ ;
    $y_k \leftarrow \nabla \ell_{k+1} - \nabla \ell_k$ ;
    $s_k \leftarrow \beta_{k+1} - \beta_k$ ;
    Compute $H_{k+1}$ using BFGS formula ;
    if convergence criteria reached then
        break;
end
Return $\beta$ estimate matrix
```

Algorithm 2: BFGS inverse Hessian method

(B) Our initial guess for β, $\hat{\beta}_0$ is some distance from the result of β from Newton's method. That distance is 5 plus some noise from the Exp(1) distribution in either the positive or negative direction.

For the line-search algorithm, we set $c = 0.001$, $\alpha_{max} = 1, \rho = 0.5$. We initialize the approximation of the inverse Hessian matrix with the identity matrix.

Log-likelihood results from our optimation techniques are shown in Figure 1. A dot is put on each line on the point at which convergence is determined to be reached. There were 18,430 iterations for gradient descent, 4,651 iterations for gradient descent with line search (a 75% reduction compared to regular GD), and just 1,438 iterations for quasi-Newton method with line search (a 92% reduction compared to regular GD). Our heuristic for reaching convergence is a reduction in log-likelihood of less than 10^{-7}. Clearly, quasi-Newton requires far, far fewer iterations to reach convergence. It is a middle

Problem 2 continued on next page. . .

ground between gradient descent and Newton's method, which only required fewer than 10 iterations to converge. However, in quasi-Newton, we do not need to invert any matrices, and therefore this approach is more numerically stable and less computationally expensive. This is the major trade-off between Newton and quasi-Newton.

Interestingly, the log-likelihood for gradient descent with a fixed step size decreases far faster than the other two methods, and even the log-likelihood for gradient descent with line search falls much faster than that for quasi-Newton. This is perhaps because, with quasi-Newton, our initial guess for the inverse Hessian is the identity matrix, which is likely very far from the truth. It takes a few iterations for our approximation to line up reasonably well with the truth.

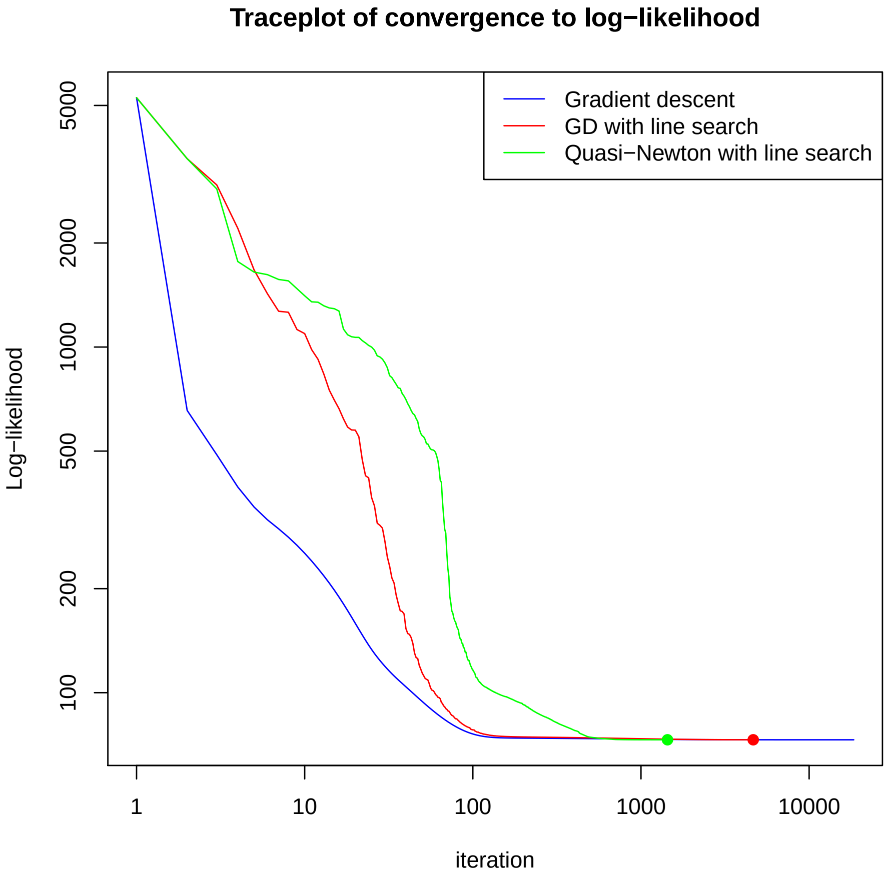

Figure 1: Comparative log-likelihood plots for gradient descent, gradient descent with line search, and quasi-Newton method with line search

R script linesearch.R

```
############################################################
######### Created by Spencer Woody on 18 Sep 2016 #########
############################################################

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

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

# Function for full likelihood

lik <- 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 <- 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)
}

# Hessian function

Hess <- function(beta, y, X, mi) {
    w.i <- as.numeric(comp.wi(X, beta))
    Hessian <- t(X) %*% diag(w.i*(1-w.i)) %*% X
    return(Hessian)
}


# Line search algorithm

linesearch <- function(beta.k, y, X, m.i,
    direct, lik.k, grad.k,
    c = 0.001, max.alpha = 1, rho = 0.75) {
    best.alpha <- max.alpha
    # Initial Boolean value for armijo condition
    armijo.cond <-   ( lik(beta.k + best.alpha * direct, y, X, m.i)
                     <= lik.k + c * best.alpha * crossprod(grad.k, direct) )
    while (!armijo.cond) {
        # Reduce alpha, recompute Boolean value for Armijo condition
        best.alpha <- rho * best.alpha
        armijo.cond <-   ( lik(beta.k + best.alpha * direct, y, X, m.i)
                         <= lik.k + c * best.alpha * crossprod(grad.k, direct) )
    }
    return(best.alpha)
```

```r
}


# Gradient descent function

grad.desc <- function(X, y, m.i, beta.init, n.iter, stepfactor) {
    p <- ncol(X)
    beta <- matrix(rep(NA, p * (n.iter + 1)), nrow = p)
    beta[, 1] <- beta.init

    lik.trace <- rep(NA, n.iter + 1)
    lik.trace[1] <- lik(beta[, 1], y, X, m.i)

    message <- "Convergence not reached!"
    for (iter in 1:n.iter) {
        beta.iter <- beta[, iter]

        # Calculate gradient and new beta, update beta and likelihood
        grad.iter <- grad(beta.iter, y, X, m.i)

        newbeta <- beta.iter - stepfactor * grad.iter
        beta[, iter + 1] <- newbeta
        lik.trace[iter + 1] <- lik(newbeta, y, X, m.i)

        # Convergence check
        lik.diff <- lik.trace[iter] - lik.trace[iter + 1]

        if (lik.diff < 1e-7) {
            beta <- beta[, -(iter:ncol(beta))] # remove excess cols of beta
            lik.trace <- lik.trace[-(iter:length(lik.trace))] # same for lik
            message <- sprintf("Stopped after %i iterations", iter)
            break
        }
    }
    mylist <- list(Beta = beta, Lik.trace = lik.trace, Conv = message)
    return(mylist)
}

# Gradient descent function with line search

gd.line <- function(X, y, m.i, beta.init, n.iter) {
    # Initialize beta matrix and likelihood trace
    p <- ncol(X)
    beta <- matrix(rep(NA, p * (n.iter + 1)), nrow = p)
    beta[, 1] <- beta.init

    lik.trace <- rep(NA, n.iter + 1)
    lik.trace[1] <- lik(beta[, 1], y, X, m.i)

    message <- "Convergence not reached!"
    for (iter in 1:n.iter) {
        beta.iter <- beta[, iter]

        # Calculate gradient and direction
        grad.iter <- grad(beta.iter, y, X, m.i)
        direct.iter <- -(grad.iter)

        # Store current likelihood
        lik.iter <- lik.trace[iter]

        # Perform linesearch
        stepsize.iter <- linesearch(beta.iter, y, X, m.i,
                                    direct.iter, lik.iter, grad.iter)

        # Update beta using calculated stepsize and direction
        newbeta <- beta.iter + stepsize.iter * direct.iter

        # Store new values of beta and likelihood
        beta[, iter + 1] <- newbeta
        lik.trace[iter + 1] <- lik(newbeta, y, X, m.i)

        # Convergence check
        lik.diff <- lik.trace[iter] - lik.trace[iter + 1]
        if (lik.diff < 1e-7) {
            beta <- beta[, -(iter:ncol(beta))] # remove excess cols of beta
            lik.trace <- lik.trace[-(iter:length(lik.trace))] # same for lik
            message <- sprintf("Stopped after %i iterations", iter)
            break
        }
    }
    mylist <- list(Beta = beta, Lik.trace = lik.trace, Conv = message)
    return(mylist)
}

# Quasi—Newton method

qn.line <- function(X, y, m.i, beta.init, n.iter) {
    # Initialize beta matrix, likelihood trace, and gradient
    p <- ncol(X)
    beta <- matrix(rep(NA, p * (n.iter + 1)), nrow = p)
    beta[, 1] <- beta.init

    lik.trace <- rep(NA, n.iter + 1)
    lik.trace[1] <- lik(beta[, 1], y, X, m.i)

    grad.new <- grad(beta.init, y, X, m.i)

    # Initialize inverse Hessian approximation with identity matrix
    id.mat <- diag(1, p)
    Hk <- id.mat

    message <- "Convergence not reached!"
    for (iter in 1:n.iter) {
        # "New" values for grad and beta from last iteration become "old" values
        grad.old <- grad.new
        beta.old <- beta[, iter]
        lik.iter <- lik.trace[iter]

        # Calculate direction from previous gradient and inverse Hessian
        direct.iter <- - Hk %*% grad.old

        # Perform linesearch
        stepsize.iter <- linesearch(beta.old, y, X, m.i,
                                    direct.iter, lik.iter, grad.old)

        # Create new estimate of beta from calculated stepsize and direction
        beta.new <- beta.old + stepsize.iter * direct.iter

        # Update gradient
        grad.new <- grad(beta.new, y, X, m.i)

        # Update betas and likelihood trace
        beta[, iter + 1] <- beta.new
        lik.trace[iter + 1] <- lik(beta.new, y, X, m.i)

        # Update Hk for next iteration
        y.k <- grad.new - grad.old
        s.k <- beta.new - beta.old

        ys.k <- tcrossprod(y.k, s.k)
        rho.k <- as.numeric(1 / (crossprod(y.k, s.k)))
        Hk <- (id.mat - rho.k * ys.k) %*% Hk %*% (id.mat - rho.k * t(ys.k)) +
               rho.k * s.k %*% t(s.k) # Formula from N&W p. 25

        # Convergence check
        lik.diff <- lik.trace[iter] - lik.trace[iter + 1]
        if (lik.diff < 1e-7) {
            beta <- beta[, -(iter:ncol(beta))] # remove excess cols of beta
            lik.trace <- lik.trace[-(iter:length(lik.trace))] # same for lik
            message <- sprintf("Stopped after %i iterations", iter)
            break
        }
    }
    mylist <- list(Beta = beta, Lik.trace = lik.trace, Conv = message)
    return(mylist)
}
```

R script for `exercises03.R`

```
############################################################
######### Created by Spencer Woody on 18 Sep 2016 #########
############################################################

# Import functions from linesearch.R

source("linesearch.R")

# 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)

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

# Obtain estimates of beta from Newton’s method

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(beta.N, y, X, m.i))
}

# Produce initial guesses for betas, based on values from Newton’s method.

beta0 <- beta.N + (-1) ^ rbinom(ncol(X), 1, 0.5) * (5 + rexp(ncol(X), rate = 1))

# Perform minimization techniques

gd  <- grad.desc(X, y, m.i, beta0, 5e4, 0.025)
gd.l <- gd.line(X, y, m.i, beta0, 5e4)
qn.l <- qn.line(X, y, m.i, beta0, 5e4)


gd.lik   <- gd$Lik.trace
gd.l.lik <- gd.l$Lik.trace
qn.l.lik <- qn.l$Lik.trace

pdf("complikplot.pdf")
plot(gd.lik,
    log = "xy",
    type = "l",
    col = "blue",
    xlab = "iteration",
    ylab = "Log-likelihood",
    main = "Traceplot of convergence to log-likelihood")
lines(gd.l.lik, col = "red")
lines(qn.l.lik, col = "green")
points(length(gd.lik), bb[length(gd.lik)], col = "blue", pch = 19)
points(length(gd.l.lik), gd.l.lik[length(gd.l.lik)], col = "red", pch = 19)
points(length(qn.l.lik), qn.l.lik[length(qn.l.lik)], col = "green", pch = 19)
legend("topright",
    c("Gradient descent",
    "GD with line search",
    "Quasi-Newton with line search"),
    lty = c(1, 1, 1),
    col = c("blue", "red", "green"))
dev.off()
```