SDS 385: Homework 2

G. Paulon

September 8, 2016

Problem 1. SGD for logistic regression

(A) In part (A) of the last homework we proved that the gradient of the negative log-likelihood can be expressed as

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

where

$$g_i(\beta) = x_i(y_i - m_i w_i) = x_i(y_i - \hat{y}_i)$$

and

$$\hat{y}_i = \mathbb{E}(y_i|\beta) = m_i w_i(\beta) = m_i \frac{1}{1+\exp(-x_i^T\beta)}.$$

A nice interpretation can be given to this latter expression: the gradient is large when the data y_i's differ from their maximum likelihood estimates, i.e. the probabilities w_i's.

(B) Suppose that we draw a single data point at random from the sample, obtaining the pair $\{y_i, x_i\}$. We show that the random vector $ng_i(\beta)$ is an unbiased estimate of $\nabla l(\beta)$, that is, $E[ng_i(\beta)] = \nabla l(\beta)$, where the expectation is under random sampling from the set of all $\{y_i, x_i\}$ pairs. In fact,

$$\begin{aligned} E[ng_i(\beta)] &= E[n(x_i y_i - x_i m_i \frac{1}{1+e^{-x_i^T\beta}})] \\ &= \sum_{i=1}^{n} \left\{ n \left(x_i y_i - x_i m_i \frac{1}{1+e^{-x_i^T\beta}} \right) \frac{1}{n} \right\} \\ &= \sum_{i=1}^{n} \left\{ x_i y_i - x_i m_i \frac{1}{1+e^{-x_i^T\beta}} \right\} \\ &= \sum_{i=1}^{n} g_i(\beta) = \nabla l(\beta). \end{aligned}$$

(C) The SGD exploits this fact in order to use an update step which is faster to compute. In fact, instead of using the gradient $\nabla l(\beta)$ calculated from all n data points to choose the step direction, we use the gradient $g_i(\beta)$ calculated from a single data point, sampled randomly from the whole data set. In this version of the algorithm, sampling without replacement has been performed.

In order to asses the validity of the implementation, we run the algorithm on the real data for a reasonable number of iterations $N = 100000$. We tried different values for the step size γ, which was kept constant over the iterations. We report in Figures 1 - 3 the traceplots of all of the β parameters (i.e. the values of the parameters over the iterations). In order to facilitate the visualization, we thinned the samples by a factor 20 (we displayed one value of β every 20 iterations).

We can see that for small values of the step size γ, convergence is not reached for many of the components of β. When we increase the step size, on the other hand, convergence is reached more rapidly because the algorithm explores well the parameters space. However, the asymptotic variance of the parameters is higher, since at every iteration the weight attributed to the direction given by any new sampled data is high.

In this first implementation of the algorithm we do not worry about the choice of the tuning parameter, that is instead crucial for SGD. As a convergence diagnostics, we plot the iterates of the running average of $l_t(\beta)$, the individual log-likelihood contribution from the data point sampled at step t (see Figure 4).

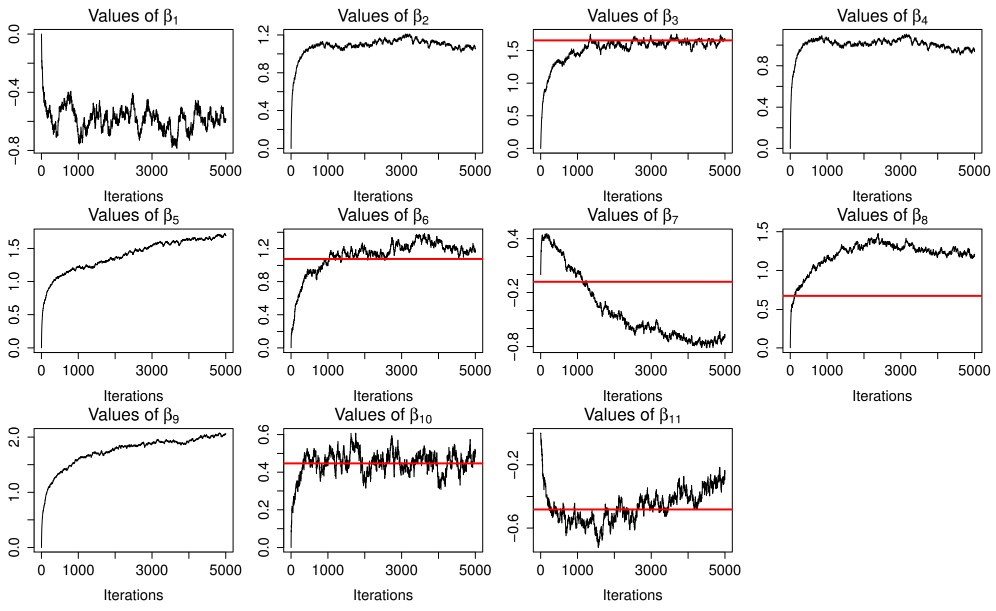

Figure 1: Traceplots of the β parameters when $\gamma = 0.01$.

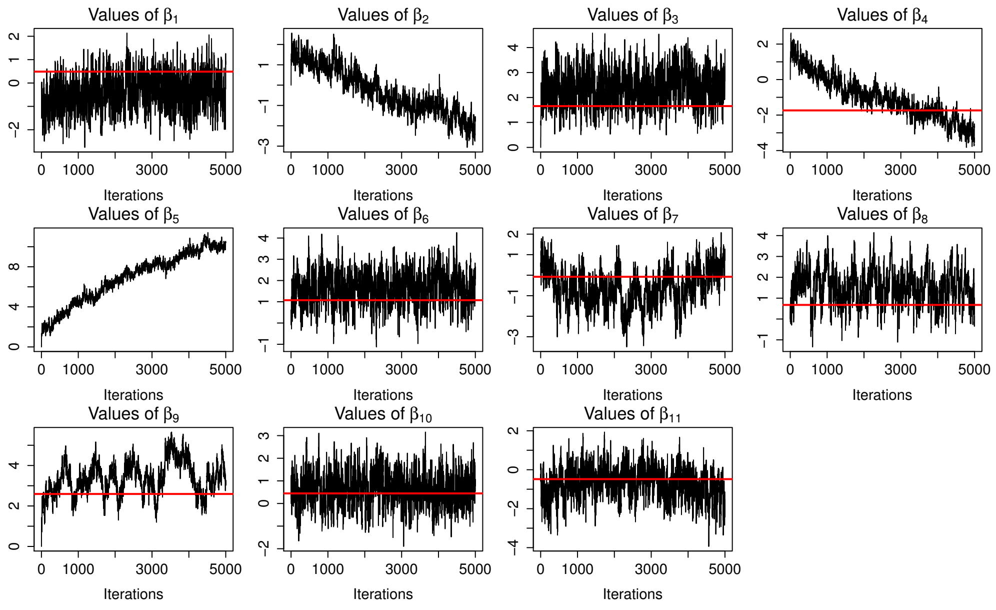

Figure 2: Traceplots of the β parameters when $\gamma = 0.5$.

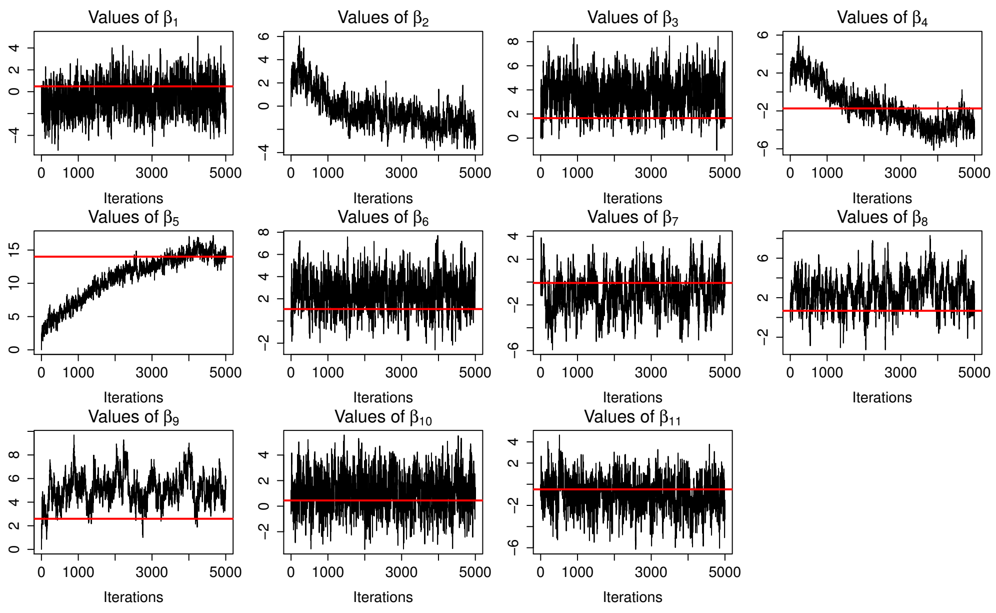

Figure 3: Traceplots of the β parameters when $\gamma = 1$.

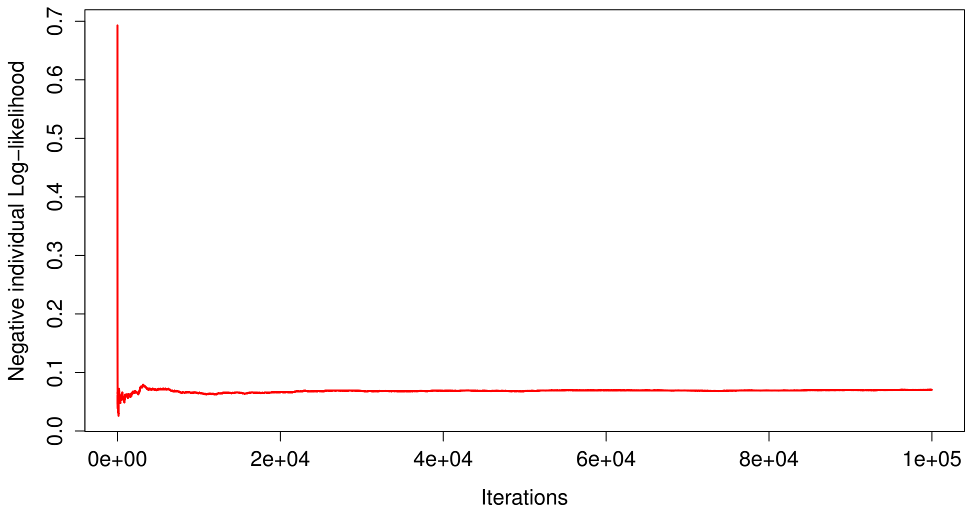

Figure 4: Running average of the single log-likelihood contributions.

(D) Now we try using a decaying step size. Specifically, we use the Robbins–Monro rule for step sizes:

$$\gamma(t) = C(t + t_0) - \alpha,$$

where $C > 0$, $\alpha \in [0.5, 1]$, and t_0 (the "prior number of steps") are constants. The exponent α is usually called the learning rate.

Ideally, we want to obtain large values of γ at the beginning of the algorithms, so that the "true" values of the parameters are rapidly reached. Afterwards, we want the step sizes to be reduced in order to diminish the variance of the estimates and to, eventually, converge.

After several trials, we set the values $\alpha = 0.8, C = 4000, t_0 = 1000$, yielding the results displayed in Figure 5.

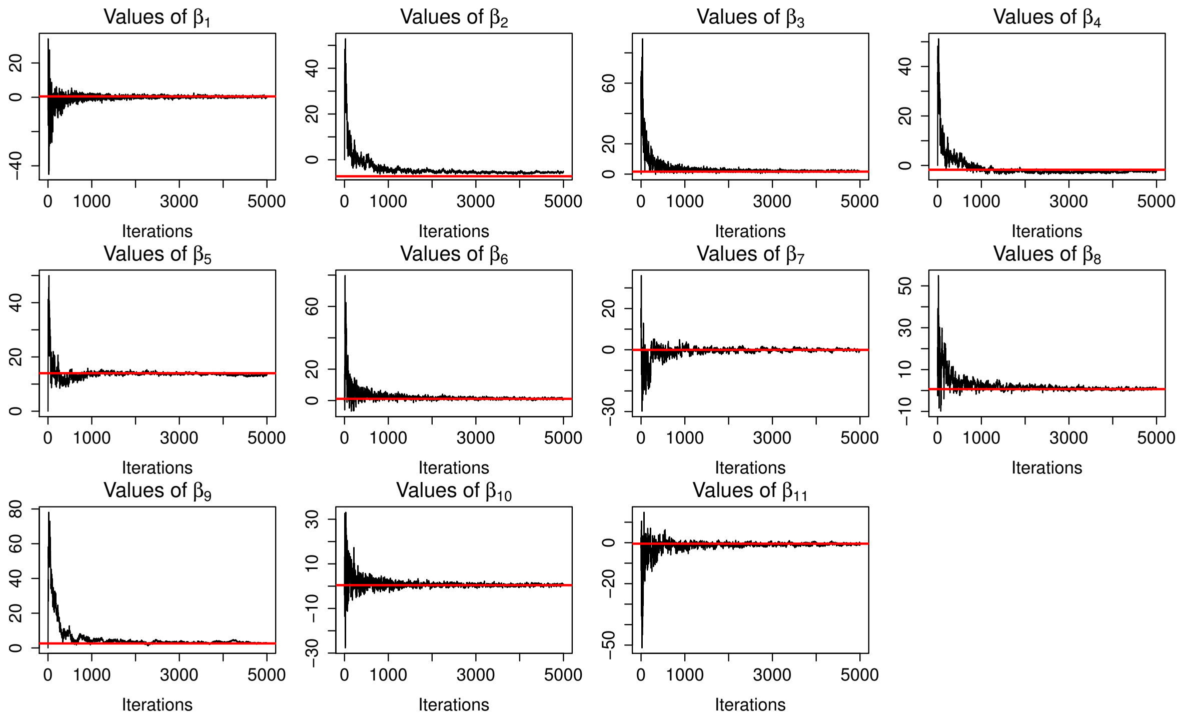

Figure 5: Running average of the single log-likelihood contributions.

(E) At last, in order to estimate the parameters, we report the time-averaged iterates instead of the iterates. The comparison between all the algorithms is found in Listing 1.

```
                        Beta0     Beta1    Beta2     Beta3    Beta4    Beta5       Beta6
Glm                 0.4870168 -7.221851 1.654756 -1.737630 14.00485 1.074953 -0.07723455
Gradient            0.4830816 -7.020847 1.654900 -1.922239 13.97312 1.073116 -0.06648360
Newton              0.4870168 -7.221851 1.654756 -1.737630 14.00485 1.074953 -0.07723455
SGD                 3.5570133 -3.245963 2.779636 -2.102697 17.71371 1.398131 -0.39040924
SGD Robbins-Monro   0.5527124 -5.426020 2.036747 -2.446775 13.33293 1.152240  0.06014270
SGD Polyak-Ruppert 0.2865011 -5.538449 1.980589 -2.581991 13.60858 1.281796 -0.05755391
                        Beta7    Beta8     Beta9     Beta10
Glm                 0.6751231 2.592874 0.4462563 -0.4824842
Gradient            0.6767472 2.595408 0.4459843 -0.4832114
Newton              0.6751231 2.592874 0.4462563 -0.4824842
SGD                 1.2890052 5.722428 1.7388858  0.4833998
SGD Robbins-Monro   1.0255726 2.835010 0.5391915 -0.4159569
SGD Polyak-Ruppert 0.9174220 3.249271 0.5015554 -0.6899736
```

Listing 1: Comparison of the estimates using five different algorithms

Appendix A

R code

```r
# Function for the SGD of the logit model
SGD <- function(y, X, mi, beta0, maxiter, alpha, tol){

  N <- dim(X)[1]
  # Sample the data points to calculate the gradient
  idx <- sample(1:N, maxiter, replace = TRUE)

  betas <- array(NA, dim=c(maxiter, length(beta0)))
  betas[1,] <- beta0 # Initial guess

  av_ll <- array(NA, dim = maxiter)
  av_ll[1] <- log_lik(betas[1,], matrix(y[idx[1]],nrow=1), matrix(X[idx[1],],nrow=1), mi[idx[1]])

  for (iter in 2:maxiter){
    gradient <- grad_loglik(betas[iter-1,], matrix(y[idx[iter]],nrow=1), matrix(X[idx[iter],],
        nrow=1), mi[idx[iter]])
    betas[iter,] <- betas[iter-1,] - alpha*gradient
    ll <- log_lik(betas[iter,], matrix(y[idx[iter]],nrow=1), matrix(X[idx[iter],],nrow=1), mi[idx
        [iter]])
    av_ll[iter] <- ((av_ll[iter-1])*(iter-1) + ll)/iter
  }
  return(list("ll" = av_ll, "beta" = betas))
}
```

Listing A.1: Function implementing the SGD for the logit model.

```r
# Function for the SGD of the logit model
SGD_Robbins <- function(y, X, mi, beta0, maxiter, C, t0, alpha, tol){

  N <- dim(X)[1]
  # Sample the data points to calculate the gradient
  idx <- sample(1:N, maxiter, replace = TRUE)

  betas <- array(NA, dim=c(maxiter, length(beta0)))
  betas[1,] <- beta0 # Initial guess

  av_ll <- array(NA, dim = maxiter)
  av_ll[1] <- log_lik(betas[1,], matrix(y[idx[1]],nrow=1), matrix(X[idx[1],],nrow=1), mi[idx[1]])

  for (iter in 2:maxiter){
    step_size <- C*(iter+t0)^(-alpha)

    gradient <- grad_loglik(betas[iter-1,], matrix(y[idx[iter]],nrow=1), matrix(X[idx[iter],],
        nrow=1), mi[idx[iter]])
    betas[iter,] <- betas[iter-1,] - step_size*gradient
    ll <- log_lik(betas[iter,], matrix(y[idx[iter]],nrow=1), matrix(X[idx[iter],],nrow=1), mi[idx
        [iter]])
    av_ll[iter] <- ((av_ll[iter-1])*(iter-1) + ll)/iter
  }
  return(list("ll" = av_ll, "beta" = betas))
}
```

Listing A.2: Function implementing the SGD for the logit model using the Robbins-Monro rule.