Lab 1

August 11, 2016

Part 1. We will work through some details on logistic regression. We first set some notation. Let $x_i \in \mathbb{R}^p, y_i \in \{0, 1\}$ for $i = 1, \ldots, n$. Let $X \in \mathbb{R}^{n \times p}$ denote the matrix with x_i as its ith row. Recall that the negative log-likelihood $\ell(\beta)$ can be written as follows:

$$\ell(\beta) = \sum_{i=1}^{n} \left[-y_i x_i^T \beta + \log(1 + \exp(x_i^T \beta)) \right].$$

1. Write the gradient and Hessian of $\ell(\beta)$.

Answer:

$$\begin{aligned} \nabla \ell(\beta) &= \sum_{i=1}^{n} \left[\frac{\exp(x_i^T \beta)}{1 + \exp(x_i^T \beta)} - y_i \right] x_i \\ &= X^T [p(\beta) - y], \end{aligned}$$

where the ith element of the predicted value vector $p(\beta)$ is given by

$$[p(\beta)]_i = \frac{\exp(x_i^T \beta)}{1 + \exp(x_i^T \beta)}.$$

$$\begin{aligned} \nabla^2 \ell(\beta) &= \sum_{i=1}^{n} \left[\frac{\exp(x_i^T \beta)}{1 + \exp(x_i^T \beta)} \frac{1}{1 + \exp(x_i^T \beta)} \right] x_i x_i^T \\ &= X^T W(\beta) X, \end{aligned}$$

where $W(\beta)$ is a diagonal matrix with ith diagonal entry $w_{ii} = [p(\beta)]_i [1 - p(\beta)]_i$.

2. What is the computational complexity for a calculating the gradient and Hessian of $\ell(\beta)$?

Answer:

The gradient costs $\mathcal{O}(np)$ to compute, and the Hessian costs $\mathcal{O}(np^2)$ to compute.

3. Under what condition is $\ell(\beta)$ strictly convex?

Answer: $\ell(\beta)$ is strictly convex if and only if $\nabla^2 \ell(\beta)$ is positive definite for all β. For any $z \neq 0$ we require that

$$z^T X^T W(\beta) X z > 0.$$

Since $W(\beta)$ is positive definite, we just need to make sure that $Xz \neq 0$ for $z \neq 0$. Thus, $\ell(\beta)$ is strictly convex whenever X has full column rank.

4. Prove that $\ell(\beta)$ is L-Lipschitz differentiable with $L = \frac{1}{4} \|X\|_{\text{op}}^2$.

Answer: Consider the Taylor expansion of $\nabla \ell(\beta)$ at $\tilde{\beta} \neq \beta$.

$$\nabla \ell(\beta) = \nabla \ell(\tilde{\beta}) + \nabla^2 \ell(\beta^\star)(\beta - \tilde{\beta}),$$

where $\beta^\star = \alpha\beta + (1-\alpha)\tilde{\beta}$ for some $\alpha \in (0,1)$. Rearranging the Taylor expansion and taking the 2-norm of both sides gives

$$\begin{aligned}
\|\nabla \ell(\beta) - \nabla \ell(\tilde{\beta})\|_2 &= \|\nabla^2 \ell(\beta^\star)(\beta - \tilde{\beta})\|_2 \\
&= \|\nabla^2 \ell(\beta^\star)\frac{\beta - \tilde{\beta}}{\|\beta - \tilde{\beta}\|_2}\|_2 \|\beta - \tilde{\beta}\|_2 \\
&\leq \left[\sup_{\|v\|_2 = 1} \|\nabla^2 \ell(\beta^\star) v\|_2\right] \|\beta - \tilde{\beta}\|_2 \\
&= \|\nabla^2 \ell(\beta^\star)\|_{\text{op}} \|\beta - \tilde{\beta}\|_2.
\end{aligned}$$

Note that $\|A^T A\|_{\text{op}} = \|A\|_{\text{op}}^2$. This is straightforward to show using the SVD of A. Thus, $\|\nabla^2 \ell(\beta^\star)\|_{\text{op}} = \|W(\beta)^{1/2} X\|_{\text{op}}^2$. But

$$\begin{aligned}
\|W(\beta)^{1/2} X\|_{\text{op}} &= \sup_{\|v\|_2 = 1} \sqrt{v^T X^T W(\beta) X v} \\
&\leq \sup_{\|v\|_2 = 1} \sqrt{\frac{1}{4} v^T X^T X v} \\
&= \frac{1}{2} \sup_{\|v\|_2 = 1} \|Xv\|_2 \\
&\leq \frac{1}{2} \|X\|_{\text{op}}
\end{aligned}$$

The first inequality follows from the fact that for any z

$$z^T W z = \sum_{i=1}^n w_{ii} z_i^2 \leq \sum_{i=1}^n \frac{1}{4} z_i^2 = \frac{1}{4}\|z\|_2^2$$

5. Suppose that there is a vector $w \in \mathbb{R}^p$ such that $x_i^T w > 0$ if $y_i = 1$ and $x_i^T w < 0$ if $y_i = 0$. Prove that $\ell(\beta)$ does not have a global minimum. In other words, when the classification problem is completely separable, there is no maximum likelihood estimator.

Answer: It is easier to see what is happening in the completely separable case with the likelihood.

$$L(\beta) = \prod_{i=1}^n f(x_i^T \beta)^{y_i} (1 - f(x_i^T \beta))^{1 - y_i},$$

where $f(u) = \exp(u)/(1 + \exp(u))$. There are a couple of facts that you should verify:

(i) $f(u)$ takes on values strictly between 0 and 1.

(ii) $f(u)$ is strictly increasing for all u.

Fact (i) implies that $L(\beta)$ is strictly less than 1. We will show, however, that there is a sequence β_n such that $L(\beta_n)$ converges to 1. The fact that $L(\beta) < 1$ for all β but there is a sequence of β_n such that $L(\beta_n) \rightarrow 1$ proves that there is no maximum likelihood estimator.

So, if Student A says that I have found it! I have found a maximum likelihood estimator β_{M}. Student B will counter that he can construct an estimator β_{T} that bests β_{M}, namely $L(\beta_{\text{T}}) > L(\beta_{\text{M}})$. Here's how Student B constructs β_{T}. Let $\epsilon = 1 - L(\beta_{\text{M}})$. Since $L(\beta_{\text{M}}) < 1$, it follows that $\epsilon > 0$. Student B then selects n_{T} sufficiently large such that $|L(\beta_{n_{\text{T}}}) - 1| < \epsilon$ and sets $\beta_{\text{T}} = \beta_{n_{\text{T}}}$. Then

$$1 - L(\beta_{\text{T}}) < \epsilon = 1 - L(\beta_{\text{M}}).$$

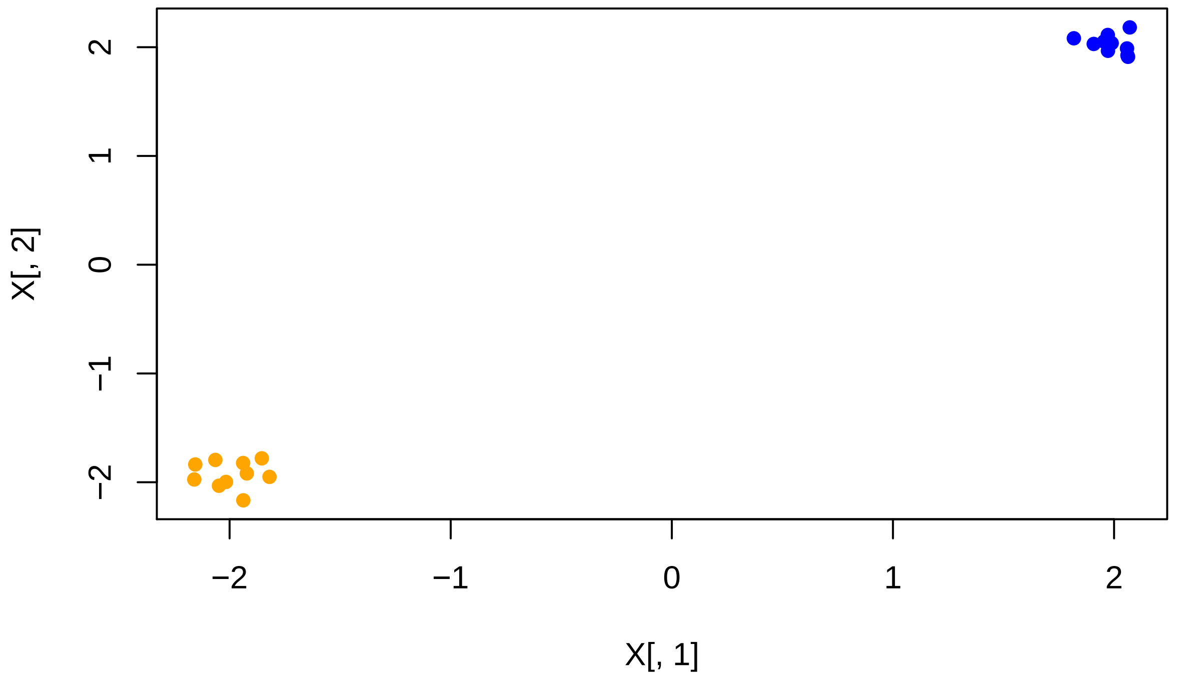
Rearranging the inequality above gives $L(\beta_{\text{M}}) < L(\beta_{\text{T}})$ and ends the discussion.

Student B's construction of β_{T}, however, relies on the existence of a sequence β_n such that $L(\beta_n) \to 1$. We provide a constructive proof of such a sequence. Take $\beta_n = nw$. Note in the argument that follows we could have taken any monotonically increasing function of n, e.g. $\beta_n = n^2 w$ or $\beta_n = n\log(n)w$ would work as well.

Consider the likelihood evaluated at β_n. If the ith point has $y_i = 1$, then the ith point contributes to the likelihood the term
$$f(x_i^T\beta_n) = f(nx_i^T w).$$
Using Fact (ii) and the fact that $x_i^T w > 0$ in this case, $\lim_{n\to\infty} f(nx_i^T w) = 1$. If the ith point has $y_i = 0$, then the ith point contributes to the likelihood the term
$$1 - f(x_i^T\beta_n) = 1 - f(nx_i^T w).$$
Using Fact (ii) and the fact that $x_i^T w < 0$ in this case, $\lim_{n\to\infty}[1 - f(nx_i^T w)] = 1$. Therefore, when we consider all n data points in the likelihood we arrive at the conclusion that $L(\beta_n) \to 1$.

A natural follow up question is does this result have any practical consequences? Consider what happens when you perform logistic regression using the glm function on a separable problem.

```
set.seed(12345)
n <- 10; p <- 2
mu1 <- matrix(2,n,p)
mu2 <- - mu1
X1 <- mu1 + 0.1*matrix(rnorm(n*p),n,p)
X2 <- mu2 + 0.1*matrix(rnorm(n*p),n,p)
X <- rbind(X1,X2)
y <- c(rep(1,n),rep(0,n))
plot(X[,1],X[,2],type='n')
points(X1[,1],X1[,2],col='blue',pch=16)
points(X2[,1],X2[,2],col='orange',pch=16)
```



```
model <- glm(y~X, family='binomial')
model
```

```
##
## Call:  glm(formula = y ~ X, family = "binomial")
##
## Coefficients:
## (Intercept)           X1           X2
##     -0.5149       6.2069       6.8350
##
## Degrees of Freedom: 19 Total (i.e. Null);  17 Residual
## Null Deviance:       27.73
## Residual Deviance: 3.277e-10     AIC: 6
```

You see that the regression coefficients are very large. The fitted values are correspondingly very close to 1 and 0 (Remember machine precision is 10^{-16}).

```
model$fitted.values
```

```
##            1            2            3            4            5
## 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
##            6            7            8            9           10
## 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
##           11           12           13           14           15
## 7.933668e-12 3.110683e-11 7.636957e-12 3.267635e-12 1.239135e-12
##           16           17           18           19           20
## 1.204230e-11 1.668339e-12 1.324903e-12 1.374240e-11 2.583607e-12
```

If the glm code did not stop at 24 iterations the regression coefficients would diverge to infinity as the likelihood can be made arbitrarily close to 1.

Part 2. Gradient Descent

You will now implementation of gradient descent to your R package. Your function will include using both a fixed step-size as well as one chosen by backtracking.

Please complete the following steps.

Step 1: Write a function "gradient_step."

```r
#' Gradient Step
#'
#' @param gradf handle to function that returns gradient of objective function
#' @param x current parameter estimate
#' @param t step-size
gradient_step <- function(gradf, t, x) {

}
```

Your function should return $x^+ = x - t\nabla f(x)$.

Step 2: Write a function "gradient_descent_fixed." The function should terminate either when a maximum number of iterations has been taken or if the relative change in the objective function has fallen below a tolerance

$$\frac{|f(x_k) - f(x_{k-1})|}{|f(x_{k-1})| + 1)} < \text{tolerance}$$

```r
#' Gradient Descent (Fixed Step-Size)
#'
#' @param fx handle to function that returns objective function values
#' @param gradf handle to function that returns gradient of objective function
#' @param x0 initial parameter estimate
#' @param t step-size
#' @param max_iter maximum number of iterations
#' @param tol convergence tolerance
gradient_descent_fixed <- function(fx, gradf, t, x0, max_iter=1e2, tol=1e-3) {

}
```

Your function should return

- The final iterate value
- The objective function values
- The 2-norm of the gradient values
- The relative change in the function values
- The relative change in the iterate values

Step 3: Write a function "backtrack."

```r
#' Backtracking
#'
#' @param fx handle to function that returns objective function values
#' @param x current parameter estimate
#' @param t current step-size
#' @param df the value of the gradient of objective function evaluated at the current x
#' @param alpha the backtracking parameter
#' @param beta the decrementing multiplier
backtrack <- function(fx, t, x, df, alpha=0.5, beta=0.9) {

}
```

Your function should return the selected step-size.

Step 4: Write a function "gradient_descent_backtrack" that performs gradient descent using backtracking.

```r
#' Gradient Descent (Backtracking Step-Size)
#'
#' @param fx handle to function that returns objective function values
#' @param gradf handle to function that returns gradient of objective function
#' @param x0 initial parameter estimate
#' @param max_iter maximum number of iterations
#' @param tol convergence tolerance
gradient_descent_backtrack <- function(fx, gradf, x0, max_iter=1e2, tol=1e-3) {

}
```

Your function should return

- The final iterate value
- The objective function values
- The 2-norm of the gradient values
- The relative change in the function values
- The relative change in the iterate values

Step 5: Write a function "gradient_descent" that is a wrapper function for "gradient_descent_fixed" and "gradient_descent_backtrack." The default should be to use the backtracking.

```r
#' Gradient Descent
#'
#' @param fx handle to function that returns objective function values
#' @param gradf handle to function that returns gradient of objective function
#' @param x0 initial parameter estimate
#' @param t step-size
#' @param max_iter maximum number of iterations
#' @param tol convergence tolerance
gradient_descent <- function(fx, gradf, x0, t=NULL, max_iter=1e2, tol=1e-3) {

}
```

Your function should return

- The final iterate value
- The objective function values
- The 2-norm of the gradient values
- The relative change in the function values
- The relative change in the iterate values

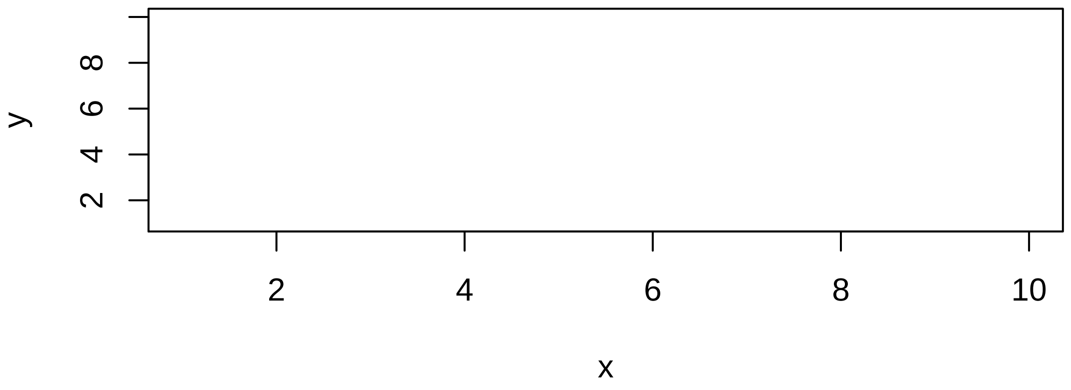
Step 6: Write functions 'fx_logistic' and 'gradf_logistic' to perform ridge logistic regression

```r
#' Objective Function for Logistic Regression
#'
#' @param y binary response
#' @param X design matrix
#' @param beta regression coefficient vector
#' @param lambda regularization parameter
fx_logistic <- function(y, X, beta, lambda=0) {

}

#' Gradient for Logistic Regession
#'
#' @param y binary response
#' @param X design matrix
#' @param beta regression coefficient vector
#' @param lambda regularization parameter
gradf_logistic <- function(y, X, beta, lambda=0) {

}
```

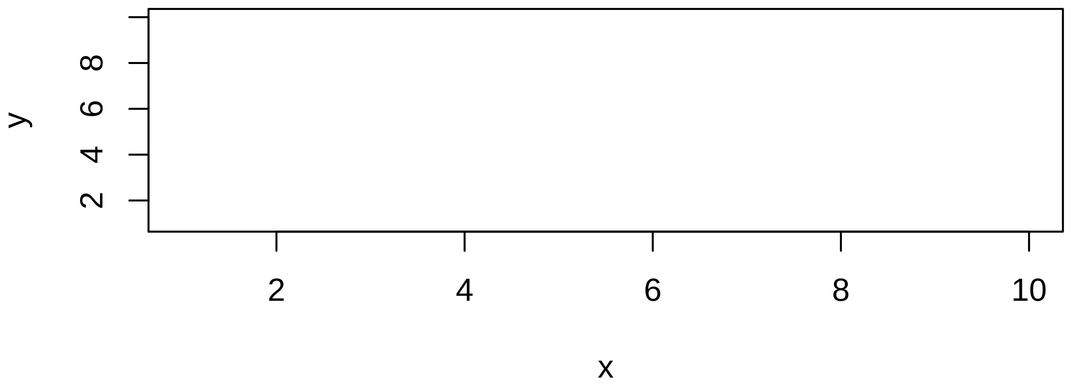
Step 7: Perform logistic regression (with $\lambda = 0$) on the following data example (y, X) using the fixed step-size. Use your answers to Part 1 to choose an appropriate fixed step-size. Plot the difference $\ell(\beta_k) - \ell(\beta_{10000})$ versus the iteration k. Comment on the shape of the plot given what you know about the iteration complexity of gradient descent with a fixed step size.

```r
set.seed(12345)
n <- 100
p <- 2
```

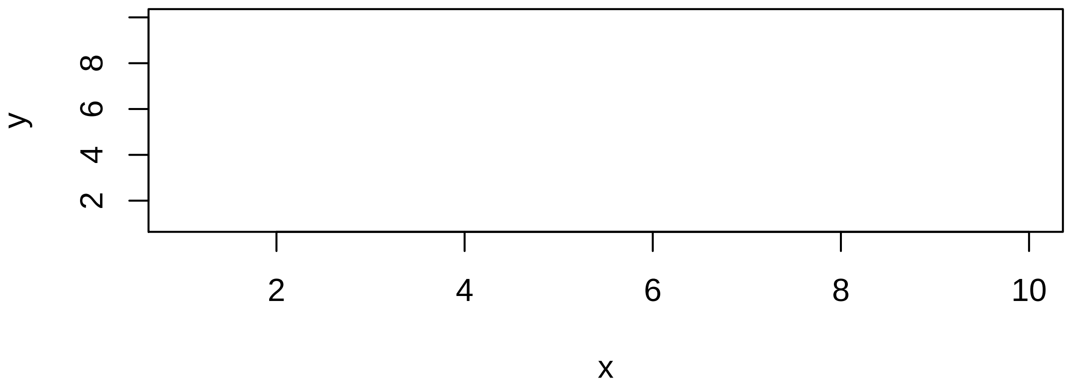
```r
X <- matrix(rnorm(n*p),n,p)
beta0 <- matrix(rnorm(p),p,1)
y <- (runif(n) <= plogis(X%*%beta0)) + 0
```



Step 8: Perform logistic regression (with $\lambda = 0$) on the simulated data above using backtacking. Plot the difference $\ell(\beta_k) - \ell(\beta_{10000})$ versus the iteration k. Comment on the shape of the plot given what you know about the iteration complexity of gradient descent with backtracking.



Step 9: Perform logistic regression (with $\lambda = 10$) on the simulated data above using the fixed step-size. Plot the difference $\ell_\lambda(\beta_k) - \ell_\lambda(\beta_{10000})$ versus the iteration k. Comment on the shape of the plot given what you know about the iteration complexity of gradient descent applied to strongly convex functions.



Part 3. If you have time, try out higher order methods. Install the R package lbfgs. How does the run time compare with gradient descent? You might find the 'system.time()' command useful. Try implementing Newton's method as discussed in lecture. How does the run time compare with gradient descent? BFGS?