Homework 5

Steven Xu

Due @ 5pm on December 7, 2018

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

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

1. Write the gradient and Hessian of $\ell(\boldsymbol{\beta})$. **Hint:** The Hessian $\nabla^2 \ell(\boldsymbol{\beta})$ can be written as $\mathbf{X}^{\mathsf{T}} \mathbf{W}(\boldsymbol{\beta}) \mathbf{X}$ where $\mathbf{W}(\boldsymbol{\beta})$ is a diagonal matrix that depends on $\boldsymbol{\beta}$.

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} [-y_i \mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta} + log(1 + exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta}))]$$

$$\nabla \ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} [-y_i \mathbf{x}_i + \frac{exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta}) \mathbf{x}_i}{1 + exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta})}]$$

$$= \sum_{i=1}^{n} \mathbf{x}_i [\frac{exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta})}{1 + exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta})} - y_i]$$

$$\nabla^2 \ell(\boldsymbol{\beta}) = \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathsf{T}}}$$

$$= \sum_{i=1}^{n} \mathbf{x}_i \frac{\mathbf{x}_i^{\mathsf{T}} exp(\mathbf{x_i^t} \boldsymbol{\beta})(1 + exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta})) - (exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta}))^2}{(1 + exp(\mathbf{x_i^t} \boldsymbol{\beta}))^2}$$

$$= \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^{\mathsf{T}} \frac{exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta})}{(1 + exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta}))^2}$$

$$\text{Let} \quad w_i(\boldsymbol{\beta}) = \frac{exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta})}{(1 + exp(\mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta}))^2}$$

$$\nabla^2 \ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} w_i \boldsymbol{\beta} \mathbf{x}_i \mathbf{x}_i^{\mathsf{T}}$$

$$= \mathbf{X}^{\mathsf{T}} \mathbf{W}(\boldsymbol{\beta}) \mathbf{X}$$

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

For gradient:

$$c_i = \frac{exp(\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})}{1+exp(\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})} - y_i \sim \mathbf{x}_i^\mathsf{T}\boldsymbol{\beta} \ \sim \mathcal{O}(p)$$

$$\sum_{i=1}^{n} \mathbf{x}_i c_i \ \sim \mathcal{O}(np)$$

Therefore total is $\mathcal{O}(np)$.

For Hessian:

$$w_i(\boldsymbol{\beta}) = \frac{exp(\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})}{(1+exp(\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta}))^2} \sim \ \mathcal{O}(p)$$

$$\sum_{i=1}^{n} w_i(\boldsymbol{\beta})\mathbf{x}_i\mathbf{x}_i^\mathsf{T} \sim \ \mathcal{O}(np^2)$$

Therefore total is $\mathcal{O}(np^2)$

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

$\ell(\boldsymbol{\beta})$ is strictly convex if $\sum_{i=1}^{n} w_i\mathbf{x}_i\mathbf{x}_i^\mathsf{T} \succ 0$.

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

WTS $||\nabla\ell(\boldsymbol{\beta}) - \nabla\ell(\boldsymbol{\alpha})||_2 \leq L||\boldsymbol{\beta} - \boldsymbol{\alpha}||_2 \ \ \forall\boldsymbol{\beta},\boldsymbol{\alpha} \ \in \mathbb{R}^m$.

$$||\nabla\ell(\boldsymbol{\beta}) - \nabla\ell(\boldsymbol{\alpha})||_2 = ||\sum_{i=1}^{n}\mathbf{x}_i\left(\frac{exp(\mathbf{x}_i^\mathsf{T}\beta)}{1+exp(\mathbf{x}_i^\mathsf{T}\beta)} - \frac{exp(\mathbf{x}_i^\mathsf{T}\alpha)}{1+exp(\mathbf{x}_i^\mathsf{T}\alpha)}\right)||_2$$

Let $f(x) = \frac{exp(x)}{1+exp(x)}$. We know that $f(x)$ is continuous on $(-\infty,\infty)$. By Lagrange mean value theroem, $f(b) - f(a) = f'(\xi)(b-a)$ for some $\xi \in (a,b)$. Now $f'(x) = \frac{exp(x)}{(1+exp(x))^2}$ which has a global maximum at $x = 0$. Therefore $f'(\xi) \leq f'(0) = \frac{1}{4}$. Therefore

$$||\sum_{i=1}^{n}\mathbf{x}_i\left(\frac{exp(\mathbf{x}_i^\mathsf{T}\beta)}{1+exp(\mathbf{x}_i^\mathsf{T}\beta)} - \frac{exp(\mathbf{x}_i^\mathsf{T}\alpha)}{1+exp(\mathbf{x}_i^\mathsf{T}\alpha)}\right)||_2 \leq ||\sum_{i=1}^{n}\frac{1}{4}\mathbf{x}_i(\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta} - \mathbf{x}_i^\mathsf{T}\boldsymbol{\alpha})||_2$$

$$= \frac{1}{4}||\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^\mathsf{T}(\boldsymbol{\beta} - \boldsymbol{\alpha})||_2$$

$$\leq \frac{1}{4}||\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^\mathsf{T}||_2||\boldsymbol{\beta} - \boldsymbol{\alpha}||_2$$

$$= \frac{1}{4}||\mathbf{X}^\mathsf{T}\mathbf{X}||_2||\boldsymbol{\beta} - \boldsymbol{\alpha}||_2$$

$$= \frac{1}{4}||\mathbf{X}||_{op}^2||\boldsymbol{\beta} - \boldsymbol{\alpha}||_2$$

Therefore $\ell(\boldsymbol{\beta})$ is L-Lipschitz differentiable with constant $\frac{1}{4}||\mathbf{X}||_{op}^2$.

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

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n_0} log(1 + exp(\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})) + \sum_{j=1}^{n_1} \left[-\mathbf{x}_j^\mathsf{T}\boldsymbol{\beta} + log(1 + exp(\mathbf{x}_j^\mathsf{T}\boldsymbol{\beta}))\right]$$

$$\ell(\mathbf{w}) = \sum_{i=1}^{n_0} log(1 + exp(\mathbf{x}_i^\mathsf{T}\mathbf{w})) + \sum_{j=1}^{n_1} \left[-\mathbf{x}_j^\mathsf{T}\mathbf{w} + log(1 + exp(\mathbf{x}_j^\mathsf{T}\mathbf{w}))\right]$$

By assymption $\mathbf{x}_i^\mathsf{T}\mathbf{w} < 0$ in the first sum component and $\mathbf{x}_j^\mathsf{T}\mathbf{w} > 0$ in the second sum component.

Our objective is to find $\tilde{\beta}$ that minimizes $\ell(\boldsymbol{\beta})$. Now notice that if we choose $c > 1$, $\mathbf{x}_i^\mathsf{T}(c\mathbf{w})$ will always make the first sum to be smaller, let's look at the derivative of the second term supposing $\mathbf{x}_j^\mathsf{T}\mathbf{w} = z_j$. The derivative for the j^{th} component is given by $-1 + \frac{exp(z_j)}{1+exp(z_j)} < 0$, so it decrease with larger z_j. Therefore if we choose $c > 1$, $\mathbf{x}_i^\mathsf{T}(c\mathbf{w})$ will always make the second sum smaller as well. That means if the data is completely separable we can always decrease the negative log-likelihood. Therefore no global minimizer exists.

To address the completely separable situation, we can modify the negative log-likelihood by adding a ridge penalty, namely we minimize

$$\ell_\lambda(\boldsymbol{\beta}) = \ell(\boldsymbol{\beta}) + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2,$$

where $\lambda > 0$ is a tuning parameter. By choosing large λ, we penalize solutions that have large 2-norms.

6. Write the gradient and Hessian of $\ell_\lambda(\boldsymbol{\beta})$.

$$\ell_\lambda(\boldsymbol{\beta}) = \nabla\ell(\boldsymbol{\beta}) + \frac{\lambda}{2}\boldsymbol{\beta}^\mathsf{T}\boldsymbol{\beta}$$

$$\nabla\ell_\lambda(\boldsymbol{\beta}) = \nabla\ell(\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}$$

$$= \sum_{i=1}^{n} \mathbf{x}_i[\frac{exp(\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})}{1 + exp(\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta})} - y_i] + \lambda\boldsymbol{\beta}$$

$$\nabla^2\ell_\lambda(\boldsymbol{\beta}) = \nabla^2\ell_\lambda(\boldsymbol{\beta}) + \lambda\mathbf{I}$$

$$= \sum_{i=1}^{n} w_i\boldsymbol{\beta}\mathbf{x}_i\mathbf{x}_i^\mathsf{T} + \lambda\mathbf{I}$$

$$= \mathbf{X}^\mathsf{T}\mathbf{W}(\beta)\mathbf{X} + \lambda\mathbf{I}$$

7. What is the computational complexity for a calculating the gradient and Hessian of $\ell_\lambda(\boldsymbol{\beta})$?

The most computational expensive part is caclulating the gradient and Hessian of $\ell(\boldsymbol{\beta})$. $\lambda\boldsymbol{\beta}$ or $\lambda\mathbf{I}$ only needs $\mathcal{O}(p)$. Therefore $\nabla\ell_\lambda(\beta) \sim \mathcal{O}(np)$ and $\nabla^2\ell_\lambda(\beta) \sim \mathcal{O}(np^2)$

8. Prove that $\ell_\lambda(\boldsymbol{\beta})$ has a unique global minimizer for all $\lambda > 0$.

First notice that $\mathbf{W}(\beta) \succ 0$ since $w_i(\beta) > 0 \ \forall\ i$. Then $\mathbf{X}^\mathsf{T}\mathbf{W}(\beta)\mathbf{X} = \mathbf{X}^\mathsf{T}\mathbf{R}\mathbf{R}\mathbf{X}$ where $\mathbf{R}$ is the symmetric square root of $\mathbf{W}\boldsymbol{\beta}$. Therefore $\mathbf{X}^\mathsf{T}\mathbf{A}\mathbf{X} = (\mathbf{R}\mathbf{X})^\mathsf{T}(\mathbf{R}\mathbf{X})$ which we know is positive semidefinite. For $\lambda > 0$, $\lambda\mathbf{I} \succ 0$, hence $\mathbf{X}^\mathsf{T}\mathbf{A}\mathbf{X} + \lambda\mathbf{I} \succ 0$. Thus global minimizer exists.

Part 2. Gradient Descent and Newton's Method

You will next add an implementation of gradient descent and Newton's method to your R package.

Please complete the following steps.

Step 0: Make an R package entitled "unityidST790".

Step 1: Write a function "gradient_step."

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

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

Step 2: Write a function "gradient_descent_fixed." Terminate the algorithm the relative change in the objective function falls below the tolerance parameter `tol`.

```
# #' 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
# #' @export
# 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 functions 'fx_logistic' and 'gradf_logistic' to perform ridge logistic regression

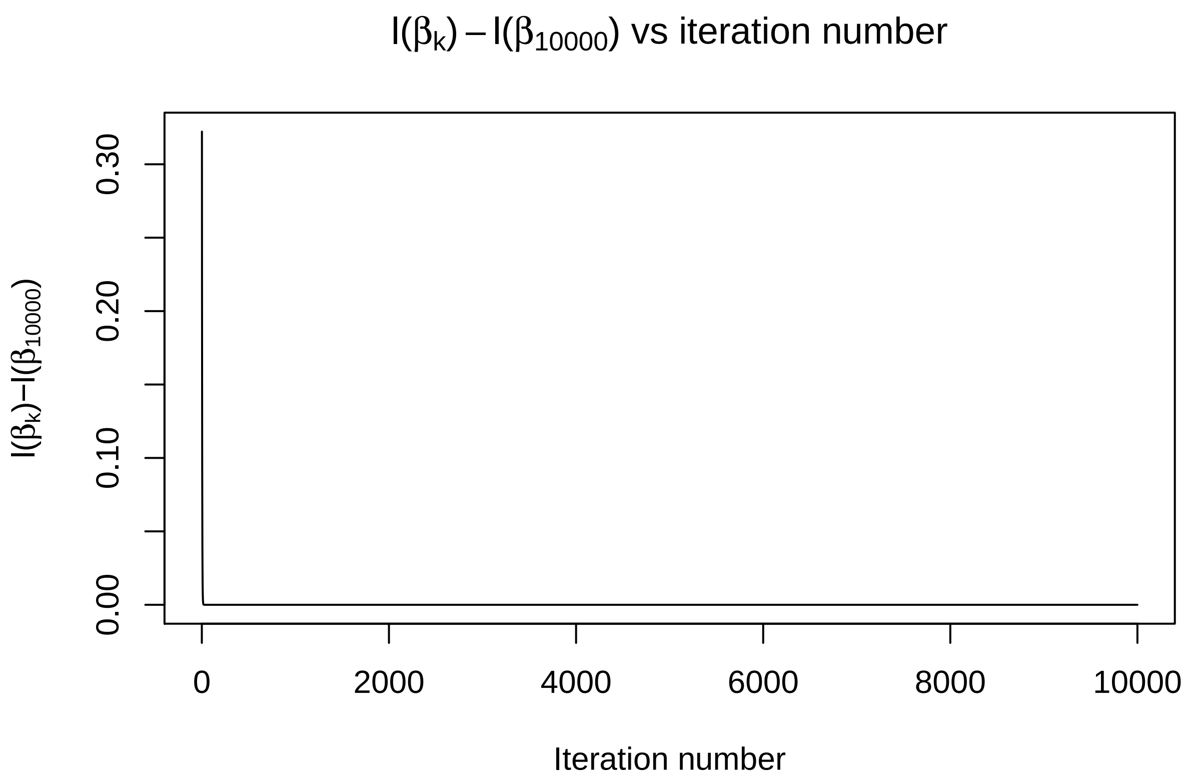
```
# #' Objective Function for Logistic Regression
# #'
# #' @param y binary response
# #' @param X design matrix
# #' @param beta regression coefficient vector
# #' @param lambda regularization parameter
# #' @export
# fx_logistic <- function(y, X, beta, lambda=0) {
#
# }
#
# #' Gradient for Logistic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

Step 4: Perform logistic regression (with $\lambda = 0$) on the following data example $(\mathbf{y}, \mathbf{X})$ using the fixed step-size. Use your answers to Part 1 to choose an appropriate fixed step-size. Plot the difference $\ell(\boldsymbol{\beta}_k) - \ell(\boldsymbol{\beta}_{10000})$ 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.

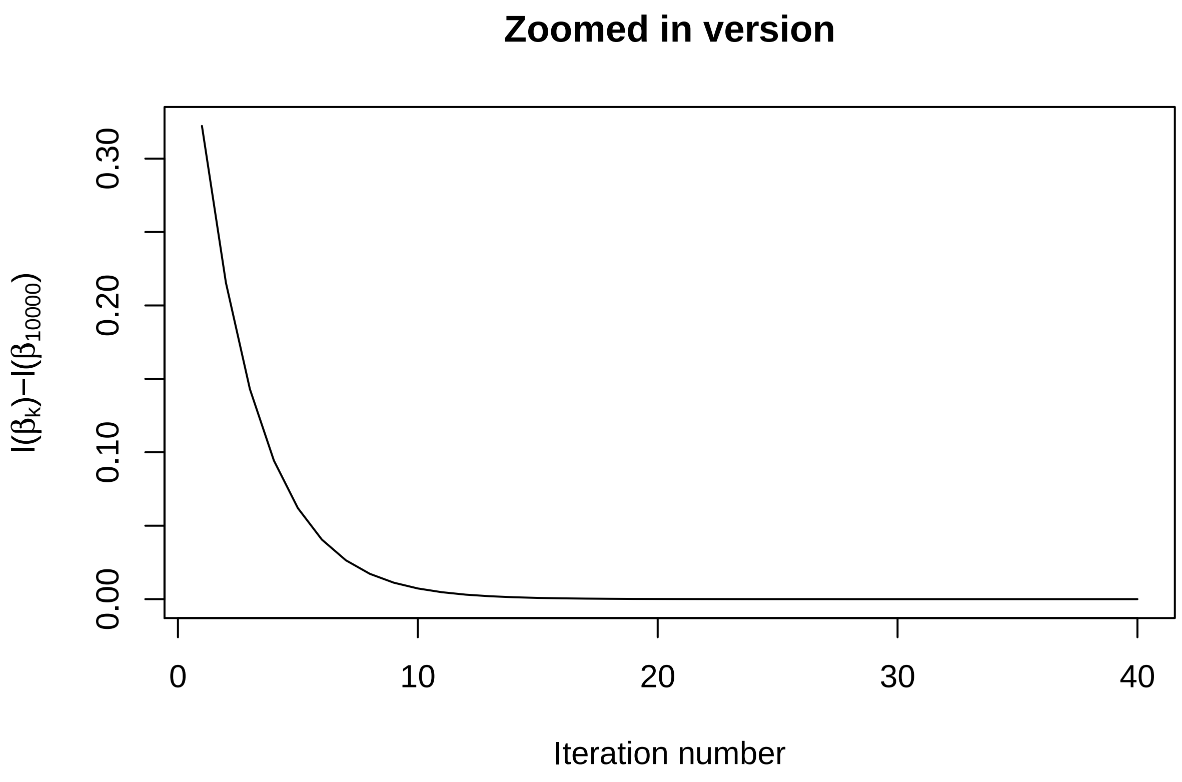
- **Hint:** Write wrapper functions around `fx_logistic` and `gradf_logistic` that take in only one argument, the regression coefficient vector $\boldsymbol{\beta}$.

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

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

L = (norm(X,type="2")^2)/4
t = 1/(2*L)
lambda = 0

fx_logistic_wrapper <- function(beta) {
  return(fx_logistic(y, X, beta,lambda))
}

gradf_logistic_wrapper <- function(beta) {
  return(gradf_logistic(y, X, beta,lambda))
}

fsum = gradient_descent_fixed(fx_logistic_wrapper, gradf_logistic_wrapper,
                              t, beta0, max_iter=1e4, tol=-1)
l_beta = fsum$fun_val
l_change = l_beta - l_beta[1e4]
```

```
effn = sum(l_change>1e-4)

plot(1:1e4,l_change,type='l',main = TeX('$l(\\beta_k)-\\l(\\beta_{10000})$ vs iteration  number'),
     ylab=TeX('$l(\\beta_k)-l(\\beta_{10000})$'),xlab = 'Iteration number')
```



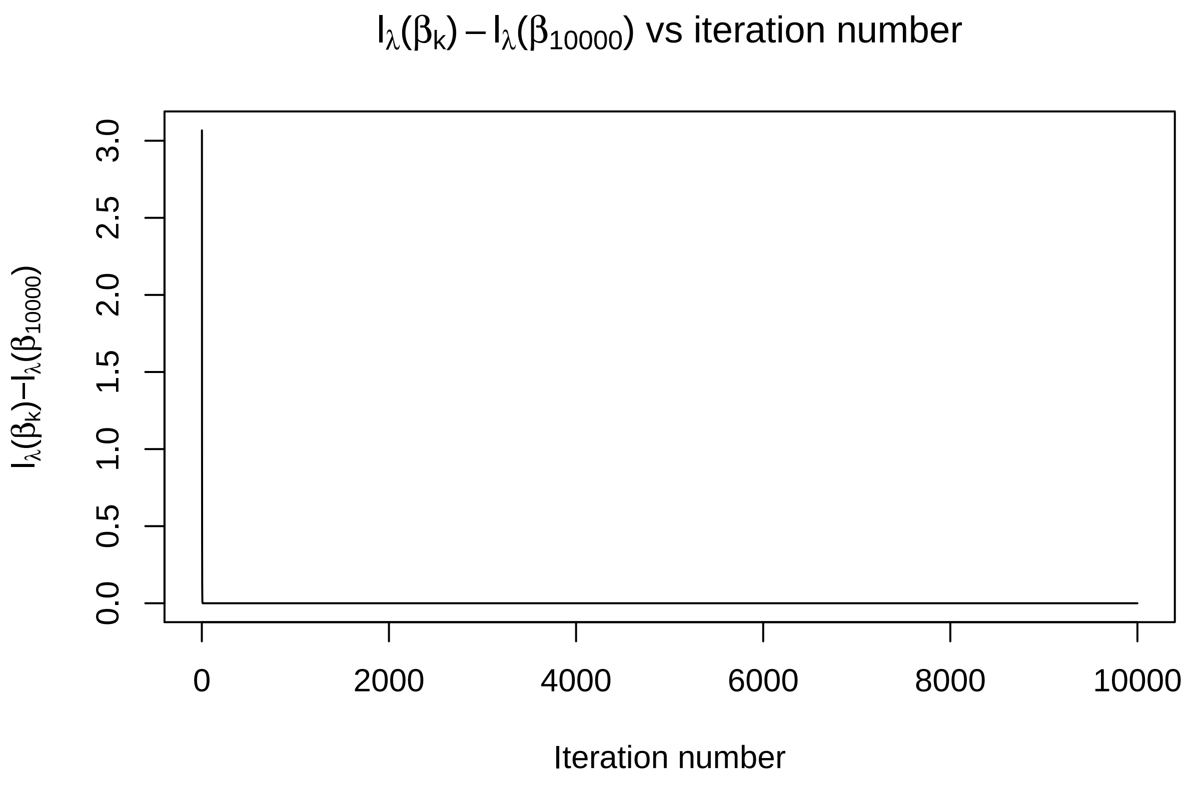
```
plot(c(1:40),l_change[1:40],type='l',main = "Zoomed in version",
     ylab=TeX('$l(\\beta_k)-l(\\beta_{10000})$'),xlab = 'Iteration number')
```



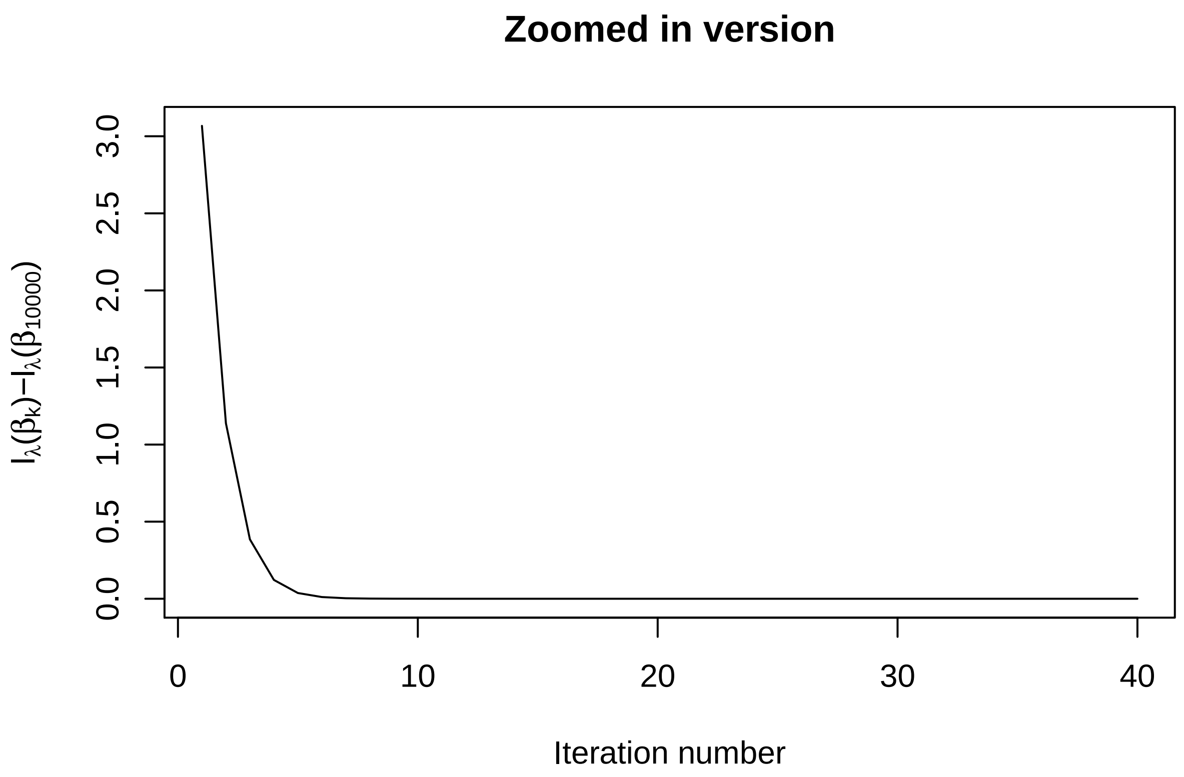
Since we know $f(x_k) - f^* \leq C * \frac{1}{k}$, where k is the iteration number and $C_1 = \frac{1}{2\alpha}\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2$, a constant. From the upper bound of this inequality, we expect the shape to resemble that of $f(x) = \frac{1}{x}$. The zoomed in plot support our belief. The tolerance was set to negative to ensure 10000 iterations was run but clearly we need way less than that, in fact the effective number (iterations took for $\ell(\beta_k) - \ell(\beta_{10000})$ to reach 10^{-4}) was 19. The zoomed in plot was created so that we can examine the shape.

Step 5: Perform logistic regression (with $\lambda = 10$) on the simulated data above using the fixed step-size. Plot the difference $\ell_\lambda(\boldsymbol{\beta}_k) - \ell_\lambda(\boldsymbol{\beta}_{10000})$ versus the iteration k. Comment on the shape of the plot given what you know about the iteration complexity of gradient descent.

```
lambda = 10
fsum = gradient_descent_fixed(fx_logistic_wrapper, gradf_logistic_wrapper,
                              t, beta0, max_iter=1e4, tol=-1)

l_beta = fsum$fun_val
l_change = l_beta - l_beta[1e4]
effn = sum(l_change>1e-4)

plot(1:1e4,l_change,type='l',main = TeX('$l_{\\lambda}(\\beta_k)-\\l_{\\lambda}(\\beta_{10000})$ vs ite
     ylab=TeX('$l_{\\lambda}(\\beta_k)-l_{\\lambda}(\\beta_{10000})$'),xlab = 'Iteration number')
```



```
plot(c(1:40),l_change[1:40],type='l',main = 'Zoomed in version', ylab=TeX('$l_{\\lambda}(\\beta_k)-l_{\
```



Since we know $f(x_k) - f^* \leq C * \frac{1}{k}$, where k is the iteration number and $C_1 = \frac{1}{2\alpha}\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2$, a constant. From the upper bound of this inequality, we expect the shape to resemble that of $f(x) = \frac{1}{x}$. The zoomed in plot support our belief. The tolerance was set to negative to ensure 10000 iterations was run but clearly we need way less than that, in fact the effective number (iterations took for $\ell(\beta_k) - \ell(\beta_{10000})$ to reach 10^{-4}) was 9. Also with a penalty term the difference decrease more faster, implying that the optimazation problem with penalty is better than that without penalty. The zoomed in plot was created so that we can examine the shape.

For the rest of Part 2, we will investigate the effect of using the Sherman-Morrison-Woodbury identity in improving the scalability of the Newton's method algorithm for ridge logistic regression. We seek to minimize the following objective function

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[-y_i \mathbf{x}_i^\mathsf{T}\boldsymbol{\beta} + \log(1 + \exp(\mathbf{x}_i^\mathsf{T}\boldsymbol{\beta}))\right] + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2$$

Step 6: Write a function "newton_step_naive" that computes the solution $\Delta\boldsymbol{\beta}_{\text{nt}}$ to the linear system

$$(\lambda\mathbf{I} + \mathbf{X}^\mathsf{T}\mathbf{W}(\boldsymbol{\beta})\mathbf{X})\Delta\boldsymbol{\beta}_{\text{nt}} = \nabla\ell(\boldsymbol{\beta}).$$

Use the **chol**, **backsolve**, and **forwardsolve** functions in the base package. The **plogis** function will also be helpful for computing $W(\boldsymbol{\beta})$.

```
# #' Compute Newton Step (Naive) for logistic ridge regression
# #'
# #' @param X Design matrix
# #' @param y Binary response vector
# #' @param beta Current regression vector estimate
# #' @param g Gradient vector
# #' @param lambda Regularization parameter
# newton_step_naive <- function(X, y, beta, g, lambda) {
#
# }
```

Your function should return the Newton step $\Delta\boldsymbol{\beta}_{\text{nt}}$.

Step 7: Write a function "newton_step_smw" that computes the Newton step using the Sherman-Morrison-Woodbury identity to reduce the computational complexity of computing the Newton step from $\mathcal{O}(p^3)$ to $\mathcal{O}(n^2p)$. This is a reduction when $n < p$.

```
# #' Compute Newton Step (Sherman-Morrison-Woodbury) for logistic ridge regression
# #'
# #' @param X Design matrix
# #' @param y Binary response vector
# #' @param beta Current regression vector estimate
# #' @param g Gradient vector
# #' @param lambda Regularization parameter
# newton_step_smw <- function(X, y, beta, g, lambda) {
#
# }
```

Your function should return the Newton step $\Delta\boldsymbol{\beta}_{\text{nt}}$.

Step 8 Write a function “backtrack_descent”

```
# #' Backtracking for steepest descent
# #'
# #' @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 d descent direction vector
# #' @param alpha the backtracking parameter
# #' @param beta the decrementing multiplier
# backtrack_descent <- function(fx, x, t, df, d, alpha=0.5, beta=0.9) {
#
# }
```

Your function should return the selected step-size.

Step 9: Write functions “logistic_ridge_newton” to estimate a ridge logistic regression model using damped Newton’s method. Terminate the algorithm when half the square of the Newton decrement falls below the tolerance parameter `tol`.

```
# #' Damped Newton's Method for Fitting Ridge Logistic Regression
# #'
# #' @param y Binary response
# #' @param X Design matrix
# #' @param beta Initial regression coefficient vector
# #' @param lambda regularization parameter
# #' @param naive Boolean variable; TRUE if using Cholesky on the Hessian
# #' @param max_iter maximum number of iterations
# #' @param tol convergence tolerance
# logistic_ridge_newton <- function(X, y, beta, lambda=0, naive=TRUE, max_iter=1e2, tol=1e-3) {
#
# }
```

Step 10: Perform logistic regression (with $\lambda = 10$) on the following 3 data examples (y, X) using Newton’s method and the naive Newton step calculation. Record the times for each using **system.time**.

```
lambda = 10
iter = 1e2

set.seed(12345)
## Data set 1
n <- 200
p <- 400

X1 <- matrix(rnorm(n*p),n,p)
beta01 <- matrix(rnorm(p),p,1)
y1 <- (runif(n) <= plogis(X1%*%beta01)) + 0

X = X1
y = y1
beta0 = beta01

time1 = system.time({logistic_ridge_newton(X, y, beta0, lambda, naive=TRUE,
                                           max_iter=iter, tol=1e-3)})[3]
## Data set 2
p <- 800
```

```r
X2 <- matrix(rnorm(n*p),n,p)
beta02 <- matrix(rnorm(p),p,1)
y2 <- (runif(n) <= plogis(X2%*%beta02)) + 0

X = X2
y = y2
beta0 = beta02
time2 = system.time({logistic_ridge_newton(X, y, beta0, lambda, naive=TRUE,
                                           max_iter=iter, tol=1e-3)})[3]

## Data set 3
p <- 1600
X3 <- matrix(rnorm(n*p),n,p)
beta03 <- matrix(rnorm(p),p,1)
y3 <- (runif(n) <= plogis(X3%*%beta03)) + 0

X = X3
y = y3
beta0 = beta03
time3 = system.time({logistic_ridge_newton(X, y, beta0, lambda, naive=TRUE,
                                           max_iter=iter, tol=1e-3)})[3]
```

Step 11: Perform logistic regression (with $\lambda = 10$) on the following 3 data examples (y, X) using Newton's method and the Newton step calculated using the Sherman-Morrison-Woodbury identity. Record the times for each using **system.time**.

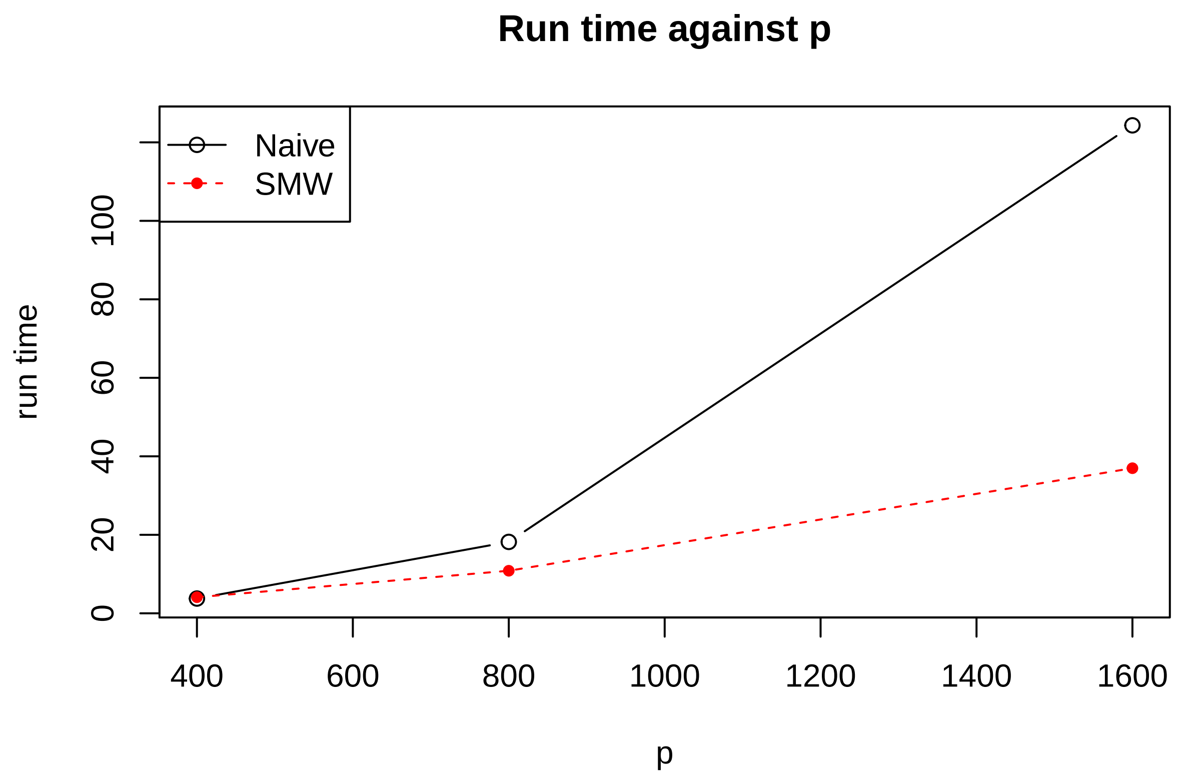
```r
## Data set 1
X = X1
y = y1
beta0 = beta01

time4 = system.time({logistic_ridge_newton(X, y, beta0, lambda, naive=FALSE,
                                           max_iter=iter, tol=1e-3)})[3]

## Data set 2
X = X2
y = y2
beta0 = beta02

time5 = system.time({logistic_ridge_newton(X, y, beta0, lambda, naive=FALSE,
                                           max_iter=iter, tol=1e-3)})[3]

## Data set 3
X = X3
y = y3
beta0 = beta03

time6 = system.time({logistic_ridge_newton(X, y, beta0, lambda, naive=FALSE,
                                           max_iter=iter, tol=1e-3)})[3]
```

Step 12: Plot all six run times against p. Comment on the how the two run-times scale with p and compare it to what you know about the computational complexity for the two ways to compute the Newton update.

```r
time = c(time1,time2,time3,time4,time5,time6)
time
```

```
## elapsed elapsed elapsed elapsed elapsed elapsed
##    3.76   18.19  124.33    4.10   10.84   36.97
```

```
plot(c(400,800,1600),time[1:3],type=c("b"),lty=1,main = 'Run time against p',ylab='run time',xlab = 'p'
lines(c(400,800,1600),time[4:6],lty=2,col=2)
points(c(400,800,1600),time[4:6],type = 'p',pch = 20,col='red')
legend("topleft",legend=c("Naive","SMW"), col=1:2,pch=c(1,20),lty=1:2)
```



When $p > n$, for naive Newton step, the computational complexity is $O(p^3)$, so time1 : time2 : time3 should be 1:8:64. My result gives 1:5:30. Using the Sherman-Morrison- Woodbury identity, the computational complexity is $O(n^2p)$, so time4 : time5 : time6 should be 1:2:4. My result gives 1:2.64:8.23. Also the ratio of two methods should be, time1/time4 = 4, time2/time5 = 16 and time3/time6 = 64. From my result, time1/time4 = 1, time2/time5 = 1.73 and time3/time6 = 3.17. The mismatch between simulated and theoretical results might be due to relative small n or p. However we can clearly see that when $p > n$, using the Sherman-Morrison-Woodbury identity reduces the computational complexity.