BIS557 Homework 3

Zibo Tian

2020/10/30

```
library(bis557)
```

```
##   missForest iteration 1 in progress...done!
##   missForest iteration 2 in progress...done!
##   missForest iteration 3 in progress...done!
##   missForest iteration 4 in progress...done!
##   missForest iteration 5 in progress...done!
##   missForest iteration 6 in progress...done!
```

Question 1

From the question, it seems that our goal is to give examples of vector p and design matrix X such that the condition number of the linear Hessian $X^T X$ is small (well-consitioned) but the condition number of $X^T DX$ is large (ill-conditioned). The D matrix is a diagonal matrix with diagonal terms $p(1-p)$.
To make large jump in the condition numbers, we need to force the singular value of the matrix have large range. This can be easily achieved by tring to make D matrix with close to 0 diagonal terms or making p approach 0 or 1. Here show the example.

```
set.seed(1222222)
X <- matrix(rnorm(30,0,1),ncol=3, nrow=10)
X <- cbind(rep(1,10),X)
beta <- rep(200,4)
p <- as.vector(1/(1+exp(-X%*%beta)))

D <- diag(p*(1-p))

#Compare the consition number
#XtX
kappa(t(X)%*%X)
```

```
## [1] 4.02226
```

```
#XtDX
kappa(t(X)%*%D%*%X)
```

```
## [1] 1.328496e+18
```

It is clear that the linear Hessian is well-conditioned but the logistic variation is not under this case.

Question 2

In this question, I write a function called "my_glm" that can implement the first-order gradient descent algorithm in two feature. The first one uses a constant step size and the second one uses an adaptive step size which take the advantage of the Momentum optimization algorithm introduced in the webpage:https://ruder.io/optimizing-gradient-descent/$//$ For the adaptive algorithm (Momentum), the updates goes like

$$v_t = 0.9 \times v_{t-1} + \gamma \bigtriangledown_\beta J(\beta)$$

$$\beta = \beta + v_t$$

While the adaptive step size is properly self-adjusted given every new set of coefficient estimates in each iteration, it would be more efficient analytically.

By doing some experiment, I noticed that the constant step size seemed to do bad job in estimating coefficients in logistic models. To make a better comparison, here I try to simulate some data and create a list of outcomes from a poisson distribution generater. Hence, it imitates the process that we have some count data and want to estimate the coefficients in our model by using GLM. The link function and the inverse of it are specified in the functions. Comparison was made between "my_glm" and the original function "glm".

```
#Check constant step size
#Generate some data
set.seed(11111)
n <- 1000
X <- cbind(rep(1,n),matrix(rnorm(n*3), ncol = 3))
beta <- c(-5, 0.6, 0.3, 1.3)
Y <- rpois(n, lambda = exp(X%*%beta))
data.pois <- data.frame(cbind(Y,X[,-1]))
form <- Y~.

my_beta1 <- my_glm(form, data.pois, mu_fun = function(eta) exp(eta), method=1)$coefficients
beta_glm <- coef(glm(form, family = "poisson", data.pois))
#Compare
table.1<-data.frame(cbind(beta_glm, my_beta1))
colnames(table.1)<-c("GLM beta", "beta under constant step size")
table.1
```

```
##               GLM beta beta under constant step size
## (Intercept) -4.8399731                    -3.8808889
## V2           0.6985405                     0.4183672
## V3           0.3523379                     0.2066152
## V4           1.0625312                     0.6080166
```

```
#Check adaptive step size
my_beta2 <- my_glm(form, data.pois, mu_fun = function(eta) exp(eta), method=2)$coefficients
#Compare
table.2<-data.frame(cbind(beta_glm, my_beta2))
colnames(table.2)<-c("GLM beta", "beta under adaptive step size")
table.2
```

```
##               GLM beta beta under adaptive step size
## (Intercept) -4.8399731                    -4.7064567
## V2           0.6985405                     0.6650088
```

```
## V3           0.3523379                    0.3353183
## V4           1.0625312                    1.0066200
```

```
my_glm(form, data.pois, mu_fun = function(eta) exp(eta), method=1)$count
```

```
## [1] 812
```

```
my_glm(form, data.pois, mu_fun = function(eta) exp(eta), method=2)$count
```

```
## [1] 182
```

In the two tables and the two counters above, it is clear that the estimated coefficients under the constant step size spend more iterations to get a difference lower than the thresholds. Also, even if both of the method "converge" on aspect of the criterior set (tolerance), the former one did not give a good estimate of the true coefficients. I guess the problem come from the drawbacks of a constant step size itself, or the simulated data. From this comparison, the algorithm with the adaptive step size is better.

Question 3

In this question, I use two functions to conduct the multi-class classification process. Firstly, the function "multiclass_logistic" helps to estimate the coefficient estimates for each comparison group. The comparison groups there consist of one specific class of our outcome of interest and all the others which do not belong to the class. Under this setting, the function will produce a matrix whose rows record the corresponding estimates for the comparison groups. Although this might be different from what formal functions such as "lme" do, this estimated coefficients can be easily used to calculated the predicted probabilities conditioning on each observations covariates (More details later). And because it is intractable to check convergence for a matrix in the function, I will manully check its convergence in the code below. Actually, because hessian as an adaptive measure was used, the betas converge to specific values quickly. Then, another function called "predict_logistic" helps to convert the estimates to a series of crude predictions. Other high-level assessments of the accuracy of the prediction are also conducted below.

```
#Call example data from the package
data(penguinsi)
test <- penguinsi
form <- species ~ bill_length_mm + bill_depth_mm
X <- model.matrix(form, test)
Y <- penguinsi$species
beta <- multiclass_logistic(X,Y,maxit=20)$coefficients
beta
```

```
##           [,1]       [,2]       [,3]
## [1,]  24.13555 -2.2102828  3.9988179
## [2,]  49.05198  0.5578984 -4.5352088
## [3,] -31.62272  0.3533512  0.8025893
```

```
beta2 <- multiclass_logistic(X,Y,maxit=30)$coefficients
beta2
```

```
##           [,1]       [,2]       [,3]
## [1,]  24.13555 -2.2102828  3.9988179
## [2,]  49.05198  0.5578984 -4.5352088
## [3,] -31.62272  0.3533512  0.8025893
```

From the output above, it is clear that the estimates converge in less than 20 times iterations.

```
#Prediction accuracy
mean(predict_logistic(X,Y,multiclass_logistic(X,Y,maxit=23))$fitted.Y==Y)
```

```
## [1] 0.9563953
```

By using crude prediction (assign predicted class to the highest probability), we actually get a pretty high rate of accuracy.

```
#Marginal probabilities
mean(predict_logistic(X,Y,multiclass_logistic(X,Y,maxit=23))$p[,1])
```

```
## [1] 0.4531638
```

```
mean(predict_logistic(X,Y,multiclass_logistic(X,Y,maxit=23))$p[,2])
```

```
## [1] 0.3701455
```

```
mean(predict_logistic(X,Y,multiclass_logistic(X,Y,maxit=23))$p[,3])
```

```
## [1] 0.1766907
```

```
table(penguinsi$species)
```

```
##
##    Adelie Chinstrap    Gentoo
##       152        68       124
```

The marginal probabilities computed from the prediction are close to the true proportion of each species.