homework-3

```
library(bis557)
library(dplyr)
#> Warning: package 'dplyr' was built under R version 3.6.2
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
library(palmerpenguins)
#> Warning: package 'palmerpenguins' was built under R version 3.6.2
```

1. CASL 5.8 Exercise number 2

We mentioned that the Hessian matrix in Equation 5.19 can be more ill-conditioned than the matrix X^tX itself. Generate a matrix X and propabilities p such that the linear Hessian (X^tX) is well-conditioned but the logistic variation is not.

The Hessian matrix in 5.19 is $H(l) = X^tDX$ where D is a diagonal matrix with $D_{i,i} = p_i(1 - p_i)$.

According to [https://www.quora.com/What-does-it-mean-to-have-a-poorly-conditioned-Hessian-matrix, and https://www.quora.com/What-is-an-ill-conditioned-matrix], a Hessian matrix is said to be ill-conditioned if it has some large positive eigenvalues and some eigenvalues very close to zero. So it would be hard to invert the matrix. We could use the singular value decomposition to obtain the singular values and compute the conditional number which is the ratio of the largest singular value and smallest singular value. A simple example is shown below:

```
X <- matrix(c(1,1,0,1),ncol = 2)
p <- c(1e-5,0.8)

D <- diag(as.vector(p))
H_log <- t(X) %*% D %*% X
H_lin <- t(X) %*% X

max(svd(H_log)$d)/min(svd(H_log)$d)
#> [1] 320002
max(svd(H_lin)$d)/min(svd(H_lin)$d)
#> [1] 6.854102
```

The singular values were obtained by the singular value decomposition and since we introduce p, the largest and smallest values are very far from each other. We could see that the conditional number for logistic Hessian is 320002, which is ill-conditioned, while for linear is 6.8, which is well-conditioned.

2.Describe and implement a first-order solution for the GLM maximum likelihood problem using only gradient information, avoiding the Hessian matrix. Include both a constant step size along with an adaptive one. You may use a standard adaptive update Momentum, Nesterov, AdaGrad, Adam, or your own. Make sure to explain your approach and compare it's performance with a constant step size.

In this part, we ask the user to input their X and y's, select to use adaptive(which is time-decay in this case), or constant step size for gradient descent. We initialize beta to zero. If the user select decay, then we update the gamma, which is learning rate, by gamma = gamma/(1+decay * step number). Else, we could keep a constant learning rate either using the default or the input values. We compute the gradient by $X^t(y - Ey)$ and update betas, until we reach the max number of iterations or the update won't bring much change to beta.

Use the code from text book p130 to generate data

```
set.seed(100)
n <- 5000; p <- 3
beta <- c(-1, 0.2, 0.1)
X <- cbind(1, matrix(rnorm(n * (p- 1)), ncol = p - 1))
eta <- X %*% beta
lambda <- exp(eta)
y <- rpois(n, lambda = lambda)

my_glm <- first_order_GLM(X,y,family = poisson(link = "log"), lr = "constant")
beta_glm <- glm(y ~ X[,-1], family = "poisson")

cbind(my_glm$coefficients, beta_glm$coefficients, beta)
#>                                   beta
#> (Intercept) -1.0377842 -1.0377842 -1.0
#> X[, -1]1     0.1834389  0.1834389  0.2
#> X[, -1]2     0.1192523  0.1192523  0.1
```

We get similar results to the glm function, but both do a decent job but are not very close to true betas.

Using the adaptive step size:

```
my_glm_ad <- first_order_GLM(X,y,family = poisson(link = "log"), lr = "step", decay = 0.01)
cbind(my_glm_ad$coefficients, beta_glm$coefficients, beta)
#>                                    beta
#> (Intercept) -0.96402198 -1.0377842 -1.0
#> X[, -1]1     0.14831973  0.1834389  0.2
#> X[, -1]2     0.09534918  0.1192523  0.1
```

Using decay of 0.01, with time based update, [https://en.wikipedia.org/wiki/Learning_rate#Adaptive_learning_rate], we get estimates that is different from the glm function in R. Some estimates are better while some are worse. Some other adaptive step size may do better job than this.

3.Describe and implement a classification model generalizing logistic regression to accommodate more than two classes.

Referring to textbook 5.5, for multiclass model, there is one-vs-one and one-vs-all approach. This model uses the one-vs-all method. We first determine the unique values of y and for each y, we have either 1, representing y is the value, or 0, representing y is not the value, and we fit a logistic model for it. The we use the beta's to compute the probabilities for assignment and pick the largest p-value for each data point and predict the y's

```
make_model_matrices <- function(formula, df){
  # create model matrix
  df_no_na <- model.frame(formula,df)
  X <- model.matrix(formula, df_no_na)
  # get dependent variable
  yname <- as.character(formula)[2]
  y <- matrix(df_no_na[,yname],ncol = 1)
  list(X = X, y = y)
}
```

```
data(penguins)
form <- species ~ bill_length_mm + body_mass_g
mat <- make_model_matrices(form,penguins)
X <- mat$X
y <- mat$y
peng_spec <- multi_logreg(X,y)

peng_spec$coefficients
#>                 [,1]         [,2]         [,3]
#> [1,] 50.8871800831 -31.99094548 -44.84123232
#> [2,] -1.1503500714   0.09125184   2.26879868
#> [3,] -0.0003725055   0.00625705  -0.01494231

peng_new <- penguins
peng_new$is_adelie <- ifelse(peng_new$species == "Adelie", 1, 0)
peng_new$is_gentoo <- ifelse(peng_new$species == "Gentoo", 1, 0)
peng_new$is_chinstrap <- ifelse(peng_new$species == "Chinstrap", 1, 0)


glm(is_adelie ~ bill_length_mm + body_mass_g, data = peng_new, family = binomial)
#>
#> Call:  glm(formula = is_adelie ~ bill_length_mm + body_mass_g, family = binomial,
#>     data = peng_new)
#>
#> Coefficients:
#>    (Intercept)  bill_length_mm     body_mass_g
#>     50.8871801      -1.1503501      -0.0003725
#>
#> Degrees of Freedom: 341 Total (i.e. Null);  339 Residual
#>   (2 observations deleted due to missingness)
#> Null Deviance:       469.4
#> Residual Deviance: 90.95     AIC: 96.95
glm(is_gentoo ~ bill_length_mm + body_mass_g, data = peng_new, family = binomial)
#>
#> Call:  glm(formula = is_gentoo ~ bill_length_mm + body_mass_g, family = binomial,
#>     data = peng_new)
#>
#> Coefficients:
#>    (Intercept)  bill_length_mm     body_mass_g
#>     -31.990945        0.091252        0.006257
#>
#> Degrees of Freedom: 341 Total (i.e. Null);  339 Residual
#>   (2 observations deleted due to missingness)
#> Null Deviance:       446.8
#> Residual Deviance: 115.3     AIC: 121.3
glm(is_chinstrap ~ bill_length_mm , data = peng_new, family = binomial,control=glm.control(maxit=15))
#>
#> Call:  glm(formula = is_chinstrap ~ bill_length_mm, family = binomial,
#>     data = peng_new, control = glm.control(maxit = 15))
#>
#> Coefficients:
#>    (Intercept)  bill_length_mm
#>       -13.9756          0.2735
#>
#> Degrees of Freedom: 341 Total (i.e. Null);  340 Residual
#>   (2 observations deleted due to missingness)
#> Null Deviance:       341.2
#> Residual Deviance: 263.1     AIC: 267.1
```

Each column of coefficients corresponding to one class of y's. The test code went through comparisons with one-vs-all for each class by comparing the coefficients. We could say that the multi_logreg provided pretty close coefficients as glm.

```
sum(peng_spec$pred_y != y)/length(y)
#> [1] 0.04385965
```

About 4.38% of data was misclassified by the model, which is a pretty decent result.