homework-3

```
library(bis557)
#> Warning: package 'rsample' was built under R version 3.6.3
#> Warning: package 'foreach' was built under R version 3.6.3
#> Warning: package 'tibble' was built under R version 3.6.3
#> Warning: package 'ggplot2' was built under R version 3.6.3
#> Warning: package 'purrr' was built under R version 3.6.3
#> Warning: package 'glmnet' was built under R version 3.6.3
#> Warning: package 'Matrix' was built under R version 3.6.3
```

Problem 1

> CASL 5.8 Exercise number 2. Include the write up in your homework-2 vignette.
>
> We mentioned that the Hessian matrix in Equation 5.19 can be more illconditioned 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.

Linear Hessian

First, calculate linear Hessian (X^tX). Calculate the ratio of max singular value to minimun singular value. The result is small, which means linear Hessian is well-conditioned.

```
X <- cbind(rep(1,6), c(-3, -3, -3, 3, 3, 3))
H <- t(X) %*% X

# Refer to CASL p100
singular_values <- svd(H)$d
max(singular_values) / min(singular_values)
#> [1] 9
```

Logistic Variation

Then, calculate logistic variation(X^tDX). Calculate the ratio of max singular value to minimun singular value. The result is large, which means logistic variation is ill-conditioned.

```
beta <- c(0.2, 2)
mu <- 1 / (1 + exp(-X %*% beta))
D <- diag(as.vector(mu), 6, 6)
H <- t(X) %*% D %*% X
singular_values <- svd(H)$d
max(singular_values) / min(singular_values)
#> [1] 921.9756
```

Problem 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.

Set up the dataset for comparison

```
data("penguinsi")
penguinsi$sex <- as.numeric(penguinsi$sex) - 1
form <- sex ~ bill_length_mm + bill_depth_mm
d = penguinsi
mms <- make_model_matrices(form, d)
X <- mms$X
Y <- mms$Y
```

Results for constant step

The function `glm_constant` has been established in a seperate file `gradient_descent_glm.R`. Now check the results for a constant step.

```
(fit_c <- glm_constant(X, Y, mu_fun = function(eta) 1/(1+exp(-eta)), var_fun = function(eta) eta))
#> $beta
#>                       [,1]
#> (Intercept)    -21.3144545
#> bill_length_mm   0.2314536
#> bill_depth_mm    0.6548705
```

Results for adaptive step

We can incorporate a term known as the *momentum* into the algorithm. The gradient computed on each mini-batch is used to update the momentum term, and the momentum term is then used to update the current weights.This setup gives the SGD algorithm three useful properties: if the gradient remains relatively unchanged over several steps it will 'pick-up speed' (momentum) and effectively use a larger learning rate; if the gradient is changing rapidly, the step-sizes will shrink accordingly; when passing through a saddle point, the built up momentum from prior steps helps propel the algorithm past the point. (refer to CASL p230)

The function `glm_adapt` has been established in a seperate file `gradient_descent_glm.R`. Now check the results for an adaptive step.

```
(fit_a <- glm_adapt(X, Y, mu_fun = function(eta) 1/(1+exp(-eta)), var_fun = function(eta) eta))
#> $beta
#>                       [,1]
#> (Intercept)    -25.7497962
```

```
#> bill_length_mm   0.2805351
#> bill_depth_mm    0.7894320
```

Reference results

Use the results from `glm` function as the reference.

```
(fit_ref <- glm(form, penguinsi, family = binomial)$coefficients)
#>    (Intercept) bill_length_mm  bill_depth_mm
#>    -25.7498909      0.2805361      0.7894348
```

Conclusion

Combine the results together to compare, we can see that the constant step method fits fairly good, but not quite close to the reference. The adaptive step method fits much better, and even equivalent to the reference.

```
compare <- as.data.frame(cbind(fit_c$beta, fit_a$beta, fit_ref))
colnames(compare) <- c('constant','adaptive','reference')
compare
#>                  constant    adaptive   reference
#> (Intercept)    -21.3144545 -25.7497962 -25.7498909
#> bill_length_mm   0.2314536   0.2805351   0.2805361
#> bill_depth_mm    0.6548705   0.7894320   0.7894348
```

Problem 3

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

Set up the dataset for comparison

```
data("penguinsi")
# penguinsi$species <- as.numeric(penguinsi$species)
# 1:"Adelie" 2:"Chinstrap" 3:"Gentoo"
form <- species ~ bill_length_mm + bill_depth_mm
d <- penguinsi
```

Implement a classification model

The function `multiclass_logistic` has been established in a seperate file `multiclass_logistic.R`. Now check the classification results. The results contain: (1) beta coefficients; (2) the misclassification error; (3) a detailed classification table

```
multiclass_logistic(form, d)
#> $Coefficients
```

```
#>           (Intercept) bill_length_mm bill_depth_mm
#> Adelie       24.13554     -2.2102803     3.9988123
#> Chinstrap   -31.63739      0.3543506     0.8010768
#> Gentoo       49.03383      0.5577040    -4.5335857
#>
#> $MisclassificationError
#> [1] 0.04360465
#>
#> $ClassificationTable
#>            Y
#> Y_hat       Adelie Chinstrap Gentoo
#>   Adelie       151         5      0
#>   Chinstrap      1        56      2
#>   Gentoo         0         7    122
```

Use multinomial regression as a reference answer

```
library(nnet)

# build model
multinomModel <- multinom(form, data = penguinsi)
#> # weights:  12 (6 variable)
#> initial  value 377.922627
#> iter  10 value 26.680270
#> iter  20 value 24.006223
#> iter  30 value 23.973300
#> iter  40 value 23.957800
#> iter  40 value 23.957800
#> final  value 23.957800
#> converged
summary(multinomModel)
#> Call:
#> multinom(formula = form, data = penguinsi)
#>
#> Coefficients:
#>           (Intercept) bill_length_mm bill_depth_mm
#> Chinstrap   -26.30741       2.252261     -3.979117
#> Gentoo       23.22270       2.737199     -8.327064
#>
#> Std. Errors:
#>           (Intercept) bill_length_mm bill_depth_mm
#> Chinstrap    14.00946      0.7109681      1.505729
#> Gentoo       20.23527      0.7283796      1.825504
#>
#> Residual Deviance: 47.9156
#> AIC: 59.9156

# prediction performance
predicted_class <- predict(multinomModel, penguinsi)
table(predicted_class, penguinsi$species)
#>
#> predicted_class Adelie Chinstrap Gentoo
```

```
#>       Adelie       149         3      0
#>       Chinstrap      3        61      2
#>       Gentoo         0         4    122

# misclassification error
mean(as.character(predicted_class) != as.character(penguinsi$species))
#> [1] 0.03488372
```

Conclusion

The misclassification error is 4.4% for my method and 3.5% for `multinom` function in `nnet` package. They are similar while `multinom` performs better in this case.