BIS557_HW3

```
library(bis557)
```

(1)CASL 5.8 Exercise number 2:

We know the logistic variation of Hessian matrix is $X^T DX$ where D is the diagonal matrix with elements $p_i(1-p_i)$. $X^T DX$ can be ill-conditioned if the eigenvalues of D is very small. In this case, the inverse of Hessian will be very large. Based on the theory, we can generate matrix X and probability p_i to make $X^T X$ well-conditioned but $X^T DX$ ill-conditioned.

```
X <- matrix(rnorm(16,mean=0,sd=1),ncol = 4, nrow = 4)
solve(t(X)%*%X)
```

```
#>             [,1]        [,2]        [,3]        [,4]
#> [1,]  1.12148176  0.15800331 -0.07024989 -0.24300130
#> [2,]  0.15800331  1.18470906  0.23014928 -0.01324702
#> [3,] -0.07024989  0.23014928  0.47092955  0.02962443
#> [4,] -0.24300130 -0.01324702  0.02962443  0.22604236
```

```
#Set ture pi vector
pi <- c(0.001,10e-5,10e-8,10e-12)
D <- diag(pi*(1-pi))
solve(t(X)%*%D%*%X)
```

```
#>              [,1]        [,2]         [,3]         [,4]
#> [1,]  21569574216  7341657721  19450327342 -15686997475
#> [2,]   7341657721  2501863260   6624862002  -5338992504
#> [3,]  19450327342  6624862002  17546238060 -14145104161
#> [4,] -15686997475 -5338992504 -14145104161  11408806542
```

We can see that the $X^T X$ is well-conditioned since its entries have reasonable values while logistic variation is ill-conditioned because the entries are very large. This will lead to bad convergence of Newton-Rhapson algorithm.

(2) Here we implement first-order GLM maximum likelihood method. We generate Poisson distributed data and compare our fitted coefficients to the golden standard `glm()` function.

I created a function `first_order_glm()` to implement first-order GLM maximum likelihood problem. This function can give the user option to use constant step size or adaptive step size.

```
#Simulate Poisson distributed data
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)


#constant step size
beta_hat <- bis557::first_order_glm(X, y,mu_fun = function(eta) exp(eta), size = 0.00001,
```

```
                                    adapt = FALSE,maxit = 1000)
beta_glm <- coef(glm(y ~ X[,-1], family = "poisson"))
cbind(beta, beta_hat, as.numeric(beta_glm))
```

```
#>      beta
#> [1,] -1.0 -1.00788141 -1.00788151
#> [2,]  0.2  0.21095809  0.21095817
#> [3,]  0.1  0.08541857  0.08541861
```

We can see that our results is quite similar to the `glm()`with the step size 0.00001 and maximum iteration 1000.

Next, we implement adaptive gradient descent by setting the variable `adapt=TRUE` Which will implement momentum method. The momentum update is the weighted average of previous update and gradient, and the weight is controlled by the parameter γ. Comparing to the constant size, this can help our algorithm converge faster since it can average out the vertical oscillation but maintain the horizontal oscillation. Empirically speaking, $\gamma = 0.9$ is a good choice.

```
#Momentum gradient descent
beta_hat1 <- bis557::first_order_glm(X, y,mu_fun = function(eta) exp(eta), size = 0.001,gamma=0.9,
                                    adapt = TRUE,maxit = 50)
cbind(beta,beta_hat1)
```

```
#>      beta
#> [1,] -1.0 -1.01314521
#> [2,]  0.2  0.21289238
#> [3,]  0.1  0.08618368
```

We can see that our fitted coefficients is quite close to the true β, and we only need 50 iterations to converge. If we use constant step size, we need more iterations to converge.

(3) Here we extend binary logistic regression into K-classes multinomial logistic regression. We use second-order method to update coefficients to ensure fast convergence. The function here is `multi_logistic()`. Just like the binary logistic regression function, I extended it to K-classes by looping over each class and evaluate the probability by softmax calculation. We will measure our model's performance by computing its misclassification rate.

```
#Use randomized iris data to test our code
ind <- sample(150,replace = F)
iris_new <- iris[ind,]
X <- as.matrix(iris_new[,-5])
y <- iris_new[,5]

fit <- bis557::multi_logistic(X,y,maxit=20)
```

Then we check our function by inspecting the fitted probabilities:

```
#We check the fitted class probabilities for each class
head(fit$fitted.p,10)
```

```
#>               [,1]         [,2]         [,3]
#>  [1,] 9.972888e-01 6.803637e-13 2.711166e-03
#>  [2,] 4.582742e-13 1.000000e+00 9.183681e-30
#>  [3,] 2.156204e-02 5.526729e-23 9.784380e-01
#>  [4,] 2.052958e-11 1.000000e+00 8.820558e-26
#>  [5,] 9.273516e-02 2.025727e-19 9.072648e-01
#>  [6,] 1.562753e-02 4.849604e-24 9.843725e-01
#>  [7,] 9.999109e-01 1.040241e-13 8.906050e-05
```

```
#>  [8,] 2.116961e-18 1.000000e+00 1.924352e-39
#>  [9,] 1.107612e-03 1.126508e-28 9.988924e-01
#> [10,] 2.440549e-05 1.899546e-31 9.999756e-01
```

We will assign the responses to the class with largest probability. For example, the 1st observation has largest probability in class 1, which is Virginica. Hence, we classify it into Virginica.

We check the fitted classes:

```
fit$fitted.y
```

```
#>   [1] versicolor setosa     virginica  setosa     virginica  virginica
#>   [7] versicolor setosa     virginica  virginica  virginica  setosa
#>  [13] virginica  virginica  versicolor virginica  versicolor versicolor
#>  [19] versicolor virginica  virginica  virginica  versicolor versicolor
#>  [25] virginica  setosa     setosa     versicolor virginica  setosa
#>  [31] virginica  virginica  setosa     virginica  virginica  virginica
#>  [37] setosa     setosa     setosa     versicolor setosa     versicolor
#>  [43] versicolor versicolor virginica  setosa     versicolor virginica
#>  [49] setosa     setosa     versicolor setosa     virginica  versicolor
#>  [55] versicolor versicolor virginica  setosa     versicolor setosa
#>  [61] setosa     virginica  virginica  setosa     setosa     virginica
#>  [67] versicolor setosa     versicolor setosa     setosa     versicolor
#>  [73] versicolor virginica  versicolor setosa     setosa     virginica
#>  [79] virginica  virginica  versicolor versicolor virginica  versicolor
#>  [85] setosa     virginica  virginica  versicolor versicolor versicolor
#>  [91] versicolor setosa     setosa     versicolor virginica  versicolor
#>  [97] setosa     virginica  virginica  virginica  virginica  versicolor
#> [103] setosa     virginica  setosa     setosa     setosa     virginica
#> [109] setosa     setosa     setosa     versicolor versicolor versicolor
#> [115] versicolor virginica  setosa     setosa     setosa     setosa
#> [121] setosa     versicolor versicolor virginica  setosa     versicolor
#> [127] versicolor setosa     setosa     virginica  virginica  versicolor
#> [133] setosa     versicolor versicolor setosa     versicolor virginica
#> [139] versicolor virginica  virginica  versicolor setosa     setosa
#> [145] versicolor virginica  setosa     versicolor virginica  virginica
#> Levels: setosa versicolor virginica
```

We compute the misclassification rate:

```
fit$misclassification
```

```
#> [1] 0.04
```

We can see that our model is quite good to make accurate classification for 3 classes in `iris` data.