Homework 3 Vignette

Kim Gannon

2020-10-30

```
library(bis557)
```

This vignette contains the answers from BIS 557 HW3.

Problem 1

Consider the specific case of logistic regression

```
  X <- matrix(c(-1e10,1e10,.0000001,10000001), nrow=2, ncol=2)
  print(X)
#>        [,1]  [,2]
#> [1,] -1e+10 1e-07
#> [2,]  1e+10 1e+07
```

We now note that the following code will allow us to invert $X^T X$. However, it will not allow us to invert $X^T DX$ and will return an error. (The line of code which attempts to invert $X^T DX$ is commented out, because the RMD file will not compile if I try to execute the code)

```
 beta <- matrix(rep(1,ncol(X)), ncol=1)
  p <- 1 / (1 + exp(-X %*% beta))
  D <- diag(as.vector(p))
  XtX <- t(X) %*% X
  XtDX <- t(X) %*% D %*% X
  solve(XtX)
#>               [,1]          [,2]
#> [1,]  1.000000e-20 -9.999999e-18
#> [2,] -9.999999e-18  2.000000e-14

  #the code below is commented out; returns error "system is computationally singular" if run:
  #solve(XtDX)
```

Note that $X^T X$ is not computationally singular, but $X^T DX$ is computationally singular; thus, we have our result.

Problem 2

The solution here is contained in the functions glm_gd.R and glm_momentum.R. Here, I opted to compare the constant step size to the Momentum algorithm for adaptive step size. For each iteration, the function glm_momentum. R updates its estimation of beta by a linear combination of the gradient and the previous update. The code used to form glm_gd and glm_momentum are based off code from CASL p. 130.

To compare our GLM functions in a specific Poisson case, we generate some fake data per CASL p. 130

```r
#Create fake Poisson 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 <- stats::rpois(n, lambda = lambda)

#run momentum GD
momentum <- glm_momentum(X, Y,
               mu_fun = function(eta) exp(eta),
              lr=0.00001, gamma=0.8, max_n = 100000, tol = 1e-10)

#run constant step size GD
gd <- glm_gd(X, Y,
             mu_fun = function(eta) exp(eta),
               lr = 0.00001,
             max_n = 100000, tol = 1e-10)
```

```r
#Compare to glm function:
glm(Y ~ X[,-1], family = "poisson")
#>
#> Call:  glm(formula = Y ~ X[, -1], family = "poisson")
#>
#> Coefficients:
#> (Intercept)     X[, -1]1     X[, -1]2
#>    -1.00297      0.17892      0.06843
#>
#> Degrees of Freedom: 4999 Total (i.e. Null);  4997 Residual
#> Null Deviance:       4620
#> Residual Deviance: 4549  AIC: 7836

#Estimates
print(momentum$beta)
#>             [,1]
#> [1,] -0.99692341
#> [2,]  0.17488349
#> [3,]  0.06706055
print(gd$beta)
#>             [,1]
#> [1,] -1.06299228
#> [2,]  0.21969711
#> [3,]  0.08929019
```

These methods both give similar estimates (which are close to the true value of beta). Momentum should converge faster than constant step size.

Problem 3

I will show how to generalize the classification algorithm to K non-ordinal categories. To solve this problem, I follow the "one-vs-all" approach suggested in 5.5 (page 138) of CASL: fitting K binary models, one for each class. My implementation is captured in the function my_multinom, and uses the palmerpenguins data set as an example.

This function does not work. It seems to diverge every time I try to run it with different constant and adaptive learning rates. However, the code in Problem 3 should at least theoretically work, given the right learning rate.