HW3

Bo Qin

10/30/2020

Part 1

> 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 is well-conditioned but the logistic variation is not.

Restate the equation from 5.19, where D is a diagonal matrix with element noted as D_{ii}:

$$H(l) = X^t DX$$
$$D_{ii} = p_i(1 - p_i)$$

We want to make the diagonal entries of D to be as close as 0 as possible so that the linear hessian is well-conditioned but logistic hessian is not.

```
# construct a X with well-conditioned linear hessian:
X = c(1, -1.5, 2.5, -1.5, 1)
X = cbind(1,X)
H = t(X) %*% X

# Use kappa function to calculate conditon number:
print(kappa(H))
```

```
## [1] 3.046716
```

```
# calculate logistic hessian and its condition number"
betas <- c(100,100)
p = 1 / (1+exp(-X %*% betas))
D <- diag(x=as.vector(p*(1-p)), nrow=5, ncol=5)
L <- t(X) %*% D %*% X
print(kappa(L))
```

```
## [1] Inf
```

We see from the above example for x and p, linear hessian is well-conditioned but the logistic variation is ill-conditioned.

Part 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.

We will use gradient descent to implement the first-order solution for the GLM maximum likelihood problem. As discussed during lecture, we want to model our response variable through an inverse link function g^{-1}described as follow:

$$\mathbb{E}[Y] = g^{-1}(X, \beta),$$

Gradient descent algorithm helps us to find the maximum likelihood by moving in the direction of the gradient listed below:

$$\bigtriangledown l(\beta) = X^T(y - \mathbb{E}[y]).$$

Therefore, for each step, we update beta according to the following equation with η being the learning rate.

$$\beta_{new} = \beta + \eta \bigtriangledown l(\beta),$$

In my package BIS557, I created a new function called glmfo. This function helps to achieve the above specification. It takes input X and Y where X is the features and Y the response vector. You can specify the distribution and the update methods you want to do on gradient descent, along with other hyperparameters like learning rate.

```
set.seed(100)

# create X and Y variables
n <- 1000
X <- cbind(1, matrix(rnorm(n * 3), ncol = 3))

# poisson family with log link
beta <- c(-1, 0.2, 0.5, 0.6 )
Y = rpois(n,exp(X %*% beta))
f = poisson(link='log')

# estimate the results:
b1 = glmfo(X,Y,f,method = "constant")
b2 = glmfo(X,Y,f,method = "momentum")
tab = as.data.frame(cbind(beta,b1$coefficients,b2$coefficients))
names(tab) = c("Real Beta","Constant Step"," Adaptive Momentum Step")
print(tab)
```

```
##   Real Beta Constant Step  Adaptive Momentum Step
## 1      -1.0    -0.9772032              -0.9772139
## 2       0.2     0.1343196               0.1343201
## 3       0.5     0.4743599               0.4743652
## 4       0.6     0.5609762               0.5609817
```

As we see from the above comparison, whether you choose to use Constant step learning rate or adaptive momentum learning rate does not have a huge impact in our case. We can also see that both methods achieve good approximation.

Part 3

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

There are two ways to achieve multiclass logistic classification: 1, One-vs-Rest, 2, One-vs-One. We chose to implement One-vs-Rest for this probelm. One-vs-rest is a heuristic method for using binary classification algorithms for multi-class classification.It involves splitting the multi-class dataset into multiple binary classification problems. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident.

For example, in a multi-class classification problem with classes "Setosa", "Versicolr" and "virginica". This could be divided into three binart classification datasets as follows:

- Classification problem 1: Setosa against [Versicolr and Virginica]
- Classification problem 2: Versicolr against [Setosa and Virginica]
- Classification problem 3: Virginica against [Versicolr and Setosa]

For each observation we have 3 probabilities corresponding to each classification problem. We designate the class of that particular observation to the class that has the highest probability.

In my package BIS557, I created a new function called multiclass. This function helps to achieve the above specification. It takes a formula as input, which also requires a data frame. Make sure your response variable is categorical.

```
# we will continue using the iris dataset
classy = multiclass(form = Species~.,iris)

#compare with true label
acc = sum(classy == iris$Species)/length(iris$Species)
print(paste('accuracy rate for our classification is:',acc))
```

```
## [1] "accuracy rate for our classification is: 0.98"
```