homework-4

```r
library(bis557)
library(casl)
library(reticulate)
#> Warning: package 'reticulate' was built under R version 3.6.2
library(stats)
use_condaenv("r-reticulate")
```

1. In Python, implement a numerically-stable ridge regression that takes into account colinear (or nearly colinear) regression variables. Show that it works by comparing it to the output of your R implementation.

We build a python function for ridge regression and fit data and compare it with the previous homework's result.

```python
import numpy as np

def ridge(X, y, lambda_param):
  u, s, vh = np.linalg.svd(X, full_matrices=False)
  D = np.diag(s/(s**2 + lambda_param))
  beta = np.dot(np.dot(np.dot(vh.T, D),u.T),y)
  return beta
```

This chuck generates matrix to fit in python so that the input data will be the same as R function.

```r
data(iris)
df_no_na <- model.frame(Sepal.Length ~ .,iris)
py$X <- model.matrix(Sepal.Length ~ ., df_no_na)
yname <- as.character(Sepal.Length ~ .)[2]
py$y <- matrix(df_no_na[,yname],ncol = 1)
```

This chuck runs ridge regression given X, y and lambda, the penalty.

```python
py_fit = ridge(X,y,.01)
```

This chuck runs previously implemented ridge regression function and compare the coefficients with python results.

```r
fit_my_ridge <- ridge(Sepal.Length ~ ., iris, lambda = .01)
cbind(py$py_fit, fit_my_ridge$coefficients)
#>            [,1]       [,2]
#> [1,]  2.1497591  2.1497591
#> [2,]  0.5032983  0.5032983
#> [3,]  0.8271273  0.8271273
#> [4,] -0.3215299 -0.3215299
#> [5,] -0.7049677 -0.7049677
#> [6,] -0.9991592 -0.9991592
```

We could see that the function in python, using np.linalg.svd, returns to same result as the previous implemented function in R.

In the colinear case:

```r
data(iris)
iris$Sepal.Width_coll <- iris$Sepal.Width*1.5+0.1
df_no_na <- model.frame(Sepal.Length ~ .,iris)
py$X <- model.matrix(Sepal.Length ~ ., iris)
```

```
yname <- as.character(Sepal.Length ~ .)[2]
py$y <- matrix(df_no_na[,yname],ncol = 1)
```

```
py_ridge_coll =ridge(X,y,.01)
```

```
py_fit <- py$py_ridge_coll
fit_my_ridge <- ridge(Sepal.Length ~ ., iris, lambda = .01)
cbind(py$py_ridge_coll, fit_my_ridge$coefficients)
#>             [,1]        [,2]
#> [1,]  2.11918470  2.11918470
#> [2,]  0.05713734  0.05713734
#> [3,]  0.82706790  0.82706790
#> [4,] -0.32169400 -0.32169400
#> [5,] -0.70444942 -0.70444942
#> [6,] -0.99850564 -0.99850564
#> [7,]  0.29762448  0.29762448
```

We could see that in the collinear case, the python function still works and has the same result as R.

2. Create an "out-of-core" implementation of the linear model that reads in contiguous rows of a data frame from a file, updates the model. You may read the data from R and send it to your Python functions fo fitting.

Out-of-core refers to processing data that is too large to fit into main memory and the algorithm would allow to access data in a sequence. [Ref: https://machinelearning.wtf/terms/out-of-core/]

We write a function for fitting linear model in python using qr decomposition. We read the data in batches from r and fit the current batch with the python function and update the model until we fit all the data.

This chuck is the python function to fit the linear model.

```
import numpy as np

def py_lm(X,y):
  qrs = np.linalg.qr(X)
  q = qrs[0]
  r = qrs[1]
  beta = np.dot(np.dot(np.linalg.inv(r), q.T),y)

  return beta
```

```
#simulate data
n <- 1e5
X <- matrix(c(rnorm(n, 10, 1), rnorm(n, 5, 1), rnorm(n, 2, 2)),nrow = n)
y <- rnorm(n,50,1)

# create batches
batch <- 100
b_size <- n/batch
beta <- matrix(rep(0,ncol(X)*batch), nrow = batch)

for (i in 1:batch){
  #create batches
  y_b <- y[(b_size*(i-1)+1):(b_size*i)]
  X_b <- X[(b_size*(i-1)+1):(b_size*i),]
  #read in python and fit model
  py$X_b <- X_b
  py$y_b <- y_b
  beta[i,] <- py$py_lm(X_b,y_b)
}
#compute mean
beta_final <- apply(beta,2,mean)
beta_final
#> [1] 3.9262573 1.9856991 0.2007395

lm(y~X-1)
#>
#> Call:
#> lm(formula = y ~ X - 1)
#>
#> Coefficients:
#>     X1      X2      X3
#> 3.9266  1.9847  0.2009
```

Comparing with fitting lm directly, we get a pretty close result. If we don't have enough memory to hold all data for fitting, using out-of-core algorithm would be a decent choice.

3. Implement your own LASSO regression function in Python. Show that the results are the same as the function implemented in the `casl` package.

We build a LASSO function in python here. It differs the Ridge regression by the first order norm for the penalization term. Referring to [https://stats.stackexchange.com/questions/17781/derivation-of-closed-form-lasso-solution], we find the ridge parameters:

```
def py_lasso(X,y, lambda_param):
  n = len(X)

  qrs = np.linalg.qr(X)
  q = qrs[0]
  r = qrs[1]
  #least square
  b_ls = np.dot(np.dot(np.linalg.inv(r), q.T),y)
  #soft hold
  b_max = np.maximum(np.abs(b_ls)-lambda_param,0)
  beta = np.sign(b_ls)*b_max

  return beta
```

```
#simulate data as page 192
n <- 1000
p <- 5
X <- matrix(rnorm(n * p), ncol = p)
beta <- c(3, 2, 1, rep(0, p - 3))
y <- X %*% beta + rnorm(n = n, sd = 0.1)
bhat <- casl_lenet(X, y, lambda = 0.01)
bhat
#>           [,1]
#> [1,] 2.9945196
#> [2,] 1.9862863
#> [3,] 0.9896681
#> [4,] 0.0000000
#> [5,] 0.0000000
```

```
py$X <- X
py$y <- y
py$py_lasso(X,y,.01)
#>           [,1]
#> [1,] 2.9949377
#> [2,] 1.9873291
#> [3,] 0.9888466
#> [4,] 0.0000000
#> [5,] 0.0000000
```

```
bhat <- casl_lenet(X, y, lambda = 0.1)
bhat
#>           [,1]
#> [1,] 2.9017117
#> [2,] 1.8849432
#> [3,] 0.9095382
#> [4,] 0.0000000
#> [5,] 0.0000000

py$py_lasso(X,y,.1)
#>           [,1]
#> [1,] 2.9049377
#> [2,] 1.8973291
#> [3,] 0.8988466
#> [4,] 0.0000000
#> [5,] 0.0000000
```

Comparing lasso results with penalty .01 and .1 with casl package, we got very similar results.

4. Propose a final project for the class.

I am thinking of using CNN to classify cat and dog images from a kaggle dataset cat and dog [https://www.kaggle.com/tongpython/cat-and-dog]. I will follow the guidance from class and build a convolutional neural network with keras. It will take steps to train and test the model with two datasets of images, and classify the images.

I will try 2D layers with different activation functions, kernel sizes, and different batch sizes and epochs to get a model with relatively high prediction accuracy. The training process could provide visualization along with training process.