

homework-4

Name: Brian Deng

```
library(bis557)
library(reticulate)
library(cas1)
```

Name: Brian Deng (BIS557 HW4)

Question 1

We will use the **Python** function `bis557::ridge_py_hw4a()` for **ridge** regression (thanks to the `{reticulate}` library), where the *penalty* L equals:

$$L = \frac{1}{2n} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2.$$

From the textbook, we solve using the formula:

$$\hat{\beta}_{ridge} = (X^T X + \lambda I_p)^{-1} X^T Y$$

Remember that for SVD, we have $X = U\Sigma V^T$. Then (from the textbook), a way to write the estimated coefficients is:

$$\hat{\beta}_{ridge} = V \cdot \text{Diag}\left(\frac{\sigma_1}{\sigma_1^2 + \lambda}, \dots\right) U^T Y$$

We show that as $\lambda \rightarrow \infty$, then $\hat{\beta}_{ridge} \rightarrow 0$. Of course, we will compare **Python** and **R**.

```
# Show ridge regularization
data(iris)
y <- matrix(iris$Sepal.Length, ncol = 1)
X <- model.matrix(~. - Sepal.Length, data = iris)
b_ridge_10_py <- bis557::ridge_py_hw4a(y, X, lambda_val = 10)
b_ridge_10 <- bis557::ridge_hw2c(form = Sepal.Length ~ ., d = iris,
                                lambda_val = 10)
b_ridge_01_py <- bis557::ridge_py_hw4a(y, X, lambda_val = 1)
b_ridge_01 <- bis557::ridge_hw2c(form = Sepal.Length ~ ., d = iris,
                                lambda_val = 1)

# Python vs R
df1 <- cbind("lm()" = lm(Sepal.Length ~ ., iris)$coefficients,
             "Python: lam=1" = b_ridge_01_py$coefficients,
             "R: lam=1" = b_ridge_01$coefficients,
             "Python: lam=10" = b_ridge_10_py$coefficients,
             "R: lam=10" = b_ridge_10$coefficients)
colnames(df1) <- c("lm()", "Python: lam=1", "R: lam=1",
```

```

"Python: lam=10", "R: lam=10")
print(df1)
#>               lm() Python: lam=1    R: lam=1 Python: lam=10    R: lam=10
#> (Intercept)      2.1712663      1.2627675  1.2627675      0.5321332  0.5321332
#> Sepal.Width      0.4958889      0.7927480  0.7927480      1.0159230  1.0159230
#> Petal.Length      0.8292439      0.7551188  0.7551188      0.6328160  0.6328160
#> Petal.Width      -0.3151552     -0.4557292 -0.4557292     -0.1712601 -0.1712601
#> Speciesversicolor -0.7235620     -0.1325378 -0.1325378      0.1008790  0.1008790
#> Speciesvirginica  -1.0234978     -0.2956002 -0.2956002     -0.1099248 -0.1099248
cat("\n")

# Show that ridge regression works for colinear regression variables
data(lm_patho)
y <- matrix(lm_patho$y, ncol = 1)
X <- model.matrix(~. - y, data = lm_patho)
b_patho_py <- bis557::ridge_py_hw4a(y, X, lambda_val = 1)
b_patho <- bis557::ridge_hw2c(form = y ~ ., d = lm_patho,
                             lambda_val = 1)

# Python vs R
df2 <- cbind("lm()" = lm(y ~ ., lm_patho)$coefficients,
             "Python: lam=1" = b_patho_py$coefficients,
             "R: lam=1" = b_patho$coefficients)
colnames(df2) <- c("lm()", "Python: lam=1", "R: lam=1")
print(df2)
#>               lm() Python: lam=1    R: lam=1
#> (Intercept) 1.003095e-05  5.000000e-06  5.000000e-06
#> x1          1.000000e+00  1.000000e+00  1.000000e+00
#> x2          NA    -9.50015e-10 -9.50015e-10
cat("\n")

```

Therefore, the results from using Python are **similar** to the results from using R! Of course, the coefficients are different (*and* smaller in magnitude) as λ increases.

Question 2

We will use the **Python** function `bis557::linear_model_py_hw4b()` to fit linear models. Of course, we will first read in a data frame using **batches of contiguous rows** to find the coefficients, then take the average of all the coefficients.

The example below will use millions of rows ($n = 5e6$) and several predictors, and this implementation will use $K = 100$ batches.

```

# Create large data frame
set.seed(2020)
K <- 1e2; n <- 5e6; p <- 8
X <- matrix(rnorm(n*p, mean = 2, sd = 4), nrow = n, ncol = p)
X <- as.data.frame(X)
colnames(X) <- c("y", paste("x", 1:7, sep = ""))

# store coefficients for all K = 100 folds
betas <- matrix(nrow = K, ncol = p)

# Python linear model

```

```

for (i in 1:K) {
  b_batch <- linear_model_py_hw4b(form = y ~ .,
                                  d = X[ceiling((i-1)*n/K + 1):ceiling(i*n/K),])
  betas[i,] <- b_batch$coefficients
}

# Take the average of the coefficients
b_hat_py <- colMeans(betas)

# Test: Compare "lm()" for one batch vs lm for contiguous rows
print(df <- cbind("lm()" = lm(y ~ ., X)$coefficients,
                  "Python: K=100" = b_hat_py))

#>               lm() Python: K=100
#> (Intercept)  1.9945923101  1.9946180380
#> x1           0.0005633802  0.0005692084
#> x2          -0.0002472777 -0.0002513050
#> x3           0.0004836360  0.0004768623
#> x4           0.0003176754  0.0003342075
#> x5           0.0006664914  0.0006530239
#> x6           0.0001054424  0.0001100508
#> x7           0.0007417661  0.0007382857
cat("\n")

```

Therefore, the “out-of-core” implementation of creating a linear model from big data by **reading in contiguous rows is reliable** (but of course, there will be a few limitations).

Question 3

Here, we let j be a predictor. Here, Y is a column vector with length n , and β is a column vector with length p . Also, X is a matrix with dimension $n \times p$. For notation purposes, let X_j be the j -th column of X .

We will use the **Python** function `bis557::lasso_py_hw4c()` for **LASSO** regression (thanks to the {reticulate} library), where the *penalty* L equals:

$$L = \frac{1}{2n} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_1.$$

From the results of HW2 Question 5 (generalized from the textbook), given that X is *orthonormal*, we have:

$$\hat{\beta}_j^{LASSO} = \text{sign}(X^T Y)_j \cdot [(X^T X)^{-1} \cdot \max(|X^T Y| - n\lambda, 0)]_j.$$

We show that as $\lambda \rightarrow \infty$, then more coefficients of $\hat{\beta}_j^{LASSO} = 0$, showing **subset selection**. Of course, we will compare **Python** and **R**.

```

# Show LASSO regularization
set.seed(2020)
n <- 100; p <- 10
X <- matrix(rnorm(n*p, sd = 10), nrow = n, ncol = p)
y <- matrix(rnorm(n, sd = 10), ncol = 1)

# Orthonormalize the matrix
Q <- qr.Q(qr(X))
b_lasso1_py <- bis557::lasso_py_hw4c(y, Q, lambda_val = 1e-2)

```

```

b_lasso2_py <- bis557::lasso_py_hw4c(y, Q, lambda_val = 1e-1)
b_lasso3_py <- bis557::lasso_py_hw4c(y, Q, lambda_val = 1e0)
b_lasso1_r <- casl::casl_lenet(Q, y, lambda = 1e-2, maxit = 1e3L)
b_lasso2_r <- casl::casl_lenet(Q, y, lambda = 1e-1, maxit = 1e3L)
b_lasso3_r <- casl::casl_lenet(Q, y, lambda = 1e0, maxit = 1e3L)

# Python vs R: lambda = 0.01, 0.1
df <- cbind("lm()" = lm(y ~ Q)$coefficients,
            "Python: lam=0.01" = b_lasso1_py$coefficients,
            "R: lam=0.01" = b_lasso1_r,
            "Python: lam=0.1" = b_lasso2_py$coefficients,
            "R: lam=0.1" = b_lasso2_r)

#> Warning in cbind(`lm()` = lm(y ~ Q)$coefficients, `Python: lam=0.01` =
#> b_lasso1_py$coefficients, : number of rows of result is not a multiple of vector
#> length (arg 1)
colnames(df) <- c("lm()", "Python: lam=0.01", "R: lam=0.01", "Python: lam=0.1",
                  "R: lam=0.1")

print(df)
#>               lm() Python: lam=0.01 R: lam=0.01 Python: lam=0.1 R: lam=0.1
#> [1,]  0.6988301      2.2039765    2.2039765      0.000000    0.000000
#> [2,]  3.8840249      1.4059743    1.4059743      0.000000    0.000000
#> [3,]  3.0910818      4.5668376    4.5668376      0.000000    0.000000
#> [4,]  6.4920731     -17.7982996   -17.7982996     -8.798300   -8.798300
#> [5,] -18.8893018     -5.5370965   -5.5370965      0.000000    0.000000
#> [6,] -5.2544532     -16.7750310   -16.7750310     -7.775031   -7.775031
#> [7,] -17.9454628     13.7562705    13.7562705      4.756270    4.756270
#> [8,]  15.4651843      8.2012773     8.2012773      0.000000    0.000000
#> [9,]  10.2319072     -7.1805125    -7.1805125      0.000000    0.000000
#> [10,] -7.6888157      0.6603727     0.6603727      0.000000    0.000000
cat("\n")

```

Therefore, the results from using Python are **similar** to the results from using R! Of course, the coefficients are different (*and* smaller in magnitude) as λ increases.

Question 4

For this **final** project, I propose to perform an analysis of a large dataset (with at least 100,000 rows and 70+ columns). This will use a combination of computational ML methods, multivariate statistics with creative models, and probably a deep learning model with *specialized* user-input penalty weights. Analyzing a large dataset to make ML predictions requires some **domain knowledge** to succeed in designing the loss and optimization function.

The dataset that I want to investigate is **Boston's** Property Assessment on **residential** real estate properties for 2020. I hope to **investigate** the analytics of Boston's high demand for residential real estate, in which some domain knowledge (e.g. real estate, Boston's demographics and geography, etc.) and user-specialized ML algorithms are useful. This CSV data was retrieved from **Analyze Boston**, which is the main source for open-source data of Boston. The website for Boston's Property Assessment is below:

<https://data.boston.gov/dataset/property-assessment>.

Using tools that I learned from BIS 557, I hope to use this dataset to study how Boston's residential properties are related (or different) by examining various characteristics, such as distance from downtown Boston, building style, property value, house size, year built, and annual property tax. This is where creative ML and deep learning methods (as well as multivariate stats methods) come to play - such as clustering (and

playing around with the algorithms and taking advantage of *domain knowledge* to **change parameters and weights** (i.e. assigning heavier weights or penalties to a particular zip code).