homework-4

Yijun Yang

```r
library(bis557)
library(glmnet)
library(reticulate)
library(purrr)
use_condaenv("r-reticulate")
```

Problem 1

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

- Prepare the data for comparison

```r
data("iris")

# create a colinear term
iris$colinear <- 2*iris$Petal.Width + 0.1

form <- Sepal.Length ~ .
d <- iris

mms <- make_model_matrices(form, d, contrasts = NULL)
X <- mms$X
Y <- mms$Y
```

- Implement the ridge regression function in R

```r
# run the ridge regression function - it works without error
(r_result <- bis557::ridge_regression(form, d, lambda = 10))
#> $coef
#>             [,1]
#> [1,]  0.52563557
#> [2,]  1.01231727
#> [3,]  0.67824973
#> [4,] -0.08026950
#> [5,]  0.10040623
#> [6,] -0.08580804
#> [7,] -0.10797544
#>
#> $form
#> Sepal.Length ~ .
#>
```

```
#> attr(,"class")
#> [1] "ridge_regression"
```

- Implement the ridge regression function in Python
    - The function has been built in a seperated file `ridge.py` under the folder `python`

```
source_python("D:/YALE/20Fall/BIS557/bis557/python/ridge.py")
(p_result <- ridge_reg(X, Y, 10))
#>             [,1]
#> [1,]  0.52563557
#> [2,]  1.01231727
#> [3,]  0.67824973
#> [4,] -0.08026950
#> [5,]  0.10040623
#> [6,] -0.08580804
#> [7,] -0.10797544
```

- Compare the results

```
compare <- as.data.frame(cbind(r_result$coef, p_result))
colnames(compare) <- c("R", "Python")
round(compare, 4)
#>         R  Python
#> 1  0.5256  0.5256
#> 2  1.0123  1.0123
#> 3  0.6782  0.6782
#> 4 -0.0803 -0.0803
#> 5  0.1004  0.1004
#> 6 -0.0858 -0.0858
#> 7 -0.1080 -0.1080
```

- Conclusion

The Python results are the same as the output of my R implementation.

Problem 2

> Create an “out-of-core” implementation of the linear model that reads in contiguous rows of a dataframe from a file, updates the model. You may read the data from R and send it to your Python functions for fitting.

We define the loss function as:

$$\mathcal{L} = (\hat{y} - y)^2$$

To minimize the loss, we iteratively update the values of w and b using the value of gradient:

$$w_{new} = w_{old} - \eta \frac{\partial \mathcal{L}}{\partial w_{old}}$$
$$\frac{\partial \mathcal{L}}{\partial w_{old}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_{old}} = 2(\hat{y} - y)x$$
$$b_{new} = b_{old} - \eta \frac{\partial \mathcal{L}}{\partial b_{old}}$$
$$\frac{\partial \mathcal{L}}{\partial b_{old}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial b_{old}} = 2(\hat{y} - y)$$

(Reference: Tutorial: Linear Regression with Stochastic Gradient Descent)

- Simulate a dataset for testing

```
# refer to CASL p192
import numpy as np
np.random.seed(2020)
n = 100
p = 5
X = np.random.randn(n, p)

# real coefficients
w = np.random.randint(-5, 5, p)
b = np.random.randint(-5, 5, 1)
y = X @ w + b

print(w) # print the real w
#> [ 1  0 -1  2 -5]
print(b) # print the real b
#> [-4]
```

- Implement the sgd function in Python
  - The function has been built in a seperated file `sgd.py` under the folder `python`

```
source_python("D:/YALE/20Fall/BIS557/bis557/python/sgd.py")
```

```
# initialize w and b
w = np.ones(p)
b = 0

for i in range(int(n)):
  w, b = sgd(X = X[i, :], y = y[i], w = w, b = b, eta = 0.01 )

np.around(w, decimals = 1) #print the estimated w
#> array([ 1. , -0.2, -0.7,  1.9, -4.2])
np.around(b, decimals = 1) #print the estimated b
#> -3.5
```

- Conclusion

The results of `sgd` function are quite similar to the real coefficients, which means `sgd` performs well for linear regression.

Problem 3

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

- Initiate a simulated dataset for testing

```
# refer to CASL p192
set.seed(2020)
n2 <- 100
p2 <- 500
X2 <- matrix(rnorm(n2 * p2), ncol = p2)
beta2 <- c(3, 2, 1, rep(0, p2 - 3))
y2 <- X2 %*% beta2 + rnorm(n = n2, sd = 0.1)
```

- Implement the 'casl_lenet' function

```
library(casl)
# refer to CASL p192
r_bhat <- casl_lenet(X2, y2, lambda = 0.1)
names(r_bhat) <- paste0("v", seq_len(n2))
(r_result <- r_bhat[r_bhat != 0])
#>        v1        v2        v3
#> 2.9084259 1.9035727 0.9114433
```

- Implement my own 'lasso' function
  - The function has been built in a seperated file `lasso.py` under the folder `python`

```
source_python("D:/YALE/20Fall/BIS557/bis557/python/lasso.py")
p_beta <- coord_descent(X2, y2, 0.1)

# only print the non-zero results
(p_result <- p_beta[p_beta != 0])
#> [1] 2.9084259 1.9035728 0.9114433
```

- Compare the results

```
compare <- as.data.frame(cbind(r_result, p_result))
colnames(compare) <- c("R", "Python")
round(compare, 4)
#>          R Python
#> v1 2.9084 2.9084
#> v2 1.9036 1.9036
#> v3 0.9114 0.9114
```

- Conclusion

The results of our `lasso` function are exactly the same as the function implemented in the `casl` package.

Problem 4

The proposal has been written in a seperated file `BIS557 Project Proposal` under the folder `project`.