homework-4

```
library(bis557)
devtools::load_all()
#> Loading bis557
#> Warning: package 'testthat' was built under R version 3.6.2
```

The three functions for the following three problems were all implemented in the file “pyregression.py”, under the name “pyridge”, “onlineLR”, and “pylasso” respectively.

1.

```
library(reticulate)
#> Warning: package 'reticulate' was built under R version 3.6.2
use_python("//anaconda3/envs/bis557/bin/python")
setwd("/Users/damingli/Documents/Courses/Computational statistics/homework-1/bis557")
reticulate::source_python("pyregression.py")
#> Warning: Python '//anaconda3/envs/bis557/bin/python' was requested but '/Users/
#> damingli/Library/r-miniconda/envs/r-reticulate/bin/python' was loaded instead
#> (see reticulate::py_config() for more information)
```

Here we make a toy dataset with collinearity.

Output with R code:

```
n <- 500
p <- 5
beta <- c(1, 2,-1, 0,-2)
X <- matrix(rnorm(n*p), nrow=n, ncol = p)
X[,1] <- X[,1]*0.05 + X[,2]*0.95  # make 1st and 2nd columns collinear
y <- X %*% beta + rnorm(n)

fit_my_rdige <- my_Ridge(X, y, lambda=1.0)
print(fit_my_rdige$coefficients)
#>            [,1]
#> [1,]  1.5161787
#> [2,]  1.5609513
#> [3,] -1.0292266
#> [4,]  0.0590679
#> [5,] -2.0766343
```

Output with python code:

```python
import numpy as np
n = 500
p = 5
X = np.random.randn(n,p)
X[:,0] = X[:,0]*0.05 + X[:,1]*0.95
betas = np.array([1,2,-1,0,-2])  # true coefficients
y = X.dot(betas) + np.random.randn(n)

pyridge(X,y,l=1.0)
#> (array([ 1.17804189,  1.84785521, -1.01589541,  0.04649714, -2.01011462]),
        0.0477337114479043)
```

They match very well.

2.

The following code is equivalent to stochastic gradient descent with batch size 1.

```python
import numpy as np
# create a dataset
n = 10000
p = 5
X = np.random.randn(n,p)
betas = np.array([1,2,-1,0,-2])  # true coefficients
y = X.dot(betas) + np.random.randn(n)
# feed the dataset in stream
betas = np.zeros(5)
intercept = 0
for i in range(n):
  betas, intercept = onlineLR(X[i,:], y[i], betas, intercept, eta=0.01)
print(betas, intercept)
#> [ 1.20509467  2.00955306 -0.84899517 -0.10751325 -2.14357647] -0.09177487551541272
```

The fitted coefficients are very accurate.

3.

Similar to the testing process:

```
import numpy as np
n = 500
p = 5
X = np.random.randn(n,p)
X[:,0] = X[:,0]*0.05 + X[:,1]*0.95
betas = np.array([1,2,-1,0,-2])  # true coefficients
y = X.dot(betas) + np.random.randn(n)

pylasso(X,y,l=1.0)
#> (array([ 1.54601556,  1.4767779 , -1.02337266, -0.04707291, -1.99839406]),
        0.06416963316071739)
```

The fitted coefficients are very good.

4.

Proposing a project:

Title: Effects of dimensionality and choice of distance metrics on the accuracy and stability of kmeans algorithm

Background: This topic is slightly different from the materials covered in BIS557, but is highly relevant in terms of methodology. At a first place, statistical softwares implementing kmeans algorithm with custom distance metrics are largely missing. Therefore, the first step is to build a package for this purpose using standard EM algorithm. Next, there are many directions to explore the algorithm: 1) how do distance metrics affect convergence? 2) how does the algorithm with different distance metrics perform as the dimensionality of data increases? 3) how to choose the optimal k?