Hw3_Yukai Wang

Yukai Wang

2019/11/21

reference for setting up python version: https://github.com/rstudio/reticulate/issues/368

This homework is due by the end of the day on November 22 2019. All solutions should be written in Python but you should provide both code and an RMarkdown file to present the results. If you don't receive a link to the private Github account for the assignment, you can use the link here. 1. Implement a ridge regression that takes into account colinear (or nearly colinear) regression variables. 2. Implement your own method for optimizing the ridge parameter. 3. Implement your own LASSO regression function. Show that the results are the same as the function implemented in the casl package. 4. 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, and then moves on to the next set of contiguous rows.

Check the python version

```
Sys.which("python")
```

```
##                  python
## "/anaconda3/bin/python"
```

Q1

Implement a ridge regression that takes into account colinear (or nearly colinear) regression variables.

```
#set up python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#from sklearn.linear_model import Ridge
import seaborn as sns


#set up simulated data
iris = sns.load_dataset("iris")
X = iris[["sepal_width", "petal_length", "petal_width"]]
Y = iris["sepal_length"]

#reference to http://fa.bianp.net/blog/2011/ridge-regression-path/

#based on SVD to conduct the ridge regression
def ridge_regression(X, Y, lambda_ridge):
    U, s, Vt = np.linalg.svd(X, full_matrices=False)
    d = s / (s** 2 + lambda_ridge)
    return np.dot(d * U.T.dot(Y), Vt).T
```

```
#example
print(ridge_regression(X,Y,1))
```

```
## [ 1.13241137  0.86796453 -0.75342017]
```

Q2

Implement your own method for optimizing the ridge parameter.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold

#set up simulated data
iris = sns.load_dataset("iris")
X = iris[["sepal_width", "petal_length", "petal_width"]]
X = X.values
Y = iris["sepal_length"]
Y = Y.values

#reference to http://fa.bianp.net/blog/2011/ridge-regression-path/
def ridge_regression(X, Y, lambda_ridge):
    U, s, Vt = np.linalg.svd(X, full_matrices=False)
    d = s / (s** 2 + lambda_ridge)
    return np.dot(d * U.T.dot(Y), Vt).T

#using Cross-validation to choose the best lambda
def best_lambda(X, Y, lambdas):
    Sum_error = [0 for i in range(len(lambdas))]
    for k in range(len(lambdas)):
        cv = KFold(n_splits=10,shuffle=False) #split into 10 folders
        for train_index, test_index in cv.split(X): # refer to https://medium.com/datadriveninvestor/k-
            train_x, test_x, train_y, test_y = X[train_index], X[test_index], Y[train_index], Y[test_in
            fit = ridge_regression(train_x, train_y, lambdas[k])
            predict_y = test_x.dot(fit)
            Sum_error[k] = Sum_error[k] + np.sum((predict_y - test_y) ** 2)
        Sum_error[k] = Sum_error[k] / 10

    return (lambdas[Sum_error.index(min(Sum_error))])

#example
print(best_lambda(X,Y,lambdas=[0.1,0.2,0.3,0.4,0.5,0.6,0.7]))
```

```
## 0.5
```

Q3

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

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets

#set up simulated data
np.random.seed(0)
X = np.random.normal(loc=0.0, scale=1.0, size=(1000, 6))
beta = np.array([[3],[2],[1],[0],[0],[0]])
delta = np.random.normal(loc=0.0, scale=0.1, size=(1000, 1))
Y = np.matmul(X, beta) + delta

#The following two functions(soft_threshold and coordinate_descent_lasso) are referring to https://xavi
def soft_threshold(rho,lamda):
    if rho < - lamda:
        return (rho + lamda)
    elif rho >  lamda:
        return (rho - lamda)
    else:
        return 0

#This function refers to https://xavierbourretsicotte.github.io/lasso_implementation.html
def coordinate_descent_lasso(beta,X,Y,lamda):
    #get the number of variables of X
    n = X.shape[1]
    iter=30
    #normalizing X
    X = X / (np.linalg.norm(X,axis = 0))
    #Looping for 100 iterations
    for i in range(1000):
        for j in range(n):
            #implementation the lasso here
            X_j = X[:,j].reshape(-1,1)
            y_pred = X @ beta
            rho = X_j.T @ (Y - y_pred  + beta[j]*X_j)
            beta[j] =  soft_threshold(rho, lamda)/iter
    return beta

n = X.shape[1]
start_beta = np.ones((n,1))
#lamda = 0.1
beta = coordinate_descent_lasso(start_beta,X,Y,lamda = 1)
print(beta)
```

```
## [[ 3.04711286e+00]
##  [ 2.10982698e+00]
##  [ 1.06517772e+00]
##  [-5.18466385e-03]
##  [ 2.17563483e-03]
##  [ 7.54468720e-03]]
```

```r
#using function in Casl to conduct lasso regression
library(casl)
```

```r
set.seed(557)
n <- 1000
p <- 6
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 = 1)
bhat
```

```
##           [,1]
## [1,] 1.8750677
## [2,] 0.8714267
## [3,] 0.0000000
## [4,] 0.0000000
## [5,] 0.0000000
## [6,] 0.0000000
```

The result of the casl and my own function is same, so it works.

Q4