homework-4

Zichen Zhao

11/21/2020

```
library(bis557)
library(reticulate)
use_python("/Users/zc/Library/r-miniconda/envs/r-reticulate/bin/python", required = T)
```

1. In Python, implement a numerically-stable ridge regression that takes into account colinear (or nearly colinear) regression variables:

```
import numpy as np
import random

#Define the ridge regression function
def ridge(y, x, lam):
    U, d, V = np.linalg.svd(x, full_matrices=False)
    d = d.reshape(-1,1)
    x_size = x.shape
    beta = np.empty((1, x_size[1]))
    D = np.diagflat(d/((d**2) + lam))
    beta = V.T.dot(D).dot(U.T).dot(y).reshape(1,-1)
    return beta
```

R output:

```
#Use previous function to get R result
ridge_regression <- function(form, dat, lambda=0, contrasts=NULL) {
  #Eliminating all the NAs within the dataset
  dat_no_na <- model.frame(form, dat)
  rownames(data) <- NULL
  #Define the model matrix
  X <- model.matrix(form, dat_no_na, contrasts.arg = contrasts)
  #Get the name of the dependent variable
  y_name <- as.character(form)[2]
  #Define the dependent variable
  Y <- as.matrix(subset(dat_no_na, select = y_name), ncol = 1)
  #Center the dependent variable
  mean.Y <- mean(Y)
  Y <- Y - mean.Y
  #Center the response variables
  mean.X <- colMeans(X[, -1])
  X <- X[, -1] - rep(mean.X, rep(nrow(X), ncol(X) - 1))
  #Scale the response variables
  scale.X <- drop(rep(1/nrow(X), nrow(X)) %*% X^2)^0.5
  X <- X/rep(scale.X, rep(nrow(X), ncol(X)))
  #Set up beta matrix
  beta <- matrix(NA_real_, nrow = length(lambda), ncol = ncol(X))
```

```
  #Find the coefficients
  svd <- svd(X)
  beta <- svd$v %*% diag(svd$d/(svd$d^2 + lambda)) %*% t(svd$u) %*% Y
  #Return the scale
  beta <- t(as.matrix(beta/scale.X))
  #Calculate the intercept
  intercept <- mean.Y - beta %*% mean.X
  #Add it to the vector
  beta <- cbind(intercept, beta)
  beta <- as.vector(beta)
  #Name intercept and beta
  names(beta) <- c("Intercept", colnames(X))
  #Generate output
  beta
}
iris$duplicate <- iris$Sepal.Width
ridge_regression(Sepal.Length ~ ., iris, lam = 0)
#>         Intercept       Sepal.Width      Petal.Length       Petal.Width
#>      2.241771e+00      5.386740e+12      8.296616e-01     -3.150947e-01
#> Speciesversicolor  Speciesvirginica         duplicate
#>     -7.237568e-01     -1.025154e+00     -5.386740e+12
```

Define y, x and lam in order to call R data from Python:

```
#Define the dependent variable
y_name <- as.character(Sepal.Length ~ .)[2]
y <- as.matrix(subset(iris, select = y_name), ncol=1)
#Define the independent variable
x <- model.matrix(Sepal.Length ~ ., iris)
lam <- 0
```

Python output:

```
#Use R data to python and get results
y = r.y
x = r.x
lam = r.lam
ridge(y, x, lam)
#> array([[ 2.17126629e+00, -1.52057588e+14,  7.93432241e-01,
#>         -3.39029621e-01, -5.95236804e-01, -8.51900225e-01,
#>          1.52057588e+14]])
```

As we can see from the two results, outputs from Python and R are close to each other but there are also some small deviation.

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:

```
import numpy as np
import random

#Define the "out-of-core" linear model function
def batch_lm(y, x, b, intercept, eta):
    y_prev = intercept + x.dot(b)
    db = -2*(x*(y-y_prev))
    dintercept = -2*(y-y_prev)
```

```
    update_intercept = intercept - dintercept*eta
    update_b = b - db*eta
    return update_intercept, update_b
```

```
n = 1000
p = 6
x = np.random.randn(n,p)
b = np.array([3,2,1,-1,-2,0])
y = x.dot(b) + np.random.randn(n)
intercept = 0
b = np.zeros(6)
for i in range(n):
    inter, b = batch_lm(y[i], x[i,:], b, intercept, eta=0.1)
print(inter, b)
#> 0.3356420769880493 [ 3.46893118  1.16127654  1.85316593 -1.88951205 -3.64093508 -0.15573517]
```

3. Implement your own LASSO regression function in Python:

```
import numpy as np
import random

#Define the LASSO regression function based on code from CASL section 7.5
def soft_thresh(a, b):
    if np.abs(a) <= b:
        a = 0
    elif a > 0:
        a -= b
    elif a < 0:
        a += b
    return a
def update_beta(y, x, lam, alpha, b, w):
    wx = x*w
    wx2 = (x**2)*w
    xb = np.matmul(x,b)
    for i in range(len(b)):
        xb -= np.multiply(x[:,i].reshape(-1,1), b[i])
        temp = soft_thresh(np.sum(wx[:,i]*(y-xb).reshape(-1)), lam*alpha)
        b[i] = temp/(np.sum(wx2[:,i]) +lam*(1-alpha))
        xb = xb + np.outer(x[:,i], b[i])
    return b
```

R output, results from casl package:

```
library(casl)
#Generate random x, y and beta
n <- 1000
p <- 5000
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_update_beta(x, y, lambda=0.5, alpha=1, b=matrix(0, nrow=ncol(x), ncol=1), W=rep(1, l
bhat[bhat !=0 ]
#> [1] 2.4702167 1.4466614 0.5282512
```

Python output:

```
x = r.x
y = r.y
y = y.reshape(-1,1)
b = np.zeros((x.shape[1],1))
w = np.empty([len(y),1])
w.fill(1/len(y))
update_beta(y, x, 0.5, 1, b, w)
#> array([[2.47021672],
#>        [1.44666145],
#>        [0.52825121],
#>        ...,
#>        [0.        ],
#>        [0.        ],
#>        [0.        ]])
```

Above two outputs from R and Python are really close to each other.

4. Final project proposal: I want to use deep learning techniques to build a model classify property damages. The model aims to identify properties that are damaged when there is some natural disaster (like hurricane, earthquake or wildfire) happened. I'm reading some materials related to neural networks and tensorflow, and I have started gathering my training data with different levels of property damages. Hopefully, I can get this model into work.