bis557homework4

Xining Yuan

11/23/2020

Question 1

```r
#Data
n <- 200
p <- 4
N <- 500
M <- 20
set.seed(1)
beta <- c(1, -1, 0.5, 9)
mu <- rep(0, p)
Sigma <- matrix(0.9, nrow = p, ncol = p)
diag(Sigma) <- 1
X <- MASS::mvrnorm(n, mu, Sigma)
y <- X %*% beta + rnorm(n, sd = 5)
```

```python
# python environment
import numpy as np

lam = 0.1
y = r.y
X = r.X

u,s,vh = np.linalg.svd(X) # X: n x m
n = u.shape[0]
m = vh.shape[0]
st = np.zeros(max(n,m))
st[:min(n,m)] = s / (s**2 + lam)
uty = np.transpose(u) @ y
ST = np.diag(st)
b = np.transpose(vh) @ ST[:m, :n] @ uty
b
```

```
## array([[ 1.82527254],
##        [-1.53787953],
##        [ 0.70804626],
##        [ 8.36860182]])
```

```r
# R
My_ridge_regression <- function(X, y, lambda) {
  n <- dim(X)[2]
  solve( crossprod(X) + lambda * diag(n), crossprod(X, y))
}
```

```
br <- My_ridge_regression(X, y, 0.1)
br
```

```
##             [,1]
## [1,]  1.8252725
## [2,] -1.5378795
## [3,]  0.7080463
## [4,]  8.3686018
```

Intrepretation

In this question, we want to compare the outputs – one is from SVD of Python and one is from ridge regression function of R. Test data is co-linearity. The groud truth of coefficients is $\begin{bmatrix} 1 \\ -1 \\ 0.5 \\ 9 \end{bmatrix}$. The output from Python is the matrix that $\begin{bmatrix} 1.82527254 \\ -1.53787953 \\ 0.70804626 \\ 8.36860182 \end{bmatrix}$ and the output from R is the matrix $\begin{bmatrix} 1.8252725 \\ -1.5378795 \\ 0.7080463 \\ 8.3686018 \end{bmatrix}$. Comparing those two results, we can get that this method works.

Question 2

```
library(reticulate)
```

```
## Warning: package 'reticulate' was built under R version 4.0.3
```

```
# R data
set.seed(1)
n <- 1e4; p <- 4
X2 <- matrix(rnorm(n*p), ncol = p)
beta <- c(1,2,3,4)
epsilon <- rnorm(n)
y2 <- X2 %*% beta + epsilon
```

```
import numpy as np

def recursive_ls(x, y, w, G):
    tmp = G @ (x.T @ x) @ G
    G = G - tmp / (1 + x @ G @ x.T)
    w = w - G @ x.T @ (x @ w - y)
    return w, G
```

```
# load all data, run ls model
bhat_all <- coef(lm(y2 ~ X2 - 1))
bhat_all
```

```
##       X21       X22       X23       X24
## 0.9870134 1.9876739 3.0045489 4.0102080
```

```
# load data row-by-row, run rls model
source_python("C:/Users/Xining/Dropbox/My PC (DESKTOP-Q77OU8J)/Desktop/Campus/First Year/First Semester/
n <- length(y2)
p <- 4
```

```
G <- diag(p)
bhat_one <- matrix(0, p, 1)
for (i in 1:n)
{
  Xi <- X2[i,]
  Xi <- matrix(Xi, ncol=p)
  yi <- y2[i]
  result <- recursive_ls(Xi, yi, bhat_one, G)
  bhat_one <- result[[1]]
  G <- result[[2]]
}
bhat_one
```

```
##           [,1]
## [1,] 0.9869195
## [2,] 1.9874662
## [3,] 3.0042462
## [4,] 4.0097982
```

Interpretation

In Python, we have an out-of-core implementation of linear model. We use recursive least square method. `bhat_all` is calculated with the whole data using build-in `lm` method. `bhat_one` is calculated by reading the data row-by-row, and updating the model. The comparison between those two results are similar.

Question 3

```
library(reticulate)
```

```
# test data
set.seed(42)
n <- 100
p <- 500
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)
```

```
# R
# Soft threshold function.
casl_util_soft_thresh <-
  function(a, b)
  {
    a[abs(a) <= b] <- 0
    a[a > 0] <- a[a > 0] - b
    a[a < 0] <- a[a < 0] + b
    a
  }
```

```
# Update beta vector using coordinate descent.W
casl_lenet_update_beta <-
  function(X, y, lambda, alpha, b, W)
  {
    WX <- W * X
    WX2 <- W * X^2
```

```
    Xb <- X %*% b
    for (i in seq_along(b))
    {
      Xb <- Xb - X[, i] * b[i]
      b[i] <- casl_util_soft_thresh(sum(WX[,i, drop=FALSE] *
                                          (y - Xb)),
                                    lambda*alpha)
      b[i] <- b[i] / (sum(WX2[, i]) + lambda * (1 - alpha))
      Xb <- Xb + X[, i] * b[i]
    }
    b
  }
```

```
# Compute linear elastic net using coordinate descent.
# Args:
# X: A numeric data matrix.
# y: Response vector.
# lambda: The penalty term.
# alpha: Value from 0 and 1; balance between l1/l2 penalty.
# maxit: Integer maximum number of iterations.
# tol: Numeric tolerance parameter.
# Returns:
# Regression vector beta of length ncol(X).
casl_lenet <- function(X, y, lambda, alpha = 1,
    b = matrix(0, nrow=ncol(X), ncol=1),
    tol = 1e-5, maxit = 50, W = rep(1, length(y))/length(y))
{
    for (j in seq_along(lambda))
    {
        if (j > 1)
        {
            b[,j] <- b[, j-1, drop = FALSE]
        }
        # Update the slope coefficients until they converge.
        for (i in seq(1, maxit))
        {
            b_old <- b[, j]
            b[, j] <- casl_lenet_update_beta(X, y, lambda[j], alpha,
            b[, j], W)
            if (all(abs(b[, j] - b_old) < tol)) {
                break
            }
        }
        if (i == maxit)
        {
            warning("Function lenet did not converge.")
        }
    }
    b
}

bhat_r <- casl_lenet(X, y, lambda = 0.1)
bhat_r[bhat_r != 0]
```

```
## [1] 2.8991182 1.8926516 0.8867888
```

```
import numpy as np

def soft_threshold(x, t):
    y = np.zeros(x.shape)
    y[x > t] = x[x > t] - t
    y[x < -t] = x[x < -t] + t
    return y

def lasso_ista(y, X, lam, t, nit):
    beta = np.zeros((X.shape[1],1))
    cost = np.zeros(nit)
    Xty = np.transpose(X) @ y
    XtX = np.transpose(X) @ X
    for i in range(nit):
        tmp = beta + t * (Xty - XtX @ beta)
        beta = soft_threshold(tmp, t * lam)
        cost[i] = np.linalg.norm(y - X @ beta, 2) + lam * np.linalg.norm(beta, 1)
    return beta, cost

def lasso_fista(y, X, lam, nit):
    beta = np.zeros((X.shape[1],1))
    cost = np.zeros(nit)
    t = 1
    z = beta.copy()
    L = np.linalg.norm(X, ord=2) ** 2
    for i in range(nit):
        beta_old = beta.copy()
        z = z + X.T.dot(y - X.dot(z)) / L
        beta = soft_threshold(z, lam / L)
        t0 = t
        t = (1. + np.sqrt(1. + 4.*t ** 2)) / 2.
        z = beta + ((t0 - 1.)/t) * (beta - beta_old)
        cost[i] = np.linalg.norm(y - X @ beta, 2) + lam * np.linalg.norm(beta, 1)
    return beta, cost
out = lasso_fista(y, X, 0.1, 10000)
bhat_py = out[[1]]
bhat1 = bhat_py[abs(bhat_py) > 0.02]
bhat1
```

Interpretation

In Python, we use FISTA method to solve LASSO problem. In R, we use the `casl_lenet` method to solve LASSO problem. By comparison, both of them provide sparse result, and they are very similar.

Interpretation

In Python, we use FISTA method to solve LASSO problem. In R, we use the `casl_lenet` method to solve LASSO problem. By comparison, both of them provide sparse result, and they are very similar.

Question 4

Goal

My final project will do the photo transformation from real photo to the cartoon.

Improvement

I do the research and find that there exists some papers using transform directly to do the change. However, I will add the step – face recolonization to collection face characteristics. According to the characteristics we get, my project will choose better style (Western or Asian) of cartoon for this person.

Method

I will use CNN to do the characteristics collection. Then, according to the characteristics, I will do the classification to judge which style (western style or Asian Style) is much fitter for the person. Then I will do the target transformation.