Computation in Data Science (Third Part) Homework 3

December 26, 2020

- Arthor: HO Ching Ru (R09946006, NTU Data Science Degree Program)
- Course: Computation in Data Science (MATH 5080), NTU 2020 Fall Semester
- Instructor: Tso-Jung Yen (Institute of Statistical Science, Academia Sinica)

1 Variable selection via lasso estimation

The lasso estimate for β is defined as:

$$\beta^{lasso} = \arg\min_{\beta} \left\{ \frac{1}{2} ||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda ||\beta||_1 \right\}$$

Now, consider use the m group function, the function becomes:

$$\beta^{lasso} = \arg\min_{\beta} \left\{ \frac{1}{2} \sum_{k=1}^{m} ||\mathbf{y}_i - \mathbf{X}_i\beta_i||_2^2 + \lambda ||\beta||_1 \right\}$$

Reformulate the lasso estimation problem in an ADMM form:

$$\frac{1}{2} \sum_{k=1}^{m} ||\mathbf{y}_i - \mathbf{X}_i\theta_i||_2^2 + \lambda ||\beta||_1$$
$$\text{subject to } \theta_i = \beta, \ \forall i \in \{1, m\}$$

The Lagrange function becomes:

$$\mathcal{L}(\theta, \beta, \alpha) = \frac{1}{2} ||\mathbf{y} - \mathbf{X_k}\theta||_2^2 + \lambda ||\beta||_1 + \alpha^T(\beta - \theta) + \frac{\rho}{2} ||\beta - \theta||_2^2$$

Update function of θ from iteration r to $r+1$:

$$\theta_k^{r+1} = \arg\min_{\theta} \left\{ \frac{1}{2} ||y_k - \mathbf{X}_k\theta_k||_2^2 + \frac{\rho}{2} \left\|\theta_k - \beta^r + \alpha_k^r\right\|_2^2 \right\} = \arg\min_{\theta} \{\blacksquare\}$$

Noticed that it has the closed form:

$$
\begin{aligned}
\frac{\partial \blacksquare}{\partial \theta} =& -\mathbf{X}_k^T ||y_k - \mathbf{X}_k \theta_k|| + \rho ||\theta_k - \beta^r + \alpha_k^r|| = 0 \\
\longrightarrow & \theta_k^{r+1} = \left(\mathbf{X}_k^T \mathbf{X}_k + \rho \mathbf{I}\right)^{-1} \left(\mathbf{X}_k^T y_k + \rho(\beta^r - \alpha_k^r)\right), \forall k = \{1, m\}
\end{aligned}
$$

.

Update function of β from iteration r to $r+1$:

$$
\begin{aligned}
\beta^{r+1} =& \arg\min_{\beta} \left\{ \lambda ||\beta||_1 + \frac{mp}{2} \left\| \beta - \frac{1}{m} \sum_{k=1}^{m} \theta_k^{r+1} - \frac{1}{m} \sum_{k=1}^{m} \alpha_k^r \right\|_2^2 \right\}, \forall k = \{1, m\} \\
=& S_{\lambda/(\rho m)} \left(\frac{1}{\rho m} \sum_{i=1}^{m} \rho(x_i^{k+1} + \alpha_i^k) \right)
\end{aligned}
$$

The soft thresholding operator, $S(\cdot)$ denotes:

$$
S_\kappa(a) = \begin{cases} a - \kappa, a > \kappa \\ 0, |a| \leq \kappa \\ a + \kappa, a < -\kappa \end{cases}
$$

Update function of α from iteration r to $r+1$:

$$
\alpha_k^{r+1} = \alpha_k^r + \theta_k^{r+1} - \beta^{r+1}, \forall k = \{1, m\}
$$

1.1 Data generation

Assume that:

$$
\beta^r = 0, \forall r \in \{1, 2, \cdots 500\} \diagdown \{100, 200, 300, 400, 500\}
$$

$$
\theta^r \sim N(0, 1), \forall r
$$

$$
\alpha^r = 0, \forall r
$$

and

$$
\rho = 1
$$

[1]:
```
import numpy as np
np.random.seed(5566)

# dimension
p = 500
n = 30000

# generate random x_i: from N(0,1)
X = np.random.normal(0, 1, size=(n, p))

# generate random y_i: from X
y = list()
for i in range(n):
    tmp = -2*X[i][99] - 2*X[i][199] + 2*X[i][299] + 2*X[i][399] - 2*X[i][499] +␣
↪(np.random.normal(0, 0.5))
    y.append([tmp])
y = np.array(y)

# generate beta: beta[100], ..., beta[500] given, others assume be zeros
beta_r = np.zeros(shape=(p, 1))
for i in range(p):
    if (i==99) or (i==199) or (i==499):
        beta_r[i] = [-2]
    elif (i==299) or (i==399):
        beta_r[i] = [2]

# generate theta and alpha
theta_r = np.random.normal(size=(p, 1))
alpha_r = np.zeros(shape=(p, 1))

# set parameters
rho = 1
num_iterations = 500
```

1.2 Check the dimension of each variable and parameters

[2]:
```
print("dimension of x:", np.shape(X))
print("dimension of y:", np.shape(y))
print("dimension of beta:", np.shape(beta_r))
print("dimension of theta:", np.shape(theta_r))
print("dimension of alpha:", np.shape(alpha_r))
```

```
dimension of x: (30000, 500)
dimension of y: (30000, 1)
dimension of beta: (500, 1)
dimension of theta: (500, 1)
dimension of alpha: (500, 1)
```

1.3 Run

The primal residual ($\mathbf{t}^{r+1}$) and dual residual ($\mathbf{s}^{r+1}$) in iteration $r+1$ is:

$$\mathbf{t}^{r+1} = \theta^{r+1} - \beta^{r+1}$$
$$\mathbf{s}^{r+1} = -\rho(\beta^{r+1} - \beta^{r})$$

[3]:
```
import pywt

def objective_function(X ,y, theta_r, lamb):
    return 0.5 * np.linalg.norm(np.dot(X, theta_r) - y, ord=2)**2 + lamb*np.
 ↪linalg.norm(theta_r, ord=1)

def lasso_admm(X, y, lamb, rho, n, p, theta_r, beta_r, alpha_r):
    primal_res_ls = []
    dual_res_ls = []

    # Initializations
#     print("-------------INITIALIZATION------------")
    print("lambda =", lamb)
    val = objective_function(X ,y, theta_r, lamb)
#     print("----------------------------------------")

    for iter in range(num_iterations):
        if (iter%100==0):
            print("Finish", iter, "iterations.")
        # STEP 1: Calculate theta_r
        # This has a closed form solution
        term1 = np.linalg.inv(np.dot(X.T, X) + rho*np.identity(p))
        term2 = np.dot(X.T, y) + rho*(beta_r -  alpha_r)
        theta_r = np.dot(term1, term2)

        # STEP 2: Calculate beta_r
        # Taking the prox, we get the lasso problem again, so, using␣
 ↪coordinate_descent
        beta_r_old = beta_r
        term3 = theta_r + alpha_r
        beta_r = pywt.threshold(term3, lamb/rho, "soft")

        # STEP 2.5: Calculate Euclidean norm of Residuals
        primal_res = np.linalg.norm(np.dot(np.identity(p), theta_r) - np.dot(np.
 ↪identity(p), beta_r), ord=2)
        dual_res = np.linalg.norm(rho*(-1)*(beta_r - beta_r_old), ord=2)
        primal_res_ls.append(primal_res)
        dual_res_ls.append(dual_res)
#         print(primal_res,dual_res)
```

```
        # STEP 3: Update alpha_r (alpha)
        alpha_r = alpha_r + theta_r - beta_r
        val = objective_function(X ,y, theta_r, lamb)

    val = objective_function(X ,y, theta_r, lamb)
    primal_res = np.linalg.norm(np.dot(np.identity(p), theta_r) - np.dot(np.
↪identity(p), beta_r), ord=2)
    dual_res = np.linalg.norm(rho*(-1)*(beta_r - beta_r_old), ord=2)
    primal_res_ls.append(primal_res)
    dual_res_ls.append(dual_res)
    print("------------FINISH-------------")
    # Return the Primal Residual list, Dual Residual list and the final val as␣
↪beta^LASSO
    return primal_res_ls, dual_res_ls, beta_r
```

1.4 Problem 1

Fix tuning parameter λ at 4 different values you like and run the iterative scheme under the 4 different values of λ separately. Produce 4 plots according to the 4 different values of λ with the following format: The x-axis is the number of iterations r and the y-axis is the Euclidean norm of the primal residual and dual residual of the iterative scheme.

In here, let $\lambda = [0.01, 0.1, 0.5, 1]$.

[4]:
```
lamb_ls = [0.01, 0.1, 0.5, 1]
primal_res_ls = list()
dual_res_ls = list()
beta_lasso_ls = list()
for index in range(len(lamb_ls)):
    primal_res, dual_res, beta_r = lasso_admm(X, y, lamb_ls[index], rho, n, p,␣
↪theta_r, beta_r, alpha_r)
    primal_res_ls.append(primal_res)
    dual_res_ls.append(dual_res)
    beta_lasso_ls.append(beta_r)
```

```
lambda = 0.01
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 0.1
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
```

```
------------FINISH-------------
lambda = 0.5
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 1
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
```

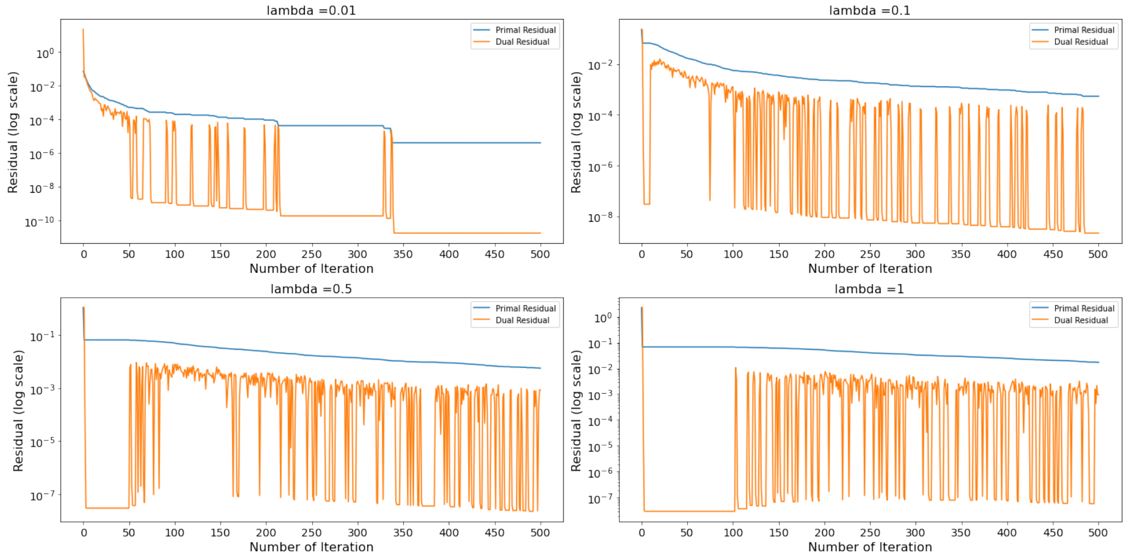
[5]:

```python
import matplotlib.pyplot as plt

plt.figure(figsize = (20, 10))

for index in range(len(lamb_ls)):
    plt.subplot(2, 2, index+1)
    plt.plot(primal_res_ls[index], label = "Primal Residual")
    plt.plot(dual_res_ls[index], label = "Dual Residual")
    plt.legend()
    plt.title("lambda ="+str(lamb_ls[index]), fontsize = 16)
    plt.xlabel("Number of Iteration", fontsize = 16)
    plt.ylabel("Residual (log scale)", fontsize = 16)
    plt.xticks(np.arange(0, num_iterations+1, num_iterations/10), fontsize = 14)
    plt.yticks(fontsize = 14)
    plt.yscale("log")

plt.tight_layout()
plt.show()
```



2 Problem 2

Select 20 different values of λ from the interval $[0.001, 5]$ and run the iterative scheme under the 20 different values of λ separately. Collect the values of $\hat{\beta}^{\text{lasso}}(\lambda)$. Produce a trace plot of $\hat{\beta}^{\text{lasso}}(\lambda)$ with the following format: The x-axis is the value of λ and the y-axis is $\hat{\beta}^{\text{lasso}}(\lambda)$. Since we have p = 500, there should be 500 such trace lines for $\hat{\beta}^{\text{lasso}}(\lambda)$. Use red color to draw the trace lines for the 100th, 200th, 300th, 400th and 500th elements of $\hat{\beta}^{\text{lasso}}(\lambda)$, and gray color to draw the trace line for the rest of elements in $\hat{\beta}^{\text{lasso}}(\lambda)$.

[6]:
```
lamb_ls_len = 20
lamb_ls = np.random.uniform(low = 0.001, high = 5, size=(20,))
len(lamb_ls)
```

[6]:
```
20
```

[7]:
```
primal_res_ls = list()
dual_res_ls = list()
beta_lasso_ls = list()
for index in range(len(lamb_ls)):
    primal_res, dual_res, beta_r = lasso_admm(X, y, lamb_ls[index], rho, n, p,␣
 ↪theta_r, beta_r, alpha_r)
    primal_res_ls.append(primal_res)
    dual_res_ls.append(dual_res)
    beta_lasso_ls.append(beta_r)
```

```
lambda = 1.590380070898616
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
```

```
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 1.6695763722971908
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 4.84768553586487
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 1.2596073619216086
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 4.988591187458686
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 2.2868549652224037
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 1.5426213311814418
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 1.3357004823076384
Finish 0 iterations.
Finish 100 iterations.
```

```
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 2.665296278983647
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 2.522650144048508
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 3.197874736766807
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 2.1549749671180813
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 3.965521127396994
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 3.5968752477631547
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 0.6151221609502391
Finish 0 iterations.
```

```
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 0.6922662210820328
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 1.4408009056010909
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 4.686215645762935
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 2.529935491576068
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
lambda = 1.9897793787475546
Finish 0 iterations.
Finish 100 iterations.
Finish 200 iterations.
Finish 300 iterations.
Finish 400 iterations.
------------FINISH-------------
```

[8]:
```
lasso_list = np.reshape(beta_lasso_ls, (len(lamb_ls), p))
```

[9]:
```
plt.figure(figsize = (10, 5))

for i in range(len(lasso_list)):
    plt.scatter([lamb_ls[i]]*p, lasso_list[i], color = 'grey')
```

```
    plt.scatter([lamb_ls[i]], lasso_list[i][99], color = 'red')
    plt.scatter([lamb_ls[i]], lasso_list[i][199], color = 'red')
    plt.scatter([lamb_ls[i]], lasso_list[i][299], color = 'red')
    plt.scatter([lamb_ls[i]], lasso_list[i][399], color = 'red')
    plt.scatter([lamb_ls[i]], lasso_list[i][499], color = 'red')

plt.title("Different lambda with beta^lasso",fontsize = 16)
plt.xlabel("lambda", fontsize = 16)
plt.ylabel("beta^lasso", fontsize = 16)
plt.xticks(np.arange(0, 5.01, 0.5), fontsize = 14)
plt.yticks(fontsize = 14)
plt.tight_layout()
plt.show()
```