HW05_stomar2

March 3, 2022

STAT-542 HW5

Sharvi Tomar (stomar2)

0.1 About HW5

We utilize the coordinate descent algorithm introduced in the class to implement the entire Lasso solution. For coordinate descent, you may also want to review HW4. This HW involves two steps: in the first step, we solve the solution for a fixed λ value, while in the second step, we consider a sequence of λ values and solve it using the path-wise coordinate descent.

0.2 Question 1 [50 Points] Lasso solution for fixed λ

For this question, you cannot use functions from any additional library, except the `MASS` package, which is used to generate multivariate normal data. Following HW4, we use the this version of the objective function:

$$\arg\min_{\beta} \frac{1}{n}\|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda\|\beta\|_1$$

The following data is used to fit this model. You can consider using similar functions in Python if needed. We use

[517]:
```
import random
import math
import numpy as np
from sklearn.preprocessing import MinMaxScaler

random.seed(24)
n = 100
p = 200
lambda_val = 0.3

# covariance matrix
V = np.array([[0.3] * p] * p)
np.fill_diagonal(V, [1] * p)

X_org = np.random.multivariate_normal(mean =[0]*p, cov=V, size=(n))
true_b = list(range(1,4)) + list(range(-3,0)) + [0] * (p-6)
```

```
y_org = np.matmul(X_org, true_b) + np.random.normal(0, 1, n)

# transformation of X_org
sd_X = np.std(X_org, axis=0)
mean_X = np.mean(X_org, axis=0)

# transformation of y_org
sd_y = np.std(y_org)
mean_y = np.mean(y_org)

X = ((X_org - mean_X)/sd_X) * math.sqrt(n/(n-1))
y = ((y_org - mean_y)/sd_y) * math.sqrt(n/(n-1))
```

a) [10 pts] State the solution x of the following problem

$$\arg\min_x \ (x-b)^2 + \lambda|x|, \quad \lambda > 0$$

Then, implement a function in the form of `soft_th <- function(b, lambda)` to return the result of the above problem. Note in the coordinate descent discussed in the slides, where b is an OLS estimator, and λ is the penalty parameter. Report the function output for the following testing cases with $\lambda = 0.3$: 1) $b = 1$; 2) $b = -1$; 3) $b = -0.1$.

[518]:
```
beta = [1, -1 ,-0.1]
lambda_val = 0.3

def soft_th(b, lambda_val):
    if b > lambda_val/2: return b - lambda_val/2
    elif b < -lambda_val/2: return b + lambda_val/2
    else: return 0

for i in beta:
    print("Soft threshold function output for lambda = 0.3 & OLS value of",␣
 ↪i,"is:", soft_th(i, lambda_val))
```

```
Soft threshold function output for lambda = 0.3 & OLS value of 1 is: 0.85
Soft threshold function output for lambda = 0.3 & OLS value of -1 is: -0.85
Soft threshold function output for lambda = 0.3 & OLS value of -0.1 is: 0
```

b) [40 pts] We will use the pre-scale and centered data `X` and `y` for this question, hence no intercept is needed. Write a Lasso algorithm function `myLasso(X, y, lambda, beta_init, tol, maxitr)`, which return two outputs (as a list with two components):

- a vector of β values **without** the intercept
- number of iterations

You need to consider the following while completing this question:

- Do not use functions from any additional library
- Start with a vector `beta_init`: $\boldsymbol{\beta} = \mathbf{0}_{p\times 1}$

- Use the soft-threshold function in the iteration when performing the coordinate-wise update.
- Use the efficient **r** updating appraoch (we discussed this in lecture and HW4) in the iteration
- Run your coordinate descent algorithm for a maximum of `maxitr` $= 100$ iterations. Each iteration should loop through all variables.
- You should implement the early stopping rule with `tol`. This means terminating the algorithm when the $\boldsymbol{\beta}$ value of the current iteration is sufficiently similar to the previous one, i.e., $\|\boldsymbol{\beta}^{(k)} - \boldsymbol{\beta}^{(k-1)}\|^2 \leq \text{tol}$.

Aftering completing your code, run it on the data we generated previously. Provide the following results:

- Print out the first 8 coefficients and the number of iterations.
- Check and compare your answer to the `glmnet` package using the following code. You should report their **first 8 coefficients** and the L_1 norm of the difference $\| \hat{\boldsymbol{\beta}}^{\text{glment}}_{[1:8]} - \hat{\boldsymbol{\beta}}^{\text{yours}}_{[1:8]} \|_1$.

[519]:
```
import copy
def myLasso(X, y, lambda_val, beta_init, tol, maxitr):
    beta_new = beta_init
    XcolNorm = np.sum(X**2, axis=0)
    iters =0
    n = X.shape[0]

    for k in range(maxitr):
        iters +=1
        beta_init = copy.copy(beta_new)
        r = y - np.dot(X, beta_init)

        for j in range(X.shape[1]):
            r += (X[:, j] * beta_new[j])

            ## soft threshold function
            beta_new[j] = soft_th(np.dot(X[:, j].T, r)/ XcolNorm[j],␣
↪lambda_val*n/XcolNorm[j])
            r -= (X[:, j] * beta_new[j])

        ## early stopping criteria
        if sum([(a_i - b_i)**2 for a_i, b_i in zip(beta_new, beta_init)]) <=␣
↪tol: break

    return[beta_new, iters]
```

[520]:
```
ret = myLasso(X, y, lambda_val, beta_init = [0] * p, tol = 1e-10, maxitr = 100)
print("First 8 coefficients are")
print(ret[0][:8])
print("Number of iterations:",ret[1])
```

```
First 8 coefficients are
[0, 0.20170394329434968, 0.5038696034732945, -0.42418064867383154,
-0.12665576228473258, -0.06882988551619673, 0, 0]
Number of iterations: 12
```

[521]:
```python
from sklearn.linear_model import Lasso
import math
model = Lasso(alpha= lambda_val/2,fit_intercept = False)
model.fit(X, y)
sklearn_lasso_coef = model.coef_
print("First 8 coefficients of sklearn_lasso are:", sklearn_lasso_coef[:8])
print("First 8 coefficients of myLasso are:", ret[0][:8])
print("L1 norm of their difference is:",sum([abs(a_i - b_i) for a_i, b_i in␣
↪zip(sklearn_lasso_coef[:8], ret[0][:8])]) )
```

```
First 8 coefficients of sklearn_lasso are: [ 0.          0.20167461  0.50384755
-0.42417685 -0.12663206 -0.06882523
  0.          0.        ]
First 8 coefficients of myLasso are: [0, 0.20170394329434968,
0.5038696034732945, -0.42418064867383154, -0.12665576228473258,
-0.06882988551619673, 0, 0]
L1 norm of their difference is: 8.354242161820458e-05
```

0.3 Question 2 [50 Points] Path-wise Coordinate Descent

Let's perform path-wise coordinate descent. The idea is simple: we will solve the solution on a sequence of λ values, starting from the largest one in the sequence. The first initial $\boldsymbol{\beta}$ are still all zero. After obtaining the optimal $\boldsymbol{\beta}$ for a given λ, we simply use this solution as the initial value for the next, smaller λ. This is referred to as a **warm start** in optimization problems. We will consider the following sequence of λ borrowed from `glmnet`. Note that this is a decreasing sequence from large to small values.

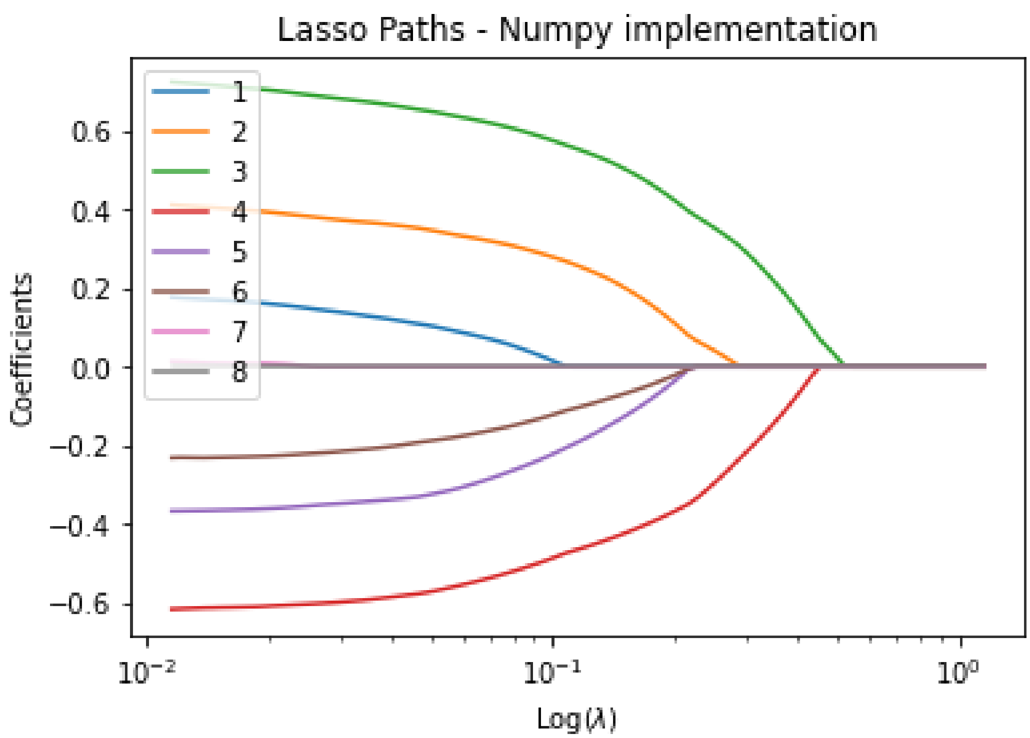
[522]:
```python
import pandas as pd
lamda_all_df = pd.read_csv("data.csv")
lamda_all = lamda_all_df.to_numpy()
```

[523]:
```python
#Run lasso regression for each lambda
theta_list = []
for i in range(len(lamda_all)):
    model = Lasso(alpha = lamda_all[i][0], fit_intercept = False)
    model.fit(X, y)
    theta_list.append(model.coef_)
```

[524]:
```python
#Stack into numpy array
theta_lasso = np.stack(theta_list).T

for i in range(8):
    plt.plot(lamda_all, theta_lasso[i], label = i+1)
```

```
plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('Coefficients')
plt.title('Lasso Paths - Numpy implementation')
plt.legend()
#plt.axis('tight')
```

[524]: <matplotlib.legend.Legend at 0x12f420d10>



a) [20 pts] Write a function `myLasso_pw <- function(X, y, lambda_all, tol, maxitr)`, which output a $p \times N_\lambda$ matrix. N_λ is the number of unique λ values. Also follow the above instruction at the beginning of this question to include the **warm start** for path-wise solution. Your `myLasso_pw` should make use of your `myLasso` in Question 1.

[525]:
```
def myLasso_pw(X, y, lamda_all, tol, maxitr):
    beta_init = [0] * p
    res = []
    res.append(myLasso(X, y, lamda_all[0][0], beta_init, 1e-7, 100)[0])

    for i in range(1,len(lamda_all)):
        add_list = None
        add_list = myLasso(X, y, lamda_all[i][0], res[-1], 1e-7, 100)[0]
```

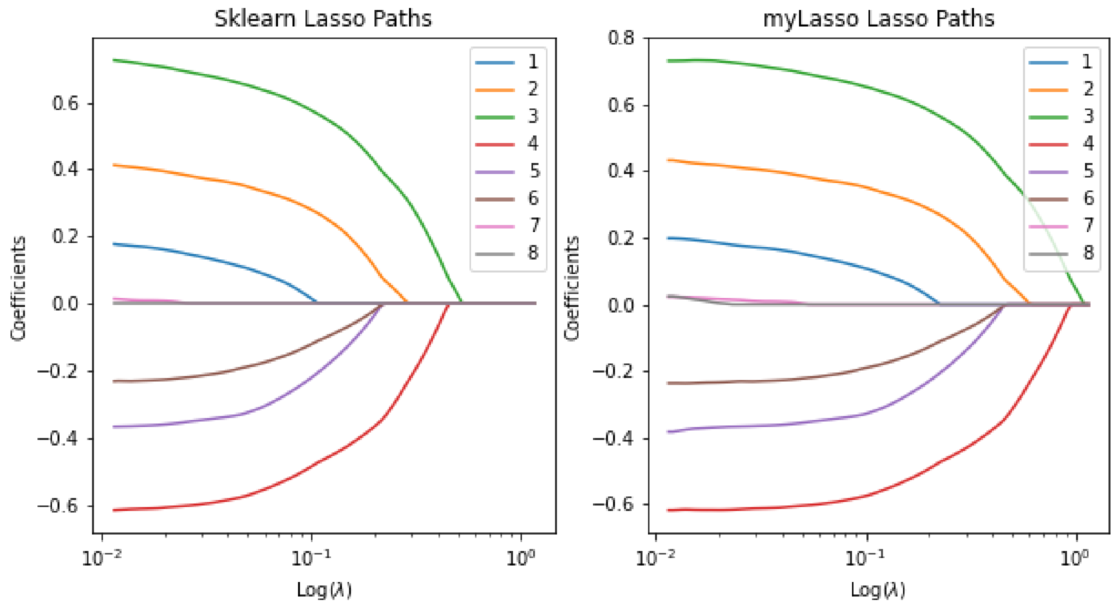
```
        res.append(add_list[:])
    return res
```

b) [5 pts] Provide the same plot as the above `glmnet` solution plot of the first 8 parameter in your solution path. Make the two plots side-by-side (e.g. `par(mfrow = c(1, 2)` in `R`) with `glmnet` on the left and your solution path on the right.

[526]:
```
b1_mylasso_pw = myLasso_pw(X, y, lamda_all, 1e-7, 100)
```

[527]:
```
#Stack into numpy array
theta_sklearn = np.stack(theta_list).T
theta_myLasso = np.stack(b1_mylasso_pw).T

plt.subplots(2,2,figsize=(10,5))

plt.subplot(1, 2, 1)
for i in range(8):
    plt.plot(lamda_all, theta_sklearn[i], label = i+1)

plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('Coefficients')
plt.title('Sklearn Lasso Paths')
plt.legend()

plt.subplot(1, 2, 2)
for i in range(8):
    plt.plot(lamda_all, theta_myLasso[i], label = i+1)
plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('Coefficients')
plt.title('myLasso Lasso Paths')
plt.legend()
```

[527]:
```
<matplotlib.legend.Legend at 0x1331d0990>
```



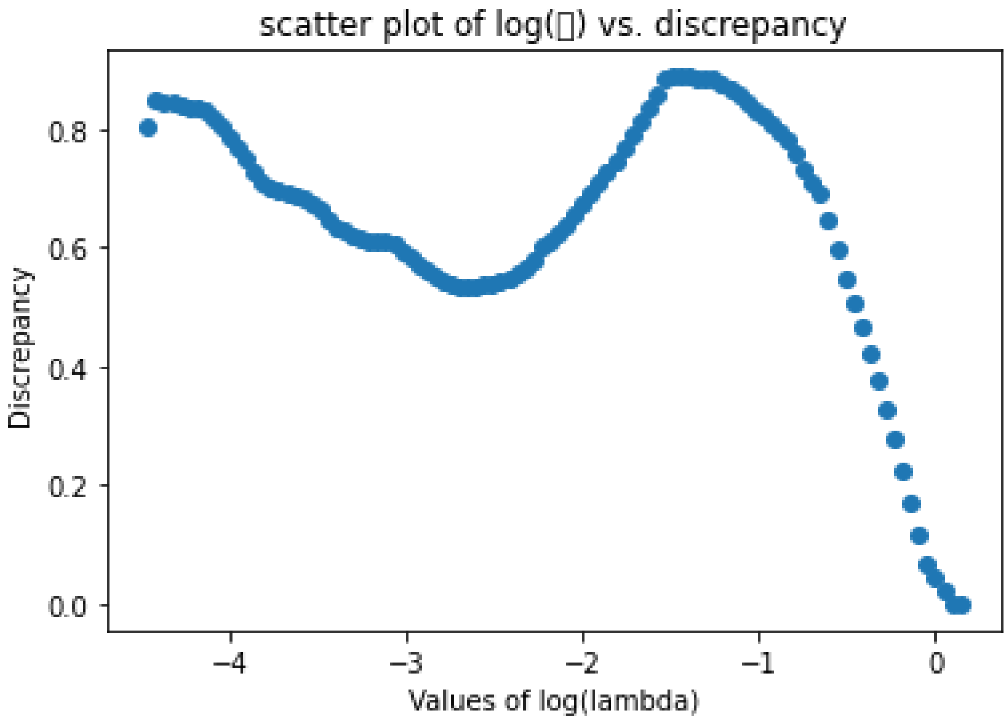
c) [5 pts] Based on your plot, if we decrease λ from its maximum value, which two variables enter (start to have nonzero values) the model first? You may denote your covariates as $X_1, ..., X_8$.

Looking from the right side of the graph we can see that the covariates 3 and X4 are the first ones that start to have nonzero values.

d) [5 pts] In Question 1, we calculated the L1 norm discrepancy between our solution and `glmnet` on the first 8 parameters. In this question, we will calculate the discrepancy on **all** coefficients, and over all λ parameters. After calculating the discrepancies, show a scatter plot of **log(λ) vs. discrepancy**. Comment on what you observe.

[528]:
```
discrepancy = []
for j in range(len(lamda_all)):
    discrepancy.append(sum([abs(a_i - b_i) for a_i, b_i in zip(theta_list[j],␣
 ↪b1_mylasso_pw[j])]))

plt.scatter(np.log(lamda_all),discrepancy)
plt.title('scatter plot of log( ) vs. discrepancy')
plt.ylabel('Discrepancy')
plt.xlabel('Values of log(lambda)')
plt.show()
```

```
/Users/sharvitomar/opt/anaconda3/lib/python3.7/site-
packages/IPython/core/pylabtools.py:128: UserWarning: Glyph 120582
(\N{MATHEMATICAL ITALIC SMALL LAMDA}) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```



With increasing value of lambda or log(lambda)) we see a decrease in discrepancy initially then it rises and again comes down with increasing lambda value(or log(lambda)).

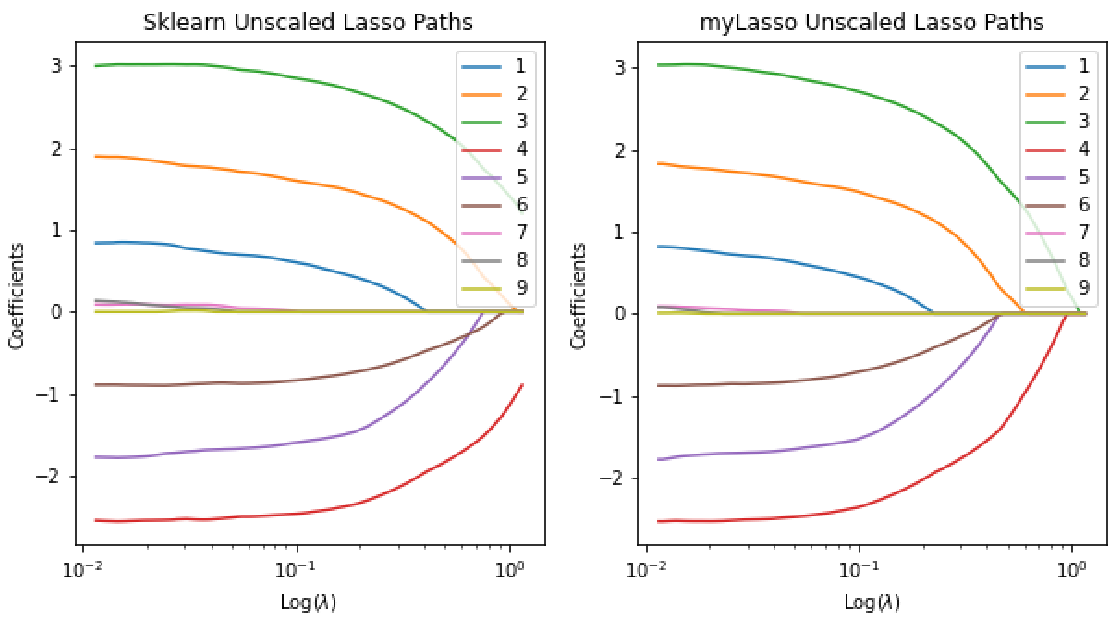
e) [15 pts] Based on the solution you obtained in the previous question, recover the unscaled coefficients using the formula in HW4. Then compare the first 9 coefficients (including the intercept term) with the following using a similar plot in b). Report the maximum value of discrepancy (see d) across all λ values.

[529]:
```
#Run lasso regression for each lambda
sklearn_unscaled = []
for i in range(len(lamda_all)):
    model = Lasso(alpha = lamda_all[i][0], fit_intercept = True)
    model.fit(X_org, y_org)
    sklearn_unscaled.append(model.coef_)
```

[530]:
```
# Getting unscaled coefficients values for mylasso() coefficients
mylasso_unscaled = b1_mylasso_pw * (sd_y/sd_X)
mylasso_intercept = mean_y - (mean_X *b1_mylasso_pw*(sd_y/sd_X))
for i in range(100):
    np.insert(mylasso_unscaled[i], 0, mylasso_intercept[i])
```

[531]:
```
#Stack into numpy array
unscaled_theta_sklearn = np.stack(sklearn_unscaled).T
unscaled_theta_myLasso = np.stack(mylasso_unscaled).T

plt.subplots(2,2,figsize=(10,5))

plt.subplot(1, 2, 1)
for i in range(9):
    plt.plot(lamda_all, unscaled_theta_sklearn[i], label = i+1)

plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('Coefficients')
plt.title('Sklearn Unscaled Lasso Paths')
plt.legend()

plt.subplot(1, 2, 2)
for i in range(9):
    plt.plot(lamda_all, unscaled_theta_myLasso[i], label = i+1)
plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('Coefficients')
plt.title('myLasso Unscaled Lasso Paths')
plt.legend()
```

[531]: <matplotlib.legend.Legend at 0x133905f10>



[532]:

```
# Max value of discrepancy for which lambda
unscaled_discrepancy = []
for j in range(len(lamda_all)):
    unscaled_discrepancy.append(sum([abs(a_i - b_i) for a_i, b_i in␣
 ↪zip(sklearn_unscaled[j], mylasso_unscaled[j])]))

min_index = unscaled_discrepancy.index(min(unscaled_discrepancy))

print("The maximum value of discrepancy across all    values␣
 ↪is",max(unscaled_discrepancy))
```

```
The maximum value of discrepancy across all    values is 3.334669625880128
```