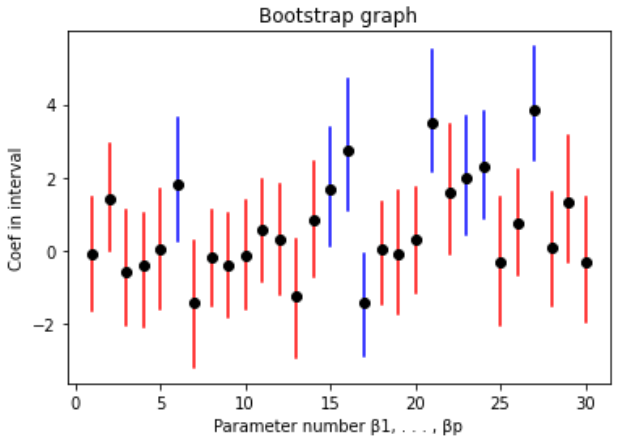
Z5305320
Dong AO
(a)

(i): It makes the penalty more useful since it sets the correct size of the loss function compare to the penalty term. Also, reasonable amount of the intervals is shown on the graph with color blue.

(ii): Yes. It will. If I take c too small, then the penalty function will take a huge effect on the loss function, which means the outcome will not be too large. This dataset is in high dimension, that is why I need l1 penalty and a larger c to do something like feature selection to get the sparse solution.




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Load dataset
data = pd.read_csv('Q1.csv', index_col=0)
data.shape
```

```
(250, 31)
```

```
data_x = data.iloc[:,0:30]
data_y = data['Y']
```

```
from sklearn.linear_model import LogisticRegression

coefs = []

c = 1000
B = 500
np.random.seed(12)
for l in range(B):
    numbers = np.random.choice(np.arange(data_x.shape[0]), size=data_x.shape[0])
    x_boot = data_x.loc[numbers]
    y_boot = data_y.loc[numbers]
    classifier = LogisticRegression(solver='liblinear', C=c, penalty='l1')
    classifier.fit(x_boot, y_boot)

    coefs.append(classifier.coef_[0])
coefs = np.array(coefs)
```

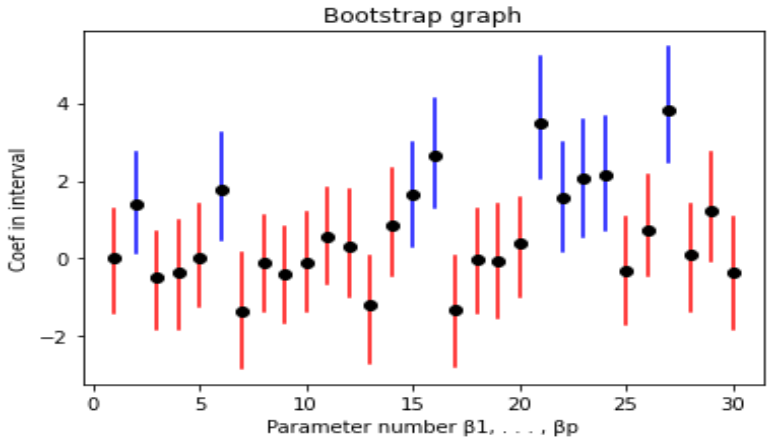
```
for i in range(30):
    some_coef = np.sort(coefs[:,i])
    some_coef = some_coef[25:475]

    if (0 >= min(some_coef) and 0 <= max(some_coef)):
        color = 'r'
    else:
        color = 'b'

    plt.plot([i + 1,i + 1], [min(some_coef), max(some_coef)], color=color)
    plt.plot(i + 1, some_coef.mean(), 'o', color='black')
plt.xlabel("Parameter number β1, . . . , βp")
plt.ylabel("Coef in interval")
plt.title('Bootstrap graph')
plt.show()
```

(b):

The outcome looks similar to that in (a). The outcome here has one less interval in blue which means it works little better.




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# load dataset
data = pd.read_csv('Q1.csv', index_col=0)
data.shape
```

```
(250, 31)
```

```
data_x = data.iloc[:,0:30]
data_y = data['Y']
```

```
from sklearn.linear_model import LogisticRegression
np.random.seed(20)
B = 500
C = 1000
coefs = []
classifier = LogisticRegression(solver='liblinear', C=C, penalty='l1')
classifier.fit(data_x, data_y)
for i in range(B):
    numbers = np.random.choice(np.arange(data_x.shape[0]), size=data_x.shape[0])
    x_boot = data_x.loc[numbers]
    p = classifier.predict_proba(x_boot)
    p = pd.DataFrame(p)
    y_predict = np.random.binomial(1, p[1])
    new_classifier = LogisticRegression(solver='liblinear', C=C, penalty='l1')
    new_classifier.fit(x_boot, y_predict)
#     print(i, "  ", new_classifier.coef_[0])
    coefs.append(new_classifier.coef_[0])
```

```
coefs = np.array(coefs)
print(len(coefs))
print(len(coefs[0]))
```

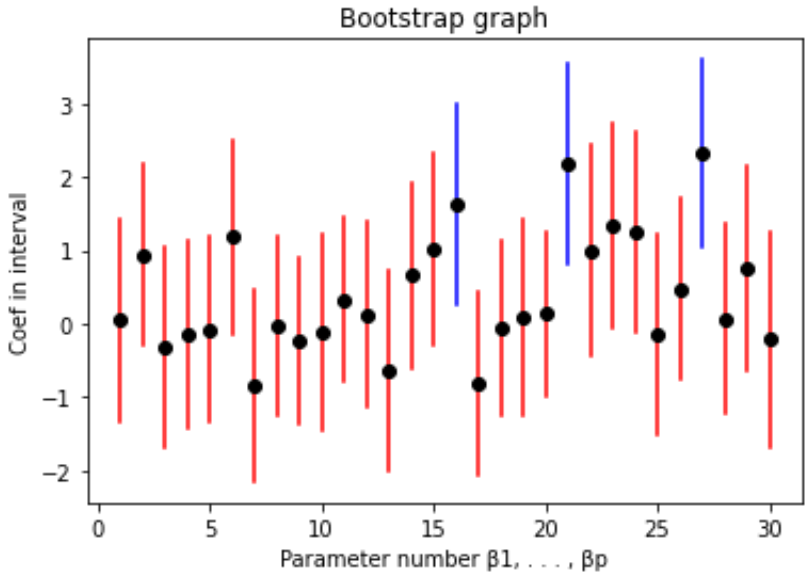
```
500
30
```

```
for i in range(30):
    some_coef = np.sort(coefs[:,i])
    some_coef = some_coef[25:475]

    if (0 >= min(some_coef) and 0 <= max(some_coef)):
        color = 'r'
    else:
        color = 'b'

    plt.plot([i + 1,i + 1], [min(some_coef), max(some_coef)], color=color)
    plt.plot(i + 1, some_coef.mean(), 'o', color='black')
plt.xlabel("Parameter number β1, . . . , βp")
plt.ylabel("Coef in interval")
plt.title('Bootstrap graph')
plt.show()
```

(c):




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# load dataset
data = pd.read_csv('Q1.csv', index_col=0)
data.shape
```

```
(250, 31)
```

```
data_x = data.iloc[:,0:30]
data_y = data['Y']
```

```
from sklearn.linear_model import LogisticRegression
np.random.seed(20)
B = 500
C = 1000
n = 250
coefs = []
D_is = []
first_term = []
classifier = LogisticRegression(solver='liblinear', C=C, penalty='l1')
classifier.fit(data_x, data_y)
classifier.coef_[0]

for i in range(30):
    first_term.append(n * classifier.coef_[0][i])

for i in range(n):
    x_boot = data_x.drop(i)
    y_boot = data_y.drop(i)
    classifier = LogisticRegression(solver='liblinear', C=C, penalty='l1')
    classifier.fit(x_boot, y_boot)
    coefs.append(classifier.coef_[0])
```

```
coefs = np.array(coefs)
print(len(coefs))
print(len(coefs[0]))
```

```
250
30
```

```
gamas = []
means = []
stds = []
S_square = []
for i in range(n):
    one_gama = []
    for l in range(30):
        one_gama.append(first_term[l] - (n-1)*coefs[i][l])
    gamas.append(one_gama)

for l in range(30):
    means.append(np.mean(gama_ma[l]))
    stds.append(np.std(gama_ma[l], ddof=0)**2)
```

```
for i in range(30):
    some_coef = np.sort(coefs[:,i])
    some_coef = some_coef[25:475]
    lower = means[i] - 1.645 * np.sqrt(stds[i] / n)
    higher = means[i] + 1.645 * np.sqrt(stds[i] / n)

    if (0 >= lower and 0 <= higher):
        color = 'r'
    else:
        color = 'b'

    plt.plot([i + 1,i + 1], [lower, higher], color=color)
    plt.plot(i + 1, means[i], 'o', color='black')
plt.xlabel("Parameter number β1, . . . , βp")
plt.ylabel("Coef in interval")
plt.title('Bootstrap graph')
plt.show()
```

(d):

Algorithm	# False Positives	# False Negatives
NP	2	2
P	5	5
JK	3	3

```
import numpy as np
a_result = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0]
b_result = [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0]
c_result = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]

def print_false(betas, result):
    False_Positives = 0
    False_Negatives = 0
    for i in range(len(betas)):
        if (betas[i] == 0 and result[i] != 0):
            False_Negatives += 1
        if (betas[i] != 0 and result[i] == 0):
            False_Positives += 1
    print(False_Positives," ", False_Positives)
```

```
# NP
np.random.seed(125)
p = 30
k = 8
betas = np.random.random(p) + 1
new_betas = betas
new_betas[np.random.choice(np.arange(p), p-k, replace=False)] = 0.0
print_false(new_betas, a_result)

k = 9
betas = np.random.random(p) + 1
new_betas = betas
new_betas[np.random.choice(np.arange(p), p-k, replace=False)] = 0.0
print_false(new_betas, b_result)

k = 3
betas = np.random.random(p) + 1
new_betas = betas
new_betas[np.random.choice(np.arange(p), p-k, replace=False)] = 0.0
print_false(new_betas, c_result)
```

```
2   2
5   5
3   3
```