AMS 598 Project 2 Report

Tinghui Wu 114719023

Oct 12, 2022

1 Introduction

In this project, we need to implement MPI by Python to run a linear regression using bootstrap method and derive a 95% confidence interval of all the coefficients.

2 Methods

2.1 Input Data and Variables

First of all, we loaded and checked the input data frame.

```
>>> import pandas as pd
>>> data_path = '/gpfs/projects/AMS598/Projects2022/project2/project2_data.csv'
>>> df = pd.read_csv(data_path)
>>> len(df)
100000
>>> df.head()
           y        X1         X2        X3        X4        X5
0  18.931489  0.833733   9.665005  0.022726 -0.585387 -0.413407
1   1.695085 -0.276048   0.392874 -0.003376  0.798216  1.691989
2  14.369461 -0.355002 -10.038700  1.532074  0.176444  0.031926
3  -5.029825  0.087487  -6.069914 -0.167393 -1.188033 -0.275416
4 -17.369313  2.252256   2.601818  0.037507 -1.376806  0.742724
```

Figure 1: Checking the input data

The input csv file contains 100000 rows in total with the column names $[y, X1, X2, X3, X4, X5]$. We would need to sample (with replacement) from all 100000 data and run the bootstrap method to simulate the best fitted model for

$$y \sim X1 + X2 + X3 + X4 + X5$$

Secondly, the number of bootstrap has been set to 10000. That is, the linear regression needed to run 10000 times. Here we simply divided by the number of processes to get the number that each process would have to run. In our cases using 4 processes, each process would run 2500 iterations of linear regression.

```
B = 10000
b = int(B/comm.size)
```

2.2 Bootstrap Linear Regression

In each iteration, we sampled the data from data frame with replacement using built-in function from `pandas`.

```
random_df = df.sample(n=len(df), replace=True)
```

For the linear regression, we also used a built-in function from `scikit-learn`. Then, we stored the coefficients to a numpy array.

```
from sklearn.linear_model import LinearRegression

X = random_df.drop(['y'], axis=1)
y = random_df['y']

linear_regression = LinearRegression()
model = linear_regression.fit(X, y)
coef[i, :] = model.coef_
```

2.3 Confidence Interval of the Coefficients

After all the processors has finished running all the bootstrap iteration, we collected the coefficients together to generate the 95% confidence interval. That is, we found the 2.5% highest number and 2.5% lowest number in each set of coefficients. In our cases running bootstrap for 10000 times, the 250th highest and lowest number for each coefficient would be used for output.

$$10000 \times 2.5\% = 250$$

We sorted the coefficients in each process first and only gather the 250 highest and lowest to the root process, then did the sorting again to generate the final result.

3 Results

3.1 Main Output

After running the code, we got the following results in Table 1.

Coefficient	Left	Right
β_1	1.44	1.57
β_2	-0.01	0.01
β_3	0	0.12
β_4	0.43	0.55
β_5	0.54	0.66

Table 1: Confidence interval of the coefficients

```
[tinghwu@login1 project2]$ ./project2_tinghui.slurm
39
[tinghwu@login1 project2]$ cat output.log
The left CI is [ 1.43808283 -0.01188919 -0.00478916  0.42634411  0.54003878]
The right CI is [ 1.56510471  0.00801704  0.11906663  0.55134679  0.6648506 ]
Node 0: 0.254955053329, 37.5269739628, 0.00247597694397
Node 1: 0.225656986237, 37.3269939423, 0.00110197067261
Node 3: 0.268023014069, 37.1853921413, 0.00105905532837
Node 2: 0.250313997269, 37.6017549038, 0.00106811523438
```

Figure 2: Output from the code

We used 4 processes to run the code and reached a total running time 39 seconds, which was way less than the requirement 4 minutes.

3.2 Box Plot for All Coefficients

We also gathered all the coefficients and made a box plot to have a full view in Figure 3, which seems consistent to our output. This part of code was commented out for the timing of last results.

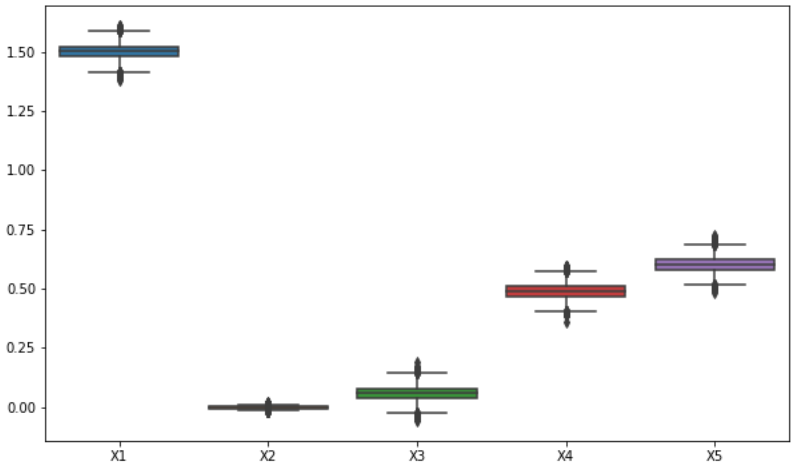

Figure 3: Box plot for all coefficients

3.3 Timing Analysis

We timed the program in each step:

- Step 1: Gathering the data file names and all the preparing
- Step 2: Iteration over all the bootstrap linear regression
- Step 3: Gathering and report the result (mainly the root node)

From the output in Figure 2 we can see that Step 2: Iteration over all the bootstrap linear regression took most of the time, which we can assume that the time for Step 2 has a linear relation with the number of iteration (coefficient `b` in our case).

We also run the code using different nodes and reported the total time as Table 2.

Number of nodes	Total time	Step 2 time
1	127	125
2	71	69
4	39	37
8	28	26
10	28	26

Table 2: Time taken running in different number of nodes

From Table 2 we can see that the total time was mainly affected by the time in Step 2, leaving the rest of the processing time (reading data, gathering results) remains 2 seconds. Also, the time taken in Step 2 wasn't perfectly half when doubling the number of nodes, making the time taken using 8 or 10 node the same. It might be that the process of starting/launching the for loop would take some fixed time.