UC Davis STA 242 2015 Spring Assignment 5 [1]

Wenhao Wu, 998587583

May 30, 2015

1 Algorithm Design

1.1 Compute the Deciles

In order to compute the deciles of the total fare less the tolls, denoted as f_{net}, we count the occurence of each value of f_{net}. The benefits are:

- There are much more records than the possible values of f_{net} in the original data. Consequently, it **saves tremendous memory usage** by counting the occurence.
- This algorithm is highly **compatible with parallel processing**. We can keep multiple tables to count the occurence of each value of f_{net} for different data files, update these tables fully in parallel, and then merge these tables to compute the deciles.

In both of our implementations, we build a table to count the occurence for each pair of data files by updating it seqentially as we read in a new piece/bulk of record(s), then combine the 12 tables to compute the deciles.

1.2 Solve The Linear Regression

Denote the trip time as t and the surcharge as f_s, respectively. In the two regression tasks, the responses are denoted as $\mathbf{y}$, a n-by-1 vector of f_{net} in all records. In the first regresssion tasks, the predictors are denoted as $\mathbf{X}_1$, a n-by-2 matrix where the first column represents the t from all records and the second column is an all 1 vector. In the second regresssion tasks, the predictors are denoted as $\mathbf{X}_2$, a n-by-3 matrix where the first and the second columns represent the t, and f_s from all records and the third column is an all 1 vector. Theoretically, the coefficients of the linear model can be computed as

$$\boldsymbol{\beta}_i = (\mathbf{X}_i^H \mathbf{X}_i)^{-1} \mathbf{X}_i^H \mathbf{y},\ i = 1, 2. \tag{1}$$

Apparently, the sufficient statistic for the linear regression tasks are $\mathbf{X}_i^H \mathbf{X}_i$ and $\mathbf{X}_i^H \mathbf{y}$, $i = 1, 2$ which has very low dimension. Moreover, these sufficient statistics can be updated sequentially as we read in a new piece/bulk of record(s), and are again highly compatible with parallel processing.
In both of our implementations, we update $\mathbf{X}_i^H \mathbf{X}_i$ and $\mathbf{X}_i^H \mathbf{y}$, $i = 1, 2$ sequentially for each pair of data files, then combine the 12 set of statistics by summing them up and solve the linear problem as in (1) to get the coefficients for the regression models.

2 Data Inspection, Pre-Processing and Extraction

Due to the limited hard drive space available on my workstation, I keep the original .zip files without decompressing them. Firstly we check that the "data" and "fare" files match each other row by row in the 3 index fields "medallion", " hack_license" and " pickup_datetime". To do so, we primarily make use of a combination of shell commands `unzip`, `cut`, `diff` , IO redirection and pipe commands to compare the 3 fields in each pair of files. (See **checkmatch.sh** in the Appendices.) We verified that the files indeed match in pairs.
During the inspection, we also notice that "trip_fare_8.csv.zip", "trip_data_9.csv.zip" and "trip_fare_9.csv.zip" contains duplicated .csv files, whcih are removed manually.
In both of our implementations, we build a "connection" to read in the output of shell pipe commands to extract the data. The shell command to extract " surcharge", "tolls_amount" and "total_amount" from the "fare" files is

```
unzip -cq ../data/trip_fare_n.csv.zip | cut -d , -f 7,10,11
```

Table 1: Deciles of the total fare less the tolls

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Pyhthon	-1430.0	6.0	7.5	8.5	9.75	11.0	13.0	15.0	18.5	26.120000000000001	685908.09999999998
R	-1430.00	6.00	7.50	8.50	9.75	11.00	13.00	15.00	18.50	26.12	685908.10

According to the data file description [], roughly 7.5% of all trips' "trip_time" is wrong so we take a safe approach to extract "pickup_datetime" and "dropoff_datetime" from the "data" files. The corresponding shell command is

```
unzip -cq ../data/trip_data_n.csv.zip | cut -d , -f 6,7
```

Later we take the differences between them as the actual trip time. Fortunately, both these two fields have a very neat format as "%Y-%m-%d %H:%M:%S" which can be easily processed.

3 Implementation in Python

Our first implementation is based on Python3. The pararllel processing is implemented with package "multiprocessing": we define a worker function `analyze_file()` to compute the count of occurence table and the sufficient statistics for the two linear regression tasks for a single pair of data/fair files. A total of 12 copies of this worker function are mapped to a pool of multiple processes and run in parallel. The results are then combined, from which the deciles are computed and the 2 linear regression problems are solved.

The worker function `analyze_file()` has a coroutine structure []: it is mainly composed of a "source" function `parse_file()` which read in one line from a pair of data/fare files, process it, and send the result to a "sink" function `accumulate_lines()`, which is in charge of updating the count of occurence table for the total amount less the toll and the sufficient statistics for the regressions.

In terms of data structure, the count of occurence table is updated as a python `dict` object and later converted to a pandas `Series` object to enable easy combination. The suffficient statistics are represented as numpy `ndarray` objects.

4 Implementation in R

Our second implementation is based on R. Similar to our first implementation, we define a worker function `analyzeFile()` to compute the count of occurence table and the sufficient statistics for the two linear regression tasks for a single pair of data/fair files, then use `parLapply()` to run it on a "cluster" for different files. After the 12 pairs of fare/data files are all processed. The results are then combined with function `reduceListSummaryNYCTaxi()` where the deciles are computed and the 2 linear regression problems are solved.

In the worker function `analyzeFile()`, instead of reading in 1 line from a pair of fare/data files at a time as in our first implementation, we use function `read.csv()` to read in a bulk of 500000 records as a data frame, and then update the count of occurence table and the sufficient statistics for regression. The benefits are two-fold. Firstly, we can use R's function `table()` to count the occurence for this bulk of record and then update the overall count of occurence table implemented with package "hash", which prove to be more efficient than updating the table directly one line at a time. Secondly, this bulk-style update would result in more accuracy in computing the sufficient statistics theoretically.

In order to further speed up function `analyzeFile()`, we implement the function to update the sufficient statistics, `updateSuffStat()`, in C++.

5 Results

5.1 Deciles and Linear Regression Results

The deciles and the two linear regression results computed using our Python and R implementation are presented in Table 1 and Table 2, respectively. As we can see, the results of the two implementaions are very close to each other. The difference is likely due to numerical accuracy issues. We believe that the R implementation is more precise.

5.2 Running Time Comparison

We test the running time of both of our implementations using different number of processes. In there tests, we use a Dell Precision T1700 workstation equipped with 16GB DDR3 RAM, a Core i7-4790K CPU and a Samsung 850 PRO

Table 2: Linear regression results

	Linear model 1		Linear model 2		
	trip time	intercept	trip time	surcharge	intercept
Python	2.02064511e-03	1.30134743e+01	2.02196225e-03	3.04104587e-01	1.29153642e+01
R	0.00202051	13.01345450	0.002021825	0.303964991	12.915390308

SSD in Ubuntu 14.04 OS. The running time are plotted in Fig. ??.

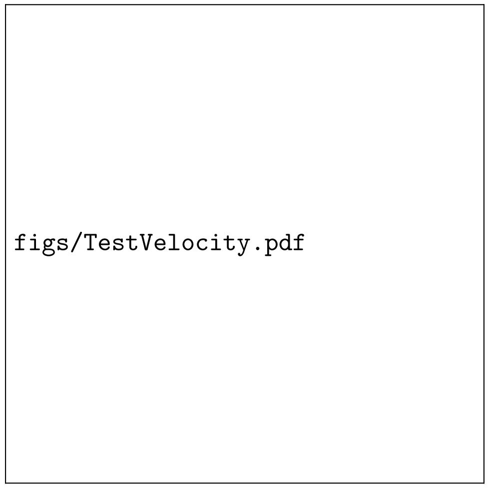

Figure 1: The evolution of average velocity of the blue cars.

6 Conclusion

References

[1] Wenhao Wu. STA 242 Assignment 5: Working with "Big Data". `git@bitbucket.org:shasqua/stat242_2015_assignment5.git`, 2015. [Online; accessed 30-May-2015].

Appendix: Source Files