Big Data - Spring 2018

Using MapReduce to explore NYC Parking Violations

In this assignment, we will analyze different aspects of parking violations in NYC using Hadoop Streaming. You will write map and reduce python programs for 7 different tasks, described later in this document.

The data we will use is open and available from NYC Open Data at:

https://data.cityofnewyork.us/City-Government/Parking-Violations-Issued-Fiscal-Year-2016/kiv2-tbus/data

https://data.cityofnewyork.us/City-Government/Open-Parking-and-Camera-Violations/nc67-uf89

The subset of the data we will use for this assignment (March 2016) can be found here:

https://vgc.poly.edu/~juliana/courses/BigData2018/Data/parking-violations.tar.gz

Note that you only need the files parking-violations.csv and open-violations.csv in the tar file above. The included file mysql-load.sql will tell you the ordering of the columns in each table (which can also be found by reading about the data on the NYC Open Data sites).

You should first test and debug your code on a smaller dataset, which you should create yourself from the available data.

For your final submission, you will run the mapreduce jobs on the complete March 2016 datasets using Hadoop streaming on dumbo. The full parking-violations.csv and open-violations.csv files for the March 2016 dataset have already been made accessible on HDFS on dumbo for you - **you should not store additional copies in your HDFS directory**. They are located at

```
/user/ecc290/HW1data/parking-violations.csv
```

and

```
/user/ecc290/HW1data/open-violations.csv
```

Submission:

You will submit the map and reduce programs for each task in a .zip file with the following structure:

* One directory per task, named "taskX", where X is the number of the task.

If you do an “ls” on your homework directory, it should contain the following sub-directories

task1

task2

task3

task4

task5

task6

task7

task8

task9

Each subdirectory for tasks 1 through 7 should include a “map.py” file, a “reduce.py” file, and a directory “taskX.out” which is the output directory generated by running your code on the complete March 2016 dataset using Hadoop Streaming with the specified number of reducers. Note that taskX.out should be a directory consisting of separate part files (one for each reduce task) rather than a single file, so use the “hfs -get” command after running your Hadoop Streaming job (and not “hfs -getmerge”).

You should not include any input files in your submission.

The subdirectory for task 9 should include one text file named “data-quality-issues.txt”.

Notes:

*** You may use combiners and partitioners, but they will not be tested, i.e., your program should work correctly without combiners/partitioners.

*** Your map.py codes are permitted to access the mapreduce_map_input_file environment variable in order to determine which input (.csv) file is being read. The CSV filenames will contain the substrings "parking" and "open". You may not use any other environment variables besides mapreduce_map_input_file. You can access this environment variable in python by doing “import os” and calling “os.environ.get(mapreduce_map_input_file)”.

*** You should use Python 3.4.4. This means you need to use the Bash wrappers provided in Lab 1, otherwise Hadoop will execute your (Python) mapper and reducer using the default version of Python.

*** You should use Hadoop Streaming only for this assignment - do not use Spark.

Assignment:

==========

Task 1

- Write a map-reduce job that finds all parking violations that have been paid, i.e., that do not occur in open-violations.csv.

- Output: A key-value pair per line — use a “tab” to separate the key and the value, a comma and space between values — where

  key = summons_number

  values = plate_id, violation_precinct, violation_code, issue_date

Here’s a sample output with 1 key-value pair:

```
1307964308 GBH2444, 74, 46, 2016-03-07
```

- For generating output for the March 2016 data, run Hadoop using 2 reducers

==========

Task 2

- Write a map-reduce job that finds the distribution of the violation types in parking_violations.csv, i.e., for each violation code, the number of violations that have this code.

- Output: A key-value pair per line — use a “tab” to separate the key and the value — where

  key = violation_code

  value = number of violations

Here’s a sample output with 1 key-value pair:

```
46    100
54    1000
```

- For generating output for the March 2016 data, run Hadoop using 2 reducers

==========

Task 3

- Write a map-reduce job that finds the total and average amount due in open violations for each license type.

- Output: A key-value pair per line — use a “tab” to separate the key and the value, a comma and space between values — where

  key = license_type

  value = total, average

where total and average are rounded to 2 decimal places.

Here’s a sample output with 1 key-value pair:

```
PAS   10000.00, 55.00

OMS   100000.00, 115.00
```

- For generating output for the March 2016 data, run Hadoop using 2 reducers

==========

Task 4

- Write a map-reduce job that computes the total number of violations for vehicles registered in the state of NY and all other vehicles.

- Output: 2 key-value pairs with one key-value pair per line — use a “tab” to separate the key and the value — following the key-value format below:

```
NY    <total number>

Other <total number>
```

- For generating output for the March 2016 data, run Hadoop using 2 reducers

==========

Task 5

- Write a map-reduce job that finds the vehicle that has had the greatest number of violations (assume that plate_id and registration_state uniquely identify a vehicle).

- Output: One key-value pair — use a “tab” to separate the key and the value, a comma and space between entries within the key/value; you should output 1 key value pair, following the key-value format below

```
plate_id, registration_state     <total_number>
```

- For generating output for the March 2016 data, run Hadoop using 1 reducer

==========

Task 6

- Write a map-reduce job that finds the top-20 vehicles in terms of total violations (assume that plate id and registration state uniquely identify a vehicle).

- Output: List of 20 key-value pairs — use a “tab” to separate the key and the value, a comma and space between entries within the key/value — using the following format:

```
plate_id, registration_state     <total_number>
```

Order results by decreasing number of violations.

- For generating output for the March 2016 data, run Hadoop using 1 reducer

==========

Task 7

- In March 2016, the 5th, 6th, 12th, 13th, 19th, 20th, 26th, and 27th were weekend days (i.e., Sat. and Sun.).

Write a map-reduce job that, for each violation code, lists the average number of violations with that code issued per day on weekdays and weekend days. You may hardcode “8” as the number of weekend days and “23” as the number of weekdays in March 2016.

- Output: List of key-value pairs — use a “tab” to separate the key and the value, a comma and space between values using the following format:

```
violation_code   weekend_average, week_average
```

where weekend_average and week_average are rounded to 2 decimal places.

- For generating output for the March 2016 data, run Hadoop using 2 reducers

==========

Task 8

- Write a map-reduce job that finds the distribution of terms for the Make and Color columns, i.e., for each value in those columns, how many times they appear in the column. Hint: you can use the wordcount algorithm we covered in class.

Please replace missing (empty) values by the string "NONE" (without quotation marks).

- Output: List of key-value pairs — use a “tab” to separate the key and the value, a comma and space between values using the following format:

`column_name` `term, term_frequency`

Order results by alphabetical order based on the term, and list the results for Make before the results for Color. Your program should make a **single pass** over the data

==========

Task 9: extra credit

- List any data quality issues you have encountered in the provided datasets in a text file called data-quality-issues.txt

Note:

We will provide an automated script for you to test the formatting of your results before you submit them.

Testing your solutions

We have provided a script to test your solutions on the March 2016 data. On dumbo, type

```
hfs -get /user/ecc290/HW1data/hw1tester.tar
```

and then type

```
tar xvf hw1tester.tar
```

To run the tests, type

```
./testall.sh <INPUTPATH>
```

where `<INPUTPATH>` is the **full** path to your directory containing the subdirectories task1, task2, etc. **without a trailing slash** (e.g., "`/home/ecc290/HW1Tasks`")

You will be told for each task, whether you pass or fail. If you fail some task X, you can view the diff of your output and the solution file in results/taskX.diff.

Note: Passing the test does not guarantee your code is correct. It does guarantee that your results are properly formatted.