Big Data - Fall 2019
Assignment #1
Using MapReduce and Spark to explore NYC Parking Violations
Due: 9AM, September 30, 2019

Goal

In this assignment, we will analyze different aspects of parking violations in NYC using both Hadoop Streaming and Spark. You will be presented with 7 tasks. Your assignment is to write 1) map and reduce Python programs for the first 3 tasks, and 2) PySpark programs using both RDDs and DataFrames for all 7 tasks. These first few pages show where you can download the required datasets and instructions for submissions. The following pages detail the tasks.

Data

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

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

Open Parking and Camera Violations
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: 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).

We advise that you 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 your MapReduce jobs and PySpark programs on the **complete** March 2016 datasets using Hadoop streaming and Spark, respectively, 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
/user/ecc290/HW1data/open-violations.csv
```

We have also made accessible versions with the headers included, which will be useful for defining schema in the SparkSQL tasks:

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

Submission Instructions

MapReduce Programs

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
```

Each subdirectory for tasks 1 through 3 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.

Remarks:

- 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 8 (extra credit) should include one text file named "data-quality-issues.txt".

Important Notes:

1. You may use *combiners* and *partitioners*, but they will not be tested, i.e., your program should work correctly without them.
2. 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 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)"*.
3. You should use Python 3.6.5. 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.

Spark Programs

You will submit the Spark programs for all tasks in a .zip file named "yournetid.zip" (e.g., "ecc290.zip"). The zip file should include 14 python files: task1.py, task1-sql.py,...,task7.py, task7-sql.py. Thus, for each task, you will use both core Spark using RDDs and SparkSQL using DataFrames. As in the MapReduce portion, you will be reading from CSV files. To do this in core Spark, reading into an RDD, you would use, e.g.,

```
from csv import reader
lines = sc.textFile(sys.argv[1], 1)
lines = lines.mapPartitions(lambda x: reader(x))
```

To do this in SparkSQL, reading into a DataFrame, you would use, e.g.,

```
parking = spark.read.format('csv').options(header='true',
inferschema='true').load(sys.argv[1])
```

Remarks:

- For this portion, you do not need to include output files in your submission.
- You should also not include any input files in your submission.
- Your code should output a directory named “taskx.out” or “taskx-sql.out” to HDFS, i.e., use the Spark RDD function saveAsTextFile(“taskx.out”) or the Spark DataFrame function save(“taskx-sql.out”,format=”text”) rather than python I/O. *(In order for the hw1tester script to work, you must name your output directories “taskx.out” and ”taskx-sql.out” where x is in 1 through 7).*

Important Notes

1. Your program should read in the path to the input file on HDFS from the command line arguments. For task 1, you are guaranteed that *parking-violations.csv* will be the first of the two files passed in. In other words, we will execute your programs with the following commands:

   For task 1:

   ```
   spark-submit --conf
   spark.pyspark.python=/share/apps/python/3.4.4/bin/python
   {your_netid}/task1.py /user/ecc290/HW1data/parking-violations.csv
   /user/ecc290/HW1data/open-violations.csv
   ```

   For task 1 - SQL:

   ```
   spark-submit --conf
   spark.pyspark.python=/share/apps/python/3.4.4/bin/python
   {your_netid}/task1-sql.py /user/ecc290/HW1data/parking-
   violations-header.csv /user/ecc290/HW1data/open-violations-
   header.csv
   ```

   For task 3:

   ```
   spark-submit --conf
   spark.pyspark.python=/share/apps/python/3.4.4/bin/python
   {your_netid}/task3.py /user/ecc290/HW1data/open-violations.csv
   ```

   For task 3-SQL:

   ```
   spark-submit --conf
   spark.pyspark.python=/share/apps/python/3.4.4/bin/python
   {your_netid}/task3-sql.py /user/ecc290/HW1data/open-violations-
   header.csv
   ```

For all other tasks:

```
spark-submit --conf
spark.pyspark.python=/share/apps/python/3.4.4/bin/python
{your_netid}/taskx.py /user/ecc290/HW1data/parking-violations.csv
```

and for SQL versions:

```
spark-submit --conf
spark.pyspark.python=/share/apps/python/3.4.4/bin/python
{your_netid}/taskx-sql.py /user/ecc290/HW1data/parking-
violations-header.csv
```

2. You should only use the most recent available versions of Python and Spark on Dumbo (3.6.5 and 2.4.0, respectively)
3. For core Spark: You may find that using a final *map()* stage is helpful for formatting your output correctly.
4. For SparkSQL: You may find that using a final *select()* which produces a single column using *format_string()* is helpful for formatting the output and writing to a text file. For example, for Task 1-SQL, if *result* is the dataframe that contains the query results, you could write:

```
result.select(format_string('%d\t%s, %d, %d,
%s',result.summons_number,result.plate_id,result.violation_precin
ct,result.violation_code,date_format(result.issue_date,'yyyy-MM-
dd'))).write.save("task1-sql.out",format="text")
```

Tasks

Task 1: Find all parking violations that have been paid, i.e., that do not occur in *open-violations.csv*.

Output: A key-value* pair per line, where:

key = summons_number

values = plate_id, violation_precinct, violation_code, issue_date

(*Note: separate key and value by the tab character (‘\t’), and elements within the key/value should be separated by a comma then a space. *This applies to all tasks below*)

Your output format should conform to the format of following examples:

```
1307964308      GBH2444, 74, 46, 2016-03-07
4617863450      HAM2650, 0, 36, 2016-03-24
```

To complete this task,

1) Write a map-reduce job. Run Hadoop using 2 reducers.
2) Write a Spark program.
3) Write a SparkSQL program.

Task 2: Find the frequencies of the violation types in *parking_violations.csv*, i.e., for each violation code, the number of violations that this code has.

Output: A key-value pair per line, where:

key = violation_code

value = number of violations

Here are sample outputs with 1 key-value pair:

```
1     159
46    100
```

To complete this task,

1) Write a map-reduce job. Run Hadoop using 2 reducers.
2) Write a Spark program.
3) Write a SparkSQL program.

Task 3: Find the *total* and *average* amounts due in open violations for each license type.

Output: A key-value pair per line, where:

key = license_type

value = total, average
where total and average are rounded to 2 decimal places.

Here are sample outputs with 1 key-value pair:

```
PAS   10000.00, 55.00
OMS   100000.00, 115.00
USC   250.00, 125.00
```

To complete this task,

1) Write a map-reduce job. Run Hadoop using 2 reducers.
2) Write a Spark program.
3) Write a SparkSQL program.

Task 4: Compute 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, following the key-value format below:

```
NY          <total number>
Other       <total number>
```

To complete this task,

1) Write a Spark program.
2) Write a SparkSQL program.

Task 5: Find 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, following the key-value format below:

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

To complete this task,

1) Write a Spark program.
2) Write a SparkSQL program.

Task 6: Find 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 using the following format:

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

*Ordered by decreasing number of violations. For items with the same number of violations, order by ascending plate_id.

To complete this task,

1) Write a Spark program.
2) Write a SparkSQL program.

Task 7 For each violation code, list 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. In March 2016, the 5th, 6th, 12th, 13th, 19th, 20th, 26th, and 27th were weekend days (i.e., Sat. and Sun.).

Output: List of key-value pairs using the following format:

```
violation_code        weekend_average, week_average
```

where weekend_average and week_average are rounded to 2 decimal places.

To complete this task,

1) Write a Spark program.
2) Write a SparkSQL program.

Task 8: extra credit

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

End of Assignment #1

Testing your solutions

We have provided a script to test your solutions on the March 2016 data.

To test of your **MapReduce** programs, 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`")

To test your **Spark** programs, on Dumbo, type

```
hfs -get /user/ecc290/HW1data/hw2tester.tar
hfs -get /user/ecc290/HW1data/README
```

To unpack the files, type

```
tar xvf hw2tester.tar
```

To run the tests, type

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

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

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.