ASSIGNMENT 1 - UE21CS343AB2 - BIG DATA

Introduction

This assignment on introduces an alternative input method - JSON and multi-stage Map-Reduce.

Assignment Objectives and Outcomes

- Get familiar with map reduce
- Get familiar with aggregation using map reduce
- Working with hadoop

Ethical Practices

Please submit original code only. You can discuss your approach with your friends but you must write original code. All solutions must be submitted through
the portal. We will perform a plagiarism check on the code and you will be penalized if your code is found to be plagiarized.

Submission Deadline

17th September, 2023.

The portal uptimes will be communicated to you in due course. Please do not wait
till the last minute to submit your code since wait times may be long.

Submission Links

Assignments need to be submitted on portal: https://bigdata-pesu.tech

(the site is offline right now, the online timings will be communicated via groups and mails).

Submission Guidelines

You will need to make the following changes to your Python scripts before submitting them.

Include the following shebang at the top of your Python scripts.

```
#!/usr/bin/env python3
```

Convert your files to executable using the following command.

```
chmod +x *.py
```

Convert line breaks in DOS format to Unix format (this is necessary if you are coding on Windows without which your code will not run our portal).

```
dos2unix *.py
```

Software/Languages to be used:

- `Python 3.10.x`
- `Hadoop v3.3.3+`

NOTE : STRICTLY NOT ALLOWED TO IMPORT ANY AND ALL PACKAGES IN MAPPER AND
REDUCER. The packages that can be imported are mentioned below.

Specification

Task 1

Problem Statement

Strike Rate is the average runs a batsman scores in 100 balls. Given the input, find the final strike rate of each batsman.

Mapper

Input : Array of JSON Objects

Example :

```
[
    { "name": "xyz", "runs": 100, "balls": 100 },
    { "name": "xyz", "runs": 10, "balls": 10 },
    { "name": "abc", "runs": 30, "balls": 10 },
    { "name": "abc", "runs": 20, "balls": 5 },
    { "name": "abc", "runs": 10, "balls": 42 }
]
```

NOTE : Do not load the entire dataset as the test case we use will be extremely large and will crash the server. You will be blacklisted in that case.

Output : format it as you please

- Output must be the name,local_strike_rate
- local_strike_rate refers to the strike rate of the particular match

Strike Rate formula = (runs/balls) * 100 [rounded upto 3 decimal places].

Local strike rate to be also rounded off to 3 decimals, not formatted to 3 decimals. Round off again in reducer after taking average.

```
Example rounding off:
10.110 -> 10.11 (0 at the end makes no sense)
10.1106 -> 10.111
10.1103 -> 10.11
10 -> 10.0 (it has to be float)
```

- Output of mapper goes to regular filesystem
- NOTE : round value to 3 decimals
- If number of balls = 0, then local strike rate is also 0.

Reducer

Input : Same format as Mapper output

Output : Independent JSON Objects

- output of the reducer has the following keys : name of batsman, and average strike rate accross all matches

average strike rate = sum of all local strike rates / total matches [rounded upto 3 decimal places]

Example :

```
{ "name": "xyz", "strike_rate": 100 }
{ "name": "abc", "strike_rate": 241.27 }
```

Sample Input

1. sample_data.json

2. expected_output_sample_data.txt for the above input

```
{"name": "Deepti", "strike_rate": 61.551}
{"name": "Harmanpreet", "strike_rate": 87.124}
{"name": "Ishan", "strike_rate": 85.077}
{"name": "Jemimah", "strike_rate": 77.407}
{"name": "Renuka", "strike_rate": 74.35}
{"name": "Rohit", "strike_rate": 71.464}
{"name": "Shubman", "strike_rate": 66.041}
{"name": "Smriti", "strike_rate": 57.807}
{"name": "VVS Laxman", "strike_rate": 64.078}
{"name": "Virat", "strike_rate": 89.928}
```

Test Dataset

Test your code with the following dataset once it passes the sample input dataset

1. **Input**: large_data.json
2. **Output**: expected_output_large_data.txt

All the files can be found here.

Instructions

1. Write a python mapper
   - Name : `mapper.py`
   - Read the specification for input and output as mentioned above
   - Only packages that can be imported are : json and sys
2. Write a python reducer to perform the aggregation
   - Name : `reducer.py`
   - Read the specification for input and output as mentioned above
   - Only packages that can be imported are : sys
3. Test it out with the sample dataset given and check the expected output
4. Adhere to the submission guidelines

Testing instructions

Local testing

```
cat <path_to_dataset>.json | ./mapper.py | sort -k 1,1 | ./reducer.py
```

Hadoop testing

Put the input file in hdfs first

- Make a dir in HDFS
  - `hdfs dfs -mkdir /example`
- Put the input file in HDFS
  - `hdfs dfs -put <path to input file in local system> /example`

- Now your input file is in /example/<input_file_name>
- This HDFS path to input file is the input path in the below command.

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-mapper "$PWD/mapper.py" \
-reducer "$PWD/reducer.py" \
-input  <path_to_input_in_hdfs> \
-output <path_to_output_folder_in_hdfs>
```

TASK 2: Join, Filter and Aggregate

Story Background:

You are working for a leading e-commerce platform, and your team is tasked with analyzing a massive dataset in the form of a text file. This dataset contains two tables, one after the other, and contains crucial information about customer orders and product reviews.

Table 1: Customer Orders

- Each line in this table represents a customer order.
- Fields: Order ID, Customer ID, Product ID, Quantity, Price per Unit.

Table 2: Product Reviews

- Each line in this table represents a product review submitted by customers.
- Fields: Review ID, Product ID, Customer ID, Rating (1-5 stars), Review Text.

Objective:

Your goal is to gain valuable insights from this dataset. Specifically, you want to understand the relationship between product reviews and the number of items sold for each product. You plan to use a multi-stage Hadoop MapReduce job to join, filter, and aggregate the data to answer the following question:

Question:

“How can you leverage Hadoop MapReduce to perform a join operation between customer order data and product review data, filter out low-rated reviews (ratings less than 3), and aggregate the results to identify products with the most negative reviews and the quantity of those products sold?”

Example:

Input Text File (input.txt):

- The file is a text file with Tab-Separated Values(TSV).
- Column 1 represents the type of record, i.e. whether it belongs to the Customer Orders Table (“order”) or the Product Reviews Table (“review”).

```
order  1  C101  P001  2  20.00
order  2  C102  P002  3  25.00
order  3  C103  P001  1  20.00
order  4  C104  P003  2  30.00
review  101  P001  C101  4  "Great product, very satisfied."
review  102  P002  C102  2  "Not happy with the quality."
review  103  P003  C104  2  "Average product, could be better."
review  104  P001  C103  1  "Terrible, wouldn't recommend."
```

Expected Output (output.txt):

```
# Products with Negative Reviews and Quantity Sold - DO NOT PRINT
# Product ID,  Quantity Sold - DO NOT PRINT
P002  3
P001  1
P003  2
```

Explanation:

In the expected output, we have identified products with negative reviews (ratings less than 3) and calculated the quantity of those products sold. In this example, “P002” had the lowest product rating (2), and it was sold in a quantity of 3. “P001” also had negative reviews (with one rating of 1 and one of 4) and was sold in a quantity of 3. “P003” had a rating of 2 (considered average) and was sold in a quantity of 2.

This output is the result of the Hadoop MapReduce job, which involved joining the customer order and product review data, filtering out low-rated reviews, and aggregating the quantity sold for each product with negative reviews.

Workflow Constraints:

1. You are obligated to use exactly three stages to account to this solution. (i.e. three pairs of Mappers and Reducers)
2. Do not import any external modules. Use modules only available in the default python package.
3. The output should be in TSV format.

While designing your solution, you may leverage the following assumptions

1. A customer does not repeat an order.
   If you encounter a row which says customer C1, ordered 10 of product P1,

you need not be worried about searching whether C1 ordered any more of P1 at a later time, you can immediately assume that the final quantity of product P1 ordered by C1 is 10

2. A customer does not leave duplicate reviews
   If you encounter a rating of 2 that customer C1 left on product P1, then you need not have to check whether the customer C1 gave another rating for product P1 at a later time.

To summarize, for each (Product ID, Customer ID) pair, there exists only one unique quantity in the orders table, and at most one rating in the reviews table.

HINT: Try to frame your workflow such that the three stages mimic the actions of JOIN, FILTER and AGGREGATE respectively.

Sample Input

dataset_sample.txt

Sample Output

expected_output_dataset_sample.txt

```
AQJZBDZL45205293	9
BICYGUPD57809078	19
GHIHXNYM43631821	13
HGRUYMDQ86957814	13
JRJRUKLO25747746	13
MAOSEHSQ24182331	19
PZSLAOMX00719228	17
```

Test Dataset

Test your code with the following dataset once it passes the sample dataset

1. **Input**: dataset_large.txt
2. **Output**: expect_output_dataset_large.txt

All the files can be found here.

Testing your code

Here is a sample script on testing your multi-stage map reduce.

```
#!/usr/bin/env bash

# Clean up the intermediate HDFS directories which could have been created as
a part of a previous run
```

```
hdfs dfs -ls /intermediate-1
if [[ $? == 0 ]]
then
        echo "Deleteing Intermediate-1 HDFS directory before starting job.."
        hdfs dfs -rm -r /intermediate-1
fi

hdfs dfs -ls /intermediate-2
if [[ $? == 0 ]]
then
        echo "Deleting Intermediate-2 HDFS directory before starting job.."
        hdfs dfs -rm -r /intermediate-2
fi

# Clean up the previously used output directory.
hdfs dfs -ls /task2/output
if [[ $? == 0 ]]
then
    echo "Deleting previous output directory"
    hdfs dfs -rm -r /task2/output
fi


echo "Initiating stage-1"
echo "============================================================"

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-mapper "$PWD/mapper_1.py" \
-reducer "$PWD/reducer_1.py" \
-input /task2/dataset.txt \
-output /intermediate-1

echo "============================================================"
echo "Stage-1 done"
echo "Initiating stage-2"
echo "============================================================"
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-mapper "$PWD/mapper_2.py" \
-reducer "$PWD/reducer_2.py" \
-input /intermediate-1/part-00000 \
-output /intermediate-2
echo "============================================================"
echo "Stage-2 done"
echo "Initializing stage-3"
echo "============================================================"

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-mapper "$PWD/mapper_3.py" \
-reducer "$PWD/reducer_3.py" \
-input /intermediate-2/part-00000 \
-output /task2/output

echo "============================================================"
echo "Stage-3 done"
```
Note: Change Hadoop version as required 3.3.3 or 3.3.6