Lab 6: Intricate Hadoop Programs

Due date: March 1, 11:59pm.

Note: Lab 7 will be assigned some time during your period of work on **Lab 6**.

Lab Assignment

Assignment Preparation

This is a pair programming lab. You can pair with anyone in the class. Please note, this is **NOT** a "team of two" lab. The difference is that you must work on **each** program in this assignment together with your partner.

Overview

In this lab we use three datasets - our `iowa.csv` data for 10,000 purchases of alcohol in Iowa, a collection of free books from Project Guttenberg, and a dataset of student performance on math, reading and writing standardized assessments.

All datasets you need to use are uploaded to the `/data` directory on HDFS. You all should have read access to all files in that directory.

For each dataset, the assignment asks you to answer one or more questions by building appropriate Hadoop MapReduce programs. As a rule, you are asked to build one program per question asked (unless otherwise mentioned). At the same time, the structure of your programs may exceed the traditional single MapReduce job we have been seeing thus far. All additional information about each dataset (data format, etc.) is linked to from the Lab 6 web page:

`http://users.csc.calpoly.edu/~dekhtyar/369-Winter2019/labs/lab06.html`

1 Iowa Liquor sales dataset.

Your first two programs will work with the Iowa Liquor sales dataset. In addition to the `iowa.csv` file, you will use the `counties.json` file (note - it contains multi-line JSON objects, I am releasing an example of how this can be managed using a third-party `InputFormat` class, but we won't have time in class to concentrate on this: use the provided example to make it work).

The iowa.csv file has the following format:

The file represents a CSV version of some of the data from our Iowa liquor sales database. Each line in the file is a single record in the format (in a single line)

```
<Invoice>, <Store Number>, <Store County>, <Vendor Name>, <Item Number>, <Item Description>,
                                                                 <Bottles Sold>, <Sale (Dollars)>
```

Here are a few sample lines from the input file:

```
'S24966600138',2614,'Scott','Brown-Forman Corporation',86817,'Southern Comfort Cherry',2,29.56
'S24043700013',2641,'Pottawattamie','Mccormick Distilling Company',36903,'Mccormick Vodka',96,163.2
'S22293800009',4925,'Polk','Sazerac Co., Inc.',64866,'Fireball Cinnamon Whiskey',12,161.64
'S15881700017',3976,'Iowa','Luxco-St Louis',81208,'Paramount Peppermint Schnapps',2,21.24
```

The `counties.json` file contains records in the following format:

```
{
        "id": <county Id>,
        "county": <county Name>,
        "population": <county population>
}
```

Here is an example of a record:

```
{
        "id": "95 ",
        "county": "Taylor County ",
        "population": "6,214"
    }
```

Please note that population is recorded as a string with a "," separating thousands.

The two files are available in the `hdfs /data/` directory.

Program 1: PerCapita.java

In what is now a well-established tradition of this class, you are going to repeat a problem from **Lab 1** again, now in Hadoop[1].

[1] You can also be assured that a Spark implementation of this and one or two other problems familiar to you is coming. This gives you an opportunity to compare and contrast

Compute per capita sales of alcohol to liquor stores by county. Output the name of the county (you can choose whether the word "county" is present in the output or not), the total sales in dollars of the alcohol to stores in the county, and the per capita sales.

Hint. This is more or less a straightforward join. You have a choice whether you want to use map-side or reduce-side join. Because another assignment will require a map-side join/use of distributed cache, I recommend a reduce-side join for this problem. Use the `om.alexholmes.json.mapreduce.MultiLineJsonInputFormat` class provided to you.

Problem 2: Correlation.java

We want to understand whether the number of sales of different types of alcohol to different counties are correlated. For this assignment, we concentrate on two types of alcohol: vodka and rum. To determine whether a specific sale documented in the `iowa.csv` file is a rum or a vodka sale, you need to detect words `"Rum"`, `"rum"`, `"Vodka"`, or `"vodka"` in the text of the `<Item Description>` column. Ignore all other drink sales[2].

For each county, compute two numbers: the total number of sales of rum and the total number of sales of vodka ("total number of sales" = number of unique receipts). With the two sets of sales, compute the Pearson correlation between them, and output just the correlation.

Let $rum = (r_1, \ldots, r_n)$ and $vodka = (v_1, \ldots, v_n)$ where n is the number of Iowa counties, and r_i and v_i are the number of sales of rum and vodka repspectively for *the same* county for each i. Then, the Pearson correlation between rum and $vodka$ is found as follows:

$$pearson(rum, vodka) = \frac{\sum_{i=1}^{n}(r_i - \mu_{rum})(v_i - \mu_{vodka})}{\sqrt{\sum_{i=1}^{n}(r_i - \mu_{rum})^2}\sqrt{\sum_{i=1}^{n}(v_i - \mu_{vodka})^2}}$$

Here,

$$\mu_{rum} = \frac{1}{n}\sum_{i=1} nr_i; \mu_{vodka} = \frac{1}{n}\sum_{i=1}^{n}$$

are the means of the rum and $vodka$ vectors respectively. In your computations you can use a hardcoded value of n for the total number of counties in Iowa (it is 99, I believe).

your implementations in different frameworks.

[2] That is - it is possible that a rum or a vodka do not have the words "Rum", "rum", "Vodka", or "vodka" in their description. You can ignore such sales for the purposes of this exercise.

Hints. This requires multiple MapReduce cycles. You need a cycle to compute histograms of rum and vodka purchases by county. You also need to compute the means for the number of rum and number of vodka sales in a single county. These means need to be used in the computation of the Pearson correlation.

There are multiple different MapReduce architectures that will allow you to perform this computation. Individual Map and Reduce functions are going to be relatively straightforward. The complexity of this problem is in proper organization of the MapReduce Jobs, and passing of information from one job to another. You *may* find using a Distributed Cache convenient at some point, although there are solutions that do not require it.

Guttenberg Dataset

The `Guttenberg` dataset is a collection of eleven[3] text files containing most downloaded Project Guttenberg English-language books during the week of Feburary 3–9. The books are found in the `/data/Guttenberg` directory:

```
$ hdfs dfs -ls /data/Guttenberg
Found 11 items
-rw-r--r--   3 hdfs hdfs     173595 2019-02-15 01:13 /data/Guttenberg/11-0.txt
-rw-r--r--   3 hdfs hdfs     724726 2019-02-15 01:13 /data/Guttenberg/1342-0.txt
-rw-r--r--   3 hdfs hdfs      51185 2019-02-15 01:13 /data/Guttenberg/1952-0.txt
-rw-r--r--   3 hdfs hdfs     234041 2019-02-15 01:13 /data/Guttenberg/219-0.txt
-rw-r--r--   3 hdfs hdfs    1276201 2019-02-15 01:13 /data/Guttenberg/2701-0.txt
-rw-r--r--   3 hdfs hdfs     616320 2019-02-15 01:13 /data/Guttenberg/76-0.txt
-rw-r--r--   3 hdfs hdfs     450783 2019-02-15 01:13 /data/Guttenberg/84-0.txt
-rw-r--r--   3 hdfs hdfs     804335 2019-02-15 01:13 /data/Guttenberg/98-0.txt
-rw-r--r--   3 hdfs hdfs      39700 2019-02-15 01:13 /data/Guttenberg/pg1080.txt
-rw-r--r--   3 hdfs hdfs     594933 2019-02-15 01:13 /data/Guttenberg/pg1661.txt
-rw-r--r--   3 hdfs hdfs     142384 2019-02-15 01:13 /data/Guttenberg/pg844.txt
```

Problem 3: Dice.java

We only have one task for this dataset, but it is relatively complex. We want to compare the 11 books based on their word usage. For this particular exercise we choose a relatively straightforward and limited means of comparison, but the overall architecture of your Hadoop program will allow you to make such comparisons more complex in the future.

This is a multi-step problem, and is the only problem where, *if you want* you are allowed to use multiple Java programs to solve. (This is because this problem has a natural off-line/on-line computing components which can be credibly separated into separate programs.) If you choose to use multiple programs, `Dice.java` shall be your final program, and you can name your other programs as you desire, and provide **full instructions** for compilation and running in your `README` file.

[3] Sorry, I was shooting for ten, and overshot by one.

First, you shall discover, for each of the documents the top 100 most frequent words.

Second, for each pair of documents, you shall compute their Dice index based on the top 100 most frequent words.

Given two sets $D_1 = \{w_1, \ldots, w_n\}$ and $D_2 = \{v_1, \ldots, v_m\}$, the Dice index $dice(D_1, D_2)$ is defined as follows:

$$dice(D_1, D_2) = \frac{2|D_1 \cap D_2|}{n + m}$$

The Dice index of 1 means that both sets coincide, the Dice index of 0 means that the two sets share no common elements (in our case - words).

The ouput of `Dice.java` shall be a collection of triples: the names of two documents (you can use file names) and the value of the Dice coefficient. Each pair needs to be considered only once so if you have `<Document1, Document2>` already reported, there is no need to report `<Document2, Document1>`.

You are allowed to hardcode the names of the documents (filenames) in your program, as well as to use 100 (number of most frequent words) as a constant.

Hints. You need to solve **three subproblems** here. The **first one** is to compute the list of 100 most common non-stopwords. This is a texbook example of a top-K problem solved on top of your standard word count problem. The **second problem** is: given two top 100 lists, compute Dice coefficient. This requires intersection operation. In turn, intersection can be viewed as a special case of a join. One key constraint is that you are not allowed to simply send all 100 words from each document into your Reducer and compute that intersection. This wont work if you are computing the Jaccard coefficient of two much larger sets. Instead, come up with an idea for a reduce-side join, possibly followed by an aggregation to compute it. In your computations, you are allowed to hardcode the use of 100 in the denominator of the Jaccard coefficient. Your third problem is how to structure the computation of the Jaccard coefficient for each pair of books. This is more of a software architecture problem, but you are going to do developing software that goes above and beyond a simple MapReduce job very often.

Student Performance Dataset

The Student Performance dataset, available as `/data/StudentsPerformance.csv` has information about individual student performance on three standard assessment tests: math, writing and reading comprehension. For each student, their demographic information is made available.

The columns in the file are:

1. **gender**: male or female.

2. **race/ethnicity**: this information is presented in the form of `Group A`, ... `Group D` values.

3. **parental level of education**. This attribute has a variety of values. We will not be using it in this lab.

4. **lunch** indicates whether the student receives a regular lunch or a discounted/free lunch.

5. **test preparation course** is an indicator of whether the student took a test prepartion course.

6. **math score**, **reading score**, **writing score**: the test scores reported in this order.

Problem 4: StudentStats.java

Find mean test scores and standard deviations of test scores for male students, female students, students in each racial category, students who receive free or discounted lunch, students who do not receive free or discounted lunch.

For each category of students, your output shall contain the name of the category, and six numbers: three means and three standard deviations, in the order: math score mean, math score standard deviation, reading score mean, reading score standard deviation, writing score mean, writing score standard deviation.

Hint. This is not a complex task. Your main challenge is to do it all in a single MapReduce cycle. This means working on multiple grouping operations in parallel in the same Map() and the same Reduce().

Problem 5: Correlations.java

Find the Pearson correlations between math and writing score for each category of students from **Program 4**.

The Pearson Correlation coefficient formula is the same as you need to apply for **Program 2**, so this is an example of a MapReduce pattern that I want you to repeat and remember well. The setup is slightly different. Basically, for each category of students (e.g., female students), you need to determine the Pearson correlation coefficient between the math and writing scores of of all the students belonging to the category. The output shall be the name of the category followed by one number – the correlation coefficient itself.

Hint. This problem should be solved using a combiner operation that potentially is different than a reducer operation. The idea is that portions of the Pearson correlation can be constructed in the combiner, and the the reducer can then simply aggregate the combiner outputs. The second issue is that Pearson correlation needs means of each column by category This is something you have just computed in Program 4. You can use the output of Program 4 as one of the input files to Program 5. This is a good place to use Distributed Cache.

Submission

Submit your Java programs and all other files necessary to run your code. Submit README file describing how to compile and run all your programs. Submit Makefiles or compile-and-run bash scripts to compile and run your programs from `ambari-head`. Submit a `README` file describing anything I should know about your implementation.

All submitted files must contain your names on them.

Submit all your code in a single archive (zip or tar.gz).

Use `handin` to submit as follows:

```
$ handin dekhtyar lab06 <FILES>
```

Good Luck!