Lab 6: Hadoop Programs

Due date: Monday, March 18, 11:59pm.

Note: There will be **minimal, if any** grace period for this lab, as the deadline has already been stretched very thin.

Lab Assignment

Assignment Preparation

This is an individual lab. I expect every person to complete it without consulting others.

Overview

In this lab you will create a number of Hadoop MapReduce programs that perform a variety of tasks on several different data inputs. The tasks are separated into simple programs (Week 5 programs) and intricate programs (Week 6 programs). Week 5 programs have the following features:

- The inputs are files with simple, easy-to-understand structure, with each record stored in a single line.
- The `map()` and `reduce()` methods do not need to use any functionality beyond core Java libraries.
- The MapReduce job you are asked to implement is an example of a specific type of a data-processing operation discussed in class.

Week 6 programs require more sophistication and may involve any of the following:

- Implementing a join between multiple data files, or a self-join

- Use of distributed cache
- Use of combiners
- Multiple input files
- Multiple MapReduce stages

Datasets

Iowa Liquor Sales.

In addition to the `iowa.csv`, which will be used for Week 5 tasks, for Week 6 tasks you will use the `counties-hadoop.json` file. 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 (we might have a quick demo of multiline JSON management in one of the labs in Week 6). Please, note, `counties-hadoop.json` has been formatted to hadoop processing of data, and it is NOT a valid JSON array. Rather, it is a list of JSON objects with no punctuation in between.

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",
 "population": 6214 }
```

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

Week 5 Programs

Program 1: vodkaratio.java

Write a Java Hadoop program that computes the following information for each county:

- How many total bottles of alcohol were purchased
- The percentage of those bottles that is vodka (see below)
- The total amount of money paid for the purchased alcohol (add up the Sale (Dollars) numbers
- The percentage of the money that went into purchasing vodka.

Name your program `vodkaRatio.java`

Notes: To determine if a purchase is of vodka search the Item description column for appearance of the substring vodka in ANY capitalization. If found - this is a vodka purchase. If not found - it is not a vodka purchase.

Program 2: Grouping and Aggregation

Create a MapReduce program named `countySales.java` that works as follows.

Input. The input to the program is the `iowa.csv` located in `/data` directory on HDFS. You all should have read access to this file.

Processing. The MapReduce job shall compute, for each county, (a) how many total bottles were purchased, (b) the total amount of money the purchased alcohol was worth, and (c) the average price per bottle.

Output. The `reduce()` method shall output the county name as the key, and the triple $\langle$ number of purchased bottles, total amount of money spent on purchasing alcohol, average price per bottle $\rangle$ as the value.

Program 3: Inverting Construct a MapReduce program `invertedIndex.java` that works as follows.

Input. The input to the program is the `iowa.csv` file introduced in the description of **Program 3**.

Processing. For this task, you are interested in the `Item Description` column of the input CSV file (sixth column in the file). Your program shall split each `Item Description` into individual words (e.g., 'Southern Comfort Cherry' is split into `'Southern'`, `'Comfort'` and `'Cherry'` words). For each word found in the entire input (all `Item Description` values), your program shall compute and output the following information:

- Number of unique occurrences of the word
- Number of unique counties that occur in the same record as the word
- Number of unique `Item Description` values in which the word occurs

Output. In the output, use the word as the key, and output a printable object that consists of the three values specified above.

Week 6 Programs

Program 4: PerCapita.java

This is essentially one of your Midterm 1 MongoDB programs.

Compute per capita sales of alcohol to liquor stores by county. Output the name of the county, 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.

Program 5: HighestPerCapita.java

This is exactly one of your Midterm 1 MongoDB programs.

This program shall extend your `PerCapita.java` program and report the county with the highest per capita alcohol purchases. Only the records for the counties with the highest per capita alcohol purchases need to be reported.

However, you **must** do this properly.

Problem 6: 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[1].

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).

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.

Problem 7: Dice.java

This problem involves different data: the `Guttenberg` dataset, a collection of eleven[2] text files containing most downloaded Project Guttenberg English-language books during the week of Feburary 3–9, 2019. The books are found in the `/data/Guttenberg` directory:

[1] 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.

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

```
$ 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
```

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.

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 won't 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 Dice coefficient. Your third problem is how to structure the computation of the Dice 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.

Submission

Use `handin` for submission. Submit the your Java programs, and a `README` file with your name and any comments you need to make about your code.

Use the `unixN.csc.calpoly.edu` servers for submission.

Use the following command for the submission:

```
$ handin dekhtyar lab06 <files>
```