Lab 9: Hadoop Programs on Non-Synthetic Datasets

Due date: March 19 (Sunday), 11:59pm.

Lab Assignment

Assignment Preparation

This is a pair programming lab. You can pair with anyone in the class.

Overview

In this lab, you are given access to a number of datasets from two sources: the University of California at Irvine Machine Learning dataset repository[1] and the Stanford Large Network Dataset Collection (SNAP)[2].

Some datasets are relatively small, some are comparatively large. All datasets contain real observations. The datasets were used for various machine learning tasks in the past. We will use them for somewhat different purposes.

All datasets you need to use are uploaded to the `/datasets` 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).

To successfully complete the assignment, each team must submit solutions to **four (4)** different problems, using **at least three (3)** different datasets. All other solutions submitted will earn each team **extra credit**.

All additional information about each dataset (data format, etc.) is linked to from the Lab 8 web page:

[1] http://archive.ics.uci.edu/ml/
[2] http://snap.stanford.edu/data/

http://users.csc.calpoly.edu/~dekhtyar/369-Winter2017/labs/lab08.html

1 Poker Dataset.

The UCI poker dataset is available at

http://archive.ics.uci.edu/ml/datasets/Poker+Hand

See the detailed description of the dataset there (it is linked to from the Lab 8 web page on the course web site). The dataset consists of two files available to you on HDFS[3]:

```
-rw-r--r--   3 dekhtyar hdfs   24538333 2017-03-05 16:36 /data/poker-hand-testing.data.txt
-rw-r--r--   3 dekhtyar hdfs     613694 2017-03-05 16:36 /data/poker-hand-traning.true.data.txt
```

Both files have exactly the same format. Each line in each of the files describes a single poker hand drawn from a deck of 52 cards (without jokers). Each card is described as a pair of numbers. The first number is in the range $1 \ldots 4$ describes the suit of the card. The second number, in the range $1 \ldots 13$ describes the value of the card (Ace is 1, Jack is 11, Queen is 12, King is 13). Each line has 11 comma-separated values. First 10 values describe the five cards. The last value is an indicator of what poker hand it is (high card, pair, two pairs, three of a kind, ..., Royal flush).

The cards are drawn randomly and are not sorted, so the same five-card hand can actually be represented in $5! = 120$ different ways.

Problem 1

We want to find the list of most common card triples that occur in **both** `poker-hand-testing.data.txt` and `poker-hand-training.true.data.txt` files. We define a common occurrence of a triple of cards as follows: it is a *unique* occurrence of three cards **in any order** in one hand of one file, **matched** by a *unique* occurrence of the same three cards (again, **in any order**) in another file.

Example. Consider an example of three cards: {3 Spades, 7 Spades, A Spades}. If this combination of cards occurs in 19 different poker hands in one file, and in 13 different poker hands in another file, the total number of co-occurrences (or common occurrences) for this set of three cards is 13 - we can match the first 13 of the 19 appearances of the triple in one file, with unique appearances in the other file, but the remaining six appearances in file 1 will be unmatched. If a triple {6 Spades, 8 Spades, K Spades} occurs 14 times in the first file, and 14 times in the second file, the latter triple is considered to be more commonly co-occurring (14 co-occurrences vs. 13) despite the former triple being more frequent in the two files.

[3]The `/data` directory also contains a `poker-hand-names.txt` file, but it is not required for the assignments of this lab.

Write a Hadoop Java program using as many MapReduce cycles as you need to discover and report the top 20 most frequently poker hands co-occurring in the two files from the dataset.

Note: Reporting top 20 most frequent triples requires a number of tricks that we will be discussing in class during the upcoming week.

Name your program `CommonHands.java`.

Problem 2

Let us define the value of a five-card hand (this is outside of the rules of poker), as the sum of the face values of all the cards in the hand, where the values of a Jack, Queen and King are considered to be 11, 12 and 13 respectively, while the value of an Ace is 14 (not 1 as it is presented in the dataset).

Write a Hadoop program to compute the number of hands with the highest value in each of the files, compare the numbers and report the file name of the file that contains more hands of the highest value (you can also report supplementary information: the highest value, and the frequencies of the highest value hands in each of the files).

Name your program `LargestValue.java`

Hint. There is a number of ways to approach this problem. Essentially, one MapReduce process needs to be used to compute the highest value of a hand in a given file. This can be done using the "find Top 1" MapReduce pattern, or by finding the highest value in one MapReduce process, and uploading it as a distributed cache value for subsequent MapReduce jobs. You are allowed any number of MapReduce repetitions (chains) to compute this.

Seeds Dataset

The UCI Machine Learning repository Seeds dataset is a fairly small dataset that contains information about 270 varieties of seeds. The dataset (including the explanation of the values) is available at

`http://archive.ics.uci.edu/ml/datasets/seeds`

The modified version of the seeds dataset is available on HDFS in one file:

```
-rw-r--r--   3 dekhtyar hdfs      10020 2017-03-05 17:08 /data/seeds_dataset.csv
```

The modification is to convert the original UCI data file into a CSV format, and to add a line number to each line to simplify completion of the assignments.

Each line in the dataset is a comma-separated vector of nine values. First value is the line/record number. The nineth value in each line can be ignored (it is used for classification purposes). The remaining seven values represent various measurements of the different seeds.

Problem 3

We are interested in performing a *k-nearest neighbors search* to discover which seeds are most similar to which other seeds.

Consider the difference between the two seeds in the `seeds` dataset to be the Eucledian distance between the vectors of seed features. The more similar the seeds are, the smaller the distance between their vectors is.

Write a Hadoop Java program that takes as input the seeds dataset file (as many times as you need it), and outputs, for each seed in the dataset, *five most similar seeds* to it (together with the distances).

Your output should use line numbers when referring to individual seeds. For example, if the five nearest neighbors of seed number 1 are seeds 14, 76, 154, 32, and 50, the output can look roughly as follows:

```
1 (14, <d1>), (76, <d2>), (154, <d3>), (32, <d4>), (50, <d5>)
```

where `<d1>, <d2>, <d3>, <d4>, <d5>` are real numbers representing the distances between the respective seeds and seed number 1.

Name your program `SeedKNN.java`

Hint. This is essentially a direct application of the `top k` MapReduce design pattern.

Problem 4

One of the disadvantages of using Eucledian distance directly on the vectors stored in the `seeds` dataset is that the scales of different features of the seeds may be different, and the contribution to the distance between two seeds of the features with smaller ranges (e.g., column 3 of the dataset) has the potential for being insiginificant when compared to the contribution of features with larger ranges (e.g., features 1 and 2).

To fix this problem, we can normalize the vectors first. Given a vector $s = (f_1, \ldots, f_7)$, each column/feature value f_i can be normalized as follows:

$$nf_i = \frac{f_i - \mu_i}{\max_i - \min_i},$$

where μ_i is the average value of feature i in the dataset, $\max_i$ is the largest observed value of feature i in the datset, and $\min_i$ is the smallest observed value of feature i in the dataset.

Write a Hadoop program that completes the same task as `SeedKNN.java` - i.e., reports the 5 nearest neighbors for each seed, but does it by first normalizing every seed vector using the procedure described above, and then using the Eucledian distance between the normalized vectors to measure the difference between the seeds.

Name your program `SeedNormKNN.java`.

Slashdot

Slashdot dataset is one of a series of social network datasets available on SNAP. The dataset we are using is the Slashdot social network, November 2008, available from this URL:

`http://snap.stanford.edu/data/soc-Slashdot0811.html`

The dataset consists of a single file, available on HDSF as:

```
-rw-r--r--   3 dekhtyar hdfs   10747200 2017-03-05 17:21 /data/soc-Slashdot0811.txt
```

Here are the first few lines of the data file:

```
# Directed graph (each unordered pair of nodes is saved once): Slashdot0811.txt
# Slashdot Zoo social network from Noveber 6 2008
# Nodes: 77360 Edges: 905468
# FromNodeId    ToNodeId
0       0
0       1
0       2
0       3
```

Your code shall ignore the lines that start with the `"#"` character. The remaining lines contain edges of the directed graph for the Slashdot Zoo social network. A directed edge $(first, second)$ represents an directed interaction from user $first$ to user $second$.

Note: For this dataset, you may want to use `NLineInputFormat` to process the data using multiple mapper instances (the size of the dataset is just under the 128Meg limit on a single split).

Problem 5

We are interested in the reachability relationship between the nodes in the Slashdot Zoo graph. Specifically, if we look at interactions between the users, we want to find out if the "Kevin Bacon" rule holds.

Write a Hadoop Java program which, for each node in the graph, finds all other nodes in the graph reachable from it in **six or fewer steps**. The program must report the list of 50 nodes which can reach the largest number of other nodes. For each node report the number of nodes that can be reached.

Name your program `SlashdotReach.java`.

Problem 6

Let us invert the problem. Rather than reporting the top 50 nodes based on how many other nodes are reachable *from them*, write a Hadoop Java program that reports the top 50 nodes *based on how many nodes can reach them* in six or fewer steps.

Name your program `SlashdotVIP.java`.

Household Power Consumption Dataset

The household power consumption dataset reports the power consumption in a single household over a period of a number of years at one minute intervals. It is available from the UCI Machine Learning repository:

```
http://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption
```

The dataset is stored in a single file available on HDFS as:

```
-rw-r--r--   3 dekhtyar hdfs  132960755 2017-03-05 17:26 /data/household_power_consumption.txt
```

The data is stored in the semicolon-separated format, with each line in the file documenting a single reading of the data. The first few lines of the file look as follows:

```
Date;Time;Global_active_power;Global_reactive_power;Voltage;Global_intensity;Sub_met
ering_1;Sub_metering_2;Sub_metering_3
16/12/2006;17:24:00;4.216;0.418;234.840;18.400;0.000;1.000;17.000
16/12/2006;17:25:00;5.360;0.436;233.630;23.000;0.000;1.000;16.000
16/12/2006;17:26:00;5.374;0.498;233.290;23.000;0.000;2.000;17.000
16/12/2006;17:27:00;5.388;0.502;233.740;23.000;0.000;1.000;17.000
16/12/2006;17:28:00;3.666;0.528;235.680;15.800;0.000;1.000;17.000
16/12/2006;17:29:00;3.520;0.522;235.020;15.000;0.000;2.000;17.000
```

Your code shall ignore the first line of the file (check to see if the first letter is "D" for example).

The UCI website describes the data in the file as follows:

```
Attribute Information:

1.date: Date in format dd/mm/yyyy
2.time: time in format hh:mm:ss
```

```
3.global_active_power: household global minute-averaged active power (in kilowatt)
4.global_reactive_power: household global minute-averaged reactive power (in kilowatt)
5.voltage: minute-averaged voltage (in volt)
6.global_intensity: household global minute-averaged current intensity (in ampere)

7.sub_metering_1: energy sub-metering No. 1 (in watt-hour of active energy).
     It corresponds to the kitchen, containing mainly a dishwasher, an oven
     and a microwave (hot plates are not electric but gas powered).

8.sub_metering_2: energy sub-metering No. 2 (in watt-hour of active energy).
     It corresponds to the laundry room, containing a washing-machine,
     a tumble-drier, a refrigerator and a light.

9.sub_metering_3: energy sub-metering No. 3 (in watt-hour of active energy).
     It corresponds to an electric water-heater and an air-conditioner.
```

Also of interest to you is this note:

```
(global_active_power*1000/60 - sub_metering_1 - sub_metering_2 - sub_metering_3)
represents the active energy consumed every minute (in watt hour) in the household by electrical
equipment not measured in sub-meterings 1, 2 and 3.
```

The most interesting attribute here is attribute 3 - the global active power used by the household.

Problem 7

We want to find the days with the highest active global power consumption in each of the years during which the observations were made. To find out the power consumption on each day, we need to add up all the global active power values from all observations that took place during that day.

Write a Hadoop Java program that reports for each year, the date of the day with the highest power consumption and the power consumption itself. Use the power conversion listed above (except do not subtract sub meterings) to compute the power consumption each minute, add up the computed values.

Name your program `PowerDays.java`.

Problem 8

We want to find the times of the highest difference in the power consumption in the kitchen. The highest difference shall be measured in the intervals 5 minutes. That is, you want to compare the power consumption in the kitchen at some time t to the power consumption at time $t + 5$ minutes and find the times when the difference was the largest (this means that in the five minutes that elapsed someone has seriously ramped up the use of the kitchen).

Write a Hadoop Java program that finds the 20 largest differences in power consumption in the kitchen. Report the pairs of times, the power consumptions in the kitchen and the difference between them.

Name your program `KitchenUse.java`

Problem 9

Find the difference between the average power consumption on the days when the household actively used **both** the laundry facilities and the heating/air-conditioning, and the average power consumption on the days when the household used neither.

In this problem, it is left up to you to determine empirically what it means to "actively use laundry facilities" and what it means to "actively use heating/air-conditioning". Once you establish these parameters, use them to split the days into the three categories (active use of both, use of one, use of neither), and compare the power consumptions for the two categories that we are interested in.

Name your program `PowerComp.java`

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 cslvm57).

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 lab08 <FILES>
```

Good Luck!