cse6242-2019-fall-hw3

CSE 6242 / CX 4242: Data and Visual Analytics | Georgia Tech | Fall 2019

Homework 3 : Hadoop, Spark, Pig and Azure

Prepared by our 30+ wonderful TAs of CSE6242A,Q,OAN,O01,O3/CX4242A for our 1200+ students

Submission Instructions and Important Notes:

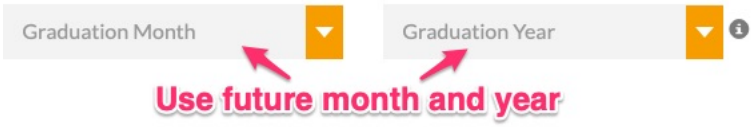
It is important that you read the following instructions carefully and also those about the deliverables at the end of each question or **you may lose points**.

- ❏ Always check to make sure you are using the most up-to-date assignment (version number at bottom right of this document).
- ❏ Submit a single zipped file, called "HW3-{GT username}.zip", containing all the deliverables including source code/scripts, data files, and readme. Example: "HW3-jdoe3.zip" if GT account username is "jdoe3". Only .zip is allowed (no other format will be accepted). Your GT username is the one with letters and numbers.
- ❏ You may discuss high-level ideas with other students at the "whiteboard" level (e.g., how cross validation works, use hashmap instead of array) and review any relevant materials online. However, each student must write up and submit his or her own answers.
- ❏ All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures (e.g., reported to and directly handled by the Office of Student Integrity (OSI)). Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.
- ❏ At the end of this assignment, we have specified a folder structure about how to organize your files in a single zipped file. 5 points will be deducted for not following this strictly.
- ❏ In your final zip file, do not include any intermediate files you may have generated to work on the task, unless your script is absolutely dependent on it to get the final result (which it ideally should not be).
- ❏ We may use auto-grading scripts to grade some of your deliverables, so it is extremely important that you strictly follow our requirements.
- ❏ Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
- ❏ Every homework assignment deliverable and every project deliverable comes with a 48-hour "grace period". Any deliverable submitted after the grace period will get zero credit. We recommend that you plan to finish by the beginning of the grace period in order to leave yourself time for any unexpected issues which might arise.
- ❏ We will not consider late submission of any missing parts of a homework assignment or project deliverable. To make sure you have submitted everything, download your submitted files to double check. You may re-submit your work before the grace period expires. Canvas automatically appends a "version number" to files that you re-submit. You do not need to worry about these version numbers, and there is no need to delete old submissions. We will only grade the most recent submission.

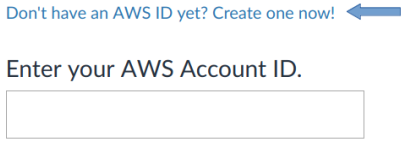
===== DO THIS NOW: apply for AWS Educate account, and set up alarms =====

Apply for an **AWS Educate** account **RIGHT AWAY** to get $100 free AWS credit, and verify that the credit has been properly applied to your account. Creating the account can take days and this assignment's computation can take hours to run, so if you do not do this now, you may not be able to submit your work in time, because you may still be *waiting* for jobs to complete.

- Go to AWS Educate page
- Click the **Join AWS Educate** button.
- Choose **Student**, and click **Next**
- **Read all the following important points**, then fill out the application form
  - ○ You must use your **@gatech.edu** email address (GT is an AWS member school).
  - ○ For "Graduation Month" and "Graduation Year" (see below) **both of them should be in the future**, since you are still a student and have not graduated yet.



  - ○ You must stay with the default option--- **you MUST NOT choose the "starter" option** --- or you will NOT receive the full $100 credit.
  - ○ You will need to provide the AWS account ID of your AWS account --- **your AWS account is NOT the same as your AWS Educate account**. If you do not have an AWS account yet (thus, no AWS account ID), sign up for one as indicated in the form (see screenshot below). After your AWS account has been created, you MUST return to your AWS Educate application to finish signing up, **or your AWS Educate will not be created**.



  - ○ Your AWS Account ID is at the top of this screen.
- For any emails that you may be receiving from AWS, please remember to check your "spam" email folders. Many students reported emails get pushed to those folders automatically.

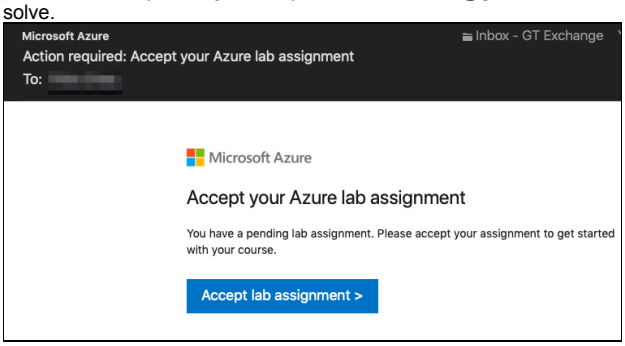
EXTREMELY IMPORTANT: enable billing alerts AND set up a billing alarm on AWS to notify you when your credits are running low. The AWS Setup Guidelines document in Q3 will show you how to enable alarms.

SHUT DOWN EVERYTHING when you're done with the instances (don't leave them on over weekends/holidays!), or you may get surprising credit card bills. Highest record from previous classes went above $2000! **The good news is you can call AWS to explain the situation and they should be able to waive the charges** -- call (phone) AWS customer care **immediately** (not email) so you can resolve the issue quickly (usually within minutes).
==========

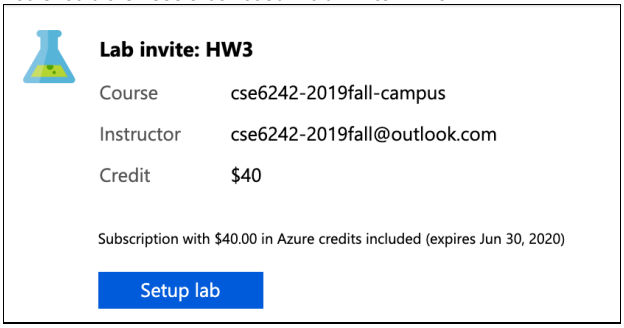
===== DO THIS NOW: Accept Azure Lab Assignment =====

VERY IMPORTANT: Use Firefox or Chrome in incognito/private browsing mode when configuring anything related to Azure (e.g., when using Azure portal), to prevent issues due to browser caches. Safari sometimes loses connections.

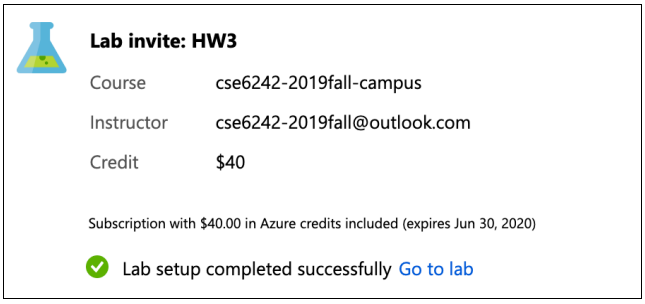
By now, you should have received an email sent to your **GTUsername@gatech.edu** email account (your GT username is the one with letters and numbers, e.g., jdoe3). The email is titled "**Action required: Accept your Azure lab assignment**" (see image below). Click "**Accept lab assignment**" to access $40 Azure credit that we have pre-allocated to you. Log in with your **GTUsername@gatech.edu** email address. You must use this email address, or you may run into issues that we do not know how to solve.



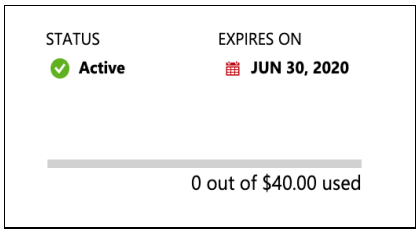
You should then see a box titled "**Lab invite: HW3**".



Click the "**Setup Lab**" button -- you may need to wait for 1-2 minutes (and you may be asked again to login using your GT account). Then, you should see the message that says "**Lab setup completed successfully**". Please note that it might take a while until you can use the allocated Azure credit to create a cluster.



Click the "**Go to Lab**" text link. Then, you should see that credit has been applied to your Azure account and how much of that credit has been used, as shown in the image below:



Keep in mind that there is **always a lag** between your actual credit consumption and what you see in that panel.

The Azure credit should be more than enough for this assignment. If you run out of credit, request extra credit by sending us (instructors) a private Piazza post (tag the post with "azurecredit"), with a brief explanation of why you need the credit. **It can take a day or more before you can use the extra credit; if you make last-minute requests, you may not be able to turn in your work in time.**

VERY IMPORTANT: Your virtual machine over Azure should only be enabled when you are using Hadoop service. If you are not running any computation over Azure, you should **deallocate your virtual machine** (different from a stopped state) using Azure management portal, to avoid getting charged. Microsoft's logic here is that you are still occupying some resources over cloud, even if you are not using them for running any computation.

==========

Grading

You can score a maximum of 120 points in this assignment.

Download the HW3 Skeleton before you begin.

Setting up Development Environment for Q1

Installing the Virtual Machine

Follow the Instructions to download, setup a preconfigured virtual machine (VM) image that you will use for this assignment

Loading Data into HDFS

Now, let us load our dataset into the HDFS (Hadoop Distributed File System), an abstract file system that stores files on clusters. Your Hadoop code will directly access files on HDFS. Paths on the HDFS look similar to those on the UNIX system, but you cannot explore them directly using standard UNIX commands. Instead, you need to use ***hadoop fs*** commands. For example

```
hadoop fs -ls
```

Download the following two graph files: graph1.tsv[1] (~5MB when unzipped) and graph2.tsv[2] (~1.1GB when unzipped) into the VM and unzip them. Also place the Q1 skeleton code in the VM. Use the following commands to setup a directory on the HDFS to store the two graph datasets. Do not change the directory structure below (data/) since we will grade your homework using the scripts that assume the following directory structure. First, navigate to the folder in your skeleton code that contains the **src** directory, **pom.xml**, **run1.sh**, and **run2.sh** files and enter the following commands.

```
hadoop fs -mkdir data
hadoop fs -put <insert path to data file>/graph1.tsv data
hadoop fs -put <insert path to data file>/graph2.tsv data
```

Please check the VM setup instructions if you are facing any file permission errors while running the commands.

Now both files (graph1.tsv and graph2.tsv) are on HDFS, at **data/graph1.tsv** and **data/graph2.tsv**. To check this, try:

```
hadoop fs -ls data
```

Setting up Development Environments

We found that compiling and running Hadoop code can be quite complicated. So, we have prepared some skeleton code, compilation scripts, and execution scripts for you that you can use, in the HW3 skeleton folder. You should use this structure to submit your homework.
In the directory of *Q1*, you will find **pom.xml**, **run1.sh, run2.sh** and the **src** directory.

- The **src** directory contains a main Java file that you will primarily work on. We have provided some code to help you get started. Feel free to edit it and add your files in the directory, but the main class should be Q1. Your code will be evaluated using the provided **run1.sh** and **run2.sh** file (details below).

- **pom.xml** contains the necessary dependencies and compile configurations for each question. To compile, you can simply call Maven in the corresponding directory (i.e., Q1 where pom.xml exists) by this command:

```
mvn package
```

It will generate a single JAR file in the target directory (i.e., target/q1-1.0.jar). Again, we have provided you some necessary configurations to simplify your work for this homework, but you can edit them as long as our run script works and the code can be compiled using mvn package command.

- ***run1.sh***, ***run2.sh*** are the shell script files that run your code over graph1.tsv (run1.sh) , graph2.tsv (run2.sh) and download the output to a local directory. The output files are named based on its question number and graph number (e.g. q1output1.tsv). You can use these run scripts to test your code. Note that these scripts will be used in grading.

To execute the shell scripts from the command prompt, enter:

```
./run1.sh
./run2.sh
```

Here's what the above scripts do:

1. Run your JAR on Hadoop specifying the input file on HDFS (the first argument) and output directory on HDFS (the second argument)
2. Merge outputs from output directory and download to local file system.
3. Remove the output directory on HDFS.

Q1 [15 points] Analyzing a Graph with Hadoop/Java

Imagine that your boss gives you a large dataset which contains an entire email communication network from a popular social network site. The network is organized as a directed graph where each node represents a person's email address and an edge between two nodes (e.g., address A and address B) has a weight stating how many times A has written to B. You have been tasked with finding the person that each person has written to the most, along with that count (see the example below for more clarification). Your task is to write a MapReduce program in Java to report, for each node X (the "source", or "src" for short) in the graph, the person Y (the "target" or "tgt" for short) that X has written to the most, and the number of times X has written to Y (the outbound "weight", from X to Y). **If a person has written to multiple targets that have exactly the same (largest) number of times, return the target with smallest node id.**

First, go over the Hadoop word count tutorial to familiarize yourself with Hadoop and some Java basics. You will be able to complete this question with only some knowledge about Java. You should have already loaded two graph files into HDFS and loaded into your HDFS file system in your VM. Each file stores a list of edges as tab-separated-values. Each line represents a single edge consisting of three columns: (source node ID, target node ID, edge weight), each of which is separated by a tab (\t). Node IDs and weights are positive integers. Below is a small toy graph, for illustration purposes (on your screen, the text may appear out of alignment).

```
src         tgt         weight
10         110         3
10         200         1
200         150         30
100         110         10
110         130         15
110         200         67
10         70         3
```

Your program should not assume the edges to be sorted or ordered in any ways (i.e., your program should work even when the edge ordering is random).

Your code should accept two arguments. The first argument (*args[0]*) will be a path for the input graph file on HDFS (e.g., data/graph1.tsv), and the second argument (*args[1]*) will be a path for output directory on HDFS (e.g., data/q1output1). The default output mechanism of Hadoop will create multiple files on the output directory such as part-00000, part-00001, which will be merged and downloaded to a local directory by the supplied run script. Please use the run1.sh and run2.sh scripts for your convenience.

The format of the output: each line contains a node ID, followed by a tab (\t), and the expected "target node ID,weight" tuple (without the quotes; and note there is no space character before and after the comma). Lines do not need to be sorted. The following example result is computed based on the toy graph above. Please exclude nodes that do not have outgoing edges (e.g., those email addresses which have not sent out any communication).

For the toy graph above, the output is as follows.

```
10         70,3
200         150,30
100         110,10
110         200,67
```

Deliverables

1. **[5 points] Your Maven project directory including Q1.java.** Please see detailed submission guide at the end of this document. You should implement your own MapReduce procedure and should not import external graph processing library.

2. **[5 points] q1output1.tsv**: the output file of processing graph1.tsv by run1.sh.

3. **[5 points] q1output2.tsv**: the output file of processing graph2.tsv by run2.sh.

Q2 [25 pts] Analyzing a Large Graph with Spark/Scala on Databricks

Tutorial: First, go over this Spark on Databricks Tutorial, to get the basics of creating Spark jobs, loading data, and working with data.

You will analyze mathoverflow.csv[3] using Spark and Scala on the Databricks platform. This graph is a temporal network of interactions on the stack exchange web-site MathOverflow. **The dataset has three columns in the following format: ID of the source node (a user), ID of the target node (a user), Unix timestamp (seconds since the epoch).**

Your objectives:

1. Remove the pairs where the *questioner* and the *answerer* are the same person. Very important: all subsequent operations must be performed on this filtered dataframe.
2. List the top 3 **answerers** who answered the highest number of questions, sorted in descending order of questions answered count. If there is a tie, list the individual with smaller node ID first.
3. List the top 3 **questioners** who **asked** the highest number of questions, sorted in descending order of questions asked count. If there is a tie, list the individual with the smaller node ID first.
4. List the top 5 most common *answerer-questioner* pairs, sorted in descending order of pair count. If there is a tie, list the pair with the smaller answerer node ID first. If there is still a tie, use the questioner node ID as the tie-breaker by listing the smaller questioner ID first.
5. List, by month, the number of interactions (questions asked/answered) from September 1, 2010 (inclusively) to December 31, 2010 (inclusively).
6. List the top 3 individuals with the most overall activity (i.e., highest total questions asked and questions answered).

You should perform this task using the DataFrame API in Spark. Here is a guide that will help you get started on working with data frames in Spark.

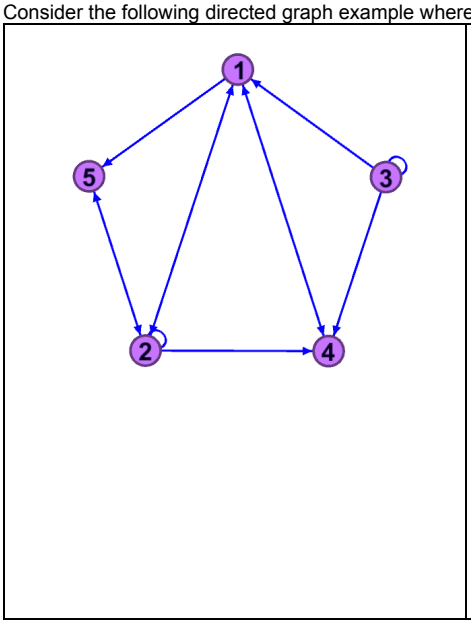
A template Scala notebook, q2-skeleton.dbc has been included in the HW3-Skeleton that reads in a sample graph file *examplegraph.csv*. In the template, the input data is loaded into a dataframe, inferring the schema using reflection (Refer to the guide above).

Note: You **must** use only Scala DataFrame operations for this task. You will lose points if you use SQL queries, Python, or R to manipulate a dataframe.

You may find some of the following DataFrame operations helpful:
`toDF, join, select, groupBy, orderBy, filter`

Upload the data file examplegraph.csv and q2-skeleton.dbc to your Databricks workspace before continuing. Follow the Databricks Setup Guide for further instructions.

Consider the following directed graph example where we show the result of achieving the objectives on *examplegraph.csv*.



answerer	questioner	timestamp
1	4	1283296645
3	4	1283297908
1	4	1283298280
2	2	1283298467
3	4	1283299092
1	2	1283300824
2	1	1283300967
1	2	1283301485
3	1	1283302207
2	4	1283303844
4	1	1283304158
3	1	1283304547
1	2	1283305468
2	1	1283305625
1	7	1283306548
3	3	1283306910
7	2	1283309524
4	1	1283310284
1	4	1283310295
2	7	1283310872

1. Remove the pairs where the questioner and the answerer are the same person. For example, the instance of the edge `answerer: 2 - questioner: 2` and `answerer: 3 - questioner: 3` should be removed from *examplegraph*.

answerer	questioner	timestamp
1	4	1254192988
3	4	1254194656

```
|       1|         4|1254202612|
|       2|         2|1254232804|
|       3|         4|1254263166|
…
|       1|         7|1254392595|
|       3|         3|1254395022|
|       7|         2|1254396925|
…
|       2|         7|1254436186|
+--------+----------+----------+
```

2. List the top 3 answerers who answered the highest number of questions, sorted in descending order of question count. If there is a tie, list the individual with the smaller node ID first. For the above *examplegraph* the results would look like this:

```
+--------+------------------+
|answerer|questions_answered|
+--------+------------------+
|       1|                 7|
|       2|                 4|
|       3|                 4|
+--------+------------------+
```

3. List the top 3 questioners who asked the highest number of questions, sorted in descending order of question count. If there is a tie, list the individual with the smaller node ID first. For the above *examplegraph* the results would look like this:

```
+----------+---------------+
|questioner|questions_asked|
+----------+---------------+
|         1|              6|
|         4|              6|
|         2|              4|
+----------+---------------+
```

4. List the top 5 most common questioner-answerer pairs, sorted in descending order of pair count. If there is a tie, list the pair with the **smaller answerer node ID** first. If there is still a tie, use the questioner node ID as the tie-breaker by listing the smaller questioner ID first. For the above *examplegraph* the results would look like this:

```
+--------+----------+-----+
|answerer|questioner|count|
+--------+----------+-----+
|       1|         2|    3|
|       1|         4|    3|
|       2|         1|    2|
|       3|         1|    2|
|       3|         4|    2|
+--------+----------+-----+
```

5. List, by month, the number of interactions (questions asked/answered) from September 1, 2010 (inclusively) to December 31, 2010 (inclusively). The month of September is represented by the number 9, month of October by 10 and so on. For the above *examplegraph* the results would look like this:

```
+-----+------------------+
|month|total_interactions|
+-----+------------------+
|    9|                14|
+-----+------------------+
```

6. List the top 3 individuals with the most overall activity, i.e., highest total questions asked and questions answered. For the above *examplegraph* the results would look like this:

```
+------+--------------+
|userID|total_activity|
+------+--------------+
|     1|            13|
|     2|             8|
|     4|             8|
+------+--------------+
```

We have provided you with the walkthrough of the steps you need to perform with an example graph. Now it is your turn to **replace the examplegraph with the mathoverflow graph** and list the results of your experiments in the provided ***q2_results.csv*** file.

Deliverables

1. **[10 pts]**
   a. **q2.dbc** Your solution as Scala Notebook archive file (.dbc) exported from Databricks. See the Databricks Setup Guide on creating an exportable archive for details.
   b. **q2.scala**, Your solution as a Scala source file exported from Databricks. See the Databricks Setup Guide on creating an exportable source file for details.

Note: You should export your solution as both a .dbc & a .scala file.

2. **[15 pts] q2_results.csv:** The output file of processing ***mathoverflow.csv*** from the q2 notebook file. You must copy the output of the display()/show() function into the file titled ***q2_results.csv*** into the relevant sections.

Q3 [35 points] Analyzing Large Amount of Data with Pig on AWS

You will try out Apache Pig for processing n-gram data on Amazon Web Services (AWS). This is a fairly simple task, and in practice you may be able to tackle this using commodity computers (e.g., consumer-grade laptops or desktops). However, we would like you to use this exercise to learn and solve it using distributed computing on Amazon EC2, and gain experience (very helpful for your career), so you will be prepared to tackle problems that are more complex.

The services you will primarily be using are Amazon S3 storage, Amazon Elastic Cloud Computing (EC2) virtual servers in the cloud, and Amazon Elastic MapReduce (EMR) managed Hadoop framework.

For this question, you will only use up a very small fraction of your $100 credit. AWS allows you to use up to 20 instances in total (that means 1 master instance and up to 19 core instances) without filling out a "limit request form". For this assignment, you should not exceed this quota of 20 instances**.** Refer to details about instance types, their specs, and pricing. In the future, for larger jobs, you may want to use AWS's pricing calculator.

AWS Guidelines

Please read the AWS Setup Guidelines provided to set up your AWS account.

Datasets

In this question, you will use a dataset of over 130 million customer reviews from Amazon Customer Reviews. (Further details on this dataset are available here).

You will perform your analysis on two datasets based off of this data, which we have prepared for you: a small one (~1GB) and a large one (~32GB).

VERY IMPORTANT: Both the datasets are in the **US East (N. Virginia)** region. Using machines in other regions for computation would incur data transfer charges. Hence, set your region to **US East (N. Virginia)** in the beginning (not Oregon, which is the default). **This is extremely important, otherwise your code may not work and you may be charged extra.**

The files in these two S3 buckets are stored in a tab ('\t') separated format. **An example of their structure is available here.**

Goal

Output the top 15 product categories having the highest **average star rating per review** along with their corresponding averages, in **tab-separated format**, sorted in descending order. Only consider entries whose review bodies are 100 characters or longer, have 30 or more total votes, and were verified purchases. If multiple product categories have the same average, **order them alphabetically on product category**. Refer to the example and calculations below:

```
product_id                review_id                star_rating
Toys                        RDIJS7QYB6XNR        3
Toys                        BAHDID7JDNAK0        4
Toys                        YHFKALWPEOFK4        2
Books                        BZHDKTLJYBDNE        4
Books                        ZZUDNFLGNALDA        3

Books                (4 + 3 ) / 2 = 5.5
Toys                (3 + 4 + 2) / 3        = 3.0
```

Note: The small dataset only includes 5 categories, so you will only get 5 rows for its output.

Sample Output

To help you evaluate the correctness of your output, we provide you with the output for the small dataset

Note: Please strictly follow the formatting requirements for your output as shown in the small dataset output file. You can use https://www.diffchecker.com/ to make sure the formatting is correct. Improperly formatting outputs may not receive any points.

Using PIG (Read these instructions carefully)

There are two ways to debug PIG on AWS (all instructions are in the AWS Setup Guidelines):

1. **Use the interactive PIG shell** provided by EMR to perform this task from the command line (grunt). Refer to Section 8: Debugging in the AWS Setup Guidelines

for a detailed step-by-step procedure. You should use this method if you are using PIG for the first time as it is easier to debug your code. However, as you need to have a persistent ssh connection to your cluster until your task is complete, this is suitable only for the smaller dataset.

2. **Upload a PIG script** with all the commands which computes and direct the output from the command line into a separate file. Once you verify the output on the smaller dataset, use this method for the larger dataset. You don't have to ssh or stay logged into your account. You can start your EMR job, and come back after when the job is complete!

Note: In summary, verify the output for the smaller dataset with Method 1 and submit the results for the bigger dataset using Method 2.

Sample Commands: Load data in PIG

To load data for the **small example** use the following command:

```
grunt> reviews = LOAD 's3://amazon-reviews-pds/tsv/amazon_reviews_us_M*' AS
(marketplace:chararray,customer_id:chararray,review_id:chararray,product_id:chararray,product_parent:chararray,product_title:cha
review_date:chararray);
```

To load data for the **large example** use the following command:

```
grunt> reviews = LOAD 's3://amazon-reviews-pds/tsv/*' AS
(marketplace:chararray,customer_id:chararray,review_id:chararray,product_id:chararray,product_parent:chararray,product_title:cha
review_date:chararray);
```

Note:

- Refer to other commands such as LOAD, USING PigStorage, FILTER, GROUP, ORDER BY, FOREACH, GENERATE, LIMIT, STORE, etc.
- Copying the above commands directly from the PDF and pasting on console/script file may lead to script failures due to the stray characters and spaces from the PDF file.
- Your script will fail if your output directory already exists. For instance, if you run a job with the output folder as **s3://cse6242oan-<GT account username>/output-small**, the next job which you run with the same output folder will fail. Hence, please use a different folder for the output for every run.
- You might also want to change the input data type for **star_rating** to handle floating point values.
- While working with the interactive shell (or otherwise), you should first test on a small subset of the data instead of the whole data (the whole data is over 100 GB). Once you believe your PIG commands are working as desired, you can use them on the complete data and wait since it will take some time.

Deliverables

- **pig-script.txt**: The PIG script for the question (using the **larger** data set).
- **pig-output.txt**: Output **(tab-separated)** (using the **larger** data set).

Note: Please strictly follow the guidelines below, otherwise your solution may not be graded.

- Ensure that file names (case sensitive) are correct.
- Ensure file extensions (.txt) are correct.
- The size of each pig-script.txt and pig-output.txt file should not exceed 5 KB.
- Double check that you are submitting the correct set of files --- we only want the script and output from the larger dataset. Also double check that you are writing the right dataset's output to the right file.
- Ensure that unnecessary new lines, brackets, commas etc. aren't in the file.
- Please do not make any manual changes to the output files

Q4 [35 points] Analyzing a Large Graph using Hadoop on Microsoft Azure

VERY IMPORTANT: Use Firefox or Chrome in incognito/private browsing mode when configuring anything related to Azure (e.g., when using Azure portal), to prevent issues due to browser caches. Safari sometimes loses connections.

Goal

The goal is to analyze graph using a cloud computing service - Microsoft Azure, and your task is to write a MapReduce program to compute the distribution of a graph's node degree differences (see example below). Note that this question shares some similarities with Question 1 (e.g., both are analyzing graphs). Question 1 can be completed using your own computer. This question is to be completed using Azure. We recommend that you first complete Question 1. Please carefully read the following instructions.

You will use two data files in this questions:

- small.tsv[4] (zipped as ~3MB small.zip; ~11MB when unzipped)
- large.tsv[5] (zipped as 247MB large.zip; ~1GB when unzipped)

Each file stores a list of edges as tab-separated-values. Each line represents a single edge consisting of two columns: (Source, Target), each of which is separated by a tab. Node IDs are positive integers and the rows are already sorted by Source.

```
Source        Target
0               0
0               1
```

```
1                1
1                2
2                3
```

Your code should accept two arguments upon running. The first argument (args[0]) will be a path for the input graph file, and the second argument (args[1]) will be a path for output directory. The default output mechanism of Hadoop will create multiple files on the output directory such as part-00000, part-00001, which will have to be merged and downloaded to a local directory (instructions on how to do this are provided below).

The format of the output should be as follows. Each line of your output is of the format

```
diff        count
```

where
(1) `diff` is the difference between a node's out-degree and in-degree (i.e., out-degree minus in-degree); and
(2) `count` is the number of nodes that have the value of `difference` (specified in 1).

The out-degree of a node is the number of edges where that node is the Source. The in-degree of a node is the number of edges where that node is the Target. `diff` and `count` must be separated by a tab (\t), and the lines do not have to be sorted. When the source and target is the same node, such as [0, 0] in the example, node "0" should be counted in both in-degree and out-degree.

The following result is computed based on the graph above.

```
-1         1
0          2
1          1
```

The explanation of the above example result:

Output	Explanation
`-1  1`	There are 1 nodes (node 3) whose degree difference is -1
`0  2`	There are 2 nodes (node 1 and node 2) whose degree is 0.
`1  1`	There is 1 node (node 0) whose degree difference is 1.

Hint: One way of doing it is using the mapreduce procedure twice. The first one for finding the difference between out-degree and in-degree for each node, the second for calculating the node count of each degree difference. You will have to make changes in the skeleton code for this.

In the Q4 folder of the hw3-skeleton, you will find the following files we have prepared for you:

- **src** directory contains a main Java file that you will work on. We have provided some code to help you get started. Feel free to edit it and add your files in the directory, but the main class should be called "Q4".
- ***pom.xml*** contains necessary dependencies and compile configurations for the question.

To compile, you can run the command in the directory which contains ***pom.xml***.

```
mvn clean package
```

This command will generate a single JAR file in the target directory (i.e. target/q4-1.0.jar).

Creating Clusters in HDInsight using the Azure portal

Azure HDInsight is an Apache Hadoop distribution. This means that it handles large amounts of data on demand. The next step is to use Azure's web-based management tool to create a Linux cluster. Follow the recommended steps shown here (or the full Azure's documentation here) to create a new cluster.

At the end of this process, you will have created and provisioned a New HDInsight Cluster and Storage (the provisioning will take some time depending on how many nodes you chose to create). Please record the following important information (we also recommend that you take screenshots) so you can refer to them later:

- Cluster login credentials
- SSH credentials
- Resource group
- Storage account
- Container credentials

VERY IMPORTANT: HDInsight cluster billing starts once a cluster is created and stops when the cluster is deleted. To save the credit, you'd better to delete your cluster when it is no longer in use. You can find all clusters and storages you have created in "All resources" in the left panel. Please refer here for how to delete an HDInsight cluster.

Uploading data files to HDFS-compatible Azure Blob storage

We have listed the main steps from the documentation for uploading data files to your Azure Blob storage here:

1. Follow the documentation here to install Azure CLI.
2. Open a command prompt, bash, or other shell, and use `az login` command to authenticate to your Azure subscription. When prompted, enter the username and password for your subscription.
3. `az storage account list` command will list the storage accounts for your subscription.
4. `az storage account keys list --account-name <storage-account-name> --resource-group <resource-group-name>` command should return "key1" and "key2". Copy the value of "key1" because it will be used in the next steps.
5. `az storage container list --account-name <storage-account-name> --account-key <key1-value>` command will list your blob containers.
6. `az storage blob upload --account-name <storage-account-name> --account-key <key1-value> --file <small or large .tsv> --container-name <container-name> --name <new-blob-name>/<small or large .tsv>` command will upload the source file to your blob storage container. `<new-blob-name>` is the folder name you will create and

where you would like to upload tsv files to. If the file is uploaded successfully, you should see “Finished [#####] 100.0000%”.

Using these steps, upload ***small.tsv*** and ***large.tsv*** to your blob storage container. The uploading process may take some time. After that, you can find the uploaded files in storage blobs at Azure (portal.azure.com) by clicking on “Storage accounts” in the left side menu and navigating through your storage account (<Your Storage Account> -> <”Blobs” in the overview tab> -> <Select your Blob container to which you’ve uploaded the dataset> -> <Select the relevant blob folder>). For example, “jonDoeStorage” -> “Blobs” -> “jondoecluster-xxx” -> “jdoeSmallBlob” -> “small.tsv”. After that write your hadoop code locally and convert it to a jar file using the steps mentioned above.

Uploading your Jar file to HDFS-compatible Azure Blob storage

Azure Blob storage is a general-purpose storage solution that integrates with HDInsight. Your Hadoop code should directly access files on the Azure Blob storage.

Upload the jar file created in the first step to Azure storage using the following command:

```
scp <your-relative-path>/q4-1.0.jar <ssh-username>@<cluster-name>-ssh.azurehdinsight.net:
```

You will be asked to agree to connect by typing “yes” and your cluster login password. Then you will see q4-1.0.jar is uploaded 100%.

SSH into the HDInsight cluster using the following command:

```
ssh <ssh-username>@<cluster-name>-ssh.azurehdinsight.net
```

`<ssh-username>` is what you have created in step 10 in the flow.
Note: if you see the warning - REMOTE HOST IDENTIFICATION HAS CHANGED, you may clean /home/<user>/.ssh/known_hosts” by using the command `rm ~/.ssh/known_hosts`. Please refer to host identification.

Run the `ls` command to make sure that the `q4-1.0.jar` file is present.

To run your code on the small.tsv file, run the following command:

```
yarn jar q4-1.0.jar edu.gatech.cse6242.Q4 wasbs://<container-name>@<storage-account-name>.blob.core.windows.net/<new-blob-name>/small.tsv wasbs://<container-name>@<storage-account-name>.blob.core.windows.net/smalloutput

Command format: yarn jar jarFile packageName.ClassName dataFileLocation outputDirLocation
```

Note: if “`Exception in thread "main" org.apache.hadoop.mapred.FileAlreadyExistsException...`” occurs, you need to delete the output folder and files from your Blob. You can do this at portal.azure.com. Click “All resources” in the left panel, you will see you storage and cluster. Click your storage, then “Blobs” and then your container, you will see all folders including the blob created by you and the “smalloutput” folder. You need to click “Load more” at the bottom to see all folders/files. You need to delete output at different places: 1. smalloutput; 2. user/sshuser/*

The output will be located in the directory: `wasbs://<container-name>@<storage-account-name>.blob.core.windows.net/smalloutput`

If there are multiple output files, merge the files in this directory using the following command:

```
hdfs dfs -cat wasbs://<container-name>@<storage-account-name>.blob.core.windows.net/smalloutput/* > small.out

Command format: hdfs dfs -cat location/* >outputFile
```

Then you may exit to your local machine using the command:

```
exit
```

You can download the merged file to the local machine (this can be done either from Azure Portal or by using the scp command from the local machine). Here is the scp command for downloading this output file to your local machine:

```
scp <ssh-username>@<cluster-name>-ssh.azurehdinsight.net:/home/<ssh-username>/small.out <local directory>
```

Using the above command from your local machine will download the small.out file into the local directory. Repeat this process for large.tsv. Make sure your output file has exactly two columns of values as shown above. Your output files should be able to be opened and readable in text editors like notepad++.

Deliverables

1. [15pt] **Q4.java & q4-1.0.jar:** Your java code and converted jar file. You should implement your own map/reduce procedure and should not import external graph processing library.
2. [10pt] **small.out**: the output file generated after processing small.tsv.
3. [10pt] **large.out**: the output file generated after processing large.tsv.

Q5 [10 points] Regression: Automobile price prediction, using Azure ML Studio

Note: Create and use a free workspace instance on Azure Studio instead of your Azure credit for this question. Please use your Georgia Tech username (e.g., jdoe3) to login.

Goal

The main purpose of this question is to introduce you to Microsoft Azure Machine Learning Studio and familiarize you with its basic functionality and typical machine

learning workflow. Go through the "Automobile price prediction" tutorial and complete the tasks below.

You will modify the given file, **results.csv**, by adding your results for each of the tasks below. We will autograde your solution, therefore DO NOT change the order of the questions or anything else. Report the exact numbers that you get in your output, DO NOT round the numbers.

1. [3 points] Repeat the experiment mentioned in the tutorial and report the values of the metrics as mentioned in the '*Evaluate Model*' section of the tutorial.
2. [3 points] Repeat the same experiment, change the '*Fraction of rows in the first output*' value in the split module to 0.8 (originally set to 0.75) and report the corresponding values of the metrics.
3. [4 points] Evaluate the model with the 5-fold cross-validation (CV), select the parameters in the module *'Partition and sample'* (Partition and Sample) (see figure below). Report the values of Root Mean Squared Error (RMSE) and Coefficient of Determination for each fold. (1st fold corresponds to fold number 0 and so on).

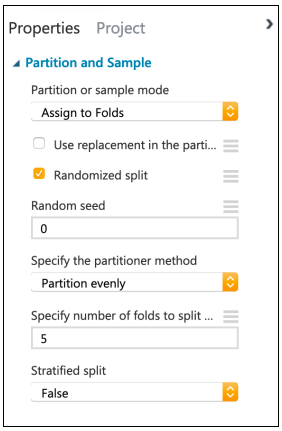


Figure: Property Tab of Partition and Sample Module

Specifically, you need to do the following:

A. Import the entire dataset (Automobile Price Data (Raw))
B. Clean the missing data by removing rows that have any missing values
C. Partition and Sample the data.
D. Create a new model (Linear Regression)
E. Finally, perform cross validation on the dataset.
F. Visualize/Report the values.

Deliverables

1. [10pt] **results.csv:** a csv file containing results for all of the three parts.

Important: folder structure of the zip file that you submit

You are submitting a single zip file HW3-GTUsername.zip (e.g., HW3-jdoe3.zip, where "jdoe3" is your GT username), which must unzip to the following directory structure (i.e., a folder "HW3-jdoe3", containing folders "Q1", "Q2", etc.). The files to be included in each question's folder have been clearly specified at the end of each question's problem description above.

```
HW3-GTUsername/
    Q1/
        src/main/java/edu/gatech/cse6242/Q1.java
        pom.xml
        run1.sh
        run2.sh
        q1output1.tsv
        q1output2.tsv
        (do not attach target directory)
    Q2/
        q2.dbc
        q2.scala
        q2_results.csv
    Q3/
        pig-script.txt
        pig-output.txt
    Q4/
          src/main/java/edu/gatech/cse6242/Q4.java
          pom.xml
          q4-1.0.jar (from target directory)
          small.out
          large.out
       (do not attach target directory)
    Q5/
        results.csv
```

Version 0

---

[1] Graph1 is a modified version of data derived from the LiveJournal social network dataset, with around 30K nodes and 320K edges.
[2] Graph2 is a modified version of data derived from the LiveJournal social network dataset, with around 300K nodes and 69M edges.
[3] Graph derived from the Stanford Large Network Dataset Collection
[4] subset of Pokec social network data
[5] subset of Friendster data