Project 1: Hadoop

This project will consist of three parts. Students are expected to submit all three parts in their git repos. Please note team sizes can be at most 2 students.

Updates:

- 09/24/2018: Not much time to go, we're nearing the Final Countdown!
- 09/20/2018: For Part B: the output files for the gutenberg and yelp dataset are too large to fit everyone's data into GitHub. Instead, we are slightly modifying the hand-in directions for the Gutenberg and Yelp data. Instead of turning in all of the part-*** files, **we'll instead find the top 2000 most frequent words in the yelp and gutenberg dataset**. The top 2000 words should be sorted by frequency (ties should be sorted alphabetically), and the data format of your final part-** file should look like:
    - <word1><tab><frequency> //omit the brackets
    - <word2><tab><frequency> //and so on

  Just as before, ensure the final part-*** file is put into a file called output.tar.gz (tar and gzip the part-** file). The output.tar.gz file for yelp and gutenberg should decompress to a single file with your answer. You should have one output file for yelp and one output file for gutenberg. The IMDB and Sherlock Holmes instructions will remain the same: output.tar.gz will contain multiple part-*** files with all of the word counts. **Please do not check-in large output files to github. If you've already checked-in yelp and gutenberg, please remove the large output.tar.gz from your github repo.** Please post to Piazza if any questions. Apologies for the inconvenience.
- 09/19/2018: For Part B, we have created two S3 buckets to make processing easier. You are allowed to use S3, but must still use the command line interface. The two datasets you can use are:
    - Gutenberg: s3://csci5253-gutenberg-dataset
    - Yelp: s3://wordcount-datasets
    - Students should perform wordcount on the whole bucket. For Gutenberg, we've concatenated many of the small files into a few small (but large) input files. Hadoop works much more efficiently on large input files: processing for Gutenberg took about an hour in Surya's setup.
    - Please post any questions on Piazza.
    - How did we get the large dataset on S3? Eric created an EC2 instance with enough local hard-disk space. Then he decompressed the 7zip file and untarred. Surya concatenated the files. From there, we can use the "aws s3" tools to copy from local filesystem to S3.
- 09/17/2018: For the stopword list in Part B, remove all non-alphabetic characters from the stopwords before removing stopwords from the dataset. Please see the Piazza thread "**Clarification on Part B Word Mapping**" for more details.

- 09/16/2018: For Part C, apparently two items will have ratios > 1
- 09/12/2018: Github classroom instructions added to bottom of document.
- 09/12/2018: AWS EMR should work for Amazon Educate Starter accounts!

Part A [20% of grade]

Objective: Learn AWS and learn about AWS EMR.

We will try to use a lot of Amazon service for the course. As a result, think of Part A as a "Hello World" into AWS, as well as a very simple introduction to EMR-- Amazon's managed Hadoop service. This part of the project is very easy: simply go to the link below and follow the tutorial: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html

What to turn in?

- Create a directory named "PartA" in your git repo (keep the case exactly the same as noted, do not include quotation marks in the directory name).
- Hand in the output file you obtain after running through Step 4. Name the file "000000_0" and put it in the PartA directory.
- Note: if for some reason you cannot spin up an Amazon EMR instance, then please read through all of the directions anyway. The assignment is a bit trivial: the "answer" is already located at the end of the Step 4 description.
    - If your Amazon EMR works, please actually go through all steps and don't just steal the answer!
    - **UPDATE (9/12/2018):** Some students are reporting this works with an Educate Starter account. Everyone please try to do this!

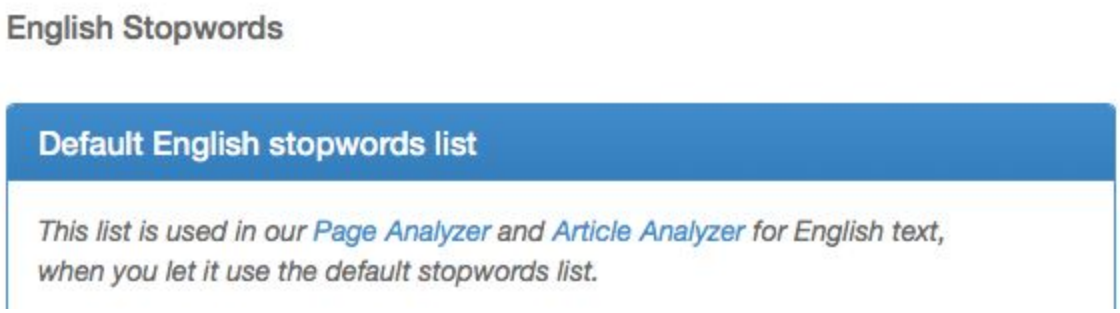
Part B [40% of grade]

Objective: Learn about using native Hadoop command-line interface, and also dip your toes into Hadoop programming.

We will perform a simple word count on a few different datasets. The goal is to learn how to use Hadoop through the command line, and also become familiar with Hadoop programming. For word count, the turned-in Hadoop program must ingest a large text corpus (which will be defined by us) and return how often each word appeared in the corpus. This should be done for each corpus individually, and not combined over all corpuses. The word count program must satisfy the following constraints:

- Text will be noisy: there will be some "words" (more accurately tokens parsed from the data) that are actual English words, and other "words" that are dates, numbers, alphanumeric values (e.g., maybe hex), or other weird formatting (say a bunch of "#"

symbols). The first constraint is to remove all non-alphabetic characters for any given input word. This means that “abc123” should be counted as “abc”. And “456abc” should also be counted at “abc”. So remove all numbers, punctuation, and any other character that is not [a-z] or [A-Z]. A few more examples:
    - “#4a90ct” → “act”
    - “Turtle,” → “Turtle”
    - “Base-ball” → “Baseball”
    - “123” → “” //special case, see below
    - “Who’s” → “Whos”
    - “(baby)” → “baby”

  All “words” that are converted to the empty string should be ignored. Said another way, the empty string should not be in your output at all.
- All output should be lower-case. This means that “Dog” will map to the same word as “dog”, which also maps to the same word as “DOG”. All output words must be in lower-case.
- Stop words should not appear in the final output.
    - None of the “Default English stopwords list” from the following page should appear in the final output:
        - https://www.ranks.nl/stopwords
        - 

      Examples of stop words are “a”, “the”, “had’, etc. (note: use the list above)

The student’s code must be run over four different datasets:

- The Project Gutenberg Library (note: the download file is 6 GB, and 70 GB of disk space is required when uncompressed). Visit the following page:
  http://www.gutenberg-tar.com
  And download the gutenberg_txt.7z file from the link at the very top.
- IMDB quotes page. Go to the following link:
  http://ftp.sunet.se/mirror/archive/ftp.sunet.se/pub/tv+movies/imdb/
  And download the quotes.list.gz file (Note: 67M compressed)
- The Adventure of Sherlock Holmes (6 MB uncompressed). Download the file here:
  https://norvig.com/big.txt
- Yelp reviews data (3 GB uncompressed):
  https://drive.google.com/open?id=1ih-xNA5h5CpPLz3pI8IN1iVnTdR4LhG0
  Note: you will have to be logged in with you CU Google Suite to access the above file.

Tips:

- Try small, known files first
- You can get a Hadoop cluster running locally if you want to debug/try on just a single machine first. Only do local runs with small data.
- Create a bunch of corner cases (weird words, formatting, etc) to make sure your code can handle all of the above constraints.
- Some useful links to SSH into an EMR cluster:
    - https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs-ssh.html
    - https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-ssh.html

What to turn in:

- **Update 9/20: See update at top of page for modified yelp and gutenberg hand-in instructions. IMDB and Holmes directions will stay the same.**
- Create a directory named “PartB” in your git repo.
- Create four subdirectories in PartB: “gutenberg”, “imdb”, “holmes”, and “yelp”. Make sure the case is correct!
- For a given dataset (say gutenberg), put all of the “part-*” output files that you get from Hadoop into a single file named “output.tar.gz”, which is tar’d and gzip’d. When you uncompress the output.tar.gz file, only the “part-*” files should be listed (in other words, do not include the output directory in your tarball). So you should have four files to turn in with data:
    - PartB/gutenberg/output.tar.gz
    - PartB/imdb/output.tar.gz
    - PartB/holmes/output.tar.gz
    - PartB/yelp/output.tar.gz
- Create a directory named “PartB/Java” and put in the following:
    - Your *.jar file, which should be named wordcount.jar
    - All source code (any *.java files and anything else needed to compile your code).
    - A Maven file (named pom.xml) to compile your code.
    - A file, named “run.sh”, that is put in each sub-directory (create the following: PartB/Java/gutenberg, PartB/Java/imdb, PartB/Java/holmes, PartB/Java/yelp) that contains *all* commands to:
        - Import data and copy to HDFS
        - Run Hadoop analysis
        - Output result

      Change the permissions of run.sh to “a+x” (everyone can execute). The TA should be able to run the “run.sh” script to verify your program works and it can generate the output you turn in. In other words, the TA should be able to download your github repo to his Amazon machine (which is running an EMR cluster) and type “./run.sh” into the command line after cd’ing into the appropriate

github directory. The code should run through all steps necessary to obtain the output. You can assume the input files are located in an "input/" subdirectory within the the current directory, one for each input above (i.e., the data is already put into the directories "PartB/Java/gutenberg/input", "PartB/Java/imdb/input", …) but **do not check in the input files to your github repo!**

- A "PartB/Readme.md" file (in the base directory) documenting your project with any special instructions or comments.

Part C [40% of grade]

Objective: Slight push into Hadoop.
We will analyze some click and buy events from user data from an e-commerce website in 2014. Information about the data we are using can be found here: http://recsys.yoochoose.net/challenge.html

Dataset: Download the following files (Note: log in to Google-suite with your CU credentials to gain access):

- buy.txt
  - https://drive.google.com/open?id=1kLR92O38aqSKi8Oe9Y5L95xORziy1tis
- clicks.txt
  - https://drive.google.com/open?id=19SLL4FXN71bp1616fl6TVid7zIY7i4JE

You can refer to the webpage for the file format. The original files were in a *.dat format, but the above links have been converted to text files for your convenience. Complete the following three tasks:

- **ItemClick**: What items were the most clicked for the month of April? Your output should contain a single file, with a ranked list of the top 10 item numbers. The first row should be the most clicked item, the second row should be second most-clicked item, etc.
  - An example file format for a row would look like:
    - #itemID (\tab) frequency

    In other words, the output should be TAB delimited (no other whitespace in a row), with the first column being the item number, and the second column being how many times that item number was clicked.

- **TimeBlocks**: break the day into one hour time blocks. Use 24-hour format to represent hours, with two digits per hour. So "00" represents 12:00AM to 12:59AM, and "01" represents 1:00AM to 1:59AM. Furthermore, "13" represents 1:00PM to 1:59PM, and "23" represents 11:00PM to 11:59PM.

  For each hour time block, compute the total revenue generated for that time of day. Note that in the buy.txt file, there is a quantity field, so total revenue for a given purchase is

price*quantity.

Do this over the whole dataset and sort the output by revenue, with the highest hour in the first row and the lowest hour last in the last row. An example row format:

- Hour (\tab) Revenue

So one example row may look like “05 2349223”

- **Success Rate**: For all items, determine their success rate. If a user clicks on an item, and then buys the item, the purchase is deemed a success. If the user clicks an item and doesn’t buy it, then it is not a success. The success rate for a given item is defined by:
  - [total_transactions_with_item_purchased] / [total_clicks_on_item]

  Note the numerator is a bit tricky: a user may purchase multiple quantities of a given item in a single purchase. We count this as a single transaction. In other words, the success rate for an item over all purchases is always <= 1. (Say a very unpopular item was only bought by one user, but that user bought a quantity of 2 of the item: then the success rate would be “1” and not “2”). Assume that a user had to click on an item before purchasing it. Turn in the top 10 success rate items, with the first row being the most successful item, and the last row being the least successful item. Ties should be broken by numerical sort (so item “1” would appear before item “2”).
  UPDATE (9/16): *For Part C, apparently two items will have success rate > 1.*

What to turn in?

- Create a “PartC” directory in your git repo. Put everything below within that directory.
- Create three subdirectories in PartC: ItemClick, TimeBlocks, and SuccessRate
- Put the output into a file named “output.txt” for each subdirectory
- Create a “Java” subdirectory in each of the above subdirectories:
  - For example: “PartC/ItemClick/Java”, etc
  - Fill in the “Java” subdirectory just as we did with Part B. In other words, all your Java files, your Maven file, a run.sh file, etc. Name each *.jar file as one of: ItemClick.jar, TimeBlocks.jar, or SuccessRate.jar
- **Do not check-in** the buy.txt and click.txt files into your git repo!

Additional turn-in files:

- In the root of your github folder, create a file called “Names.txt”. In Names.txt, each row should contain the following information for each group member (one row per group member):
  - FullStudentName (\tab) CU email address (\tab) IdentiKey

Notes:

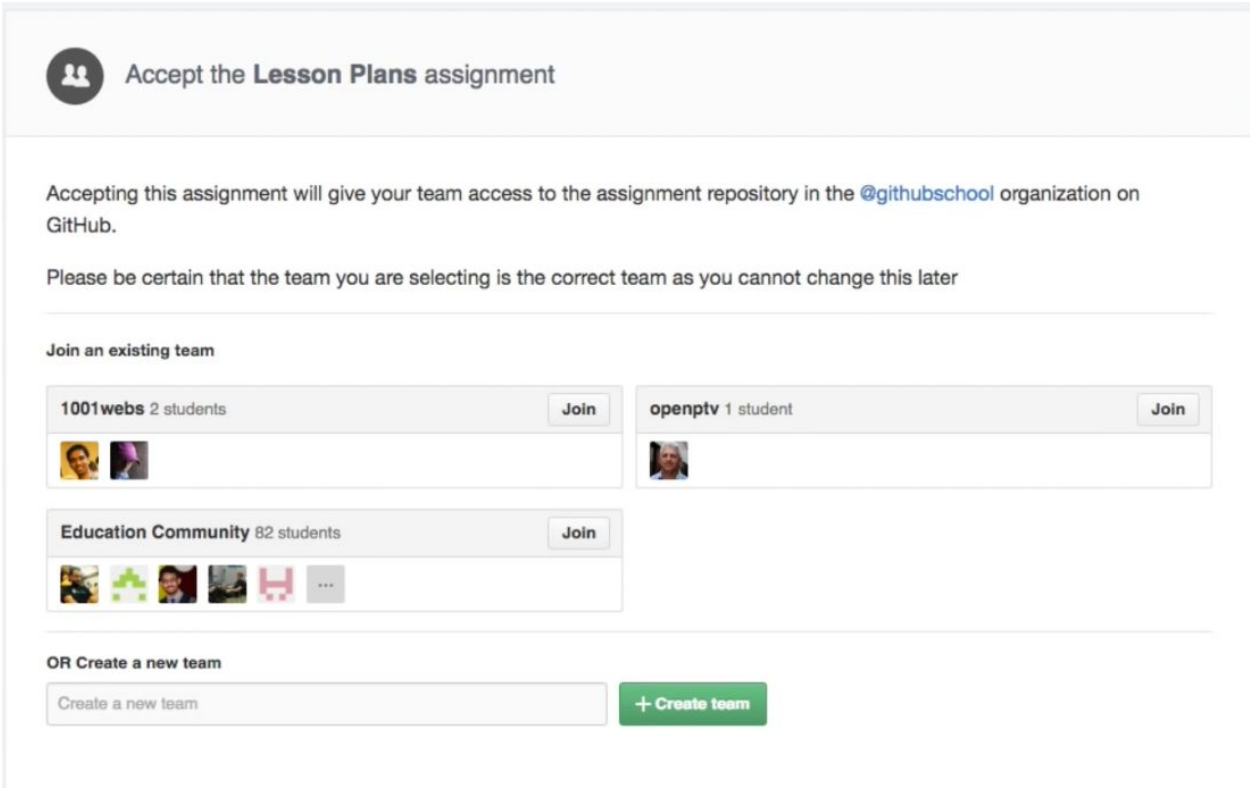
- Question: Why are the instructions so crazy? Does directory names and case sensitivity really matter?
    - Answer: Yes! We need to script the grading. Everything needs to be to spec.
- Check-in your code early and often.
- Remember that we have Google Cloud credits, Cloudlab.us resources, and you can use your local machine. Check moodle for slides on joining Cloudlab.
- **Remember to close all VMs, clusters, etc when not using them!**
- If you have an Amazon AWS Educate Starter account, please try Step 1 (Brett said he was now able to get it working, and the instructors have different accounts so we can't verify).
- **All questions** should be posted publicly on Piazza. Students are encouraged to help one another. Students helping often will get extra credit towards their participation grade at the end of the semester. Remember to abide by all Conduct guidelines (ie: be polite).
- Remember the Academic Integrity policy from the course! Don't copy code without attribution. Clearly document what parts were copied. Try to be cool and not copy anything though. Writing from scratch will be useful later on.
- Apologies for the AWS EMR mix-ups! As stated during the first class day, this is Eric's first time teaching the course and a few hiccups are to be expected.
- I've created Piazza threads for distance learners and in-class students to find partners. No groups of three allowed without explicit permission. Single students allowed, but not encouraged.
- Let Eric know ASAP if there are any issues with the working relationships in your group.
- **For each git check-in,** write a comment on who was responsible for what parts of the commit. This will help create a paper trail to deal with any issues that may arise. If two students sat side-by-side to code parts of the program, that should be explicitly noted in the git comment for the commit.
- May the force be with you.

Turn in instructions:

We are using Github classroom to turn in the project, and also to check project progress by team members. The first step is creating your team. To do so, **one** of your team members should follow this link:

https://classroom.github.com/g/MKmZE5Ga

The page will eventually come to an "Accept the assignment" page, like this:



The **first** student on the team should use the “Create a new team” function. **The team name must be formatted as follows: LASTNAMEMEMBER1-LASTNAMEMEMBER2.** So, for example, if Eric Rozner was partnering with Kamal Chaturvedi, then the team name would be:

Rozner-Chaturvedi

The ordering of names doesn’t matter, but both last names must be entered. Students working alone (not recommended), should just call their team name their last name. **Students working alone will be required to post on a Piazza thread (created on Sept 12 by Eric Rozner with the title “Project 1 Solo Checkin”) that they are working alone so we can set some additional permissions.**

Enter the team name and hit the “Create team” button. It will take some time for the team to be created. After creation, the **second team member** must join the Project assignment link:

https://classroom.github.com/g/MKmZE5Ga

Then, the second team member should hit the “Join” button next to the team name that the first team member created. If there are any issues with team creation, please post to Piazza.

Some good github documentation:

- Merging conflicts
  - https://services.github.com/on-demand/merge-conflicts/
- Github guides:
  - https://guides.github.com

- Git “Hello World”
    - https://guides.github.com/activities/hello-world/
- Git ebook (free)
    - https://www.amazon.com/Rys-Git-Tutorial-Ryan-Hodson-ebook/dp/B00QFIA5OC

Due date:

- Tuesday, Sept 25 at 11:59:59 PM MST
- Please start early-- assume hiccups will occur!