Project 2: Spark & MongoDB

Objective: The objective of this assignment is for students to gain some hands-on experience with Spark and MongoDB.

This project will consist of two parts. Students are expected to submit both parts in their git repos. Please note team sizes can be at most 2 students. ***If students have trouble with their partners (from Project 1 or otherwise), they should privately email the instructor (Eric) ASAP. Otherwise, assume the same project pairings for this project. Some groups have already asked for reassignment, so if you want reassignment please let me know.*** Due to logistical reasons, we cannot accommodate requests to be paired with a specific person at this time (sorry!). Also please understand that we can reasonably create new pairs early-on, but our ability to do so becomes less as the deadline nears.

Updates:

- **Oct 8**: please see Piazza for new MongoDB address:
    - http://ec2-54-210-44-189.compute-1.amazonaws.com/test
- Please reference this document frequently for updates.

Programming languages:

Spark is supported by a variety of languages. We will allow for students to use a programming language of their choice, but **we will only officially support python.** We will try to help students with Scala, Java, etc, but cannot guarantee any troubleshooting or debugging. We'll focus on each project's code functionality (does it work) and correctness for the grading.

Part A [30% of grade]

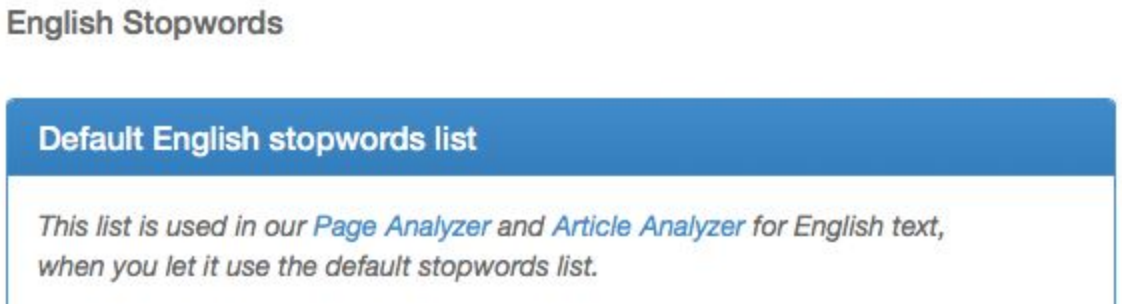
Objective: learn Spark framework in AWS.
We will *mostly* implement Part B from Project 1 in Spark. Project 1 pointer:

- https://docs.google.com/document/d/1rnymrIav1gOYxte47bVNtQc2vtStT-9POZKIrrS7AU0/edit?usp=sharing

In more detail, we will perform a simple word count on a few different datasets. The word count program must satisfy the following constraints:

- **One clarification:** any whitespace (tab, space, newline,etc) should be a delimiter to separate two words from one another.
- All workloads should use the **command line** (you can use the GUI for testing, debug, and sanity check if needed-- note we will not support the GUI)

- 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 (remember to remove non-alphabetic characters from the stop words):
        - 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 two different datasets:

- 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
- **Note:** No need to do Yelp or Gutenberg this time, wahoo!

For all datasets, **we'll find the top 2000 most frequent words in each dataset**. The top 2000 words should be sorted by frequency (ties should be sorted alphabetically: so when there is a tie, the word "aardvark" would appear before the word "dog"), and the data format of your final output file should look like:

- <word1><tab><frequency> //omit the brackets
- <word2><tab><frequency> //and so on

Ensure the final output file is put into a file called output.txt. You should have one output file for each dataset.

References for Spark on AWS:

- https://aws.amazon.com/emr/details/spark/
- https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark.html
- If you are running Python 3, this might be useful (it appears Python 2.7 is default):
  - https://aws.amazon.com/premiumsupport/knowledge-center/emr-pyspark-python-3x/
    - Any version of python is fine for hand-in

What to turn in:

- Create a directory named "PartA" in your git repo.
- Create 2 subdirectories in PartA: "imdb" and "holmes". Make sure the case is correct!
- For a given dataset (say imdb), put the single output file that you get into a file named "output.txt". So you should have 2 files to turn in with data:
  - PartA/imdb/output.txt
  - PartA/holmes/output.txt

- Create a directory named "PartA/Code" and put in the following:
  - All source code (and anything else needed to compile your code).
  - *(If required)* A Maven file (named pom.xml) to compile your code.
  - Create two subdirectories: PartA/Code/imdb, PartA/Code/holmes and in each put:
    - A file, named "run.sh" that contains all commands to:
      - Import data and copy to HDFS
      - Run Spark 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. In

other words, you can assume “input” holds the imdb data when we run the imdb test, and “input/” holds the holmes data when we run the holmes test. **Do not check in the input files to your github repo!**

- A “PartA/Readme.md” file (in the base directory) documenting your project with any special instructions or comments. **Document which language you used. Please note the version of python you are using if you are using python.**

Part B [70% of grade]

Objective: Learn about MongoDB, some Spark integration with datastore
For this project, we’ll use some data from Amazon. All datasets can be downloaded from this page (*but students should not need to download*):

- http://jmcauley.ucsd.edu/data/amazon/links.html

References for MongoDB:

- General reference:
  - https://docs.aws.amazon.com/quickstart/latest/mongodb/overview.html
- How to connect MongoDB to Spark:
  - https://docs.mongodb.com/spark-connector/master/python-api/

We have populated a *subset* of the datasets below into a MongoDB instance that should be accessible to all (so students don’t need to download from the link above):

mongo -u student -p student ec2-54-173-174-196.compute-1.amazonaws.com/test
ServerAddress: ec2-54-173-174-196.compute-1.amazonaws.com **(10/8: see Update note!)**
Username: student
Password: student
DB name: test
Collections: reviews(11,933,575 documents), metadata (880,025 documents)

- **5-core** (9.9gb) - subset of the data in which all users and items have at least 5 reviews (41.13 million reviews) *[Eric’s note: we will work*

with a subset of this data, see below]

Sample review:


```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.
He is having a wonderful time playing these old hymns.  The music  is
at times hard to read because we think the book was published for
singing from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```


where

- `reviewerID` - ID of the reviewer, e.g. A2SUAM1J3GNN3B
- `asin` - ID of the product, e.g. 0000013714
- `reviewerName` - name of the reviewer
- `helpful` - helpfulness rating of the review, e.g. 2/3
- `reviewText` - text of the review
- `overall` - rating of the product
- `summary` - summary of the review
- `unixReviewTime` - time of the review (unix time)
- `reviewTime` - time of the review (raw)

- Metadata includes descriptions, price, sales-rank, brand info, and co-purchasing links: **metadata** (3.1gb) - metadata for 9.4 million products *[Eric's note: we will work with a subset of this data, see*

below]

Sample metadata:


```
{
  "asin": "0000031852",
  "title": "Girls Ballet Tutu Zebra Hot Pink",
  "price": 3.17,
  "imUrl": "http://ecx.images-
amazon.com/images/I/51fAmVkTbyL._SY300_.jpg",
  "related":
  {
    "also_bought": ["B00JHONN1S", "B002BZX8Z6", "B00D2K1M3O",
"0000031909", "B00613WDTQ", "B00D0WDS9A", "B00D0GCI8S", "0000031895",
"B003AVKOP2", "B003AVEU6G", "B003IEDM9Q", "B002R0FA24", "B00D23MC6W",
"B00D2K0PA0", "B00538F5OK", "B00CEV86I6", "B002R0FABA", "B00D10CLVW",
"B003AVNY6I", "B002GZGI4E", "B001T9NUFS", "B002R0F7FE", "B00E1YRI4C",
"B008UBQZKU", "B00D103F8U", "B007R2RM8W"],
    "also_viewed": ["B002BZX8Z6", "B00JHONN1S", "B008F0SU0Y",
"B00D23MC6W", "B00AFDOPDA", "B00E1YRI4C", "B002GZGI4E", "B003AVKOP2",
"B00D9C1WBM", "B00CEV8366", "B00CEUX0D8", "B0079ME3KU", "B00CEUWY8K",
"B004FOEEHC", "0000031895", "B00BC4GY9Y", "B003XRKA7A", "B00K18LKX2",
"B00EM7KAG6", "B00AMQ17JA", "B00D9C32NI", "B002C3Y6WG", "B00JLL4L5Y",
"B003AVNY6I", "B008UBQZKU", "B00D0WDS9A", "B00613WDTQ", "B00538F5OK",
"B005C4Y4F6", "B004LHZ1NY", "B00CPHX76U", "B00CEUWUZC", "B00IJVASUE",
"B00GOR07RE", "B00J2GTM0W", "B00JHNSNSM", "B003IEDM9Q", "B00CYBU84G",
"B008VV8NSQ", "B00CYBULSO", "B00I2UHSZA", "B005F50FXC", "B007LCQI3S",
"B00DP68AVW", "B009RXWNSI", "B003AVEU6G", "B00HSOJB9M", "B00EHAGZNA",
"B0046W9T8C", "B00E79VW6Q", "B00D10CLVW", "B00B0AVO54", "B00E95LC8Q",
"B00GOR92SO", "B007ZN5Y56", "B00AL2569W", "B00B608000", "B008F0SMUC",
"B00BFXLZ8M"],
    "bought_together": ["B002BZX8Z6"]
  },
  "salesRank": {"Toys & Games": 211836},
  "brand": "Coxlures",
  "categories": [["Sports & Outdoors", "Other Sports", "Dance"]]
}
```


where

- `asin` - ID of the product, e.g. 0000031852
- `title` - name of the product
- `price` - price in US dollars (at time of crawl)
- `imUrl` - url of the product image
- `related` - related products (also bought, also viewed, bought together, buy after viewing)
- `salesRank` - sales rank information
- `brand` - brand name
- `categories` - list of categories the product belongs to

There are 2 parts to Part B, each 50% of the grade (0.1% of the grade will be given if the student checks in a poem to at least one git check-in… just kidding, but still feel free to check in a poem). See below for each part.

Due to the large size of the dataset, we have picked a subset of categories to import in our MongoDB instance. As a result, when we say “over all categories”, we mean over all categories that are in our MongoDB instance. The categories we have selected are:

- Movies & TV
- CDs & Vinyl
- Video Games
- Toys & Games

- **[Part1]:** For each category, find highest rated item of all items that have at least 100 reviews. Note the categories should be defined in the dataset as one of the following:
  - **Movies & TV, CDs & Vinyl, Video Games, Toys & Games**

  The output should look like the following:
  - [Category][tab][Item title][tab][number of reviews][tab][average user rating]
  - Note that “Item title” is the “title” field in the metadata table (in other words, the actual English name, not the item number).
  - Sort the categories alphabetically in the output.
  - Name your file output.txt (see below for directory information)
  - **Note**: Notice from the above image that the “categories” field for each item has multiple categories. Therefore, if an item has multiple categories, it should be counted in each category (of the categories that are in our dataset, categories outside our dataset should be ignored)
- **[Part2]:** For every individual review, we are going to bucket each of them by its review score. So your code should take each product review and logically assign it to one of the following buckets:
  - Bucket 1: review score of 1 (where score is the “overall” field in each review)
  - Bucket 2: review score of 2
  - Bucket 3: review score of 3
  - Bucket 4: review score of 4
  - Bucket 5: review score of 5

  As an example, if an item was given a rating of 3 by a user, then we’d put that review into bucket 3. If *the same item* was given a review of 5 by another user, then we’d put that second review into bucket 5. Complete this for all reviews. When complete, a bucket should contain any user review that received the bucket’s score. At the end, output the top 500 words (same stop words removed as in Part A) from all aggregated “reviewText” fields for all reviews in a given bucket. For example, for the Bucket 1 output, the top 500 words appearing over all “reviewText” fields from all reviews appearing in Bucket 1. Do the same for Bucket 2, etc. Just as before, remove all non-alphabetic characters for stopwords and input words. Turn-in should contain one file for each bucket:
  - PartB/Part2/Bucket1/output.txt
  - PartB/Part2/Bucket2/output.txt
  - PartB/Part2/Bucket3/output.txt
  - PartB/Part2/Bucket4/output.txt

        - PartB/Part2/Bucket5/output.txt

    The format of the output file should be:

        - <Word 1><tab><frequency>
        - Sort by frequency first (highest frequency words appears first), and break ties by alphabetic sort (first word alphabetically comes first).

What to turn in:

- Create a directory named “PartB” in your git repo.
- Create 2 subdirectories in PartB: “Part1” and “Part2”. Make sure the case is correct!
    - Put answers in each subdirectory
    - Put all code, anything necessary to run/compile your code in each subdirectory
    - Include a “run.sh” file for each subdirectory, that has commands to:
        - (*Optionally*) Compile code
        - Run Spark 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 a Spark 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.
- A “PartB/Readme.md” file (in the base directory) documenting your project with any special instructions or comments. Add some poems here.

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. Document who has done what with each check-in!
- 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!**

- **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.
- 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 odds be ever in your favor!

Turn in instructions:

We are using Github classroom to turn in the project, and also to check project progress by team members. **The team name must be formatted as follows: project-2-spark-LASTNAMEMEMBER1-LASTNAMEMEMBER2.** So, for example, if Eric Rozner was partnering with Kamal Chaturvedi, then the team name would be:

project-2-spark-Rozner-Chaturvedi

Link to join the project 2 assignment via github classroom:

- https://classroom.github.com/g/hO35mE9E

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:

- Tues, Oct 9 at 11:59:59 PM MST
- Please start early-- assume hiccups will occur! Students have already had issues with AWS, Vocareum, etc. Do not wait until the last minute to work on the project! Students with cloud issues *will not receive an extension!*