EECS 476: Data Mining (Winter 2021)

Project 1
Data preprocessing & Frequent Items

DUE: Feb. 19th, 2021, 11:50 pm

Instructions

- This project has to be done individually.
- The project (Part1 & Part2) is **due at 11:50 pm on Feb. 19th, 2021**.
- We allow 5 total late days across all assignments throughout the semester to accommodate extreme circumstances. For each 24-hour period a 5% penalty will be applied. You will have 10 minute grace period, but beyond that the corresponding penalty will be applied. Late days are rounded up to the nearest integer. For example, 11 minutes late means 5% off, 1 day and 11 minutes late means 10%, 2 days and 11 minutes or 3 days late means 15% off, and so on. Submissions after the 5 late days will get a zero.
- The project includes two parts: (1) **code** for a specific task and (2) written answers to related questions. You will need to submit your code to the autograder and a PDF file containing your answers to the short-answer questions. The **structure of your submission** file should be as follows:
  1. Upload source files to the autograder at `https://autograder.io/web/courses`. The full list of submission files is:
     - MostRated.zip
     - PopularGenre.zip
     - TransactionFormat.zip
     - FrequentItemsets.zip
     - AssociationRules.zip

     In each zip file, please include the source code you write and the tools to build the jar file (gradle dependencies).
  2. Submit `writeup.pdf` to Canvas. The write-up should be a single pdf that includes both Part 1 and Part 2.
- Starter code can be found at:
  `https://github.com/datamining-class/Project1-Starter-code`
- Do NOT submit any data, unless we ask explicitly.
- In this project, you are asked to write MapReduce code and run it on the Great Lakes Hadoop cluster. Make sure you run your jobs on Great Lakes by setting the queue name as “eecs476”. The datasets are under `/var/eecs476w21/` in the HDFS on Great Lakes.

- Make sure that your code is independent of Great Lakes. We will run your code on the autograder on both subsets of the input data that you are given, and extra non-released data to check for correctness. **You will not receive credit if your code fails to run on the autograder.**
- We recommend that you submit your project solutions to both the autograder and canvas at least 30 minutes before the regular deadline as a safety measure. It is common to experience delays close to the submission deadlines (e.g., due to slower servers).
- There are submission instructions related to each question, make sure to carefully follow them.

- **Policy of collaboration:** All students (including LS&A and Engineering) are required to observe the Engineering Honor Code in all assignments and exams. A copy of the honor code can be found at `http://ossa.engin.umich.edu/honor-council/`. The University takes honor code violations seriously, and penalties can be severe. You may discuss the projects with your fellow students at the conceptual level, **but must complete coding and write-up, from scrap to final form, on your own**. Verbatim copying of another group's work is forbidden.
- We will run plagiarism detection software on all submitted code. Any violations will be reported.

Part 1. [35 points] Data Analysis on Hadoop

You have studied Hadoop during the lecture, you can perform some more meaningful analysis. Suppose that you are working at a movie company, NoFlix, and as a leading data scientist, you are tasked to better understand the user preferences. Your manager provided you with a dataset called `MovieLens20M` includingthree files: “ratings.csv”, ‘genres.csv” and “movies.csv”.

Data. The files are located on HDFS:

- `/var/eecs476w21/Project1/genres.csv` contains information about what genres a movie belongs to
- `/var/eecs476w21/Project1/movies.csv` translates each movieId to movieName
- `/var/eecs476w21/Project1/ratings.csv` provides the user-movie ratings and their timestamps

Table 1: MovieLens20M dataset: illustration of the file formats.

(a) Genres per movie.

`Movie ID`	`Movie Genres`
1	Romance, Comedy
2	Horror
3	Action

(b) Movie names.

`Movie ID`	`Movie Name`
1	Toy Story
2	Jumanji
3	Grumpier Old Men

(c) Movie ratings over time. The timestamp is given in the number of seconds since the epoch.

`User Id`	`Movie ID`	`Rating`	`Timestamp`
1	1	3.0	1112489921
1	2	4.0	1112480123
2	4	2.5	1112480980
3	2	3.5	1112413219

Q1. Find the Most Rated Movie Genres

1. **[10 points]** Write a MapReduce program which calculates the number of ratings for each genre over three year spans. For example, you will calculate the number of ratings of all action movies based on reviews from the years 1995-1997, 1998-2000, ... until 2013-2015. Your program should accept three command line arguments:

```
--ratingsFile: the file with the movie reviews and their timestamps

--genresFile: the file with mappings between movie ids and genres
```

```
--outputScheme: the prefix of the output files produced by your jobs
```

If there are multiple map-reduce phases in your implementation, the output directories should have as suffix natural numbers starting from 1. For instance, if the argument of –outputScheme is OUTPUT, then the output directories should be OUTPUT1, OUTPUT2, etc.

Your program should output a csv where each row is of the following format:

`<genreName,beginningOfTimespan,#GenreRatings>`

For example, "`action,1995,125`" would indicate that action movies had 125 ratings in total over the years 1995-1997.

2. **[5 points]** Run your code on `MovieLens20M` dataset. Per interval, report the most rated movie genre along with number of ratings in your write-up. You should be providing one answer for each time interval. Ties should be put into alphabetical order. That is, if two genres share the same number of ratings in a given time interval, give the genre which is closest to the beginning of the alphabet.

3. **[5 points]** Plot the the number of ratings (y-axis) of `Mystery`, `Film-Noir`, `Comedy` and `Fantasy` across the seven time intervals (x-axis). The plot should contain 4 time series, one per genre of interest.

TO SUBMIT

(a) A zip file, named `MostRated.zip`, which can be run with Hadoop. Please include source code and any dependencies needed to run the code. Submissions which do not contain source code will receive no credit.

(b) The most rated movie genres by time interval along with their number of ratings in `writeup.pdf`.

(c) The plot of question Q1.3 in `writeup.pdf`.

(d) **DO NOT** submit any data.

Q2. Find the Most Popular Movie in Each Genre

Now that you know what type of movies people watch most, let's find some examples of the most popular (i.e., highly rated) movies.

1. **[10 points]** Write a MapReduce program which produces the most highly rated (i.e., average user rating) movies for each genre over each 3 year span. That is, if there are 2 time periods, and 3 genres, you should produce 6 results, one for each time period, genre combination. Using the same command line arguments and input files, along with the additional argument:

   ```
   --movieNameFile: the file which contains a mapping from movie id
                    to movie name
   ```

   Your program should produce output:

   ```
   <beginningOfTimespan,genreName,movieScore,movieId,movieName>
   ```

   Any ties should be broken by choosing the movie with the name closest to the beginning of the alphabet.

2. **[5 points]** Report the most popular movie for each genre and time period that you reported as the most rated in Q1. That is, if you found `action` to be the most rated movie genre in the years 1995-1997, you should report this information, along with the most popular `action` movie in that period. You should report one answer for each genre reported in Q1.

To Submit

(a) A zip file, named `PopularMovies.zip`, which can be run with Hadoop. Please include source code and any dependency needed to run the code. Submissions which do not contain source code will receive no credit.

(b) The most rated genre for each time period, as well as the most popular movie of that genre and time period in `writeup.pdf`.

(c) **DO NOT** submit any data.

Part 2. [65 points] Frequent Item Sets

To gain a deeper understanding of a user's preference, you decide to use association rules and frequent itemset analysis. As soon as you start working on the task, you realize that you need to do some preprocessing and trasform the data into the 'transaction' format, where a user is followed by a list of movies that they viewed/rated.

Q1. [5pts] Data Preprocessing: Transaction Format

Write a MapReduce program that transforms an input file given in compact format (e.g., Table 1c) into a 'transaction' format. The output should be a csv `<userID, <list of rated movies>>`. Run your code using `MovieLens20M` (`ratings.csv`) as the input. Your program should accept two command line arguments:

```
--ratingsFile: the file with the movie reviews and their timestamps

--outputScheme: the prefix of the output files produced by your jobs
```

Table 2: "Transaction" format of Table 1c.

`user ID`	`movie IDs`
1	1, 2
2	4
3	2

To Submit

(a) A jar file named `TransactionFormat.zip`, which can be run with Hadoop. The jar should include any dependencies as well as source code for this portion of the project. Submissions which do not contain source code will receive no credit.

(b) **DO NOT** submit any data.

Q2. [23pts] Frequent item sets

Now that you have transformed the data to the right format, you are ready to analyze it using frequent itemset algorithms.

1. **[15pts]** Write a MapReduce program that finds sets of frequently watched movies of size k, for a given support s. The input should be the output of Q1, a csv file of the format shown in Table 2. Your program should accept command line arguments:

```
-k: indicates the length of the output sets, (e.g. k = 2 or k = 3)

-s: indicates the minimum support an item set must have to be
    included in the final results (e.g. 35000)
```

```
--ratingsFile: the file which contains movie reviews as shown in
               Table 2

--outputScheme: the name your program will use your jobs
    (e.g. --outputScheme WordCount would produce outputs
    WordCount1, WordCount2 if your program required 2 jobs)
```

The output should have

tuples: `<movie ID 1, movie ID 2, support>`

and triplets: `<movie ID 1, movie ID 2, movie ID 3, support>`

The order of the movie IDs doesn't matter, but be careful not to duplicate output by including the same set of IDs in a different order.
Hint: *Start by writing a program for the most frequent pairs. You will need this version of the program in Q3.*

If there are multiple map-reduce phases in your implementation, say the arguemnt parsed to –outputScheme is OUTPUT, then your output directory should be OUTPUT1, OUTPUT2, ... i.e., name suffix of the output directory with natural numbers starting from 1.

2. **[5pts]** Run your code using the `MovieLens20M` dataset as input for $s \geq 35000$. Report the top-5 most frequent pairs of movie **names**.

3. **[3pts]** Report the top-3 most frequent triplets of movie **names**. If necessary, break ties by choosing the smallest movie IDs. You may need to modify the support level in order to get at least 3 frequent triplets.

To Submit

(a) A zip file named `FrequentItemsets.zip`, which can be run with Hadoop. Please include source code and any dependencies needed to run the code. Submissions which do not contain source code will receive no credit.

(b) The top-5 frequent pairs for Q2(2) in the `writeup.pdf`.

(c) The top-3 frequent triplets for Q2(3) in the `writeup.pdf`.

(d) **DO NOT** submit any data.

Q3. [17pts] Association rules

1. **[12pts]** Start from the version of MapReduce program in Q2 that finds only pairs of frequently rated movies $< X, Y >$. Modify your program to also compute all the rules of type $X \Rightarrow Y$ with support s. The input format should be the same as in Q2, and the output should be in the form (C is the confidence score, no spaces):

   `< X->Y,C >.`

   All command line arguments should be the same as in Q2, however you only need to handle k = 2 (don't worry about k = 3).

2. **[5 pts]** For all the frequent pairs with support $s \geq 35000$, compute the confidence scores of the corresponding association rules: $X \Rightarrow Y$ and $Y \Rightarrow X$. List the top-5 rules in the writeup using the following format: $< X \Rightarrow Y$, confidence $c >$.

To Submit

(a) A zip file named `AssociationRules.zip`, which can be run with Hadoop. Please include source code and any dependencies needed to run the code. Submissions which do not contain source code will receive no credit.

(b) The top-5 rules in Q3(2) in the `writeup.pdf`.

Q4. [20pts] Data Analysis at Scale

In this part, you will use `Amazon Review Data`, a HUGE dataset containing product reviews from Amazon. This question aims to familiarize you with some practical problems that data scientists may face daily.

Data. For the following analysis you will use three new rating files, all of which are located on HDFS. They consist of (user,item,rating,timestamp) tuples. You can ignore the timestamp; it was added only to keep the format consistent with the `MovieLens20M` dataset that you used in the previous analysis.

- `/var/eecs476w21/Project1/AmazonReview20M.csv` contains 20M ratings for ~980K products
- `/var/eecs476w21/Project1/AmazonReview40M.csv` contains 40M ratings for ~1.6M products
- `/var/eecs476w21/Project1/AmazonReview60M.csv` contains 60M ratings for ~1.9M products

1. **[5pts]** Run your code for Q1 on all three datasets in order to transform them into the transaction format. Then run your frequent itemsets code in Q2 with:
   - $k = 2$ and $s = 30$ on `AmazonReview{20M,40M,60M}.csv`

- $k = 2$ and $s = 45$ on `AmazonReview60M.csv`, and
- $k = 2$ and $s = 60$ on `AmazonReview60M.csv`.

Report the runtime of `FrequentItemsets.zip` for each of the five runs.

Hints: Although we will not grade based on your runtime, please try your best to optimize your code. This question will take up to several hours, so you may need to write bash scripts and let your program run overnight. Please remember to log the runtime information. To avoid sessoin timeout, you may need some software tools like `tmux` or `screen`. We recommend that you start working on this early!

2. **[3pts]** For $k = 2$ and $s = 30$, give the scatterplot of the runtime (y-axis) vs. the number of ratings in the three `AmazonReview` datasets (i.e., 20M, 40M, 60M). Briefly describe the trend that you observe.

3. **[3pts]** For the `AmazonReview60M.csv`, give the scatterplot of the runtime (y-axis) vs. the support threshold (s=30, 45, 60). Briefly describe the trend that you observe.

4. **[3pts]** Plot the histogram of the support for the frequent itemsets in `AmazonReview40M` with $k = 2$, and $s = 30$. Use bins of size 5 for the support, i.e., give the number of frequent itemsets with support 30-34, 35-39, etc. as shown in the sample plot in Fig. 1 (only for illustration, not necessarily depicting the true trend). What do you observe in your plot?

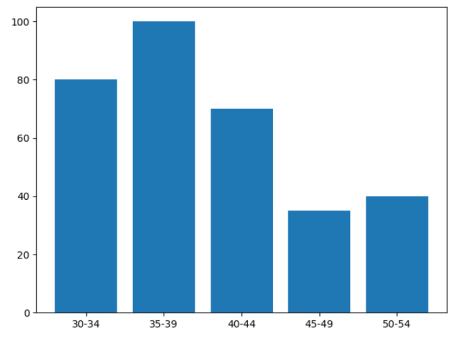


Figure 1: Sample plot for Q4(4).

5. **[2pts]** What do you think is the bottleneck for your program? Why do you think so?

6. **[4pts]** What difficulties have you encountered in Part2? How did you solve them?

To Submit

(a) Report five runtimes in Q4(1) in the `wrietup.pdf`.

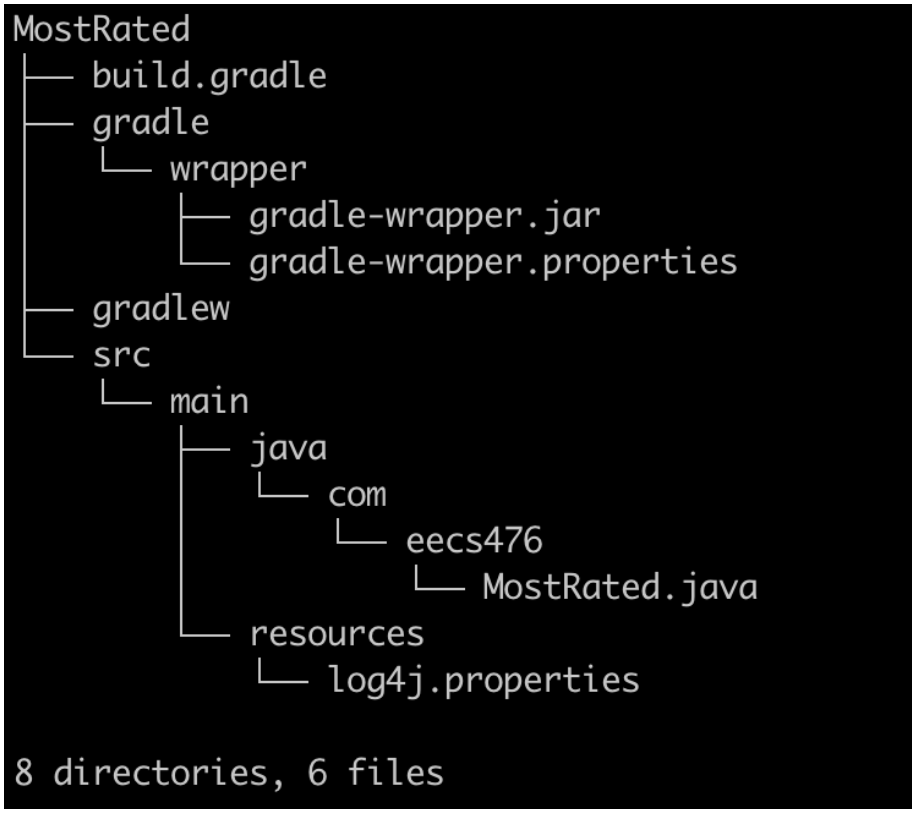
(b) Give the plots and brief description of your observations for Q4(2)-(4).

(c) Answer questions Q4(5)-(6) by reflecting on the computational complexity of the tasks in part 2, as well as the size of the input datasets.

Q5. Clarifications and Hints

We provide some clarifications and hints to help you with this project:

- Refer to the following structure for what to include in the zip file submitted to Autograder. This is an example for the MostRated part.



```
MostRated
├── build.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
└── src
    └── main
        ├── java
        │   └── com
        │       └── eecs476
        │           └── MostRated.java
        └── resources
            └── log4j.properties

8 directories, 6 files
```

- You should not assume Movie IDs or User IDs to be integers.
- Treat ”No Genre” as a regular genre.
- To compute the timestamp, use the timezone of the current local machine. You can setup a calendar with the command: Calendar.getInstance().
- Precision does not matter when you compute average scores or confidence scores.
- For Part 2 Q2, the algorithm needs to generalize for all k’s.
- If you want to run your code on Great Lakes, you should take into account the distributed setting. That is, your code should be able to run on multiple machines in parallel.