Following the previous Movie Similarities project, you will need to provide same recommendations for given movies. There are two objectives for this project:

1. Using [Spark](https://databricks.com/spark/about) instead of Hadoop for the similarity computation;
2. Learning how to submit spark jobs to the cloud computing clusters.

**Data** -- You are going to use the same datasets, which make it possible to compare between Hadoop and Spark results. In this project, you only have to use the small set (https://s3.amazonaws.com/cds-1/spark/ratings.dat | https://s3.amazonaws.com/cds-1/spark/movies.dat). A difference when importing the two data files: you don’t have to read them together. i.e. the data reading process is hard-coded in the provided sample code.

**The input arguments** -- You don’t have to parse all the arguments in the command line, which means that the arguments can also be hard-coded inside the script. Just don’t forget to specify them in the beginning of your submitted script:

* **-m [movie title]:** The title of a movie for which we'd like to see similar titles. You should be able to look at multiple movie titles.
* **-k [number of items]:** For each of the movies specified using **-m**, this specifies how many of the top matches we'd like to see. In other words, running with "-m The Godfather (1972) -k 10" would be asking for "the top ten movie recommendations if you liked The Godfather." (Default 25)
* **-l [similarity lower bound]:** When computing movie similarity metrics, we'll produce a floating-point value in the range [-1, 1]. This input says to ignore any movie parings whose similarity metric is below this value. (Default 0.4)
* **-p [minimum rating pairs]:** When computing similarity metrics, ignore any pair of movies that don't have at least this many shared ratings. (Default 3)

**You only need to calculate the statistical correlation. The output lines** will still be sorted descend based on the correlation. Each line should look like this:

[The input movie, a similar-rated movie, statistical correlation, number of co-occurrence]

**Steps**

1). (This step is already included in the provided sample code) You need to take movie IDs and names from movies file, user IDs, movie IDs, and ratings from ratings file, and connect them by movie IDs (you can even do all the computations with only movie IDs and go back to find corresponding movie names before printing the results);

Useful spark command: sc.textFile || rdd.map()

2). Then you create rating pairs for movies that were rated by same user;

Useful spark command: rdd.flatMap() || rdd.groupByKey() || rdd.sortByKey()

3). You make sure that all pairs only show up once in your created pairs and compute the correlations and co-occurrence for each pair;

Useful spark command: rdd.sortByKey() || rdd.filter() || rdd.reduceByKey()

4) You filter the output down to the several input movies of your choice and print the result;

5) Once the script is finished, please make a submission to the **Google Dataproc cluster** or **AWS EMR cluster**. The brief instructions will be provided.

**The deliverables:**

* Your code -- can be either your script file or the record of codes that you run line by line in Python/Java/Scala/R;
* The required screenshots include: the launch of the cluster, submitting the job, and the final output (the output can be either printed in the terminal or written to a file).

**Please make sure that any of your screenshot is not identical to others’**.

**Useful information for the spark script:**

Spark tutorials

https://spark.apache.org/docs/latest/quick-start.html

https://www.tutorialspoint.com/apache\_spark/index.htm

https://www.edureka.co/blog/spark-tutorial/

Useful commands (Python&Scala): sc.textFile || rdd.map() || rdd.filter() || rdd.flatMap() || rdd.reduceByKey() || rdd.sortByKey() || rdd.take()

Scala tutorials

https://www.supergloo.com/spark-tutorial/spark-tutorials-scala/

https://blog.matthewrathbone.com/2015/12/14/spark-scala-tutorial.html

https://www.analyticsvidhya.com/blog/2017/01/scala/

**Instruction for submitting the jobs to the clusters**

You can choose Google Cloud or AWS depends on your preference and available credit. It is very **IMPORTANT** that once you stop using the cluster, it **MUST** be destroyed or terminated to avoid further charging. You can create/clone a new one next time if you plan to use the clusters again.

**Google Dataproc cluster**

1. Install gcloud SDK and connect to the platform:

https://cloud.google.com/sdk/install

https://cloud.google.com/sdk/gcloud/reference/auth/activate-service-account

For example, my connection code is:

gcloud auth activate-service-account \

cds-class@ethereal-argon-179902.iam.gserviceaccount.com \

--key-file=/Users/rqiu/Downloads/MyProject-cfc0a86e39c7.json

1. Prepare the data and code in your LOCAL folder (In the exact one you type in the command)

Data: I have made the converted data available to the public: https://s3.amazonaws.com/cds-1/spark/movies.dat | https://s3.amazonaws.com/cds-1/spark/ratings.dat

Put your finished mr-spark.py it in the same folder

1. Create a cluster on Google Dataproc

On your Google Dataproc page, create a cluster. You can use the default 4vCPUs with 15GB memory, or 8vCPUs with 30GB memory to make it faster.

Once the cluster is ready, run this command (change the cluster name to yours):

gcloud dataproc jobs submit pyspark --cluster cluster-5e41 --region global mr-spark.py

AWS EMR cluster

1. Create an EMR cluster following this YouTube video: <https://www.youtube.com/watch?v=An8tw4lEkaI>
2. ssh to the cluster and type in following commands to download the data:

wget https://s3.amazonaws.com/cds-1/spark/movies.dat

wget <https://s3.amazonaws.com/cds-1/spark/ratings.dat>

1. nano mr-spark.py to add your code.
2. Run the command: spark-submit ./mr-spark.py

The provided sample code “mr-spark.py” is ready to run on both platforms, only that the results are not final. You can use it as a sample to test if the job submission is successful. If you cannot finish the Spark code on time, successful submission of the sample code on the online clusters can still earn you part of the credit.

Output of the sample code:

