**CS 6301- Big Data Analytics and Management**

**Fall 2014 Homework Solution # 5**

**Due Date : Y**

In this homework you will learn how to solve problems using Apache Spark. Please apply ***Apache Spark*** interactive shell (for **scala** or **python**) / run from command line (**scala**/**java**/**python**) to derive some statistics from **IMDB** movie data. You can find the dataset in ***elearning***. Copy the data into your hadoop cluster and use it as input data. You can use the put or ***copyFromLocal*** HDFS shell command to copy those files into your HDFS directory. There are 3 datafiles: ***movies.dat***, ***ratings.dat***, ***users.dat*** Please read the “***README\_Important***” file to know about the data organization and to know about the Attribute of the data. All are very well explained in that ***README\_Important*** file. Please read the questions carefully and use only the data file that you need.

***Q1.*** Given a input ***zipcode***, find all the ***user-ids*** that belongs to that ***zipcode***. You must take the input ***zipcode*** in command line. [For example, if the input ***zipcode*** is ***75252*** then you need to find all users who lives in ***75252***] [You only need ***users.dat*** file to get the answer.]

Ans:

val lines = sc.textFile("hdfs://10.176.148.60:9000/movie/users.dat")

val inputZip = "55117" /\* This is input parameter \*/

val user\_zip = lines.filter{x => val tokens = x.split("::")

(tokens(4) == inputZip)

}

val users = user\_zip.flatMap(x => x.split("::")(4)).distinct

println ("User Ids for zip " + inputZip + ": " + users.collect.mkString(","))

Output: User Ids for zip 55117: 7,5,1

***Q2***. Find ***top*** **10 average** rated ***movies*** with ***descending*** order of rating. [Clue : From the dataset we know that, each user can rate mulitple movies and each movie can be rated by multiple users and this information is found in ***ratings.dat*** file . So, first we have to find the average rating of a movie and second, we need to find the top 10 average rated movies. So, First RDD will find the average rating of each movies and the second RDD will take the previous RDD as the input and find the top 10 from them.] [You only need ***ratings.dat*** file to get the answer.]

val lines = sc.textFile("hdfs://10.176.148.60:9000/movie/ratings.dat")

val movie\_avg = lines.map{

x => val tokens = x.split("::")

(tokens(1), (tokens(2).toInt, 1))

}.reduceByKey{

(x,y) => (x.\_1 + y.\_1, x.\_2 + y.\_2)

}.mapValues(x=> (1.00 \* x.\_1)/x.\_2)

val movies = sc.textFile("hdfs://10.176.148.60:9000/movie/movies.dat")

val movie\_name = movies.map{

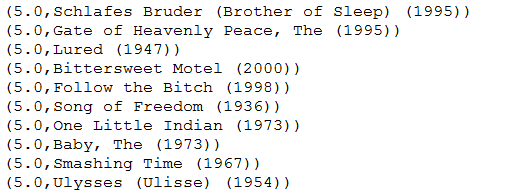
x => val tokens = x.split("::")

(tokens(0),tokens(1))

}.join(movie\_avg)

val movie\_avg\_dec = movie\_name.map(x => (x.\_2.\_2,x.\_2.\_1)).sortByKey(false).take(10).foreach(println)

Output:



***Submission***:

You have to upload your submission via e-learning before due date. Please upload the following to eLearning:

1. Three scripting file like, Q1.txt, and Q2.txt separately if you use scala / python interactive shell. Each file contains the scala /python code.

If you use java, then submit all the java files.

**Q3.** Read the following link for co-occurrence based recommendation implementing in mahout.

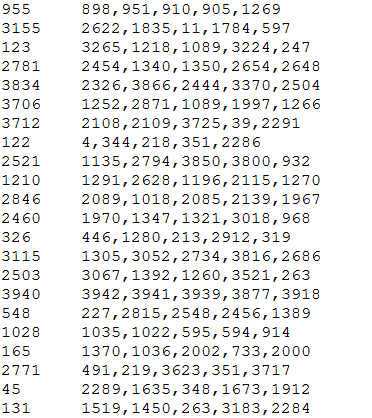
[https://mahout.apache.org/users/recommender/intro-cooccurrence-spark.html](https://mahout.apache.org/users/recommender/intro-cooccurrence-spark.html%20)

Currently Mahout switches from MapReduce to Apache Spark. It has an interactive shell also (showed in the class, lecture contains how to install it). Using that, apply item-based collaborative filtering using mahout’s ***spark-itemsimilarity***. ***spark-itemsimilarity*** can be used to create ***"other people also liked these things"*** type recommendations.

**“A user rates some movies as 4. Recommend some movies for his that has the same rating. For each movie, show maximum 5 recommended movies”**

**Steps to follow:**

1. Read the above link carefully and construct the item-similarity matrix of each movie having rating 4 (use ratings.dat). The output should be like this:



1. Save the above file to HDFS. Now Run Apache spark interactive shell. From the shell, take the user id as input (you can fix the id, e.g., val userID = 1). Now finds all the movies that he rates as 4.
2. Load/read the above file (item-similarity file) and find the movies that match with the user’s rated movies with the key of the item-similarity file.

For example, suppose a user has id 7 and he rates 955 and 123 as 4. After executing this, you will get a matrix like following:

955 898,951,910,905,1269

123 3265,1218,1089,3224,247

1. Now replace the movie Id with movie name.

For example,

Movie\_955 Movie\_898, Movie\_951, Movie\_910, Movie\_905, Movie\_1269

Movie\_123 Movie\_3265, Movie\_1218, Movie\_1089, Movie\_3224, Movie\_247

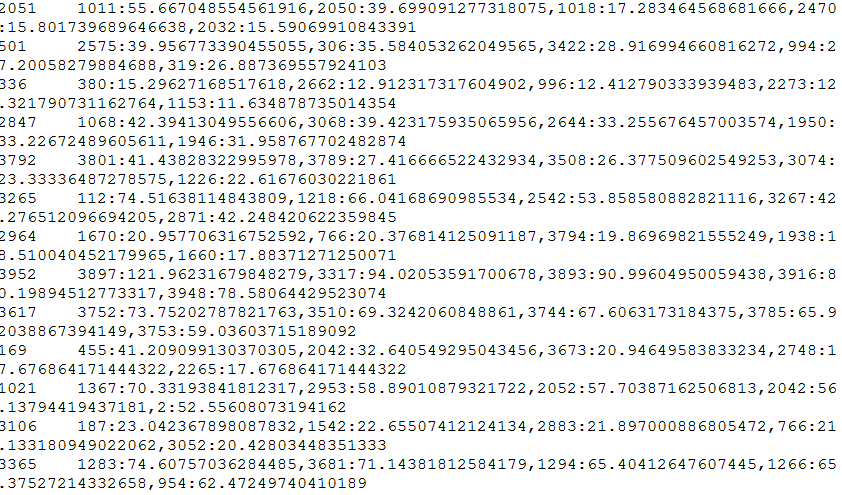
You can apply join if it necessary. (Use movies.dat and ratings.dat)

**Mahout part:**

Go to your mahout directory then type following:

$ bin/mahout spark-itemsimilarity -i /movie/ratings.dat -o /movie/ -ic 1 -fc 2 -f1 4 -m 5 -td "," -id "::"

A similarity matrix will be generated undere /movie/indicator-matrix/part-00000



Now open spark-shell from spark/bin directory (/usr/local/spark)

$./spark-shell

**Scala programing**

val user\_rated\_movie = List(955, 123)

val lines = sc.textFile("hdfs://10.176.148.60:9000/movie/indicator-matrix/part-00000")

val movies\_from\_matrix = lines.filter{

x => val tokens = x.split("\t")

val key = tokens(0).toInt

user\_rated\_movie.contains(key)

}

val movies = sc.textFile("hdfs://10.176.148.60:9000/movie/movies.dat").map{

x => val tokens = x.split("::")

(tokens(0).toInt, tokens(1))

}

val rated\_movie\_name = movies\_from\_matrix.flatMap{

x => val tokens = x.split("\t")

val key = tokens(0).toInt

val values = tokens(1).split(",")

val buf = scala.collection.mutable.ListBuffer.empty[(Int, Int)]

for (token <- values){

val token\_split = token.split(":")

val movie\_id = token\_split(0).toInt

val item = (movie\_id, key)

buf += item

}

buf.toList

}.join(movies).map(x => (x.\_2.\_1, x.\_2.\_2)).groupByKey.join(movies)

/\* Printing \*/

rated\_movie\_name.collect.foreach (x => println("Rated Movie: " + x.\_2.\_2.toString + " -> Recommended Movies: " + x.\_2.\_1.mkString("/")))

Output:



***Submission***:

You have to upload your submission via e-learning before due date. Please upload the following to eLearning:

1. A scripting file like, Q4\_1.txt that shows the building of spark-itemsimilarity and another scripting file Q4\_2.txt shows the scala/java program (contains codes for step 2 - 4).

If you use java, then submit all the java files.