

Scalable Data Mining (Autumn 2021)

Assignment 1

Name : Altaf Ahmad
Roll no: 18MA20005

Question 1

(a) Download the ratings file, parse it and load it in an RDD named ratings.

Ans: First we download the given file and then parse it by separating the elements from the format UserID::MovieID::Rating::Timestamp , using split with "::"

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/s...  
scala> def openRat(name : String) : ra = {  
    |   val spl = name.split("::").map(_.trim).toList  
    |   return ra(spl(0).toInt , spl(1).toInt , spl(2).toInt, spl(3))  
    | }  
openRat: (name: String)ra  
scala> val ratings = sc.textFile("data1/ratings.dat").map(name => openRat(name))  
ratings: org.apache.spark.rdd.RDD[ra] = MapPartitionsRDD[2] at map at <console>:26
```

(b)How many lines does the ratings RDD contain?

Ans : We found that the number of lines of RDD is 1000209

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/s...  
scala> ratings  
res1: org.apache.spark.rdd.RDD[ra] = MapPartitionsRDD[2] at map at <console>:26  
scala> ratings.count  
res2: Long = 1000209
```

(c) Count how many unique movies have been rated.

Ans : We will have to count the number of distinct movies. We can create the map and count of that using distinct. The number of unique movies is 3706

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th...  
scala> ratings.map(rat => rat.mov).distinct.count()  
res4: Long = 3706  
scala>
```

(d) Which user gave the most ratings? Return the userID and number of ratings.

Ans : In order to find the user who gave the most ratings, we first calculate the number of ratings of each user and then sort it in descending order. After this we can find out the maximum rating user along with the number of ratings.

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/spark-3.1.2-bin-hadoop2...  
scala> val maxUser = ratings.keyBy(u => u.user)  
maxUser: org.apache.spark.rdd.RDD[(Integer, ra)] = MapPartitionsRDD[7] at keyBy at <console>:25  
scala> val mostRat = maxUser.mapValues(a => 1).reduceByKey((a,b) => a + b)  
mostRat: org.apache.spark.rdd.RDD[(Integer, Int)] = ShuffledRDD[9] at reduceByKey at <console>:25  
scala> val sorted = mostRat.sortBy(_._2, false).take(1)  
sorted: Array[(Integer, Int)] = Array((4169,2314))  
scala>
```

The userID is 4169 and the number of ratings is 2314

(e) Which user gave the most '5' ratings? Return the userID and number of ratings.

Ans : Similarly in this question, we count the number of '5' star ratings given and then sort it in descending order. After that we can find the userID and the number of ratings.

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/spark-3.1.2-bin-had...  
scala> val fiveStar = ratings.filter(a => a.rat == 5)  
fiveStar: org.apache.spark.rdd.RDD[ra] = MapPartitionsRDD[9] at filter at <console>:25  
  
scala> fiveStar.count  
res1: Long = 226310  
  
scala> val users = fiveStar.keyBy(a => a.user).aggregateByKey(0)((usr, m) => user + 1, (a,b) => a+b)  
<console>:25: error: not found: value user  
      val users = fiveStar.keyBy(a => a.user).aggregateByKey(0)((usr, m) => user + 1, (a,b) => a+b)  
                                ^  
scala> val users = fiveStar.keyBy(a => a.user).aggregateByKey(0)((usr, m) => usr + 1, (a,b) => a+b)  
users: org.apache.spark.rdd.RDD[(Integer, Int)] = ShuffledRDD[11] at aggregateByKey at <console>:25  
  
scala> val sorted = users.sortBy(_._2, false)  
sorted: org.apache.spark.rdd.RDD[(Integer, Int)] = MapPartitionsRDD[16] at sortBy at <console>:25  
  
scala> val ans = sorted.take(1)  
ans: Array[(Integer, Int)] = Array((4277,571))  
  
scala>
```

userID = 4277 and number of ratings = 571

Question 2

(a) Read the movies and users files into RDDs. How many records are there in each RDD?

Ans :

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/spark-3.1.2-bin...  
scala> def openMov(name: String): mov = {  
  |   val spl = name.split("::").map(_.trim).toList  
  |   return mov(spl(0).toInt, spl(1), spl(2))  
  | }  
openMov: (name: String)mov  
  
scala> val movies = sc.textFile("data1/movies.dat").map(a => openMov(a)).cache  
movies: org.apache.spark.rdd.RDD[mov] = MapPartitionsRDD[25] at map at <console>:26  
  
scala> def openUser(name: String): user = {  
  |   val spl = name.split("::").map(_.trim).toList  
  |   return user(spl(0).toInt, spl(1), spl(2).toInt, spl(3), spl(4))  
  | }  
openUser: (name: String)user  
  
scala> val users = sc.textFile("data1/users.dat").map(a => openUser(a)).cache  
users: org.apache.spark.rdd.RDD[user] = MapPartitionsRDD[28] at map at <console>:26  
  
scala> movies.count  
res3: Long = 3883  
  
scala> users.count  
res4: Long = 6040  
  
scala>
```

movies.dat has 3883 records and users.dat has 6040 records

(b) How many of the movies are a comedy?

Ans : There are a total of 1200 movies with genre "Comedy"

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/...  
scala> movies.filter(a => a.genre.contains("comedy")).count  
res5: Long = 0  
  
scala> movies.filter(a => a.genre.contains("Comedy")).count  
res6: Long = 1200  
  
scala>
```

(c) Which comedy has the most ratings? Return the title and the number of rankings. Answer this question by joining two datasets.

Ans : For solving this, we will have to join the movies RDD with the ratings RDD and then count the number of movies with genre "Comedy". Sort this in decreasing order and then take the first value,

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/spark-3.1.2-bin-hadoop2.7/bin
scala> val comedy = movies.filter(a => a.genre.contains("Comedy")).keyBy(a => a.movie)
comedy: org.apache.spark.rdd.RDD[(Integer, mov)] = MapPartitionsRDD[47] at keyBy at <console>:25

scala> val joined = comedy.join(ratings.keyBy(a => a.mov))
joined: org.apache.spark.rdd.RDD[(Integer, (mov, ra))] = MapPartitionsRDD[51] at join at <console>:27

scala> val titles = joined.map(a => (a._2._1.title))
titles: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[52] at map at <console>:25

scala> val ans = titles.groupBy(a => a).map{case (m, usr) => (m, usr.size)}.sortBy(a => a._2, false).take(1)
ans: Array[(String, Int)] = Array((American Beauty (1999),3428))

scala>
```

The most rated comedy is American Beauty (1999) with number of rankings -> 3428

(e) Compute the number of unique users that rated the movies with movie_IDs 2858, 356 and 2329 without using an inverted index. Measure the time (in seconds) it takes to make this computation.

Ans : The number of unique users is 4213

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/spark-3.1.2-bin-hadoop2.7/bin
scala> ratings.filter(a => (List(2858, 356, 2329).contains(a.mov))).groupBy(a => a.user).keys.distinct.count
res13: Long = 4213

scala> def runTime() {
  | val start_time = (System.currentTimeMillis)
  | println(ratings.filter(a => (List(2858, 356, 2329).contains(a.mov))).groupBy(a => a.user).keys.distinct.count)
  | println(((System.currentTimeMillis) - start_time)/1000.0)
  | }
runTime: ()Unit

scala> runTime()
4213
0.348

scala>
```

It takes around 0.3 - 0.38 seconds to make this computation

(f) Create an inverted index on ratings, field movie_ID. Print the first item.

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/spark-3.1.2-bin-hadoo...
scala> val invIndex = ratings.groupBy(a => a.mov).cache
invIndex: org.apache.spark.rdd.RDD[(Integer, Iterable[ra])] = ShuffledRDD[111] at groupBy at <console>:25

scala> invIndex.lookup(1).take(1)
res15: Seq[Iterable[ra]] = ArrayBuffer(CompactBuffer(ra(1,1,5,978824268), ra(6,1,4,978237008), ra(8,1,4,978233496), ra(9,1,5,978225952), ra(10,1,5,978226474), ra(18,1,4,978154768), ra(19,1,5,978555994), ra(21,1,3,978139347), ra(23,1,4,978463614), ra(26,1,3,978130703), ra(28,1,3,978985309), ra(34,1,5,978102970), ra(36,1,5,978061285), ra(38,1,5,978046225), ra(44,1,5,978019369), ra(45,1,4,977990044), ra(48,1,4,977975909), ra(49,1,5,977972501), ra(51,1,5,977947828), ra(56,1,5,977938855), ra(60,1,4,977931983), ra(65,1,5,991368774), ra(68,1,3,991376026), ra(73,1,3,977867812), ra(75,1,5,977851099), ra(76,1,5,977847069), ra(78,1,4,978570648), ra(80,1,3,977786904), ra(90,1,3,993872933), ra(92,1,4,977646817), ra(96,1,4,980563195), ra(99,1,3,982873678), ra(109,1,4,9775192...))

scala>
```

(g) Compute the number of unique users that rated the movies with movie_IDs 2858, 356 and 2329 using the above calculated index. Measure the time (in seconds) it takes to compute the same result using the index.

Ans:

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/spark-3.1.2-bin-hadoo...  
scala> def invRunTime() {  
  | val start_time: Long = (System.currentTimeMillis)  
  |  
  | val items = sc.parallelize(List[Integer](2858, 356, 2329)).keyBy(a => a)  
  | println(invIndex.join(items).flatMap{usr => usr._2._1}.map(a => a.user).distinct.count)  
  |  
  | println(((System.currentTimeMillis) - start_time)/1000.0)  
  | }  
invRunTime: ()Unit  
  
scala> invRunTime()  
4213  
7.943  
  
scala> |
```

The number of unique users is still the same but the time taken is approximately 7.9 secs

Question 3

(a). Create a function that given an RDD and a field (e.g. download_id), it computes an inverted index on the RDD for efficiently searching the records of the RDD using values of the field as keys.

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/spark-3.1.2-bin-hadoop2.7/bin  
scala>  
  
scala> Assignment_1.count  
res0: Long = 9669634  
  
scala> def invIndex(rdd : org.apache.spark.rdd.RDD[LogLine], downID : Int):  
  | org.apache.spark.rdd.RDD[(Any, Iterable[LogLine])] = {  
  |   return rdd.groupBy((x : LogLine) => x.productElement(downID));  
  | }  
invIndex: (rdd: org.apache.spark.rdd.RDD[LogLine], downID: Int)org.apache.spark.rdd.RDD[(Any, Iterable[LogLine])]  
  
scala> |
```

(b) Compute the number of different repositories accessed by the client 'gthorren-22' (without using the inverted index).

```
altaf@altaf-HP-Laptop-15q-bu1xx: ~/7th Sem/Scalable Data Mining/spark-3.1.2-bi...  
scala> val down22 = Assignment_1.filter(_.download_id == 22)  
down22: org.apache.spark.rdd.RDD[LogLine] = MapPartitionsRDD[3] at filter at <console>:27  
  
scala> val cl22 = down22.map(_.rest.split("/").slice(4,6).mkString("/").takeWhile(_ != '?'))  
cl22: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[4] at map at <console>:27  
  
scala> val ans = cl22.distinct.count  
ans: Long = 3973  
  
scala> |
```