CS60021: Scalable Data Mining

Assignment 1 Solution

==========

Assignment 1 Submission
Name: RADHIKA PATWARI
Roll number: 18CS10062
Date: 31/8/2021
==========

My Setup:

- Installed Spark + Scala on Ubuntu 16.04 linux system

Answers to the questions :

Question 1:

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

Approach :

1. Declared the file path to the ratings.dat file
2. Defined the token "::" which is used in ratings.dat to separate field values
3. Used spark context 'sc'
4. textFile() loads the ratings.dat file as RDD
5. map() is used to transform the RDD by splitting each line using "::" through split() function
6. Transformed RDD is stored in ratings

```
def main(args: Array[String]): Unit = {
      val filePath = "/home/radhika/Downloads/semester7/sdm/data/ratings.dat"
      val sepToken: String = "::"
      val ratings = sc.textFile(filePath).map(line => line.split(sepToken))

      // Remaining code
}
```

---

b) How many lines does the ratings RDD contain?
- 100209 lines

Approach :

1. Used count() action to count the no. of lines in the RDD ratings

```
      val lineCount = ratings.count()
      println("Ratings file has " + lineCount + " lines")
```

---

c) Count how many unique movies have been rated.
- 3706 distinct movies

Approach :

1. Each record of RDD is of form Array[String] having field {UserID, MovieID, Rating, Timestamp}
2. Transforming Array[String] to String by extracting MovieID at row(1) using map()
3. Removing duplicate MovieID from the transformed RDD using distinct()
4. Using count() action to count no. of distinct movies

```
      val movieCount = ratings.map(row => row(1)).distinct().count()
      println("Ratings file has " + movieCount + " distinct movies")
```

---

d) Which user gave the most ratings? Return the userID and number of ratings.
- User id : 4169 rated 2314 movies

Approach :

1. Each record of RDD is of form Array[String] having field {UserID, MovieID, Rating, Timestamp}
2. Transforming Array[String] to String by extracting UserID at row(0) using map()
3. Transforming String to (String, Int) to assign a value 1 to each User Id
4. Summing up the Int values for each user which gives the no. of times each user has rated any movie
5. userCount contains (userId, moviesRated) for all users
6. Use maxBy() to find the userId with maximum count of movies rated
7. mostActiveUser store (userId, moviesRated) for the user who has rated maximum no. of movies

```
      val userIds = ratings.map(row => row(0))
      val userCount = userIds.map(user => (user, 1)).reduceByKey(_+_).collect
      val mostActiveUser = userCount.maxBy(user => user._2)
      var userId = mostActiveUser._1
      var moviesRated = mostActiveUser._2
      println("User ID : " + userId + " has rated " + moviesRated + " movies,
which is the highest.")
```

---

e) Which user gave the most '5' ratings? Return the userID and number of ratings.
- User id: 4277 (who has rated 1743 movies) has given most '5' star ratings.

Approach :

1. Each record of RDD is of form Array[String] having field {UserID, MovieID, Rating, Timestamp}
2. Transforming Array[String] to (String, String) by extracting UserID at row(0) and Rating given by that user at row(2) using map()
3. Transforming (String, String) to (String, Int). If the user has given “5” rating then a value 1 is assigned else, assign 0
4. Sum up the values for each userId to check how many 5 star ratings each user has given using ReduceByKey() function
5. Use maxBy() to find the userId with maximum count of 5 star ratings
6. userWithMost5StartRatings contain (userId, count of 5 star ratings) for the user who has given max 5 star ratings
7. userId store the user id for max 5 star giving user
8. moviesRated contain the total no. of movies rated by the user “userId”

```
      val userIdsAndRatings = ratings.map(row => (row(0), row(2)))
      val histogramOf5StarRatings = userIdsAndRatings.map(user => if (user._2 == 
"5") (user._1, 1) else (user._1, 0)).reduceByKey(_+_).collect
      val userWithMost5StarRatings = histogramOf5StarRatings.maxBy(user => 
user._2)
      val userId = userWithMost5StarRatings._1
      val moviesRated = ratings.map(row => row(0)).map(user => if (user == 
userId) 1 else 0).reduce((x, y) => x+y)

      println("user with id: " + userId + " (who has rated " + moviesRated + " 
movies) has given most '5' star ratings.")
```

Final Output for all parts of Question 1 (linux terminal):

```
scala> :load /home/radhika/Downloads/semester7/sdm/p1.scala
Loading /home/radhika/Downloads/semester7/sdm/p1.scala...
defined object MovieLensRatings

scala> MovieLensRatings.main(Array(""))
Ratings file has 1000209 lines
Ratings file has 3706 distinct movies
user with id: 4169 has rated 2314 movies, which is the highest.
user with id: 4277 (who has rated 1743 movies) has given most '5' star ratings.

scala>
```

Question 2:

a) Read the movies and users files into RDDs. How many records are there in each RDD?
- Movies file has 3883 lines
  Users file has 6040 lines

Approach :

1. Declared the file path to the movies.dat and users.dat files
2. Defined the token "::" which is used in ratings.dat to separate field values
3. Used spark context 'sc'
4. textFile() loads the movies.dat and users.dat files as RDDs
5. map() is used to transform the RDD by splitting each line using "::" through split() function
6. Transformed RDD is stored in movies and users
7. Used count() action to count the no. of lines in the RDD movies and users

```
def main(args: Array[String]): Unit = {

      val filePath1 = "/home/radhika/Downloads/semester7/sdm/data/movies.dat"
      val filePath2 = "/home/radhika/Downloads/semester7/sdm/data/users.dat"

      val sepToken: String = "::"
      val movies = sc.textFile(filePath1).map(_.split(sepToken))
      val users = sc.textFile(filePath2).map(_.split(sepToken))

      val lineCount1 = movies.count()
      println("Movies file has " + lineCount1 + " lines")

      val lineCount2 = users.count()
      println("Users file has " + lineCount2 + " lines")

      // Remaining Code

}
```

---

b) How many of the movies are comedies?
- 521 movies have Comedy genre

Approach :

1. Each record of RDD movies is of form Array[String] having field {MovieID::Title::Genres}
2. Using filter() to extract records with Genre == "Comedy"
3. Transforming Array[String] to String by extracting MovieID at row(0) using map()
4. Removing duplicate movie ids using distinct() transformation
5. Count the no. of distinct movies having Comedy genre using count() action and store in comedyMoviesCount

```
      val comedyMoviesCount = movies.filter(row => (row(2) == "Comedy")).map(row
=> row(0)).distinct.count
      println(comedyMoviesCount + " movies have Comedy genre")
```

---

c) Which comedy has the most ratings? Return the title and the number of rankings. Answer this question by joining two datasets.
- Movie Being John Malkovich (1999) has highest rating of 9245

[Assumption : most ratings imply sum of all ratings given for that movie]

Approach :

1. Each record of RDD movies is of form Array[String] having field {MovieID::Title::Genres}
   Each record of RDD ratings is of form Array[String] having field {UserID::MovieID::Rating::Timestamp}
2. Using filter() to extract records with Genre == “Comedy” such that comedyMovies of form [String, String] for [MovieId, Title]
3. Transforming Array[String] to [String,String] for [MovieId, Rating] by extracting MovieID at row(1) and corresponding rating at row(2) in RDD ratings using map()
4. Joining the above RDDs and extracting [MovieTitle, Rating] using map()
5. Find sum of all ratings for each movie using reducebyKey
6. Using maxBy() to find the movie with the maximum rating

```
      val comedyMovies = movies.filter(row => row(2) == "Comedy").map(row =>
(row(0),row(1)))
      val movieRatings = ratings.map(row => (row(1),row(2)))

      val comedyMovieRatings = comedyMovies.join(movieRatings).map(row =>
(row._2._1, row._2._2.toInt)).reduceByKey(_+_).collect
      val comedyMovieWithHighestRating = comedyMovieRatings.maxBy(movie =>
movie._2)

      println("Movie " + comedyMovieWithHighestRating._1 + " has highest rating
of " + comedyMovieWithHighestRating._2)
```

---

e) Compute the number of unique users that rated the movies with movie_IDs 2858, 356 and 2329 without using an inverted index.
- 379 distinct users
Time elapsed : 0.863722061 seconds

[Assumption : considering users who have rated all the 3 movies]

Approach :

1. Each record of RDD ratings is of form Array[String] having field {UserID::MovieID::Rating::Timestamp}
2. Using filter() to extract records with MovieId == “2858”
3. Transforming Array[String] to [String] for [UserId] by extracting UserID at row(0) in RDD ratings using map()
4. Removing duplicate userIds from RDD using distinct() and storing the final set of userIds in usersForMovie2858
5. Repeating similar approaches from step 2 to 4 for movie ids 356 and 2329
6. Using intersection() transformation to obtain an RDD containing users who have rated all 3 movies.
7. Using count() to find distinct users

```
      val t1 = System.nanoTime

      val usersForMovie2858 = ratings.filter(row => row(1) == "2858").map(row =>
row(0)).distinct()
      val usersForMovie356 = ratings.filter(row => row(1) == "356").map(row =>
row(0)).distinct()
      val usersForMovie2329 = ratings.filter(row => row(1) == "2329").map(row =>
row(0)).distinct()

      val usersWithSpecificMovies =
usersForMovie2858.intersection(usersForMovie356).intersection(usersForMovie2329).
count

      val duration = (System.nanoTime - t1) / 1e9d

      println(usersWithSpecificMovies + " distinct users have rated all 3 movies
with movie_IDs 2858, 356 and 2329")
      println("Time elapsed : " + duration + " seconds")
```

---

f) Create an inverted index on ratings, field movie_ID. Print the first item.
- Movie Id 2828 with its first 10 user ids

2828
16 | 70 | 75 | 123 | 148 | 163 | 202 | 209 | 216 | 245 |

Approach :

1. Each record of RDD ratings is of form Array[String] having field {UserID::MovieID::Rating::Timestamp}

2. Transforming RDD ratings from Array[String] to [String, String] corresponding to [MovieId, UserId]
3. Using groupByKey to make MovieId as the index for storing userIds
4. Printing the first element after inversion.
   [The first MovieId is 2828 which has many users. To restrict output, printed only frst 10 users]

```
	val ratingsInvertedOnMovieId = ratings.map(x => (x(1),x(0))).groupByKey

	println("First Item after inversion : ")
	ratingsInvertedOnMovieId
		.collect()
		.take(1)
		.foreach(a => {
				println(a._1)
				a._2.take(10).foreach(user => print(user + " | "))
			})
```

---

g) Compute the number of unique users that rated the movies with movie_IDs 2858, 356 and 2329 using the above calculated index.
- 379 distinct users have rated all 3 movies with movie_IDs 2858, 356 and 2329
Time elapsed : 0.766054001 seconds

[Assumption : considering users who have rated all the 3 movies]

Approach :

1. Each record of RDD ratingsInvertedOnMovieId is of form Array[String, Iterable[String]] having field {MovieID, {UserId1, UserId2, UserId3}}
2. Using filter() to extract records with MovieId == “2858”
3. Transforming Array[String] to [String] for [UserId] by extracting UserID at row(0) in RDD ratingsInvertedOnMovieId using map()
4. Removing duplicate userIds from RDD using distinct() and storing the final set of userIds in usersForMovie2858
5. Repeating similar approaches from step 2 to 4 for movie ids 356 and 2329
6. Using intersection() transformation to obtain an RDD containing users who have rated all 3 movies.
7. Using count() to find distinct users

```
	val t2 = System.nanoTime

	val usersForMovie2858 = ratingsInvertedOnMovieId.filter(x =>
x._1=="2858").flatMap{case(key, row) => row}.distinct()
	val usersForMovie356 = ratingsInvertedOnMovieId.filter(x =>
```

```
x._1=="356").flatMap{case(key, row) => row}.distinct()
      val usersForMovie2329 = ratingsInvertedOnMovieId.filter(x =>
x._1=="2329").flatMap{case(key, row) => row}.distinct()

      val usersWithSpecificMoviesUsingInvertedIndex =
usersForMovie2858.intersection(usersForMovie356).intersection(usersForMovie2329).
count
      val duration2 = (System.nanoTime - t2) / 1e9d

      println("\n" + usersWithSpecificMoviesUsingInvertedIndex + " distinct users
have rated all 3 movies with movie_IDs 2858, 356 and 2329")
      println("Time elapsed : " + duration2 + " seconds")
```

Final Output for all parts of Question 2 (linux terminal):

```
scala> :load /home/radhika/Downloads/semester7/sdm/p1.scala
Loading /home/radhika/Downloads/semester7/sdm/p1.scala...
defined object MovieLensRatings

scala> MovieLensRatings.main(Array(""))
Movies file has 3883 lines
Users file has 6040 lines
521 movies have Comedy genre
Movie Being John Malkovich (1999) has highest rating of 9245
379 distinct users have rated all 3 movies with movie_IDs 2858, 356 and 2329
Time elapsed : 0.863722061 seconds
First Item after inversion :
2828
16 | 70 | 75 | 123 | 148 | 163 | 202 | 209 | 216 | 245 |
379 distinct users have rated all 3 movies with movie_IDs 2858, 356 and 2329
Time elapsed : 0.766054001 seconds
```

Question 3:

Loading and Preprocessing data

Approach :

1. Load the torrent .txt file
2. Split lines on basis of commas and colon into Array[String]
3. Define a case class torrectSchema for various fields - debug_level, timestamp, download_id, retrieval_stage and rest
4. Check timestamp of each record
5. Discard the record in case of invalid timestamp
6. Store the preprocessed RDD in Assignment_1

```
import java.util.TimeZone
import java.text.SimpleDateFormat
```

```
TimeZone.setDefault(TimeZone.getTimeZone("GMT"))
var timeFormat = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ssX")

case class torrentSchema (
      debug_level : String,
      timestamp : java.util.Date,
      download_id : String,
      retrieval_stage : String,
      rest : String)

def parseFile(filename : String): org.apache.spark.rdd.RDD[torrentSchema] = {
      var parsedRDD = sc.textFile(filename)
      .filter(line => line.size > 0)
      .map(line => line.split(", | -- ", 4))
      .map(w =>
            if (w.size == 4)
                Array(w(0), w(1), w(2)) ++ w(3).split(": ",2)
            else
                w)
      .map(x =>
            try {
                 torrentSchema(x(0), timeFormat.parse(x(1)), x(2), x(3), x(4))
                }
            catch {
                 case e: Exception => torrentSchema(null,null,null,null,null)
                })
      return parsedRDD
}

val filePath = "/home/radhika/Downloads/semester7/sdm/ghtorrent-logs.txt"
var Assignment_1 = parseFile(filePath)
```

---

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.

Approach :

1. invertedIndexBy() function takes a torrentSchema type RDD and inverts index on specific field
2. groupBy() is used to transform the RDD and invert the index on “download_id”
3. Inverted RDD is stored in invert_index

```
def invertedIndexBy(RDD : org.apache.spark.rdd.RDD[torrentSchema], field :
String) = {
      RDD.groupBy(entry =>
      field match {
            case "debug_level" => entry.debug_level
            case "timestamp" => entry.timestamp
            case "download_id" => entry.download_id
            case "retrieval_stage" => entry.retrieval_stage
            case "rest" => entry.rest
      }
      )
}

val invert_index = invertedIndexBy(Assignment_1, "download_id")
```

---

b. Compute the number of different repositories accessed by the client ‘ghtorrent-22’ (without using the inverted index).

- There are 4577 unique repositories for ghtorrent-22.
  Time elapsed : 16.030400845 seconds

Approach :

1. Filter out records with empty rest field value and “ghtorrent-22” download_id field value
2. All records with repository names contain “Repo” or “repos” substring. Filter out the records that do not contain either of these substrings in rest field
3. Extract repository names from the rest field using parseRepoName() function which splits the rest field using ‘/’
4. Filter out empty repo names
5. Use distinct() function to remove duplicate repository names
6. Use count() action to count the number of different repositories that are accessed by the client ‘ghtorrent-22’

```
def parseRepoName(rest : String): String = {
   try{
      if (rest.contains("github.com/repos/")) {
            var words = rest.split("github.com/repos/")(1).split("\\?")
            var path = words(0).split("/")
            if (path.size == 1) {
                  return path(0)
            }
            else if (path.size > 1) {
                  return path(0)+"/"+path(1)
            }
```

```
      }
      else {
            var words = rest.split(" ")
            var index = words.indexOf("Repo")
            return words(index + 1)
      }
      return null
   }
   catch {
      case e: Exception => return null
   }
}

val t1 = System.nanoTime
var repoCtr: Long = Assignment_1.filter { row =>
                                          row.rest != null &&
                                          row.download_id == "ghtorrent-22" &&
                                          (row.rest.contains("Repo") ||
                                           row.rest.contains("repos"))
                                         }.map(row =>
                                               parseRepoName(row.rest)
                                         ).filter(repoName =>
                                               repoName != null).distinct.count

val duration = (System.nanoTime - t1) / 1e9d

println("\nThere are " + repoCtr + " unique repositories for ghtorrent-22.\n")
println("Time elapsed : " + duration + " seconds")
```

Final Output for 2nd part of Question 3 (linux terminal):

```
scala> :load /home/radhika/Downloads/semester7/sdm/p1.scala
Loading /home/radhika/Downloads/semester7/sdm/p1.scala...
import java.util.TimeZone
import java.text.SimpleDateFormat
timeFormat: java.text.SimpleDateFormat = java.text.SimpleDateFormat@faabb35e
defined class torrentSchema
parseFile: (filename: String)org.apache.spark.rdd.RDD[torrentSchema]
parseRepoName: (rest: String)String
invertedIndexBy: (RDD: org.apache.spark.rdd.RDD[torrentSchema], field: String)org.apache.spark.rdd.RDD[(Comparable[
_ >: String with java.util.Date <: Comparable[_ >: String with java.util.Date <: java.io.Serializable] with java.io
.Serializable] with java.io.Serializable, Iterable[torrentSchema])]
filePath: String = /home/radhika/Downloads/semester7/sdm/ghtorrent-logs.txt
Assignment_1: org.apache.spark.rdd.RDD[torrentSchema] = MapPartitionsRDD[56] at map at /home/radhika/Downloads/seme
ster7/sdm/p1.scala:77
invertedRDD: org.apache.spark.rdd.RDD[(Comparable[_ >: String with java.util.Date <: Comparable[_ >: String with ja
va.util.Date <: java.io.Serializable] with java.io.Serializable] with java.io.Serializable, Iterable[torrentSchema]
)] = ShuffledRDD[58] at groupBy at /home/radhika/Downloads/semester7/sdm/p1.scala:78
client: String = ghtorrent-22
t1: Long = 979209926443112
numRepos: Long = 4577
duration: Double = 16.030400845

There are 4577 unique repositories for ghtorrent-22.

Time elapsed : 16.030400845 seconds

scala>
```

c. Compute the number of different repositories accessed by the client ‘ghtorrent-22’ using the inverted index calculated above.

- There are 4577 unique repositories for ghtorrent-22.

Approach :

1. Using the invert_index on download_id from part (a)
2. Filter out records with “ghtorrent-22” download_id field value
3. All records with repository names contain “Repo” or “repos” substring. Filter out the records that do not contain either of these substrings in rest field
4. Extract repository names from the rest field using parseRepoName() function which splits the rest field using ‘/’
5. Filter out empty repo names
6. Use distinct() function to remove duplicate repository names
7. Use count() action to count the number of different repositories that are accessed by the client ‘ghtorrent-22’

```
val occurences_list = invert_index.filter(x => x._1 == "ghtorrent-22")
                                  .flatMap(x => x._2).collect()
val count = sc.parallelize(occurences_list)
              .filter(x =>
                   (x.rest.contains("Repo") || x.rest.contains("repos")))
              .map(x => parseRepoName(x.rest))
              .filter(x => x != null)
              .distinct()
              .count()
```