

Big Data Programming - Assignment 7

Varaprasad Kurra

Panther ID : 002430487

Source Code:

```
import org.apache.spark.api.java.JavaDoubleRDD;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.mllib.recommendation.ALS;
import org.apache.spark.mllib.recommendation.MatrixFactorizationModel;
import org.apache.spark.mllib.recommendation.Rating;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.Session;
import org.apache.spark.sql.functions;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;

import scala.Tuple2;

public class SparkMovieRecommendation
{
    // original data files
    private static final String USER_URI =
"file:///C:/Users/VaraPrasad/Desktop/Summer_Semester/movie_data.data";
    private static final String MOVIE_URI =
"file:///C:/Users/VaraPrasad/Desktop/Summer_Semester/movie_data.item";
    // let us focus on this user, can change to any other user
    private static final int user = 356;
    public static void main(String[] args)
    {
        SparkSession spark =
SparkSession.builder().config("spark.master","local[*]").getOrCreate();
        JavaSparkContext sc = new JavaSparkContext(spark.sparkContext());
        sc.setLogLevel("WARN");

//-----
//Movie RDD that reads movieId and movieTitle
        JavaRDD<Movie> movieRdd = spark.read().textFile(MOVIE_URI).javaRDD().map(
            new Function<String, Movie>()
            {
                public Movie call(String moviedata)
                {
                    String[] tmpStrs = moviedata.split("\\|");
                    if ( tmpStrs != null && tmpStrs.length >= 2 && tmpStrs[0] != null
&& tmpStrs[1] != null )
                    {

                        return new Movie(tmpStrs[0],tmpStrs[1]);
                    }
                }
            }
        );
    }
}
```

```

        }
        else
            return null;
    }
});

Dataset<Row> movieDS = spark.createDataFrame(movieRdd.rdd(), Movie.class);
movieDS.show();
movieDS.createOrReplaceTempView("movies");

//-----

JavaRDD<Rating> ratingsRdd = spark.read().textFile(USER_URI).javaRDD().map(
    new Function<String, Rating>()
    {
        public Rating call(String userRating)
        {
            String[] tmpStrs = userRating.split("\t");
            if ( tmpStrs != null && tmpStrs.length >= 3 && tmpStrs[0] != null
            && tmpStrs[1] != null && tmpStrs[2] != null )
            {
                int userId = Integer.parseInt(tmpStrs[0]);
                int movieId = Integer.parseInt(tmpStrs[1]);
                double rating = Double.parseDouble(tmpStrs[2]);

                return new Rating(userId, movieId, rating);
            } else return null;
        }
    }
);

System.out.println(ratingsRdd.take(20).toString());

// create a DataFrame representing ratings, but using a different way
StructType ratingSchema = new StructType(
    new StructField[] {
        DataTypes.createStructField("userId", DataTypes.IntegerType, false),
        DataTypes.createStructField("movieId", DataTypes.IntegerType, false),

        DataTypes.createStructField("rating", DataTypes.DoubleType, false)
    });

Dataset<Row> originalRatingsMatrix = spark.createDataFrame(
    ratingsRdd.map( new Function<Rating, Row>()
    {
        @Override
        public Row call(Rating record)
        {
            return RowFactory.create(record.user(), record.product(), record.rating());
        }
    } ), ratingSchema);

originalRatingsMatrix.show(40);
originalRatingsMatrix.createOrReplaceTempView("originalRatings");
//-----

```

```

//Joining the User and Movie DataFrame
spark.sql("select r.userId,r.rating,m.movieTitle from originalRatings
r,movies m "
+ "where r.userId = " + user + " and r.movieId = m.movieId order by r.rating
desc").show(40);

//-----
Dataset<Row> first_Set = spark.sql("select m.movieId,r.userId,r.rating from
originalRatings r,movies m "
+ "where r.userId = " + user + " and r.movieId =
m.movieId order by r.rating desc");

Dataset<Row> second_Set = spark.sql("select m.movieId from movies m "
+ "where m.movieTitle NOT IN (select m.movieTitle
from originalRatings r,movies m "
+ "where r.userId = " + user + " and r.movieId =
m.movieId)");
second_Set = second_Set.withColumn("userId", functions.lit(0));
second_Set = second_Set.withColumn("rating", functions.lit(0));
System.out.println("First set that is with USER 356");
first_Set.show(40);
System.out.println("Second set that is without USER 356 and creating a
Dataset with 0 for userId and rating");
second_Set.show(40);
System.out.println("Generating an input for Union");
Dataset<Row> union_Set = first_Set.union(second_Set);
union_Set.show(40);
union_Set.createOrReplaceTempView("union_Set");

//-----
JavaRDD<Rating> inputForPred = union_Set.toJavaRDD().map(new
Function<Row,Rating>(){
    public Rating call(Row line) {
        int movieId = Integer.parseInt((String) line.get(0));
        int userId = (int) line.get(1);
        double rating = 0.0;
        return new Rating(userId, movieId, rating);
    }
});

JavaRDD<Tuple2<Object, Object>> finalPredInput =
inputForPred.map(new Function<Rating,Tuple2<Object,Object>>(){
    public Tuple2<Object, Object> call(Rating r) {
        return new Tuple2<Object,
Object>(r.user(),r.product());
    }
});

//-----
//Input parameters to the Model Training
int rank = 10;
int numIterations = 10;
double lambda = 0.3;

//Training the model with existing Rating Data

```

```

MatrixFactorizationModel model = ALS.train(JavaRDD.toRDD(ratingsRdd), rank,
numIterations, lambda);

// Applying the model on the new Data
JavaRDD<Rating> finalPrediction =
model.predict(JavaRDD.toRDD(finalPredInput)).toJavaRDD();
    //System.out.println(finalPrediction.take(20).toString());

//check the predicted ratings for the same user, just to compare
Dataset<Row> predictedRatingsMatrix = spark.createDataFrame(
    finalPrediction.map( new Function<Rating, Row>() {
        @Override
        public Row call(Rating r) {
            return RowFactory.create(r.user(), r.product(), r.rating());
        }
    } ), ratingSchema);

predictedRatingsMatrix.createOrReplaceTempView("FinalRating");
    spark.sql("select r.userId as userId0, u.rating as rating0,
r.rating, m.movieTitle from FinalRating r,movies m, union_Set u "
        + "where r.movieId = m.movieId and u.movieId =
r.movieId order by r.rating desc").show();

    spark.close();
}
}

```

Output:

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with packages like `JavaMinHashLSH`, `SparkMinHashLSH`, and `SparkMovieRecommendation`.
- Editor:** Displays the `SparkRecommendationSystem.java` file. The code includes:
 - Line 114: `System.out.println("Generating an input for Union");`
 - Line 116: `Dataset<Row> union_Set = first_Set.union(second_Set);`
 - Line 117: `union_Set.show(40);`
 - Line 118: `union_Set.createOrReplaceTempView("union_Set");`
 - Line 122: `JavaRDD<Rating> inputForPred = union_Set.toJavaRDD().map(new Function<Row, Rating>() {`
 - Line 123: `public Rating call(Row line) {`
 - Line 124: `int movieId = Integer.parseInt((String) line.get(0));`
 - Line 125: `int userId = (int) line.get(1);`
 - Line 126: `double rating = 0.0;`
 - Line 127: `return new Rating(userId, movieId, rating);`
 - Line 128: `}`
 - Line 129: `});`
 - Line 132: `JavaRDD<Tuple2<Object, Object>> finalPredInput = inputForPred.map(new Function<Rating, Tuple2<Object, Object>>() {`
 - Line 133: `public Tuple2<Object, Object> call(Rating r) {`
 - Line 134: `return new Tuple2<Object, Object>(r.user(), r.product());`
 - Line 135: `}`
 - Line 136: `});`
- Console:** Shows the output of the `union_Set.show(40);` command, displaying 40 rows of data with columns `userId`, `rating`, and `movieTitle`.

Top 20 Observations of the Output:

userId	rating	movieTitle
0	0.0	4.44279232316992 Great Day in Harl...
0	0.0	4.294813740107378 Santa with Muscle...
0	0.0	4.246120769251345 Someone Else's Am...
0	0.0	4.204483274967663 Saint of Fort Was...
0	0.0	4.17622469397975 Aiqing wansui (1994)
0	0.0	4.113131769846532 Some Mother's Son...
0	0.0	4.003211972399433 Pather Panchali (...
356	5.0	3.9951365241864947 Good Will Hunting...
0	0.0	3.9656305927580835 Anna (1996)
356	5.0	3.9533318568821683 Titanic (1997)
356	4.0	3.92597998914014 As Good As It Get...
0	0.0	3.915660908497915 They Made Me a Cr...
0	0.0	3.8981258980416964 Spanish Prisoner,...
0	0.0	3.8981258980416964 Butcher Boy, The ...
0	0.0	3.8981258980416964 Brothers in Troub...
0	0.0	3.8981258980416964 Butcher Boy, The ...
0	0.0	3.8845404201586136 Entertaining Ange...
0	0.0	3.7920094306341365 Star Kid (1997)
0	0.0	3.7839810558168105 Marlene Dietrich:...
0	0.0	3.778125874977406 Close Shave, A (1...

only showing top 20 rowsS