Big Data Programming Assignment 6

Varaprasad Kurra Panther ID : 002430487

Source Code:

```
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.FilterFunction;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.api.java.function.PairFunction;
import org.apache.spark.ml.feature.Binarizer;
import org.apache.spark.ml.feature.CountVectorizer;
import org.apache.spark.ml.feature.CountVectorizerModel;
import org.apache.spark.ml.feature.IDF;
import org.apache.spark.ml.feature.IDFModel;
import org.apache.spark.ml.feature.MinHashLSH;
import org.apache.spark.ml.feature.MinHashLSHModel;
import org.apache.spark.ml.feature.Normalizer;
import org.apache.spark.ml.feature.StopWordsRemover;
import org.apache.spark.ml.linalg.SparseVector;
import org.apache.spark.mllib.linalg.distributed.CoordinateMatrix;
import org.apache.spark.mllib.linalg.distributed.IndexedRow;
import org.apache.spark.mllib.linalg.distributed.IndexedRowMatrix;
import org.apache.spark.mllib.linalg.distributed.MatrixEntry;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;

import antlr.collections.impl.Vector;
import scala.Tuple2;

public class SparkRecommendationSystem {

// change this to your own file path
private static final String FILE_URI =
"file:///C:/Users/VaraPrasad/Desktop/Summer_Semester/StackOverFlow_Files";
private static final String TARGET_URI =
"file:///C:/Users/VaraPrasad/Desktop/Summer_Semester/StackOverFlow_Target";

public static void main(String[] args) {
            // initializing spark
            SparkSession spark =
SparkSession.builder().config("spark.master","local[*]").getOrCreate();
            JavaSparkContext sc = new JavaSparkContext(spark.sparkContext());
            sc.setLogLevel("WARN");
            // create RDD by reading text files
            JavaPairRDD<String,String> documents=sc.wholeTextFiles(FILE_URI);
      System.out.println(documents.take((int)documents.count()).toString());
```

```
// create RDD by reading text files - The Target File
     JavaPairRDD<String,String> tdocuments = sc.wholeTextFiles(TARGET_URI);
     System.out.println("");
     System.out.println("Current Target File or Key File is");
     System.out.println(tdocuments.take((int)tdocuments.count()).toString());

// break each document into words
JavaPairRDD<Tuple2<String, String[]>, Long> wDocuments = documents.mapValues(
new Function<String, String[]>()
{
 public String[] call(String line) throws Exception {
      return line.split("\\W+");   // use the following for English
}
            } ).zipWithIndex();
System.out.println(wDocuments.take((int)wDocuments.count()).toString());


// break each document into words  Target file
JavaPairRDD<Tuple2<String, String[]>, Long> twDocuments =
tdocuments.mapValues( new Function<String, String[]>()
                              {
                  public String[] call(String line) throws Exception
                                    {
                                          return line.split("\\W+");
                                    }
                              } ).zipWithIndex();

      System.out.println(twDocuments.take((int)twDocuments.count()).toString(
));

            // load wDocuments into dataframe
StructType schema = new StructType(new StructField[] {
DataTypes.createStructField("docID", DataTypes.LongType, false),
DataTypes.createStructField("file_path", DataTypes.StringType, false),

      DataTypes.createStructField("all_words",DataTypes.createArrayType(DataT
ypes.StringType, false),false)
                              });
Dataset<Row> documentsWithAllWords = spark.createDataFrame(
                              wDocuments.map( new
Function<Tuple2<Tuple2<String,String[]>,Long>, Row>() {
                                    @Override
public Row call(Tuple2<Tuple2<String,String[]>, Long> record) {
      return
RowFactory.create(record._2(),record._1()._1().substring(record._1._1().lastI
ndexOf("/")+1), record._1()._2());
                                    }
                              } ), schema);
documentsWithAllWords.show(true);

Dataset<Row> tdocumentsWithAllWords = spark.createDataFrame(
                              twDocuments.map( new
Function<Tuple2<Tuple2<String,String[]>,Long>, Row>() {
                                    @Override
public Row call(Tuple2<Tuple2<String,String[]>, Long> record) {
```

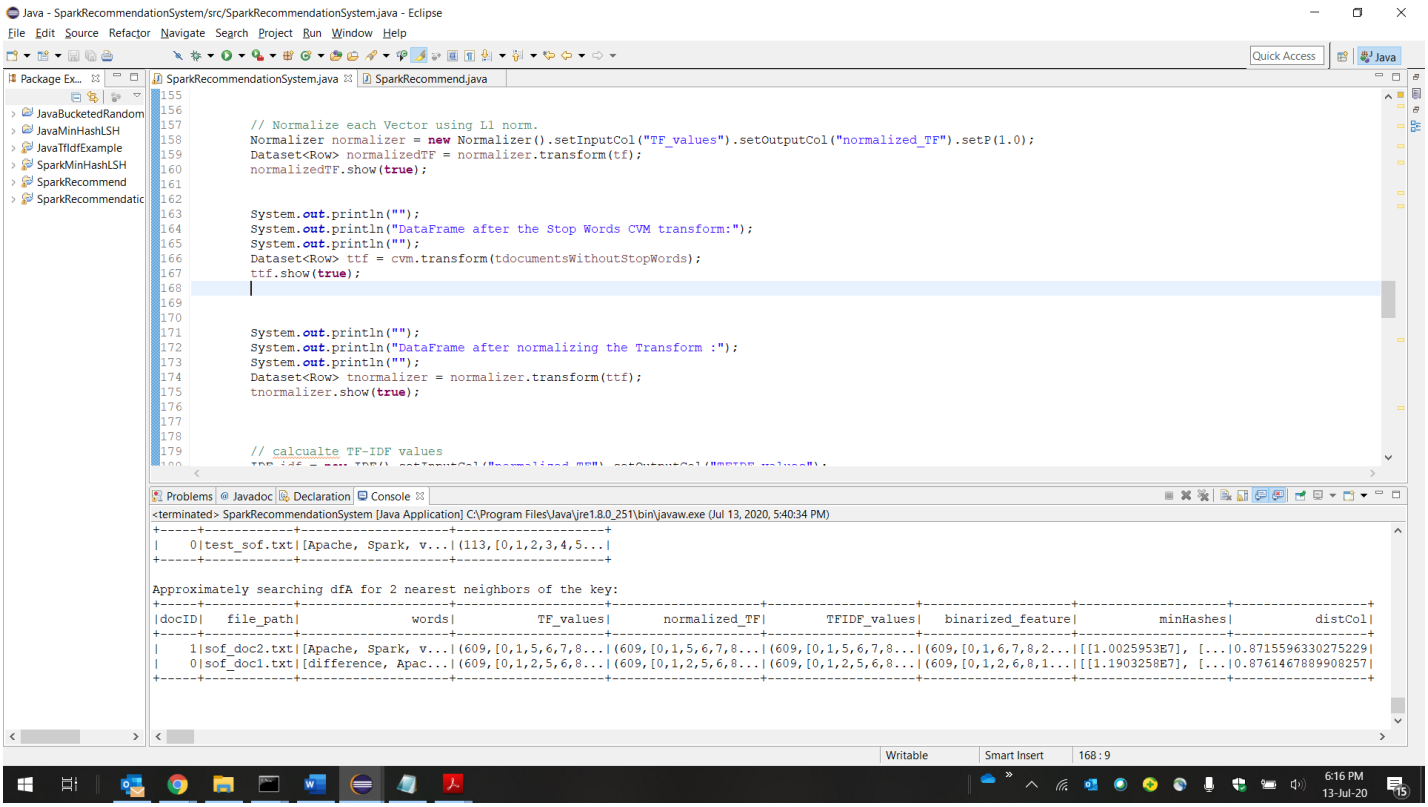
```
return
RowFactory.create(record._2(),record._1()._1().substring(record._1._1().lastI
ndexOf("/")+1), record._1()._2());
                                }
                        } ), schema);

System.out.println("");
System.out.println("");
System.out.println("DataFrame is :");
System.out.println("");
tdocumentsWithAllWords.show(true);

// remove stop words
StopWordsRemover remover = new
StopWordsRemover().setInputCol("all_words").setOutputCol("words");
Dataset<Row> documentsWithoutStopWords =
remover.transform(documentsWithAllWords).select("docID",
"file_path","words");
documentsWithoutStopWords.show(true);
System.out.println("DataFrame after the Stop Words :");
System.out.println("");
// remove stop words
Dataset<Row> tdocumentsWithoutStopWords =
remover.transform(tdocumentsWithAllWords).select("docID",
"file_path","words");
tdocumentsWithoutStopWords.show(true);


// fit a CountVectorizerModel from the corpus
CountVectorizer vectorizer = new
CountVectorizer().setInputCol("words").setOutputCol("TF_values");
CountVectorizerModel cvm = vectorizer.fit(documentsWithoutStopWords);
System.out.println("vocab size = " + cvm.vocabulary().length);
for (int i = 0; i < cvm.vocabulary().length; i ++ ) {
      System.out.print(cvm.vocabulary()[i] + "(" + i + ") ");
}
System.out.println();
Dataset<Row> tf = cvm.transform(documentsWithoutStopWords);
tf.show(true);


System.out.println("");
System.out.println("");
System.out.println("Count Vectorizer for the Vocab Size and Words");
System.out.println("");
CountVectorizer tvectorizer = new
CountVectorizer().setInputCol("words").setOutputCol("TF_values");
CountVectorizerModel tcvm = tvectorizer.fit(tdocumentsWithoutStopWords);

            System.out.println("vocab size = " + tcvm.vocabulary().length);
            for (int i = 0; i < tcvm.vocabulary().length; i ++ )
                  {
                        System.out.print(tcvm.vocabulary()[i] + "(" + i + ")
");
                  }
            System.out.println();
```

```
 // Normalize each Vector using L1 norm.
Normalizer normalizer = new
Normalizer().setInputCol("TF_values").setOutputCol("normalized_TF").setP(1.0)
;
Dataset<Row> normalizedTF = normalizer.transform(tf);
normalizedTF.show(true);
System.out.println("");
System.out.println("DataFrame after the Stop Words CVM transform:");
System.out.println("");
Dataset<Row> ttf = cvm.transform(tdocumentsWithoutStopWords);
ttf.show(true);

System.out.println("");
System.out.println("DataFrame after normalizing the Transform :");
System.out.println("");
Dataset<Row> tnormalizer = normalizer.transform(ttf);
tnormalizer.show(true);



// calcualte TF-IDF values
IDF idf = new
IDF().setInputCol("normalized_TF").setOutputCol("TFIDF_values");
          IDFModel idfModel = idf.fit(normalizedTF);
          Dataset<Row> tf_idf = idfModel.transform(normalizedTF);
          tf_idf.select("docID", "file_path", "words",
"TFIDF_values").show(true);


            System.out.println("DataFrame with TFIDF values :");
            System.out.println("");
            Dataset<Row> tTFIDF =
idf.fit(normalizedTF).transform(tnormalizer);
            tTFIDF.select("docID", "file_path", "words",
"TFIDF_values").show(true);

            //To implement KNN, the approxNearestNeighbors function accepts
the Vectors as it's input.
            //converting  the column of a DataFrame i.e TF_IDF values into a
Dense / Sparse Vector so as to fed into the model.
            //tRDD - Target or Key RDD for the KNN Model

          JavaRDD<SparseVector> TargetKey = tTFIDF.toJavaRDD().map(new
Function<Row, SparseVector>()
                        {
                              public SparseVector call(Row KeyEntryRow)
throws Exception
                              {
                                    return (SparseVector) KeyEntryRow.get(4);
                              }
                        }
                        );

          System.out.println("");
            System.out.println("Target Key is :");

      System.out.println(TargetKey.take((int)TargetKey.count()).toString());
```

```
          Binarizer binarizer = new
Binarizer().setInputCol("TFIDF_values").setOutputCol("binarized_feature").set
Threshold(0.001);
          Dataset<Row> binarizedDataFrame = binarizer.transform(tf_idf);


          Binarizer tbinarizer = new
Binarizer().setInputCol("TFIDF_values").setOutputCol("binarized_feature").set
Threshold(0.001);
          Dataset<Row> tbinarizedDataFrame = tbinarizer.transform(tTFIDF);


          System.out.println("Binarizer output with Threshold = " +
binarizer.getThreshold());
          binarizedDataFrame.show(true);

            System.out.println("Binarizer output with Threshold = " +
binarizer.getThreshold());
            tbinarizedDataFrame.show(true);


          MinHashLSH mh = new
MinHashLSH().setNumHashTables(100).setInputCol("binarized_feature").setOutput
Col("minHashes");
          MinHashLSHModel model = mh.fit(binarizedDataFrame);

          Dataset<Row> mh_data = model.transform(binarizedDataFrame);
          mh_data.select("docID", "file_path", "words",
"TFIDF_values","binarized_feature").show(true);

                  Dataset<Row> tttf =
tcvm.transform(tdocumentsWithoutStopWords);
                  tttf.show(true);

                  System.out.println("Approximately searching dfA for 2
nearest neighbors of the key:");

                  //Using first() function to get the SparseVector
                  //binarizedDataFrame since approxNearestNeighbors accepts
DataSet as its first argument

      model.approxNearestNeighbors(binarizedDataFrame,TargetKey.first() ,
2).show();


                   JavaRDD<IndexedRow> rddIndexRows =
tf_idf.toJavaRDD().map(new Function<Row,IndexedRow>()
                                  {
                                     public IndexedRow call(Row row) throws
Exception
                                           {
                                                 Object features =
row.getAs("TFIDF_values");

      org.apache.spark.ml.linalg.DenseVector dense = null;
if (features instanceof org.apache.spark.ml.linalg.DenseVector)
```

```
            {
      dense = (org.apache.spark.ml.linalg.DenseVector)features;
                                          }
      else if(features instanceof org.apache.spark.ml.linalg.SparseVector)
                  {
                        org.apache.spark.ml.linalg.SparseVector sparse =
(org.apache.spark.ml.linalg.SparseVector)features;
                                    dense = sparse.toDense();
                                    }
                              else
                                    {
                                    RuntimeException e = new
RuntimeException("Cannot convert to "+
features.getClass().getCanonicalName());
                                    throw e;
                                          }
                              org.apache.spark.mllib.linalg.Vector vec =
org.apache.spark.mllib.linalg.Vectors.dense(dense.toArray());
                              return new IndexedRow((long)
row.getAs("docID"), vec);
                                          }
                                 });


            spark.close();
      }
}
```

Output:


Java - SparkRecommendationSystem/src/SparkRecommendationSystem.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Ex...
JavaBucketedRandom
JavaMinHashLSH
JavaTfIdfExample
SparkMinHashLSH
SparkRecommend
SparkRecommendatic
SparkRecommendationSystem.java
SparkRecommend.java
// Normalize each Vector using L1 norm.
Normalizer normalizer = new Normalizer().setInputCol("TF_values").setOutputCol("normalized_TF").setP(1.0);
Dataset<Row> normalizedTF = normalizer.transform(tf);
normalizedTF.show(true);
System.out.println("");
System.out.println("DataFrame after the Stop Words CVM transform:");
System.out.println("");
Dataset<Row> ttf = cvm.transform(tdocumentsWithoutStopWords);
ttf.show(true);
System.out.println("");
System.out.println("DataFrame after normalizing the Transform :");
System.out.println("");
Dataset<Row> tnormalizer = normalizer.transform(ttf);
tnormalizer.show(true);
// calcualte TF-IDF values
Problems Javadoc Declaration Console
<terminated> SparkRecommendationSystem [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (Jul 13, 2020, 5:40:34 PM)
+-----+------------+--------------------+--------------------+
| 0|test_sof.txt|[Apache, Spark, v...|(113,[0,1,2,3,4,5...|
+-----+------------+--------------------+--------------------+
Approximately searching dfA for 2 nearest neighbors of the key:
+-----+------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|docID| file_path| words| TF_values| normalized_TF| TFIDF_values| binarized_feature| minHashes| distCol|
+-----+------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
| 1|sof_doc2.txt|[Apache, Spark, v...|(609,[0,1,5,6,7,8...|(609,[0,1,5,6,7,8...|(609,[0,1,5,6,7,8...|(609,[0,1,6,7,8,2...|[[1.0025953E7], [...|0.8715596330275229|
| 0|sof_doc1.txt|[difference, Apac...|(609,[0,1,2,5,6,8...|(609,[0,1,2,5,6,8...|(609,[0,1,2,5,6,8...|(609,[0,1,2,6,8,1...|[[1.1903258E7], [...|0.8761467889908257|
+-----+------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
Writable
Smart Insert
168 : 9
6:16 PM
13-Jul-20