**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Map;

**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.FlatMapFunction;

**import** org.apache.spark.api.java.function.Function;

**import** org.apache.spark.api.java.function.Function2;

**import** org.apache.spark.api.java.function.PairFlatMapFunction;

**import** org.apache.spark.api.java.function.PairFunction;

**import** org.apache.spark.api.java.function.VoidFunction;

**import** org.apache.spark.broadcast.Broadcast;

**import** org.apache.spark.ml.feature.CountVectorizer;

**import** org.apache.spark.ml.feature.CountVectorizerModel;

**import** org.apache.spark.ml.linalg.Vector;

**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** scala.Tuple2;

// this is the example in Chap 3, Example 3.6

// for the related homework question, see SparkBruteForceSimilarity

**public** **class** SparkMinHashLSH {

//C:/Users/89762/Desktop/CSC4760/HW5/LSH\_\*.txt

**private** **static** **final** String ***FILE\_URI*** = "C:/Users/89762/Desktop/CSC4760/HW5/LSH\_\*.txt";

**private** **static** **final** **double** ***sizeAdj*** = 1.0;

**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 using text files

JavaPairRDD<String,String> documents = sc.wholeTextFiles(***FILE\_URI***);

// System.out.println(documents.take((int)documents.count()).toString());

// convert original documents into shingle representation

**class** ShinglesCreator **implements** Function<String,String[]> {

@Override

**public** String[] call(String text) **throws** Exception {

**return** ShingleUtils.*getTextShingles*(text);

}

}

JavaPairRDD<String,String[]> shinglesDocs = documents.mapValues(**new** ShinglesCreator());

shinglesDocs.values().foreach(**new** VoidFunction<String[]>() {

**public** **void** call(String[] shingles) **throws** Exception {

**for** ( **int** i = 0; i < shingles.length; i ++ ) {

System.***out***.print(shingles[i] + "|");

}

System.***out***.println();

}

});

// create characteristic matrix representation of each document

StructType schema = **new** StructType(

**new** StructField[] {

DataTypes.*createStructField*("file\_path", DataTypes.***StringType***, **false**),

DataTypes.*createStructField*("file\_content",DataTypes.*createArrayType*(DataTypes.***StringType***, **false**),**false**)

});

Dataset<Row> df = spark.createDataFrame(

shinglesDocs.map( **new** Function<Tuple2<String, String[]>, Row>() {

@Override

**public** Row call(Tuple2<String, String[]> record) {

**return** RowFactory.*create*(record.\_1().substring(record.\_1().lastIndexOf("/")+1), record.\_2());

}

} ), schema);

df.show(**true**);

CountVectorizer vectorizer = **new** CountVectorizer().setInputCol("file\_content").setOutputCol("feature\_vector").setBinary(**true**);

CountVectorizerModel cvm = vectorizer.fit(df);

Broadcast<Integer> vocabSize = sc.broadcast(cvm.vocabulary().length);

System.***out***.println("vocab size = " + cvm.vocabulary().length);

**for** (**int** i = 0; i < vocabSize.value(); i ++ ) {

System.***out***.print(cvm.vocabulary()[i] + "(" + i + ") ");

}

System.***out***.println();

Dataset<Row> characteristicMatrix = cvm.transform(df);

characteristicMatrix.show(**false**);

// following is the new code added

// the following block is to rebuild a detaset with required field name

Dataset<Row> matrix = characteristicMatrix

.crossJoin(characteristicMatrix)

.toDF("file\_path1", "two", "feature\_vector1", "file\_path2","five","feature\_vector2");

// the following is to convert the dataset to a JavaPairRDD

JavaPairRDD<String,Double> minhashSignature = matrix.toJavaRDD().mapToPair(**new** PairFunction<Row, String, Double>() {

**public** Tuple2<String, Double> call(Row row) **throws** Exception {

//retrieve data from dataset and convert to array

**double**[] Row1 = ((Vector)row.getAs("feature\_vector1")).toArray();

**double**[] Row2 = ((Vector)row.getAs("feature\_vector2")).toArray();

**double** a=0.0;

**double** b=0.0;

//compare the two arrays and return the similarity:

//if both=0, not count; if both=1, union+1; otherwise unique+1

**for**(**int** i=0;i<128;i++) {

**if** (Row1[i]==1&&Row2[i]==0) {

b++;

}**else** **if**(Row1[i]==0&&Row2[i]==1) {

b++;

}**else** **if**(Row1[i]==1&&Row2[i]==1) {

a++;

}**else** {

}

}

//compare the filenames, only return the valid pairs.

String fileName1=(String) row.getAs("file\_path1");

String fileName2=(String) row.getAs("file\_path2");

**if**(!fileName1.equals(fileName2)) {

**int** j = fileName1.compareTo(fileName2);

**if**(j<0) {

//connect the filenames with "-"

**return** **new** Tuple2<String, Double>((String) row.getAs("file\_path1")+"-"+(String) row.getAs("file\_path2"), a/(a+b));

}**else** {

//give the repeated pairs constant value, in order to filter them later.

**return** **new** Tuple2<String, Double>("",0.0);

}

}

**return** **new** Tuple2<String, Double>("",0.0);

}

});

//use filter to eliminate the invalid pairs

JavaPairRDD<String,Double> sim = minhashSignature.filter(**new** Function<Tuple2<String, Double>,Boolean>() {

**public** Boolean call(Tuple2<String, Double> bucketDocument) {

**if** ( bucketDocument.\_1.contains("-") ) **return** **true**;

**else** **return** **false**;

}

});

System.***out***.println("===> FINAL:");

System.***out***.println(sim.take((**int**)minhashSignature.count()).toString());

vocabSize.unpersist();

vocabSize.destroy();

sc.close();

}

}