Homework 6

You will need to submit the following (**100** points total):

1. *all* your source code, with necessary comments so I know what parts you have added. (90 points)
2. a screen shot showing the result of running the code (10 points)

below is the code to use (I am pasting the whole class here, look for the comment for this new code the students should write):

```
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.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.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;

public class SparkRecommendationNearestNeighborSearch {

	// below should be their file paths, each student should be using his/her own file paths
	private static final String FILE_URI = "file:///f:/my talks and teaching/New folder/examples/datasets/sof_*.txt";
	private static final String TEST_URI = "file:///f:/my talks and teaching/New folder/examples/datasets/test_sof.txt";

	public static void main(String[] args) {

		// class name
```

```
System.out.println(SparkRecommendationNearestNeighborSearch.class.getCanonicalName());

// 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());

// 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
            // return line.split("\\|");  // use the following for Chinese
      }
} ).zipWithIndex();
System.out.println(wDocuments.take((int)wDocuments.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(DataTypes.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().lastIndexOf("/")+1), record._1()._2());
                  }
            } ), schema);
documentsWithAllWords.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");
System.out.println("everything without stop words: ");
documentsWithoutStopWords.show(true);

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);

// 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);

// 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(false);

   // FROM HERE on, should be the new code by the student
   // the YELLOW bars are the key language constructs one should use


Binarizer binarizer = new
         Binarizer().setInputCol("TFIDF_values").setOutputCol("binarized_feature").setThreshold(0.0001);
Dataset<Row> binarizedDataFrame = binarizer.transform(tf_idf);

// for debug purpose
System.out.println("Binarizer output with Threshold = " + binarizer.getThreshold());
binarizedDataFrame.select("docID", "file_path", "words", "TFIDF_values","binarized_feature").show(false);

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

// Feature Transformation
System.out.println("The hashed dataset where hashed values are stored in the column 'hashes':");
model.transform(binarizedDataFrame).show(false);

// prepare the test document, this is just repeat every step from the above

JavaPairRDD<Tuple2<String, String[]>, Long>
         newDoc = sc.wholeTextFiles(TEST_URI).mapValues( new Function<String, String[]>() {
         public String[] call(String line) throws Exception {
               return line.split("\\W+");   // use the following for English
```

```
                // return line.split("\\|");  // use the following for Chinese
            }
        } ).zipWithIndex();
        System.out.println(newDoc.take((int)newDoc.count()).toString());

        Dataset<Row> newDocWithAllWords = spark.createDataFrame(
                newDoc.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().lastIndexOf("/")+1), record._1()._2());
                                }
                } ), schema);
        newDocWithAllWords.show(true);

        // remove stop words
        Dataset<Row> newDocWithoutStopWords = remover.transform(newDocWithAllWords).select("docID", "file_path","words");
        System.out.println("everything without stop words: ");
        newDocWithoutStopWords.show(true);

        // calculate TF.IDF
Dataset<Row> newDocTF = cvm.transform(newDocWithoutStopWords);
newDocTF.show(false);
Dataset<Row> normalizedNewDocTF = normalizer.transform(newDocTF);
normalizedNewDocTF.show(true);
Dataset<Row> newDocTFIDF = idf.fit(normalizedTF).transform(normalizedNewDocTF);
newDocTFIDF.select("docID", "file_path", "words", "TFIDF_values").show(true);

// prepare the key
Dataset<Row> newFileKey = binarizer.transform(newDocTFIDF);
System.out.println("Binarizer output with Threshold = " + binarizer.getThreshold());
newFileKey.select("docID", "file_path", "words", "TFIDF_values","binarized_feature").show(false);

JavaRDD<SparseVector> testRDD = newFileKey.toJavaRDD().map(new Function<Row, SparseVector>() {
    public SparseVector call(Row row) throws Exception {
        return (SparseVector) row.get(4);
    }
});
System.out.println(testRDD.first().toString());

// approximate nearest neighbor search
System.out.println("Approximately searching dataset for 2 nearest neighbors of the give test file:");
model.approxNearestNeighbors(binarizedDataFrame, testRDD.first(), 2).select("docID", "file_path", "distCol").show();

// https://datascience.stackexchange.com/questions/13347/calculate-cosine-similarity-in-apache-spark
```

```
            spark.close();
        }

}
```

NOTE:

1. the student ***can*** use other ways to do this, but the final screen copy should have (very) close results
2. even they use a different flow/way of doing this, the YELLOW bars (see above) should be used

here is my final screen shot (result) – make sure they should be fairly close results:

```
Approximately searching dataset for 2 nearest neighbors of the give test file:
+-----+------------+------------------+
|docID|   file_path|           distCol|
+-----+------------+------------------+
|    1|sof_doc2.txt|0.8715596330275229|
|    0|sof_doc1.txt|0.8727272727272728|
+-----+------------+------------------+
```

so sof_doc2 is the number 1 recommendation, and sof_doc1 is the second.