```
package it.polito.bigdata.spark.sparkmllib;

import java.util.ArrayList;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.mllib.fpm.AssociationRules;
import org.apache.spark.mllib.fpm.FPGrowth;
import org.apache.spark.mllib.fpm.FPGrowth.FreqItemset;
import org.apache.spark.mllib.fpm.FPGrowthModel;

public class SparkDriver {

  public static void main(String[] args) {
    String inputFile;
    String outputFolderItemsets;
    String outputFolderRules;
    double minSupport;
    double minConfidence;

    inputFile = args[0];
    outputFolderItemsets = args[1];
    outputFolderRules = args[2];
    minSupport = Double.parseDouble(args[3]);
    minConfidence = Double.parseDouble(args[4]);


      // Create a configuration object and set the name of the application
      SparkConf conf=new SparkConf().setAppName("Itemset and Association rule
mining");

      // Create a Spark Context object
      JavaSparkContext sc = new JavaSparkContext(conf);

    JavaRDD<String> inputRDD = sc.textFile(inputFile);

    // Read the input data
    // Each line of the input file contains one transaction,
    // i.e., a list of items
    // Each element of the JavaRDD represents one transaction = a list of strings
    JavaRDD<ArrayList<String>>transactions =inputRDD.map(new InputRecord());

    // Create an FPGrowth object
    FPGrowth fp=new FPGrowth();

    // Set the minimum support (frequency threshold)
    fp.setMinSupport(minSupport);

    // Run the algorithm
    FPGrowthModel<String> model = fp.run(transactions);

    // Retrieve the extracted itemsets
    JavaRDD<FreqItemset<String>> freqItemsets=model.freqItemsets().toJavaRDD();

    // FreqItemset<String> is a complex data type. It contains a list of items
    // and the frequency of that set
    // The following map is used to map each FreqItemset object to a String
    // containing the set of items and the support using the format
```

```
      // [list of items], frequency=frequency
    JavaRDD<String> freqItemsetsString=freqItemsets.map(new
ItemsetAsSingleString());

    freqItemsetsString.saveAsTextFile(outputFolderItemsets);

    // Association rule mining from the set of frequent itemsets mined
    // by using FPGrowth
    // Instance an AssociationRules object
    AssociationRules arules = new AssociationRules();

    // Set the minimum confidence threshold
    arules.setMinConfidence(minConfidence);

    // Extract the rules
    JavaRDD<AssociationRules.Rule<String>> assRules = arules.run(freqItemsets);

    // AssociationRules.Rule<String> is a complex data type.
    // The following map is used to map each AssociationRules.Rule
    // object to a String
    JavaRDD<String> assRulesString=assRules.map(new AssRuleToString());

    assRulesString.saveAsTextFile(outputFolderRules);

    sc.close();
  }
}
```

```
package it.polito.bigdata.spark.sparkmllib;

import java.util.ArrayList;
import org.apache.spark.api.java.function.Function;

@SuppressWarnings("serial")
public class InputRecord implements Function<String, ArrayList<String>> {

	public ArrayList<String> call(String line) {
		ArrayList<String> itemsList=new ArrayList<String>();

		String[] items = line.split(" ");

		for (String item: items) {
			itemsList.add(item);
		}

		return itemsList;
	}

}
```

```
package it.polito.bigdata.spark.sparkmllib;

import org.apache.spark.api.java.function.Function;
import org.apache.spark.mllib.fpm.FPGrowth.FreqItemset;

@SuppressWarnings("serial")
public class ItemsetAsSingleString implements Function<FreqItemset<String>, String>
{

	@Override
	public String call(FreqItemset<String> itemset) {
		return new String("["+ itemset.javaItems()+"],
frequency="+itemset.freq());
	}

}
```

```
package it.polito.bigdata.spark.sparkmllib;

import org.apache.spark.api.java.function.Function;
import org.apache.spark.mllib.fpm.AssociationRules.Rule;

@SuppressWarnings("serial")
public class AssRuleToString implements Function<Rule<String>, String> {

      public String call(Rule<String> rule) throws Exception {


            return new String("{"+rule.javaAntecedent()+"} -> {"+
                                    rule.javaConsequent()+"} "+
                                    "confidence="+ rule.confidence());
      }
}
```