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

import org.apache.spark.api.java.*;
import org.apache.spark.sql.DataFrame;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.types.Metadata;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;
import org.apache.spark.mllib.linalg.VectorUDT;
import org.apache.spark.ml.Pipeline;
import org.apache.spark.ml.PipelineModel;
import org.apache.spark.ml.PipelineStage;
import org.apache.spark.ml.clustering.KMeans;


import org.apache.spark.SparkConf;

public class SparkDriver {


      public static void main(String[] args) {

            String inputFileTraining;
            String outputPath;

            inputFileTraining=args[0];
            outputPath=args[1];

            // Create a configuration object and set the name of the application
            SparkConf conf=new SparkConf().setAppName("MLlib - k-means");

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

            // Create an SQLContext
            SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);

            // Read training data from a textual file
            // Each lines has the format: class-label,list of numerical attribute
values
            // E.g., 1,1.0,5.0,4.5,1.2
            JavaRDD<String> inputData=sc.textFile(inputFileTraining);

            // Map each element (each line of the input file)on a Vector
            JavaRDD<Row> inputRDD=inputData.map(new InputRecord());

            // Define a DataFrame base on the input data.
            StructField[] fields = {new StructField("features", new VectorUDT(),
false, Metadata.empty())};
            StructType schema = new StructType(fields);

            DataFrame data = sqlContext.createDataFrame(inputRDD, schema).cache();

            // Create a k-means object.
            // k-means is an Estimator that is used to
            // create a k-means algorithm
            KMeans km = new KMeans();
```

```
            // Set the value of k
            km.setK(2);


            // Define the pipeline that is used to cluster
            // the input data
            // In this case the pipeline contains one single stage/step (the model
            // generation step).
            Pipeline pipeline = new Pipeline()
                        .setStages(new PipelineStage[] {km});

            // Execute the pipeline on the data to build the
            // clustering model
            PipelineModel model = pipeline.fit(data);


            // Now the clustering model can be applied on the data
            // to assign them to a cluster (i.e., assign a cluster id)
            // The returned DataFrame has the following schema (attributes)
            // - features: vector (values of the attributes)
            // - prediction: double (the predicted cluster id)
            DataFrame clusteredData = model.transform(data);

            // Save the result in an HDFS file
            JavaRDD<Row> clusteredDataRDD = clusteredData.javaRDD();
            clusteredDataRDD.saveAsTextFile(outputPath);

            // Close the Spark Context object
            sc.close();
      }
}
```

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

import org.apache.spark.api.java.function.Function;
import org.apache.spark.mllib.linalg.Vectors;
import org.apache.spark.sql.Row;
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.sql.catalyst.expressions.GenericRow;

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

	public Row call(String record) {
		String[] fields = record.split(",");

		// Create a vector of double. One value for each attribute
		double[] attributesValues = new double[fields.length];

		for (int i = 0; i < fields.length; ++i) {
			attributesValues[i] = Double.parseDouble(fields[i]);
		}

		// Create a dense vector based in the content of attributesValues
		Vector[] attrValues= {Vectors.dense(attributesValues)};
		return new GenericRow(attrValues);
	}

}
```