```
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.ml.Pipeline;
import org.apache.spark.ml.PipelineModel;
import org.apache.spark.ml.PipelineStage;
import org.apache.spark.ml.classification.LogisticRegression;
import org.apache.spark.ml.feature.Tokenizer;
import org.apache.spark.ml.feature.HashingTF;
import org.apache.spark.ml.feature.IDF;
import org.apache.spark.ml.feature.StopWordsRemover;

import org.apache.spark.SparkConf;


public class SparkDriver {


      public static void main(String[] args) {
            // Labeled and unlabeled instance types.
            // Spark SQL can infer schema from Java Beans.

            String inputFileTraining;
            String inputFileTest;
            String outputPath;

            inputFileTraining=args[0];
            inputFileTest=args[1];
            outputPath=args[2];

            // Create a configuration object and set the name of the application
            SparkConf conf=new SparkConf().setAppName("Text classification");

            // 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 words
            // E.g., 1,hadoop mapreduce
            JavaRDD<String> trainingData=sc.textFile(inputFileTraining);

            // Map each element (each line of the input file) to a LabeledDocument
            JavaRDD<LabeledDocument> trainingRDD=trainingData.map(new
InputData()).cache();

            DataFrame training = sqlContext.createDataFrame(trainingRDD,
LabeledDocument.class);

            // Configure an ML pipeline, which consists of five stages:
            // tokenizer -> split sentences in set of words
            // remover -> remove stopwords
            // hashingTF -> map set of words to a fixed-length feature vectors
            //                      (each word becomes a feature and the value of
```

```
the feature
            //                        is the frequency of the word in the sentence)
            // idf -> scales each features. Intuitively, it down-weights features
which appear
            //            frequently in the input lines
            // lr -> logistic regression classification algorithm

            // The tokenizer split each sentence in a set of words
            Tokenizer tokenizer = new Tokenizer()
              .setInputCol("text")
              .setOutputCol("words");

            // Remove stopwords
            StopWordsRemover remover = new StopWordsRemover()
              .setInputCol("words")
              .setOutputCol("filteredWords");

            // Map words to a features
            HashingTF hashingTF = new HashingTF()
              .setNumFeatures(1000)
              .setInputCol(remover.getOutputCol())
              .setOutputCol("rawFeatures");

            // Apply the IDF transformation
            IDF idf = new
IDF().setInputCol("rawFeatures").setOutputCol("features");

            // Create a classification model based on the logistic regression
algorithm
            LogisticRegression lr = new LogisticRegression()
              .setMaxIter(10)
              .setRegParam(0.01);

            // Create the pipeline
            Pipeline pipeline = new Pipeline()
              .setStages(new PipelineStage[] {tokenizer, remover, hashingTF, idf,
lr});

            // Analyze the training data and create the classification model based
            // on the specified pipeline
            PipelineModel model = pipeline.fit(training);


            // Load the test/unlabeled data from a textual file
            JavaRDD<String> testData=sc.textFile(inputFileTest);

            // Map each element (each line of the input file) a LabeledDocument
            JavaRDD<LabeledDocument> testRDD=testData.map(new InputData()).cache();

            DataFrame test = sqlContext.createDataFrame(testRDD,
LabeledDocument.class);

            // Make predictions on test documents.
            // The value of the label attribute is ignored during this step
            DataFrame predictions = model.transform(test);

            // Select only the text and
            // the predicted class for each record/document
            DataFrame predictionsDF=predictions.select("text", "prediction");
```

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


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

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

import java.io.Serializable;

@SuppressWarnings("serial")
public class LabeledDocument implements Serializable {
        private double label;
        private String text;

        public LabeledDocument(String text, double label) {
          this.text = text;
          this.label = label;
        }

        public String getText() { return this.text; }
        public void setText(String text) { this.text = text; }

        public double getLabel() { return this.label; }
        public void setLabel(double label) { this.label = label; }
      }
```

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

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

@SuppressWarnings("serial")
public class InputData implements Function<String, LabeledDocument> {

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

		// Fields of 0 contains the id of the class label
		double classLabel = Double.parseDouble(fields[0]);
		String text = fields[1];

		// The content of the document is after the comma

		// Return a new LabeledDocument
		return new LabeledDocument(text, classLabel);
	}

}
```