

```

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.regression.LinearRegression;
import org.apache.spark.mllib.regression.LabeledPoint;
import org.apache.spark.SparkConf;

public class SparkDriver {

    public static void main(String[] args) {

        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("MLlib - linear regression");

        // 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: target attribute,list of numerical
attribute values
        // E.g., 5.0,1.0,5.0,4.5,1.2
        JavaRDD<String> trainingData=sc.textFile(inputFileTraining);

        // Map each element (each line of the input file) a LabelPoint
        JavaRDD<LabeledPoint> trainingRDD=trainingData.map(new InputRecord());

        // Prepare training data.
        // We use LabeledPoint, which is a JavaBean.
        // We use Spark SQL to convert RDDs of JavaBeans
        // into DataFrames.
        // Each data point has a set of features and a label
        DataFrame training = sqlContext.createDataFrame(trainingRDD,
LabeledPoint.class).cache();

        // Create a LinearRegression object.
        // LogisticRegression is an Estimator that is used to
        // create a classification model based on logistic regression.
        LinearRegression lr = new LinearRegression();

        // We can set the values of the parameters of the
        // Linear Regression algorithm using the setter methods.
        // There is one set method for each parameter

```

```

// For example, we are setting the number of maximum iterations to 10
// and the regularization parameter. to 0.01
lr.setMaxIter(10);
lr.setRegParam(0.01);

// Define the pipeline that is used to create the logistic regression
// model on the training data
// In this case the pipeline contains one single stage/step (the model
// generation step).
Pipeline pipeline = new Pipeline()
    .setStages(new PipelineStage[] {lr});

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

// Now, the classification model can be used to predict the value of
// the target attribute of new "unlabeled" data
// Read test (unlabeled) data
JavaRDD<String> testData=sc.textFile(inputFileTest);

// Map each element (each line of the input file) a LabelPoint
JavaRDD<LabeledPoint> testRDD=testData.map(new InputRecord());

// Create the DataFrame based on the new test data
DataFrame test = sqlContext.createDataFrame(testRDD,
LabeledPoint.class);

// Make predictions on test documents using the Transformer.transform()
method. // The transform will only use the 'features' columns

// The returned DataFrame has the following schema (attributes)
// - features: vector (values of the attributes)
// - label: double (value of the class label)
// - rawPrediction: vector (nullable = true)
// - probability: vector (The i-th cell contains the probability that
the //
//           current record belongs to the i-th class
// - prediction: double (the predicted class label)

DataFrame predictions = model.transform(test);

// Select only the features (i.e., the value of the attributes) and
// the predicted value of the target attribute for each record
DataFrame predictionsDF=predictions.select("features", "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 org.apache.spark.api.java.function.Function;
import org.apache.spark.mllib.regression.LabeledPoint;
import org.apache.spark.mllib.linalg.Vectors;
import org.apache.spark.mllib.linalg.Vector;

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

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

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

        // The other cells of fields contain the (numerical) values of the
attributes
        // Create an array of doubles containing these values
        double[] attributesValues = new double[fields.length-1];

        for (int i = 0; i < fields.length-1; ++i) {
            attributesValues[i] = Double.parseDouble(fields[i+1]);
        }

        // Create a dense vector based in the content of attributesValues
        Vector attrValues= Vectors.dense(attributesValues);

        // Return a new LabeledPoint
        return new LabeledPoint(classLabel, attrValues);
    }
}

```