Big data: architectures and data analytics

Spark MLlib

Spark MLlib

- Spark MLlib is the Spark component providing the machine learning/data mining algorithms
  - Pre-processing techniques
  - Classification (supervised learning)
  - Clustering (unsupervised learning)
  - Itemset mining

3

Spark MLlib

- MLlib APIs are divided into two packages:
  - org.apache.spark.mllib
    - It contains the original APIs built on top of RDDs
  - org.apache.spark.ml
    - It provides higher-level API built on top of DataFrames for constructing ML pipelines
    - It is recommended because with DataFrames the API is more versatile and flexible
    - It provides the pipeline concept

4

Spark Mllib – Data types

Spark Mllib – Data types

- Spark MLlib is based on a set of basic local and distributed data types
  - Local vector
  - Labeled point
  - Local matrix
  - Distributed matrix
- DataFrames for ML are built on top of these basic data types

6

Local vectors

- Local **org.apache.spark.mllib.linalg.Vector** objects are used to store vectors of double values
  - Both dense and sparse vectors are supported
- The MLlib algorithms works on vectors of double
  - Non double attributes/values must be mapped to double values

Local vectors

- Dense and sparse representations are supported
- E.g., a vector (1.0, 0.0, 3.0) can be represented
  - in dense format as [1.0, 0.0, 3.0]
  - or in sparse format as (3, [0, 2], [1.0, 3.0])
    - where 3 is the size of the vector
    - [0,2] contains the indexes of the non-zero cells
    - [1.0, 3.0] contains the values of the non-zero cells

Local vectors

- The following code shows how a vector can be created in Spark

```
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.linalg.Vectors;

// Create a dense vector (1.0, 0.0, 3.0).
Vector dv = Vectors.dense(1.0, 0.0, 3.0);

// Create a sparse vector (1.0, 0.0, 3.0) by
// specifying its indices and values corresponding
// to non-zero entries
Vector sv = Vectors.sparse(3, new int[] {0, 2},
                             new double[] {1.0, 3.0});
```


Local vectors

- The following code shows how a vector can be created in Spark

```
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.linalg.Vectors;

// Create a dense vector (1.0, 0.0, 3.0).
Vector dv = Vectors.dense(1.0, 0.0, 3.0);

// Create a sparse vector (1.0, 0.0, 3.0) by
// specifying its indices and values corresponding
// to non-zero entries
Vector sv = Vectors.sparse(3, new int[] {0, 2},
                             new double[] {1.0, 3.0});
```

Indexes of non-empty cells

Size of the vector

Values of non-empty cells

Labeled points

- Local **org.apache.spark.mllib.regression.LabeledPoint** objects are local vector associated with a label
  - The label is a double value
    - For the classification problem, each class label is associated with an integer value ranging from 0 to C-1, where C is the number of distinct classes
  - Both dense and sparse vectors associated with a label are supported
- In MLlib, labeled points are used by many supervised learning algorithms

Labeled points

- The following code shows how a LabeledPoint can be created in Spark

```
import org.apache.spark.mllib.linalg.Vectors;
import org.apache.spark.mllib.regression.LabeledPoint;

// Create a labeled point with a positive label and
// a dense feature vector.
LabeledPoint pos = new LabeledPoint(1,
                          Vectors.dense(1.0, 0.0, 3.0));

// Create a labeled point with a negative label and a sparse feature
// vector.
LabeledPoint neg = new LabeledPoint(0,
    Vectors.sparse(3, new int[] {0, 2}, new double[] {1.0, 3.0}));
```

Labeled points

- The following code shows how a LabeledPoint can be created in Spark

A vector of double representing the values of the features/attributes

```
import org.apache.spark.mllib.linalg.Vectors;
...eledPoint;
...nd
// a dense feature vector.
LabeledPoint pos = new LabeledPoint(1,
                     Vectors.dense(1.0, 0.0, 3.0));

// Create a labeled point with a negative label and a sparse feature
// vector.
LabeledPoint neg = new LabeledPoint(0,
      Vectors.sparse(3, new int[] {0, 2}, new double[] {1.0, 3.0}));
```


Labeled points

- The following code shows how a LabeledPoint can be created in Spark

Class labels

```
import org.apache.spark.mllib.linalg.Vectors;
...eledPoint;
// Create a labeled point with a positive label and
// a dense feature vector.
LabeledPoint pos = new LabeledPoint(1,
                     Vectors.dense(1.0, 0.0, 3.0));

// Create a labeled point with a negative label and a sparse feature
// vector.
LabeledPoint neg = new LabeledPoint(0,
      Vectors.sparse(3, new int[] {0, 2}, new double[] {1.0, 3.0}));
```


Labeled points

- The following code shows how a LabeledPoint can be created in Spa...

When a binary classification problem is considered, usually the value 1 is associated with the positive class, while the value 0 is associated with the negative class.

```
import org.apach...
import org.apache.spark.mllib.regression.LabeledPoint;

// Create a labeled point with a positive label and
// a dense feature vector.
LabeledPoint pos = new LabeledPoint(1,
                     Vectors.dense(1.0, 0.0, 3.0));

// Create a labeled point with a negative label and a sparse feature
// vector.
LabeledPoint neg = new LabeledPoint(0,
      Vectors.sparse(3, new int[] {0, 2}, new double[] {1.0, 3.0}));
```


Sparse labeled data

- Frequently the training data are sparse
  - E.g., textual data are sparse. Each document contains only a subset of the possible words
  - Hence, sparse vectors are used
- MLlib supports reading training examples stored in the LIBSVM format
  - It is a commonly used format that represents each document/record as a sparse vector

Sparse labeled data

- The LIBSVM format
  - It is a text format in which each line represents a labeled sparse feature vector using the following format:
    - label index1:value1 index2:value2 ...
- where
  - label is an integer associated with the class label
  - the indexes are one-based (i.e., integer indexes starting from 1) representing the features
  - the values are the (double) values of the features
- After loading, the feature indexes are converted to zero-based (i.e., integer indexes starting from 0)

Sparse labeled data: example

- The following example file

```
1 1:5.8 2:1.7
0 1:4.1 3:2.5 4:1.2
```

- Contains two records/documents
  - A positive record (class 1) containing indexes 1 and 2 (i.e., features 1 and 2) with values 5.8 and 1.7 respectively
  - A negative record (class 0) containing indexes 1, 3, and 4 (i.e., features 1, 3, and 4) with values 4.1, 2.5, and 1.2 respectively

Sparse labeled data

- The MLUtils.loadLibSVMFile method reads training examples stored in the LIBSVM format
- Example code with RDDs

```
import org.apache.spark.mllib.regression.LabeledPoint;
import org.apache.spark.mllib.util.MLUtils;
import org.apache.spark.api.java.JavaRDD;
.....
JavaSparkContext sc = new JavaSparkContext(conf);
// Read the content of a LIBSVM file and store it
// in a JavaRDD of LabeledPoints
JavaRDD<LabeledPoint> examples =
        MLUtils.loadLibSVMFile(sc.sc(),
        "sample_libsvm_data.txt").toJavaRDD();
```

19

Sparse labeled data

- Example code with DataFrames

```
.....
JavaSparkContext sc = new JavaSparkContext(conf);
SQLContext jsql = new SQLContext(jsc);
// Read the content of a LIBSVM file and store it
// in a DataFrame
DataFrame data = jsql.createDataFrame(
    MLUtils.loadLibSVMFile(sc.sc(),
    "data/mllib/sample_libsvm_data.txt"),
   LabeledPoint.class);
```

20

Spark MLlib - Main concepts

Spark MLlib - Main concepts

- DataFrame
  - Spark ML uses DataFrames from Spark SQL as ML datasets, which can hold a variety of data types
    - E.g., a DataFrame could have different columns storing text, feature vectors, (true) labels, and predictions

22

Spark MLlib - Main concepts

- Transformer
  - A Transformer is an algorithm which can transform one DataFrame into another DataFrame
    - E.g., A feature transformer might take a DataFrame, read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new DataFrame with the mapped column appended
    - E.g., a classification model is a Transformer which can be applied on a DataFrame with features and transforms it into a DataFrame with also predictions

23

Spark MLlib - Main concepts

- Estimator
  - An Estimator is an algorithm which can be applied on a DataFrame to produce a Transformer (a model)
    - An Estimator implements a method fit(), which accepts a DataFrame and produces a Model of type Transformer
  - An Estimator abstracts the concept of a learning algorithm or any algorithm that fits or trains on an input dataset and returns a model
    - E.g., A classification algorithm such as Logistic Regression is an Estimator, and calling fit() on it a Logistic Regression Model is built, which is a Model and hence a Transformer

24

Spark MLlib - Main concepts

- Pipeline
  - A Pipeline chains multiple Transformers and Estimators together to specify a Machine learning/Data Mining workflow
    - The output of a transformer/estimator is the input of the next one in the pipeline
  - E.g., a simple text document processing workflow aiming at building a classification model includes several steps
    - Split each document into a set of words
    - Convert each set of words into a numerical feature vector
    - Learn a prediction model using the feature vectors and the associated class labels

25

Spark MLlib - Main concepts

- Parameter
  - All Transformers and Estimators share a common API for specifying parameters

26

Spark MLlib - Main concepts

- In the new APIs of Spark MLlib the use of the pipeline approach is preferred
- This approach is based on the following steps
  - 1) The set of Transformers and Estimators that are needed are instantiated
  - 2) A pipeline object is created and the sequence of transformers and estimators associated with the pipeline are specified
  - 3) The pipeline is executed and model is created
  - 4) (optional) The model is applied on new data

27

Classification algorithms

Classification algorithms

- Spark MLlib provides a (limited) set of classification algorithms
  - Logistic regression
  - Decision trees
  - SVMs (with linear kernel)
    - Only binary classification problem are supported by the SVMs classifiers
  - Naïve Bayes
  - …

29

Classification algorithms

- Each classification algorithm has its own parameters
- However, all the provided algorithms are based on two phases
  - Model generation based on a set of training data
  - Prediction of the class label of new unlabeled data
- All the classification algorithms available in Spark work only on numerical data
  - Categorical values must be mapped to integer values (i.e, numerical values)

30

Logistic regression and structured data

Logistic regression and structured data

- The following slides show how to
  - Create a classification model based on the **logistic regression algorithm**
  - Apply the model to new data
- The input dataset is a structured dataset with a fixed number of attributes
  - One attribute is the class label
  - The others are predictive attributes that are used to predict the value of the class label
  - We suppose the first column of the input file contains the class label

32

Logistic regression and structured data

- Consider the following example file

  1,5.8,1.7

  0,10.5,2.0
- It contains two records
- Each record has three attributes
  - The first attribute (column) is the class label
  - The second and the third attributes (columns) are predictive attributes

33

Logistic regression and structured data: example

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

34

Logistic regression and structured data: example

```
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 –
                                        logistic regression");

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

35

Logistic regression and structured data: example

```
// 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
JavaRDD<String> trainingData=sc.textFile(inputFileTraining);

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

// Prepare training data.
// 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);
```

36

Logistic regression and structured data: example

```
// 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
JavaRDD<String>

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

// Prepare training data.
// 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);
```

The InputRecord class map each string of the input RDD (i.e., each line of the input file) to a LabeledPoint

Logistic regression and structured data: example

```
// 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
```

The training data are represented by means of a DataFrame of LabeledPoint.
Each element of this DataFrame has two columns:
-label: the class label
-features: the vector of real values associated with the attributes of the input record

```
// Prepare training data.
// 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);
```


Logistic regression and structured data: example

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

// We can set the values of the parameters of the
// Logistic 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 0.0.1
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});
```


Logistic regression and structured data: example

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

// We can set the values of the parameters of the
// Logistic 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 0.0.1
lr.setMaxIter(10);
lr.setRegParam(0.01);
```

This is the sequence of Transformers and Estimators to apply on the training data.
This simple pipeline is composed only of the logistic regression estimator

```
// In this case the pipeline contains one single stage/step (the model
// generation step).
Pipeline pipeline = new Pipeline().setStages(new PipelineStage[] {lr});
```


Logistic regression and structured data: example

```
// 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 class label
// 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);
```


Logistic regression and structured data: example

```
// Make predictions on test documents using the transform()
// method.
// The transform will only use the 'features' columns
DataFrame predictions = model.transform(test);

// 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)

// Select only the features (i.e., the value of the attributes) and
// the predicted class for each record
DataFrame predictionsDF=predictions.select("features", "prediction");
```

Logistic regression and structured data: example

```
// Make predictions on test documents using the transform()
// method.
// The transform will only use the 'features' columns
DataFrame predictions = model.transform(test);

// The returned DataFrame has the following schema (attributes)
```

The model is applied to new data/records and the class label is predicted for each new data/record.
The new generated DataFrame has the same attributes of the input DataFrames plus the prediction attribute (and also some other attributes).

```
// - prediction: double (the predicted class label)

// Select only the features (i.e., the value of the attributes) and
// the predicted class for each record
DataFrame predictionsDF=predictions.select("features", "prediction");
```

43

Logistic regression and structured data: example

```
// Make predictions on test documents using the transform()
// method.
// The transform will only use the 'features' columns
DataFrame predictions = model.transform(test);

// 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
// - p

// Select only the features (i.e., the value of the attributes) and
// the predicted class for each record
DataFrame predictionsDF=predictions.select("features", "prediction");
```

The attribute values and the predicted class are selected

44

Logistic regression and structured data: example

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

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

45

Logistic regression and structured data: example

```
// This is the class InputRecord.
// It is used by the map transformation that is used to transform the input file
// in an RDD of LabeledPoints
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;
```

46

Logistic regression and structured data: example

```
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]);
        }
```

47

Logistic regression and structured data: example

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

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

}
```

48

Decision trees and structured data

Decision trees and structured data

- The following slides show how to
  - Create a classification model based on the **decision tree algorithm**
  - Apply the model to new data
- The input dataset is a structured dataset with a fixed number of attributes
  - One attribute is the class label
  - The others are predictive attributes that are used to predict the value of the class label
  - We suppose the first column of the input file contains the class label

50

Decision trees and structured data

- The structure is similar to the one used to build a classification model by means of the logistic regression approach
- However, some specific methods are needed because the decision tree algorithm needs some statistics about the class label to build the model
  - Hence, an index must be created on the label before creating the pipeline that creates the decision tree-based classification model

51

Decision trees and structured data

- Consider the following example file

  1,5.8,1.7

  0,10.5,2.0
- It contains two records
- Each record has three attributes
  - The first attribute (column) is the class label
  - The second and the third attributes (columns) are predictive attributes

52

Decision trees and structured data: example

```
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.DecisionTreeClassifier;
import org.apache.spark.ml.feature.IndexToString;
import org.apache.spark.ml.feature.StringIndexer;
import org.apache.spark.ml.feature.StringIndexerModel;
import org.apache.spark.mllib.regression.LabeledPoint;
import org.apache.spark.SparkConf;
```

53

Decision trees and structured data: example

```
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 - Decision Tree");

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

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

54

Decision trees and structured data: example

```
// 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
JavaRDD<String> trainingData=sc.textFile(inputFileTraining);

// Map each element (each line of the input file) to a LabeledPoint
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();
```

55

Decision trees and structured data: example

```
// For creating a decision tree a label attribute with specific metadata is
// needed
// The StringIndexer Estimator is used to achieve this operation
StringIndexerModel labelIndexer = new StringIndexer()
  .setInputCol("label").setOutputCol("indexedLabel").fit(training);

// Create a DecisionTreeClassifier object.
// DecisionTreeClassifier is an Estimator that is used to
// create a classification model based on decision trees
DecisionTreeClassifier dc= new DecisionTreeClassifier();

// We can set the values of the parameters of the Decision Tree
// For example we can set the measure that is used to decide if a
// node must be split
// In this case we set gini index
dc.setImpurity("gini");
// Set the name of the indexed label column
dc.setLabelCol("indexedLabel");
```

56

Decision trees and structured data: example

```
// For creating a decision tree a label attribute with specific metadata is
// needed
// The StringIndexer Estimator is used to achieve this operation
StringIndexerModel labelIndexer = new StringIndexer()
  .setInputCol("label").setOutputCol("indexedLabel").fit(training);
```

This part is specific of the Decision Tree model generation process. It is not needed for generating a logistic regression algorithm

```
DecisionTreeClassifier dc= new DecisionTreeClassifier();

// We can set the values of the parameters of the Decision Tree
// For example we can set the measure that is used to decide if a
// node must be split
// In this case we set gini index
dc.setImpurity("gini");
// Set the name of the indexed label column
dc.setLabelCol("indexedLabel");
```

57

Decision trees and structured data: example

```
// Convert indexed labels back to original labels.
// The content of the prediction attribute is the index of the
// predicted class
// The original name of the predicted class is stored in
// the predictedLabel attribute
IndexToString labelConverter = new IndexToString()
        .setInputCol("prediction").setOutputCol("predictedLabel")
        .setLabels(labelIndexer.labels());
```

58

Decision trees and structured data: example

```
// Define the pipeline that is used to create the decision tree
// 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[]{labelIndexer,dc,labelConverter});

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

59

Decision trees and structured data: example

```
// Define the pipeline that is used to create the decision tree
// 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[]{labelIndexer,dc,labelConverter});

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

In this case the pipeline is composed of three steps
1) StringIndexer
2) Decision Tree classifier
3) IndexToString

60

Decision trees and structured data: example

```
// Now, the classification model can be used to predict the class label
// 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 transform() method.
// The transform will only use the 'features' columns
DataFrame predictions = model.transform(test);
```

61

Decision trees and structured data: example

```
        //The returned DataFrame has the following schema (attributes)
        // - features: vector (values of the attributes)
        // - label: double (value of the class label)
        // - indexedLabel: double (value of the class label after the initial transformation)
        // - 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)
        // - predictedLabel: double (the predicted class label back to original labels)
        // Select only the features (i.e., the value of the attributes) and
        // the predicted class for each record (in this case the prediction is in
        // predictedLabel)
        DataFrame predictionsDF=predictions.select("features", "predictedLabel");

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

        sc.close();
    }
}
```

62

Decision trees and structured data: example

```
// This is the class InputRecord.
// It is used by the map transformation that is used to transform the input file
// in an RDD of LabeledPoints
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;
```

63

Decision trees and structured data: example

```
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]);
        }
```

64

Decision trees and structured data: example

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

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

}
```

65