Big data: architectures and data analytics

Spark MLlib

Textual data classification

Textual data classification

- The following slides show how to
  - Create a classification model based on the logistic regression algorithm for **textual documents**
  - Apply the model to new textual documents
- The input training dataset represents a textual document collection
  - Each line contains one document and its class
    - The class label
    - A list of words (the text of the document)

4

Textual data classification

- Consider the following example file

  1,The Spark system is based on scala
  1,Spark is a new distributed system
  0,Turin is a beautiful city
  0,Turin is in the north of Italy
- It contains four textual documents
- Each line contains two attributes
  - The class label (first attribute)
  - The text of the document (second attribute)

5

Textual data classification

- Input data before pre-processing

Label	Text
1	The Spark system is based on scala
1	Spark is a new distributed system
0	Turin is a beautiful city
0	Turin is in the north of Italy

6

Textual data classification

- A set of preprocessing steps must be applied on the textual attribute before generating a classification model

Textual data classification

1. Since Spark ML algorithms work only on "Table", the textual part of the input data must be translated in a set of attributes in order to represent the data as a table
   - Usually a table with an attribute for each word is generated

Textual data classification

2. Many words are useless (e.g., conjunctions)
   - Stopwords are usually removed

Textual data classification

- The words appearing in almost all documents are not characterizing the data
  - Hence, they are not very important for the classification problem
- The words appearing in few documents allow distinguish the content of those documents (and hence the class label) with respect to the others
  - Hence, they are very important for the classification problem

Textual data classification

3. Traditionally a weight, based on the TF-IDF measure, is used to assign a difference importance to the words based on their frequency in the collection

Textual data classification

- Input data after the pre-processing transformations (tokenization, stopword removal, TF-IDF computation)

Label	Spark	system	scala	.....
1	0.5	0.3	0.75	..
1	0.5	0.3	0	...
0	0	0	0	...
0	0	0	0	...

Textual data classification

- The Dataset<Row> associated with input data after the pre-processing transformations must contain, as usual, the columns
  - label
    - Class label value
  - features
    - The pre-processed version of the input text
  - There are also some other intermediate columns, related to applied transformations, but they are not considered by the classification algorithm

Textual data classification

- The Dataset<Row> associated with input data after the pre-processing transformations must contain, as usual, the columns

label	features	text	.....	.....
1	[0.5, 0.3, 0.75, ..]	The Spark system is based on scala	..	..
1	[0.5, 0.3, 0, ..]	Spark is a new distributed system	...	...
0	[0, 0, 0, ..]	Turin is a beautiful city	...	...
0	[0, 0, 0, ..]	Turin is in the north of Italy	...	...

Textual data classification

- The Dataset<Row> associated with input data after the pre-processing transformations must contain, as usual, the columns

label	features	text	.....	.....
1	[0.5, 0.3, 0.75, ..]	The Spark system is based on scala	..	..
1	[0.5, 0.3, 0, ..]	Spark is a new distributed system	...	...
0	[0, 0, 0, ..]	Turin is a beautiful city	...	...
0	[0, 0, 0, ..]	Turin is in the north of Italy	...	...

Only "label" and "features" are considered by the classification algorithm

Textual data classification: example

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

import java.io.Serializable;

public class LabeledDocument implements Serializable {
    private double label;
    private String text;

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


Textual data classification: example

```
    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; }
  }
);
```


Textual data classification: example

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

import org.apache.spark.api.java.*;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
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;
```

Textual data classification: example

```
public static void main(String[] args) {
	String inputFileTraining; String inputFileTest; String outputPath;

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

	// Create a Spark Session object and set the name of the application
	// We use some Spark SQL transformation in this program
	SparkSession ss = SparkSession.builder()
			.appName("MLlib - logistic regression").getOrCreate();

	// Create a Java Spark Context from the Spark Session
	// When a Spark Session has already been defined this method
	// is used to create the Java Spark Context
	JavaSparkContext sc = new JavaSparkContext(ss.sparkContext());
```


Textual data classification: example

```
// ************************
// Training step
// ************************

// 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);
```


Textual data classification: example

```
	// Map each element (each line of the input file) to a LabeledDocument
	// LabeledDocument is a class defined in this application. Each instance
	// of LabeledDocument is characterized by two attributes:
	// - private double label
	// - private String text
	// LabeledDocument represents a "document" and the related class label.
	JavaRDD<LabeledDocument> trainingRDD=trainingData.map(record -> {
			String[] fields = record.split(",");

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

			// The content of the document is after the comma
			String text = fields[1];
			// Return a new LabeledDocument
			return new LabeledDocument(classLabel, text);
		}).cache();
```


Textual data classification: example

```
// Prepare training data.
// We use LabeledDocument, which is a JavaBean.
// We use Spark SQL to convert RDDs of JavaBeans
// into Dataset<Row>. The columns of the Dataset are label
// and features
Dataset<Row> training = ss
		.createDataFrame(trainingRDD, LabeledDocument.class).cache();
```


Textual data classification: example

```
	// 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 -> compute the idf component of the TF-IDF measure
	// lr -> logistic regression classification algorithm

	// The Tokenizer splits each sentence in a set of words.
	// It analyzes the content of column "text" and adds the
	// new column "words" in the returned DataFrame
	Tokenizer tokenizer = new Tokenizer()
			.setInputCol("text")
			.setOutputCol("words");
```


Textual data classification: example

```
// Remove stopwords.
// the StopWordsRemover component returns a new DataFrame with
// new column called "filteredWords". "filteredWords" is generated
// by removing the stopwords from the content of column "words"
StopWordsRemover remover = new StopWordsRemover()
		.setInputCol("words")
		.setOutputCol("filteredWords");
```

Textual data classification: example

```
// Map words to a features
// Each word in filteredWords must become a feature in a Vector object
// The HashingTF Transformer performs this operation.
// This operations is based on a hash function and can potentially
// map two different word to the same "feature". The number of conflicts
// in influenced by the value of the numFeatures parameter.
// The "feature" version of the words is stored in Column "rawFeatures".
// Each feature, for a document, contains the number of occurrences
// of that feature in the document (TF component of the TF-IDF measure)
HashingTF hashingTF = new HashingTF()
        .setNumFeatures(1000)
        .setInputCol("filteredWords")
        .setOutputCol("rawFeatures");
```

25

Textual data classification: example

```
// Apply the IDF transformation.
// Update the weight associated with each feature by considering also the
// inverse document frequency component. The returned new column
// is called "features", that is the standard name for the column that
// contains the predictive features used to create a classification model
IDF idf = new IDF()
        .setInputCol("rawFeatures")
        .setOutputCol("features");
```

26

Textual data classification: example

```
// Create a classification model based on the logistic regression algorithm
// We can set the values of the parameters of the
// Logistic Regression algorithm using the setter methods.
LogisticRegression lr = new LogisticRegression()
        .setMaxIter(10)
        .setRegParam(0.01);

// Define the pipeline that is used to create the logistic regression
// model on the training data.
// In this case the pipeline is composed of five steps
// - text tokenizer
// - stopword removal
// - TF-IDF computation (performed in two steps)
// - Logistic regression model generation
Pipeline pipeline = new Pipeline()
 .setStages(new PipelineStage[] {tokenizer, remover, hashingTF, idf, lr});
```

27

Textual data classification: 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
```

28

Textual data classification: example

```
// *************************
// Prediction step
// *************************

// Read unlabeled data
// For the unlabeled data only the predictive attributes are available
// The class label is not available and must be predicted by applying
// the classification model inferred during the previous phase
JavaRDD<String> unlabeledData=sc.textFile(inputFileTest);
```

29

Textual data classification: example

```
// Map each unlabeled input document of the input file to a
LabeledDocument JavaRDD<LabeledDocument> unlabeledRDD=
unlabeledData.map(record -> {
        String[] fields = record.split(",");
        // The content of the document is after the comma
        String text = fields[1];

        // The class label in unknown.
        // To create a LabeledDocument a class label value must be
        // specified also for the unlabeled data. I set it to -1 (an invalid
        // value).
        double classLabel = -1;

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

30

Textual data classification: example

```
// Create the DataFrame based on the new unlabeled data
Dataset<Row> unlabeled =
        ss.createDataFrame(unlabeledRDD, LabeledDocument.class);

// Make predictions on unlabeled documents by 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)

Dataset<Row> predictions = model.transform(unlabeled);
```

31

Textual data classification: example

```
        // Select only the text and
        // the predicted class for each record/document
        Dataset<Row> 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();
    }
}
```

32