

Big data: architectures and data analytics

Clustering algorithms

Clustering algorithms

- Spark MLlib provides a (limited) set of clustering algorithms
 - K-means
 - Gaussian mixture
 - ...

3

Clustering

- Each clustering algorithm has its own parameters
- However, all the provided algorithms identify a set of groups of objects/clusters and assign each input object to one single cluster
- All the clustering algorithms available in Spark work only with numerical data
 - Categorical values must be mapped to integer values (i.e, numerical values)

4

K-means clustering algorithm

K-means clustering algorithm

- K-means is one of the most popular clustering algorithms
- It is characterized by one important parameter
 - The number of clusters **K**
 - The choice of **K** is a complex operation
- It is able to identify only spherical shaped clusters

K-means clustering algorithm

- The following slides show how to
 - Cluster the input data by means of the **K-means algorithm**
- The input dataset is a structured dataset with a fixed number of attributes
 - All the attributes are numerical attributes
 - Data must be normalized before applying the clustering algorithm in order to give the same importance to all attributes

7

K-means clustering algorithm

- Example of input file
 - 0.5,0.9,1.0
 - 0.6,0.6,0.7
- In the following example code we suppose that the input data are already normalized
 - E.g., All values are already in the range [0-1]

8

K-means clustering algorithm: 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.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;
```

9

K-means clustering algorithm: example

```
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);
```

10

K-means clustering algorithm: example

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

// Read data from a textual file
// Each line contains a set of real numbers
// E.g., 1.0,0.5,1.2
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 based 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();
```

11

K-means clustering algorithm: example

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

// Read data from
// Each line contains a set of real numbers
// E.g., 1.0,0.5,1.2
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 based 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();
```

In this case the input data must be represented as a JavaRDD<Row> and then as a DataFrame

12

K-means clustering algorithm: example

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

// Read data from
// Each line contains
// E.g., 1.0,0.5,1.2
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 based 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();
```

13

K-means clustering algorithm: example

```
// 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 (= number of clusters)
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});
```

14

K-means clustering algorithm: example

```

// 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();
}
}

```

15

K-means clustering algorithm: example

```

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];
    }
}

```

16

K-means clustering algorithm: example

```
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);  
}  
}
```