

# **Big data: architectures and data analytics**

---

# Clustering algorithms

---

# Clustering algorithms

---

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

# 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)

# Clustering

---

- The input of the MLlib clustering algorithms is a Dataset<Row> containing a column called features
  - Data type: org.apache.spark.ml.linalg.Vectors
- The clustering algorithm clusters the input records by considering only the content of features
  - The other columns, if any, are not considered

# Clustering: Example of input data

---

- Example of input data
  - A set of customer profiles
  - We want to group customers in groups based on their characteristics

MonthlyIncome	NumChildren
1400.0	2
11105.5	0
2150.0	2

# Clustering: Example of input data

- Input training data

MonthlyIncome	NumChildren
1400.0	2
11105.5	0
2150.0	2

- Input Dataset<Row> that must be generated as input for the MLlib clustering algorithms

features
[1400.0 , 2.0]
[11105.5, 0.0]
[2150.0 , 2.0]

# Clustering: Example of input data

The values of all input attributes are “stored” in a vector of doubles (one vector for each input record).  
The generated Dataset<Row> contains a column called features containing the vectors associated with the input records.

- Input training

MonthlyIncome	NumChildren
1400.0	2
11105.5	0
2150.0	2

- Input Dataset<Row> that must be generated as input for the MLlib clustering algorithms

features
[1400.0 , 2.0]
[11105.5, 0.0]
[2150.0 , 2.0]



# **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 apply the **K-means algorithm** provided by MLlib
- The input dataset is a structured dataset with a fixed number of attributes
  - All the attributes are numerical attributes

# 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]

# K-means clustering algorithm: Example

---

```
package it.polito.bigdata.spark.sparkmllib;
import java.io.Serializable;
import org.apache.spark.ml.linalg.Vector;

public class InputRecord implements Serializable {
    private Vector features;

    public Vector getFeatures() {
        return features;
    }
    public void setFeatures(Vector features) {
        this.features = features;
    }
    public InputRecord(Vector features) {
        this.features = features;
    }
}
```

# K-means clustering algorithm: 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.Session;
import org.apache.spark.ml.Pipeline;
import org.apache.spark.ml.PipelineModel;
import org.apache.spark.ml.PipelineStage;
import org.apache.spark.ml.linalg.Vector;
import org.apache.spark.ml.linalg.Vectors;
import org.apache.spark.ml.clustering.KMeans;
```

# K-means clustering algorithm: Example

---

```
public class SparkDriver {  
  
    public static void main(String[] args) {  
        String inputFile; String outputPath;  
  
        inputFile = args[0];  
        outputPath = args[1];  
  
        // 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 - K-means").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());  
    }  
}
```

# K-means clustering algorithm: Example

---

```
// Read training data from a textual file
// Each line contains 3 double values /the input records are
// characterized by three attributes)
// E.g., 1.0,5.0,4.5
JavaRDD<String> inputData = sc.textFile(inputFile);
```



# K-means clustering algorithm: Example

---

```
// Map each input record/data point of the input file to a InputRecord object
// InputRecord is characterized by the features attribute
JavaRDD<InputRecord> inputRDD = inputData.map(record -> {
    String[] fields = record.split(",");

    //The three cells of fields contain the (numerical) values of the
    // three input attributes.
    double[] attributesValues = new double[3];

    attributesValues[0] = Double.parseDouble(fields[0]);
    attributesValues[1] = Double.parseDouble(fields[1]);
    attributesValues[2] = Double.parseDouble(fields[2]);

    // Create a dense vector based on the content of attributesValues
    Vector attrValues = Vectors.dense(attributesValues);
    return new InputRecord(attrValues);
});
```

# K-means clustering algorithm: Example

---

```
// Create a DataFrame based on the input data.  
Dataset<Row> data = ss  
    .createDataFrame(inputRDD, InputRecord.class).cache();
```

# K-means clustering algorithm: Example

---

```
// Create a DataFrame based on the input data.
```

```
Dataset<Row> data = ss  
    .createDataFrame(inputRDD, InputRecord.class).cache();
```

The schema of the returned Dataset<Row> is  
- features: Vector

# 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});
```

# 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)
Dataset<Row> 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();
}
}
```