Clustering algorithms

Clustering algorithms

- Spark MLlib provides a (limited) set of clustering algorithms
  - K-means
  - Bisecting k-means
  - Gaussian Mixture Model (GMM)

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 DataFrame containing a column called features of type Vector
- 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 DataFrame 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 DataFrame contains a column called features containing the vectors associated with the input records.

- Input trainin

MonthlyIncome	NumChildren
1400.0	2
11105.5	0
2150.0	2

- Input DataFrame 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: main steps

- Clustering with Mllib
  1. Create a DataFrame with the features column
  2. Define the clustering pipeline and run the fit() method on the input data to infer the clustering model (e.g., the centroids of the k-means algorithm)
     - This step returns a clustering model
  3. Invoke the transform() method of the inferred clustering model on the input data to assign each input record to a cluster
     - This step returns a new DataFrame with the new column "prediction" in which the cluster identifier is stored for each input record

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

```
attr1,attr2,attr3
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
  - I.e., all values are already in the range [0-1]
  - Scalers/Normalizers can be used to normalized data if it is needed

K-means clustering algorithm: Example

```
from pyspark.mllib.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
from pyspark.ml import Pipeline
from pyspark.ml import PipelineModel

# input and output folders
inputData = "ex_datakmeans/dataClusteering.csv"
outputPath = "clusterskmeans/“

# Create a DataFrame from dataClusteering.csv
# Training data in raw format
inputDataDF = spark.read.load(inputData,\
                format="csv", header=True,\
                inferSchema=True)
```

K-means clustering algorithm: Example

```
# Define an assembler to create a column (features) of type Vector
# containing the double values associated with columns attr1, attr2, attr3
assembler = VectorAssembler(inputCols=["attr1", "attr2", "attr3"],\
                    outputCol="features")

# Create a k-means object.
# k-means is an Estimator that is used to
# create a k-means algorithm
km = KMeans()

# Set the value of k ( = number of clusters)
km.setK(2)
```

K-means clustering algorithm: Example

```
# Define the pipeline that is used to cluster
# the input data
pipeline = Pipeline().setStages([assembler, km])

# Execute the pipeline on the data to build the
# clustering model
kmeansModel = pipeline.fit(inputDataDF)

# Now the clustering model can be applied on the input 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)
# - original attributes attr1, attr2, attr3
clusteredDataDF = kmeansModel.transform(inputDataDF)
```

K-means clustering algorithm: Example

```
# Define the pipeline that is used to cluster
# the input data
pipeline = Pipeline().setStages([assembler, km])

# Execute the pipeline on the data to build the
# clustering model
kmeansModel = pipeline.fit(inputDataDF)
```

The returned DataFrame has a new column (prediction) in which the "predicted" cluster identifier (an integer) is stored for each input record.

```
# - features: vector (values of the attributes)
# - prediction: double (the predicted cluster id)
# - original attributes attr1, attr2, attr3
clusteredDataDF = kmeansModel.transform(inputDataDF)
```

K-means clustering algorithm: Example

```
# Select only the original columns and the clusterID (prediction) one
# I rename prediction to clusterID
clusteredData = clusteredDataDF\
.select("attr1", "attr2", "attr3", "prediction")\
.withColumnRenamed("prediction","clusterID")

# Save the result in an HDFS output folder
clusteredData.write.csv(outputPath, header="true")
```