Table of contents

Spark MLlib provides a (limited) set of clustering algorithms

- K-means
- Bisecting k-means
- Gaussian Mixture Model (GMM)

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).

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).

:::{.callout-note collapse="true"} ## Example The goal is to group customers in groups based on their characteristics.

Consider the following input data: a set of customer profiles.

MonthlyIncome	NumChildren
1400.0	2
11105.5	0
2150.0	2

The following input DataFrame that must be generated as input for the MLlib clustering algorithms

features
$1400.0, 2.0$
$11105.5, 0.0$
$2150.0, 2.0$

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.

1 Main steps

The steps for clustering with Mllib are the following

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.

2 K-means clustering algorithm

K-means is one of the most popular clustering algorithms, characterized by one important parameter: the number of clusters K (the choice of K is a complex operation). Notice that this method is able to identify only spherical shaped clusters.

Example

The following paragraphs show how to apply the K-means algorithm provided by MLlib. The input dataset is a structured dataset with a fixed number of attributes, and all the attributes are numerical attributes.

Example of input file

```
attr1,attr2,attr3
0.5,0.9,1.0
0.6,0.6,0.7
```

In this example code it is assumed that the input data is 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.

```python
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
)

# 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)

# 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) # <1>

# Select only the original columns and the clusterID (prediction) one
# I rename prediction to clusterID
clusteredData = clusteredDataDF \
```

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