kmeans

NAME

kmeans - A clustering algorithm commonly used in EDA (exploratory data analysis).

SYNOPSIS

import com.nec.frovedis.mllib.clustering.KMeans

KMeansModel
KMeans.train (`RDD[Vector]` data,
    Int k,
    Int iterations = 100,
    Long seed = 0,
    Double epsilon = 0.01)

DESCRIPTION

Clustering is an unsupervised learning problem whereby we aim to group subsets of entities with one another based on some notion of similarity. K-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters (K).

This module provides a client-server implementation, where the client application is a normal Apache Spark program. Spark has its own mllib providing the KMeans support. But that algorithm is slower when comparing with the equivalent Frovedis algorithm (see frovedis manual for ml/kmeans) with big dataset. Thus in this implementation, a spark client can interact with a frovedis server sending the required spark data for training at frovedis side. Spark RDD data is converted into frovedis compatible data internally and the spark ML call is linked with the respective frovedis ML call to get the job done at frovedis server.

Spark side call for KMeans.train() quickly returns, right after submitting the training request to the frovedis server with a dummy KMeansModel object containing the model information like k value etc. with a unique model ID for the submitted training request.

When operations like prediction will be required on the trained model, spark client sends the same request to frovedis server on the same model (containing the unique ID) and the request is served at frovedis server and output is sent back to the spark client.

Detailed Description

KMeans.train()

Parameters
data: A `RDD[Vector]` containing spark-side data points

k: An integer parameter containing the number of clusters
iterations: An integer parameter containing the maximum number of iteration count (Default: 100)
seed: A long parameter containing the seed value to generate the random rows from the given data samples (Default: 0)
epsilon: A double parameter containing the epsilon value (Default: 0.01)

Purpose
It clusters the given data points into a predefined number (k) of clusters.
After the successful clustering, it returns the KMeansModel.

For example,

```
// -------- data loading from sample kmeans data file at Spark side--------
val data = sc.textFile("./sample")
val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble)))
val splits = parsedData.randomSplit(Array(0.8, 0.2), seed = 11L)
val training = splits(0)
val test = splits(1)

// Build the cluster using KMeans with default parameters
val model = KMeans.train(training,2)

// Evaluate the model on test data
model.predict(test).foreach(println)
```

Note that, inside the train() function spark data is converted into frovedis side sparse data and after the training, frovedis side sparse data is released from the server memory. But if the user needs to store the server side constructed sparse data for some other operations, he may also like to pass the FrovedisSparseData object as the value of the "data" parameter. In that case, the user needs to explicitly release the server side sparse data when it will no longer be needed.

For example,

```
val fdata = new FrovedisSparseData(parsedData) // manual creation of frovedis sparse data
val model2 = KMeans.train(fdata,2) // passing frovedis sparse data
fdata.release() // explicit release of the server side data
```

Return Value
This is a non-blocking call. The control will return quickly, right after submitting the training request at frovedis server side with a KMeansModel object containing a unique model ID for the training request along with some other general information like k value etc. But it does not contain any centroid information. It simply works like a spark side pointer of the actual model at frovedis server side. It may be possible that the training is not completed at the frovedis server side even though the client spark side train() returns with a pseudo model.

SEE ALSO

kmeans_model, frovedis_sparse_data