Matrix Factorization using ALS

NAME

Matrix Factorization using ALS - A matrix factorization algorithm commonly used for recommender systems.

SYNOPSIS

import com.nec.frovedis.mllib.recommendation.ALS

MatrixFactorizationModel
ALS.trainImplicit (`RDD[Rating]` data,
    Int rank,
    Int iterations = 100,
    Double lambda = 0.01,
    Double alpha = 0.01,
    Long seed = 0)

DESCRIPTION

Collaborative filtering is commonly used for recommender systems. These techniques aim to fill in the missing entries of a user-item association matrix. Frovedis currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries.

Like Apache Spark, Frovedis also uses the alternating least squares (ALS) algorithm to learn these latent factors. The algorithm is based on a paper "Collaborative Filtering for Implicit Feedback Datasets" by Hu, et al.

This module provides a client-server implementation, where the client application is a normal Apache Spark program. Spark has its own mllib providing the ALS support. But that algorithm is slower when comparing with the equivalent Frovedis algorithm (see frovedis manual for ml/als) 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 ALS.trainImplicit() quickly returns, right after submitting the training request to the frovedis server with a dummy MatrixFactorizationModel object containing the model information like rank 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

ALS.trainImplicit()

Parameters
data: A `RDD[Rating]` containing spark-side distributed rating data
rank: An integer parameter containing the number of latent factors (also known as rank)
iterations: An integer parameter containing the maximum number of iteration count (Default: 100)
lambda: A double parameter containing the regularization parameter (Default: 0.01)
alpha: A double parameter containing the learning rate (Default: 0.01)
seed: A Long parameter containing the seed value to initialize the model structures with random values

Purpose
It trains a MatrixFactorizationModel with alternating least squares (ALS) algorithm. It starts with initializing the model structures of the size MxF and NxF (where M is the number of users and N is the products in the given rating matrix and F is the given rank) with random values and keeps updating them until maximum iteration count is reached. After the training, it returns the trained MatrixFactorizationModel model.

For example,

```
// -------- data loading from sample rating (COO) file at Spark side--------
val data = sc.textFile("./sample")
val ratings = data.map(_.split(',') match { case Array(user, item, rate) =>
                    Rating(user.toInt, item.toInt, rate.toDouble)
              })

// Build the recommendation model using ALS with default parameters
val model = ALS.trainImplicit(ratings,4)
println("Rating: " + model.predict(1,2)) // predict the rating for 2nd product by 1st user
```

Note that, inside the trainImplicit() function spark side COO rating 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() // an empty object
fdata.loadcoo(ratings) // manual creation of frovedis sparse data
val model2 = ALS.trainImplicit(fdata,4) // 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 MatrixFactorizationModel object containing a unique model ID for the training request along with some other general information like rank etc. But it does not contain any user/product components. 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

matrix_factorization_model, frovedis_sparse_data