**Movie Recommendation System**

This project is to give recommendations for users based upon the viewing history and ratings given on movie. We have used the MovieLense data from Kaggle which was updated 3 months back to the Kaggle site. We have used the same dataset for building the recommendation system using **Databricks and Scala.**

We have used the Machine learning approach to make the prediction for users depending on his past choices and ratings of the movies. We have used Advanced filtering mechanism to predict possible recommendations for concerned users depending upon their preference. Here we have used Collaborative filtering and Spark ALS to build the recommendation system.

**Collaborative filtering**

Collaborative filtering works on building a matrix.

**Step 1**: Read ratings of all users on Item (Movie ID) top builds the rating matrix.

**Step 2**: Factorize the Rating matrix into two user matrix and item matrix. user matrix where rows represent users and columns are latent factors; the other being item matrix where rows are latent factors and columns represent items.

**Step3**: While doing the factorization process. The missing rating values can be filled. Once all the missing ratings value are filled, it can be used for making the recommendation.

ALS (Alternative Least Squares) is a pre-build function available through Spark MLLib package. This pre-build function is the implementation of collaborative filtering technique that uses Alternative Least Squares to find suitable weights to minimize the least squares between the predicted and actual ratings.

Data

ALS Model

Save ALS Model for predicting

T

ALS Model Training

T

Load ALS Model with Train Data set

T

Testing Data

.

Training Data

Recommender Output

**Steps of Execution:**

1. Import data into Databricks.
2. Analyze Data and select ratings.csv as re main data for the project.
3. Randomly split ratings RDD into training data RDD (70%) and test data RDD (30%)
4. Create an ALS model with the Movie Ratings Data: Build a ALS user product matrix model with rank = 10, iterations = 10. The training is done via the Collaborative Filtering algorithm Alternating Least Squares (ALS). Essentially this technique predicts missing ratings for specific users for specific movies based on ratings for those movies from other users who did similar ratings for other movies.

This ALS model will automatically predict ratings on movies for users who haven’t rated the movie yet.

Now the User, Movie matrix will ratings completed for all movies by all users.

Now the User, Movie matrix will ratings completed for all movies by all users.

This Model can be used to do the recommendation for users to see a movie or list movies.

1. Train the ALS model and save the model using Train Data set.
2. Evaluate the Model using the Test Data set.
3. Use the model to Make recommendations by making use of methods available in the ALSModel.
4. The ALS Model offers a method called recommendProducts.
5. ALSModel.recommendProducts take two parameters: user id and number of items.
6. The ALS model will recommend the top number of items for a given user.

**Project code:**

import org.apache.spark.mllib.recommendation.{ALS, MatrixFactorizationModel, Rating}

import org.apache.spark.rdd.RDD

import org.apache.spark.sql.SparkSession

**Preprocessing of the data (if needed)**

%scala

//**Import Movielense data** for Ratings and Movies datasets into Databricks

val ratings\_DF = spark.read.format("csv").option("header", "true").load("/FileStore/tables/ratings.csv")

val movies\_DF = spark.read.format("csv").option("header", "true").load("/FileStore/tables/movies.csv")

**Exploratory analysis of the data (if needed)**

// **Analyze data**

val RatingsCount = ratings\_DF.count()

val UsersCount = ratings\_DF.select(ratings\_DF.col("userId")).distinct().count()

val MovieCount = ratings\_DF.select(ratings\_DF.col("movieId")).distinct().count()

println("Got " + RatingsCount + " ratings from " + UsersCount + " users on " + MovieCount

+ " movies.")

**// Randomly split** ratings RDD **into training data RDD** (70%) and **test data RDD** (30%)

val splits = ratings\_DF.randomSplit(Array(0.70, 0.30), seed = 12345L)

val (trainingData, testData) = (splits(0), splits(1))

**Model development**

**Build Model:**

val model = new ALS()

.setIterations(10)

.setBlocks(-1)

.setAlpha(1.0)

.setLambda(0.10)

.setRank(10)

.setSeed(22L)

.setImplicitPrefs(false)

.run(ratingsRDD)

**Evaluation of the model**

**Compute RMSE and Test**

Test RMSE = 0.9137996239271179

**Execution of the model for Movie Recommendations**

// Making Predictions. Get the top 5 movie predictions for user 232

println("Rating:(UserID, MovieID, Rating)")

println("----------------------------------")

val topRecsForUser = model.recommendProducts(232, 5)

for (rating <- topRecsForUser) {

println(rating.toString())

}

println("----------------------------------")

// Movie recommendation for a specific user. Get the top 5 movie predictions for user 668

println("Recommendations: (MovieId => Rating)")

println("----------------------------------")

val recommendationsUser = model.recommendProducts(232, 5)

recommendationsUser.map(rating => (rating.product, rating.rating )).foreach(println)

println("----------------------------------")

**Code and screenshots of how everything was executed using Databricks-Scala.**

Graphical user interface, text, application, email

Description automatically generated

**Pre-processing:**

Graphical user interface, text, application

Description automatically generated

**Analysing the Data**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Splitting the data into Train and Test**

Graphical user interface, text, application

Description automatically generated

**Model development**

Text

Description automatically generated

**Evaluation of Model:**

Graphical user interface, text, application

Description automatically generated

**Execution of the model for Movie Recommendations**

A screenshot of a computer

Description automatically generated with medium confidence

**Using the Model to Predict**

The predictive model MatrixFactorizationModel of MLlib ALS, which is now extended as ALSModel, offers a method called recommendProducts.

The Method recommendProducts takes two parameters: user id and the number of items to be returned. The model predicts the top number of items a user will like. Here the items are the Movie ID

Here the model predicts movie recommendations for user 232.