Movie Recommendation by Collaborative Filtering using the Netflix Data

-By Kavya Pothula

Under the guidance of :

Vahid Behzadan

Description:

This project is about the netflix recommendations using collaborative Filtering using Netflix Data for users.In this we would be building Recommendation models using collaborative filtering to recommend movies to users in netflix.

Motivation:

Now-a days, we often see that there are recommendations whenever we see any entertainment applications or any shopping website. Recommendations are generally an algorithm which suggests the users the products which they may like depending on their past events or behaviour.usually whenever we open youtube or Netflix, we can see a column stating Recommended for you, which contain the vedios which we may like.Those are retrieved using this collaborative filtering.

They have a huge database of users which includes the type of movies they had watched and how many times hey had watched.They may also combine the users who watch similar movies and depending on these data the perform recommendation so that they can give the most viewable content to users and thus increases their business.

Recommender system:

It is like a information filtering system which aims in predicting the preferences or ratings a user would give to a particular one.This system is now widely used in shopping websites, food applications and entertainment applications.It is of two types.We may use them seperately or combined.They are:

1.Collaborative filtering

2.Content Based Filtering

Collaborative filtering:

Collaborative Filtering is based on data collected from other users.It follows the approach that if users have watched same type of movies in the past, there is a chance that they make like same type of movies in the future also.By locating peer users with same rating history or content history we can easily generate recommendations for the future.

Collaborative filtering methods are classified as model-based and memory based.

Model based approch uses machine learning algorithms to predict the movies which the user may like.Whereas memory based alorithms depends on others users and their interests.It is of two types user-item and item-item.A user-item takes a particular user and find other users whose watch similar movies.In item-item approach, we consider an item and finds users who likes it and find other items which those users likes.This model takes item as input and gives other items liked by the user as output.

System-Configuration:

I used AWS EMR cluster and created a jupyter notebook.

Setting up an EMR Cluster:

1. Login to AWS account.

2. Go to services and select EMR.

3. Click on Create Cluster.

4. we get a page where we need to fill the details.They are:

    - We need to enter a cluster name.
    - Select spark in the applications box.
    - Select a security key, we may also select an existing one or create a new key-pair.
    - Click on create cluster
    - In few minutes, the cluster will be ready to use.
    - Once the cluster is ready, we can go to ec2 instance page and we can find master and slave craeted for the cluster.
    - By using the IPV4 node address we can open a jupyter notebook by connecting to SSH.

    We also used Spark in notebook to create this.

    Loading Files in S3 Bucket:

    We need to load the files in s3 bucket so that they can be used in jupyter notebook.Steps to create a S3 bucket are:

    - Open aws
    - Choose S3 bucket
    - Click on create bucket
    - Fill the required columns and create bucket

- Go to the bucket and click on upload files
- Now choose the files you want to upload and click on confirm
- The files will be uploaded to the S3 bucket

  Now we can use the URL of this bucket and use these files in jupyter notebook.

Implementation:

- Import pyspark
- Import libraries which we need
- Load the data and create dataframes for training and testing data
- Analyse the train and test data
- Check the estimated average overlap of items for users
- Check the estimated average overlap of users for items
- Create an approach and find the ratings,mean squared roots and root-mean square error
- Using this we can find the movie ratings and recommend the movies.

Finding the distinct users and items in testing data

```
#finding out the distinct users and items/movies in the testing set

print("The distinct items in testing set are :",df_testing.select('movieID').distinct().count())
print("The distinct users in testing set are :",df_testing.select('userID').distinct().count())
```

```
[Stage 12:>                                                         (0 + 1) / 1]21/12/15 16:49:01 WARN Utils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.debug.maxToStringFields' in SparkEnv.conf.
```

```
The distinct items in testing set are : 1701
```

```
[Stage 18:>                                                         (0 + 1) / 1]
```

```
The distinct users in testing set are : 27555
```

Finding the distinct users and items in training set

```
#finding out the distinct users and items(movies) in the training set

print("The distinct items in training set are :",df_training.select('movieID').distinct().count())
print("The distinct users in training set are :",df_training.select('userID').distinct().count())
```

```
The distinct items in training set are : 1821
```

```
[Stage 30:============================================>                (3 + 1) / 4]
```

```
The distinct users in training set are : 28978
```

Find average rate of the movies

```
#checking overall average rate of the movies with their counts  in the training set

averageMovieRatingsTraining = pd.DataFrame(training_pd.groupby('movieId')['ratings'].mean())

averageMovieRatingsTraining['counts'] =pd.DataFrame(training_pd.groupby('movieId')['ratings'].count())

averageMovieRatingsTraining.head()
```

	ratings	counts
movieId		
8	3.055104	2831
28	3.760127	12244
43	2.310345	58
48	3.620648	1666
61	2.385965	57

Average User Ratings

```
#checking overall average rate of the user with their counts  in the training set

averageUserRatingsTraining = pd.DataFrame(training_pd.groupby('userId')['ratings'].mean())

averageUserRatingsTraining['counts'] =pd.DataFrame(training_pd.groupby('userId')['ratings'].count())

averageUserRatingsTraining.head()
```

	ratings	counts
userId		
7	3.903846	104
79	3.630952	84
199	3.943662	71
481	4.351351	74
769	3.193878	98

ALS Model Approach¶

We use Alternating Least Square algorithm in this.

Importing the important libraries for the ALS algorithm to work would be our first step. Some of the libraries we will need as part of implementation are the regressor evaluator which will help us to measure the performance of our ALS model and import the ALS model itself.

ALS algorithm need some of the additional parameters to be fed like maxIter, regParam, ranks etc with specifying the actual inputs like userCol, itemCol and ratingsCol.I have considered coldStartStrategy which plays a vital role in eliminating the NAN values if the data have

ALS Model Approach

In [69]:
```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row
```

In [70]:
```
#Joining the actual dataframes with movie dataframe for accurate understandings

df_training_join = df_training.join(df_movies,on=['movieId'],how='inner')

df_testing_join = df_testing.join(df_movies,on=['movieId'],how='inner')
```

Approach 1:

```
In [71]: als = ALS(maxIter=5, regParam=0.08, userCol="userId", itemCol="movieId", ratingCol="ratings",
                   coldStartStrategy="drop")
         model1 = als.fit(df_training_join)



         # Evaluate the model by computing the RMSE on the test data
         predictions1 = model1.transform(df_testing_join)
         evaluator = RegressionEvaluator(metricName="mse", labelCol="ratings",
                                         predictionCol="prediction")

         evaluator1 = RegressionEvaluator(metricName="rmse", labelCol="ratings",
                                         predictionCol="prediction")



         rmse = evaluator1.evaluate(predictions1)
         mse = evaluator.evaluate(predictions1)

         print("Mean squared error = " + str(mse))
         print("Root-mean-square error = " + str(rmse))
```

Approach 2:

```
: als = ALS(maxIter=10, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="ratings",
            coldStartStrategy="drop")
  model2 = als.fit(df_training_join)



  # Evaluate the model by computing the RMSE on the test data
  predictions2 = model2.transform(df_testing_join)
  evaluator = RegressionEvaluator(metricName="mse", labelCol="ratings",
                                  predictionCol="prediction")

  evaluator1 = RegressionEvaluator(metricName="rmse", labelCol="ratings",
                                  predictionCol="prediction")



  rmse = evaluator1.evaluate(predictions2)
  mse = evaluator.evaluate(predictions2)

  print("Mean squared error = " + str(mse))
  print("Root-mean-square error = " + str(rmse))
```

I have considered two approaches by changing the parameters to check the model performance with root mean square error (RMSE) values and the mean square error (MAE) values. The approach in which the maxIter = 10 , regParam = 0.01 were selected, RMSE is about 0.84 and MAE with 0.7 were achieved which are less and the best comparatively.

Once the model is fit and predictions are done on testing data, its time to check the user and item predictions.

I have done that for both approaches and fot the User recommendations and movie recommendations using both approaches.

From the above two model performances we can say the RMSE value is comparitively less when maxIter = 10 and regParam = 0.01. Hence this would be our best ALS model.

I have created a user dataset which have movie id,user Id, and rating and I have randomly entered the data on my own.i created a dataframe for it and combined it with training data

Finally we can see them in actual data.By this we can recommend the movies to users.

Conclusions:

Recommendation Algorithm works well in getting the recommendations to the users.This is very helpful for the users so that they can easily have access to the products they have and this improves the constumer satisfaction, so that it helps the providers to gain more constumers.This even has a capability to change the recommendations based on the recent activity of the users.Thus I think this algorithm is useful for the providers to increase their business by providing the recommendations.