DISTRIBUTED AND SCALABLE DATA ENGG. FINAL PROJECT REPORT

(DSCI-6007-01-S20)

On

NETFLIX RECOMMENDATION SYSTEM

By

AKHIL

Under the guidance of

VAHID BEHZADAN, Ph.D.

Tagliatela College of Engineering

Department of Data Science

300 Boston Post Rd, West Haven, CT 06516

ACKNOWLEDGEMENT

Our most sincere and grateful acknowledgement is due to this sanctum, **University Of NewHaven,** for giving us the opportunity to fulfill our aspirations and for good career.

We express our heartfelt gratitude to our guide **Dr. Vahid Behzadan, Ph.D.** for his esteemed guidance. We are indebted for his guidelines that proved to be very much helpful in completing our project successfully in time.

Our special thanks to our classmates for providing their special guidance and support for the completion of our project successfully.

Submitted By

Akhil Naraharisetti (00667837)

MOTIVATION

In this project your group will predict 100,000 movie ratings for users in a subset of the original NETFLIX data issued for the NETFLIX Prize. This challenge aimed at substantially improving the accuracy of predictions about how much someone is going to enjoy a movie based on their movie preferences. It was issued by the Netflix company and on September 21, 2009 a $1mio Grand Prize was awarded to the winning team.

The purpose of this project is to analyze the NETFLIX data using SPARK in EMR cluster and, based on the outcomes of this analysis, develop a feasible and efficient implementation of the collaborative filtering algorithm in SPARK. Execute your program on Amazon EMR to get the rating predictions and evaluate those ratings by comparing them to the provided true ratings.

Why is the application important in the real-world and in the context of Data Engineering?

A recommendation system is obviously for the benefit of Users, saying that it saves lot of time for the Users who are willing to choose items (movies, w.r.t project) on the basis of ratings. Also,

Data Engineering comes into picture for accessing, collecting and cleaning data from applications and systems into usable state. Scalability and Security is ensured. So, we build the entire code in EMR cluster of AWS with PySpark and get the rating predictions.

We use Apache Spark because it is a Distributed Data Processing Engine for parallel computation techniques. Moreover, it is provided with libraries like Machine Learning, SQL, stream processing and what not for creating an entire application in itself. We are using PySpark in this project where it uses Python API on Spark. Also, implementing Python is much easier because in Spark-shell you have to be using Scala which is a different language and should be learnt in order to use Spark API. Instead we use PySpark.

APPROACH

This project is divided into 3 parts:

Firstly, setting up the environment. Like, setting up and configuring EMR cluster in AWS with PySpark.

Secondly, Analyzing the Data, implementing Algorithm by Collaborative Filtering approach and then evaluating the model predictions.

Thirdly, Testing the approach by adding myself as a new User to the dataset. I will you the same approach to output predictions for the movies I haven't rated and ranking those in the decreasing order of the rankings.

SYSTEM CONFIGURATION

We have to create an AMR cluster to start working on this project. Steps for the EMR cluster configuration are as follows;

1. Go to you AWS console and make sure you have enough credits.
2. Select option EMR under AWS console, and then select create Cluster.
3. Give a desired Cluster name, select SPARK under applications and leave everything else default. You can create an EC2 keypair for Authorized Access. Select Create Cluster.
4. The cluster starts initiating. You can clone, terminate or AWS CLI export the created Cluster.
5. Meanwhile, go to the Notebooks on EMR service.
6. Choose Create Notebook
7. Configure the Notebook and then choose create Notebook.
8. Open the created Notebook by choosing 'Open with Jupyter'.
9. A new tab pops up with Jupyter Environment. You have successfully created a Jupyter Notebook in the EMR cluster.
10. Now create a PySpark notebook. The setup is done.

Another important task is to upload the data files in S3. No need of creating an another S3 bucket to upload as the EMR cluster creates a default S3 bucket. You can upload the data files in there. For that,

1. Go to S3 under AWS services.
2. You'll find 2 buckets created by EMR cluster. Choose one of them.
3. Upload the Netflix .txt files in it.

NOTE : I will not be uploading any data files in my Github Repo as per the rules of the project. Sorry for the inconvenience.

BIG DATA APPLICATION

DATASET DESCRIPTION

You can go through the readme.MD file for more description about the project. Coming to the Dataset, I have been provided with compressed folder containing;

1. Trainingset.txt
2. Testingset.txt
3. Movie_title.txt
4. Description.txt

The Training set consists of about 3.2 million samples while Testing set have about 100,000 samples.

Both Training and Testing sets have data with labels: MovieID, UserID and Ratings. Movie_title.txt have labels: MovieID, Year of Release and Movie name.

ANALYZING THE NETLFIX DATA

For the effective implementation of my approach I have to analyze the data. Implementing the Collaborative Filtering approach for finding similar Users or similar Items (movies, in our case).So, what is Collaborative Filtering? It is a technique that can filter out items (movies, in our case) that a user might like on the basis of reactions by similar Users.

I have been exploring different Collaborative filtering algorithms and implementing on Sub sample of Training set for obtaining better similarities. But firstly, we have to find which one has a better similarity, User similarity or Item similarity.

Let me explain this theoretically. There are 2 types of Collaborative Filtering:

Memory-based: This category includes algorithms that are memory based, in which statistical techniques are applied to the entire dataset to calculate predictions. To find the rating R that a user U would give to a movie M, the approach includes :

1. Finding Users similar to U who have rated movie M
2. Calculating the rating R based on the ratings of users found in previous step.

Finding Similar Users;

We can take a subsample of the training set, which have rated 2 same movies. If you plot Movie1 to Movie2 the points on the graph would be users. Calculate the distance between 2 points using Eucledian distance formula. The lesser is the distance, the more is the Similarity. This is a basic approach.

For being more specific with the User-User similarity, we can implement Cosine Similarity. Calculate the angle between 2 points. The more is the angle ,the lesser is cosine of the angle, the lesser is similarity. We basically know, Cosine is a decreasing function. We ultimately find the Cosine distance of the vectors.

Later, you'll be adjusting these vectors by eliminating the bias and thus finding centred Cosine.

There are many others like Jaccard Similarity, which finds out the Jaccard co-efficient. You can implement either of these, based on datasets' complexity.

These are the two approaches which could be used to find both User and Movie Similarity. You can predict the a user's rating R for a movie M will be close to the average of the ratings given to movie M by the users similar to U. So, basically you take mean of all the ratings.

For a better clarity, we are going to look which approach we are going to implement i.e, User based or Movie based. User based means approach based on similar Users.

USER-BASED vs ITEM-BASED APPROACH(They come under Memory-based):

1. User based , for example, if a user 'A' rates 10 movies, out of which 'n' users have rated 6 movies. We can say, A and other 'n' have similar tastes. So, the users 'n' recommend their top rated movies other than these 6 to user 'A' to watch.
2. Item-based, for example, if you give high rating to a movie, then you be recommended with other movies by others who gave a high rating to the same movie.

In general, Item-based is much faster and stable than the User-based approach. This is because, the average rating received by movies doesn't change as quickly as the average rating received by users. I hope you understand this well. Moreover, the number of users are immensely more in number than the Movies count. What all I mean is, Item-based approach is much efficient and faster for larger datasets like these. Moreover, they perform much better when the Rating's Matrix is Sparse.

Rating's Matrix is obtained by multiplying 'n' User vector to 'm' Movies vector. Definitely it is sparse rather than dense. We have to convert this Ranking(sparse) matrix to Dense Matrix. We can achieve it by another approach which Model-based.

Model-based: This is much compatible than the Memory-based approach if it comes to the complexity of the Data. It compresses the user-movie matrix. We perform Matrix Factorization techniques to achieve this. Few of the popular algorithms which performs Matrix Factorization are:

1. Singular Value Decomposition
2. Principal Component Analysis
3. Alternative Least Squares
4. Non-negative Matrix factorization

You can find the implementations of these algorithms in various libraries for PySpark.

COLLABORATIVE FILTERING APPROACH I IMPLEMENTED

I have implemented *Alternative Least Squares* algorithm for the Matrix Factorization. ALS is an iterative optimization process where we for every iteration try to arrive closer and closer to a factorized representation of our original data.

We have our original matrix R of size u x m with our users, movies and some type of feedback data. We then want to find a way to turn that into one matrix with users and hidden features of size u x f and one with items and hidden features of size f x m. In U and V we have weights for how each user/item(movie) relates to each feature. What we do is we calculate U and V so that their product approximates R as closely as possible: $R \approx U \times V$.

The least squares approach in it's basic forms means fitting some line to the data, measuring the sum of squared distances from all points to the line and trying to get an optimal fit by minimising this value.

With the alternating least squares approach we use the same idea but iteratively alternate between optimizing U and fixing V and vice versa. We do this for each iteration to arrive closer to $R = U \times V$.

Above is some theoretical explanation about the ALS algorithm. Let's implement it.

IMPLEMENTATION

So, coming to the Code implementation (code through the code along for better understanding)

1) Importing PySpark and other necessary modules. Remember, we are performing this on Jupyter Notebook with PySpark in EMR cluster.
2) Start the Spark Application by creating the Spark session.
3) Give the file path of the Training set which would be S3 bucket, where we uploaded the datafiles in the beginning.
4) Create scheme for the Dataframe and create the Spark DataFrame using Spark.read().
5) Performing some statistical analysis for the knowabouts of the data.
6) As we see in the code, there are 3.255352 million rows in the Training dataframe
7) We obtained the Mean, Standard Deviation, Minimum and Maximum.
8) Imported ALS model from PySpark package.
9) ALS parameters : maxIter, regParam, userCol, ItemCol and ratingCol
10) After building the model, fit the Training dataset into the model.
11) In the similar fashion, we create Testing dataframe. Total movies in this dataframe are 1,701 and total Users are 27,555.
12) There are 100,478 rows in Testing dataframe.
13) Now, make predictions from the Testing dataframe.
14) Then, obtaining the first 20 predictions ranking in descending order.
15) We see Movie ID 2939 got prediction 6.639 which is the highest of all.

EVALUATION

The 2 Evaluation metrics to be used are Root Mean Squared Error and Mean Absolute Error. There are in-built modules available to calculating these 2 metrics in PySpark package.

After evaluating the model using these 2 metrics, we obtained

RSME = 0.8532122

MAE = 0.67024

Now, does my approach work for my own preferences?

I have added my myself as new user with UserID = 1, I have rated 10 Movies (if you see the code I have clearly mentioned my preferences). So I have added up 10 more rows to the Training set. I have uploaded it in S3 and then, created a new dataframe including my preferences.

I have fit the data into the new model of ALS(with same hyperparameter values). Then I made predictions for the movies I haven't watched in the decreasing order. The highest prediction of 6.427806 was the movie with MovieID = 218. So, I have mapped this to the movie_list.txt to get this movie details. Go through the code for better understanding.

The top movie the model predicted for me was 'Triumph'. I couldn't agree more.

NOTE: I have also implemented this complete project in my local system i.e, without using EMR cluster in my Anaconda environment for better understanding of PySpark Dataframes. I have produced the same results and will be uploading it in my Github. This can be avoided for evaluating purposes.

CONCLUSION

A lot of things came into my vision and understanding while contributing to this project. The complete importance of Apache Spark was very well understood. Everyone will definitely get benefitted once acquired the knowledge of PySpark and it's use cases. Distributed Computing Engine alias Apache Spark performance is very high. The ability of Multiple computations is making it far more efficient.

Things Learnt so far:

- Python Multi-processing
- EMR cluster, S3 and other AWS services and their real-time use cases
- Collaborative Filtering Approach as a Recommendation System
- Matrix Factorization Algorithms (it's vast and full of math, can dive deeper into it for much better understanding of the algorithms other than ALS. Like SVD, NMF, etc.)

Future works:

- Complete PySpark(using RDD's) and Spark-shell (Using Scala)
- Building few other recommendation systems with user-item approach.