MOVIE RECOMMENDATION SERVICE (PUMKINMETER).

SOURCE: https://www.codementor.io/@jadianes/building-a-recommender-with-apache-spark-python-example-app-part1-du1083qbw

BUAN 5315: BIG DATA ANALYTICS
PASCAL NTAGANDA

Introduction

The Pumkinmeter is a tool that uses the concept of collaborative filtering, to be able to provide recommendation services to its users. Specifically, it recommends movies and shows based on customer interests, in relation to the interests and preferences of other member users to the network. With the help of PySpark, a python API with a distributed computing framework, large data pools can be worked on with Pyspark libraries that enable real-time and large-scale data processin [1]. Therefore, a machine learning model with be used to ascertain collaborative filtering procedures for efficient movie recommendations, after necessary data processing.

Dataset

The datasets used in this case are obtained from Grouplens Research website. Grouplense has made the ratings data sets available freely, with various dataset sizes and periods of time. The small dataset has 100,000 ratings and 3600 tag application applied to 9,000 movies by 600 users and last updated in September of 2019. The large and full dataset has 27,000,000 ratings and 1,100,000 tag applications applied to 58000 movies by 280,000 users. It also has tag genome data with 14 million relevance scores across 1000 tags. It was last updated in September 2018. LINK: https://grouplens.org/datasets/movielens/

Technical Details

With the utilization of Jupyter notebooks a web based interactive computing platform, Pyspark is used to ensure efficient large and real time data processing. Firstly, data is loaded directly from the Grouplens website, and no need for deep cleaning since it's already structured. This is followed processing obtain the necessary values of ratings and movies from the datasets. With the constraints of the virtual machine used from Amazon Web Services (AWS), it makes processing the large data hectic. Therefore, during the selection of Alternate Least Squares (ALS) model parameters, the small dataset is used.

ALS is the model used to be able to find similarities in the fitted data [2]. It can create lower dimensionality matrices with a product equal to the original one, hence expressing similar features. This allows computational efficiency and better results due to the compact nature of items used. The machine learning library MlLib in Pyspark, is used to train the ALS model on the small dataset, with separation into relational distributed datasets created through partitioning the nodes or data points. Here the points are validation, testing and training relational distributed datasets (RDD) [3]. The same procedure is now used on the large dataset, by creating the training and testing RDDs. Recommendations are then estimated from the new user information that is added to the original data and running the recommendation model.

After creation of the recommendation machine learning model based on Alternate Least Squares, the model is tested on two users under two scenarios. Each user is prompted to enter and rate their ten movie options with ratings and their movie id. This information is added to the original dataset and run through the model. For each user, movie recommendations are made based on filtering movies with less than 25 ratings in scenario 1 and filtering out movies with less than 100 ratings in scenario 2.

Debugging Details

In AWS, the virtual computer required a stronger instance type to enable faster computations. More importantly, the ALS model could only be trained and validated on the small dataset. This was mostly due to the lack of enough memory in the server, in addition to the long processing time and model training time if the large dataset was used. The necessary changes were done after trials to operate on the large dataset to train and validate the model. Using the smaller dataset is therefore faster and economic, and then run the validated model on the large dataset. Another issue arose during attaining of results from the scenarios whereby the program would crush if the new user ratings were added, followed by training the model again. Therefore, each option scenario would be run independently from the rest in different files or from the beginning.

Results

There was no doubt about the ability of the model to provide efficient recommendations for users. Therefore, as expected the recommendations were in line with the new user ratings added to the dataset. Each user was able to have independent recommendations that skewed to their interests. Their interests were matched through their ratings of movies in the different scenarios. The indicator was the genre and content in movies being similar to their highest ratings and less alike to their lower rated movies. The Scenarios express the adjustment of the model depending on the interests of the users, being more precise when the ratings with less than 100 ratings was used.

Insight

Understanding the procedure is a little different from obtaining results. The recommendation service needs to be fast and efficient enough to be used by customers. During the process, it shows that even though the use of Resilient Distributed Datasets is effective to reduce processing time and work on large datasets, there is still need of some form of high computing power and storage. With effective parallelization, the process can be faster. Conclusively, Pyspark is efficient but still needs support in terms of storage and computing power, hence incurring some extra costs (not as much as would be required if they were not used). With all that a side, the recommendation algorithm works well, but could be better in terms of real time processing.

References

[1]Dianes, A Jose, *Building a Movie Recommendation Service with Apache Spark & Flask - Part 1, 2015*. https://www.codementor.io/@jadianes/building-a-recommender-with-apache-spark-python-example-app-part1-du1083qbw

[2]Ramsingh, J. (2022). PySpark toward Data Analytics. In *Big Data Applications in Industry 4.0* (pp. 297-330). CRC Press.
Singh, P. (2022). Recommender Systems. In *Machine Learning with PySpark* (pp. 157-187). Apress, Berkeley, CA.

[3]Ajitsaria, Abhinav. Build a recommendation engine with collaborative filtering, 2022. https://realpython.com/build-recommendation-engine-collaborative-filtering/#:~:text=Collaborative%20filtering%20is%20a%20technique,similar%20to%20a%20particular%20user.