MOVIE RECOMMENDATION SYSTEMS USING PYSPARK AND CLOUD COMPUTING

BIA-678 Big Data Technologies – 19 Spring

AUTHOR
Yang Liu, Jiawei Xue, Shuqiong Chen, Ashish Negi,

Abstract

In this project, we built a collaborative filtering recommendation system using PySpark and Alternating Least Squares (ALS) algorithm. Specifically, we built a model-based collaborative filtering system that issues recommendations of movies based on the relationship between queried user and movie and the rest of the rating matrix. We used the root of the mean square error (RMSE) as our model evaluation metrics and compared the model result in different data scale and different platform. The result turned out that the local, HFSC Cluster and Amazon Web Service (AWS) works well when reducing the RMSE as the scale increased, while local perform better than HFSC Cluster when comparing the runtime of the ALS building process. Finally, we improve the ALS model runtime performance utilizing the AWS.

1 Introduction

People are probably already familiar with the output of these types of recommendation engines where a website tells you something along the lines of, "If you like that, then you will probably like this." People may have likely seen these types of recommendation on their favorite retail or media streaming websites. These recommendations are generated through different types of data that others as a user or customer provide either directly or indirectly. When someone purchases something online, or watch a movie, or even read an article, he or she are often given a chance to rate that item on a scale of 1 to 5 stars, a thumbs up or thumbs down, or some other type of rating scale. Based on his or her feedback from these types of rating systems, companies can learn a lot about his or her preferences. Based on that, they can offer you recommendations based on the preferences of users that are similar to that person.

In this project, our goal is to build a recommendation engine using Altering Least Squares or "ALS" in PySpark. We also wanted to measure the model performance regarding different data scale and platform. The first thing we needed to determine is what kind of recommendation g system we should choose. In the first step, we introduced different kinds of recommendation systems and different kinds of ratings. Then we introduced our dataset and performed exploratory data analysis. Then we explained how the ALS works and executed the ALS model on different platforms: local, HFSC cluster, and AWS. After that, we compared the runtime and the root of mean squared error, or "RMSE" among these platforms. Finally, we issued our conclusion and future work.

2 Recommendation System

2.1 Type of Recommendation System

Generally, there are two types of recommendation system, the content-based filtering and the collaborative filtering. Content-based filtering methods are based on a description of the item and a profile of the user's preferences. These methods are best suited to situations where there is known data on a movie (director, genre, length, etc.), but not on the user. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on movie features.[1]

Another approach to designing the recommendation systems is collaborative filtering. Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. The system generates recommendations using only information about rating profiles for different users or

items. By locating peer users/items with a rating history similar to the current user or item, they generate recommendations using this neighborhood[1]. In other words, the recommendation system will recommend movies to users based on the users' similarity.

2.2 Types of Ratings

There are two types of ratings: the explicit ratings and the implicit ratings.
The examples of these explicit ratings are when you input a number of stars or something like a thumbs up or thumbs down, in other words, the users explicit state how much they like or dislike something.

The implicit ratings are based on the passive tracking of your behavior, like the number of movies you've seen in different genres. Fundamentally, implicit ratings are generated from the frequency of your actions. For example, if you watch 30 movies, and of those 30 movies, 22 are action movies, and only 1 is a comedy, the low number of comedy views will be converted into low confidence that you like comedies, and the high number of action movie views will be converted into a high confidence that you like action movies. These probabilities are then used as ratings.

2.3 Methodology

In this project, we decided to build a collaborative filtering recommendation system using explicit ratings, one of the most widely accepted and effective recommendation systems.

3 Data and Data Preparation

3.1 Data Source

We acquired our data from the MovieLens dataset on the grouplens website [2]. This dataset) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 27753444 ratings and 1108997 tag applications across 58098 movies. These data were created by 283228 users between January 09, 1995 and September 26, 2018. This dataset was generated on September 26, 2018. Users were selected at random for inclusion. The data are contained in the files *genome-scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv and tags.csv*. Only the ratings.csv and movies.csv are related to our recommendation system. So, we only used these two files to build our recommendation engine. Also, considered the capability of our local machine, we only took a subset, 1 million rows, nearly 65MB, from the whole dataset.

3.2 Data Processing and Exploratory Data Analysis

We used the PySpark to process the data and implement model. First, we created two RDDs by reading file from *ratings.csv* and *movies.csv*. Then we created two Spark DataFrame using SparkSession module. We took a glance of the data by *spark.show( ) method.* The outputs show that all ratings are contained in the file *ratings.csv,* each line of this file after the header row represents one rating of one movie by one user, and has the following format: *userId, movieId, rating, timestamp.* Ratings are made on a 5-star scale, with half-star increments (0.5 stars – 5.0 stars). Movie information is contained in the file *movies.csv.* Each line of this file after the header row represents one movie, and has the following format: *movieId, title, genres.*

Next, we used Spark SQL to perform some aggregate queries, such as the min( ), max( ), avg( ), and count( ), etc, to get a better understanding of the two datasets. The results show that at least

one movie was rated by one user, while the maximum movies that were rated by one single user were 4874, an unbelievable number! And, the average number of movies that were rated by each user was around 98. At this step, we didn't know whether 98 is a large number or small number compared to the whole dataset. Therefore, perform further aggregate queries, and get that the total number of unique users is 20507, and the total number of unique movies are 26030, which means the rated movies by each user, 97, is only 0.38% of the entire movie list. That makes sense because we only watch a small set of movies in real life. However, 0.38% is a really low percentage, which makes have the risk to cause "cold start", so we decided to check it in detail to make sure whether we need to take specific action to handle the cold start problem. We calculated the sparsity of the dataset. The sparsity is used to assess whether further preparation is needed in order to adequately prepare it for ALS model. The sparsity is denoted as sparsity = 1 – (numerator * 1.0 / denominator), where the numerator is the actual number of ratings in the dataset, and the denominator is the number of ratings if every user watched every movie. It turns out that the sparsity of ratings.csv is 99.63%, which means more than 99 percent of the cells in the ratings.csv are blank. This is a very sparse matrix, which will easily cause the "cold start" problem. In the next section, we set up a parameter called "cold start strategy" to solve this problem [3].

4 Alternating least squares model

The steps that we took to create the ALS model are: (a) splitting the data into training and testing dataset (b) building ALS model with hyperparameters and tuning hyperparameters by Grid-search (c) building 5-fold cross-validation (d) fitting the model on the training data (e)extracting the model from cross-validation (f) generating predictions on testing data (g) evaluating predictions using RMSE.

4.1 Sampling

We split the dataset into training and testing dataset as a 4:1 ratio in order to get as much data for training model as possible, and at the same time to avoid overfitting. Also, we wanted to measure model performance under scale. Therefore, we downsized our raw data into different sample size from 500 to 1,000,000 with a double-increasing method.

4.2 Building the ALS model with Hyperparameters

We used the spark.ml module to perform the alternating least squares (ALS) algorithm to build our recommendation system. We set up five hyperparameters in the ALS model: rank, iterations, nonnegative, implicitPrefs and cold start strategy.

4.2.1 Rank

When we have a matrix that contains users and movie ratings, ALS will factor that matrix into two matrices, one containing user information and the other containing movie information. Each matrix takes the respective labeled axis from the original matrix and is given another axis that is unlabeled. The unlabeled axes contained what's called latent features. These latent features represent groups that are created from patterns in the original rating matrix and the values in these columns represent how much each item falls into these groups. The number of latent features is referred to as the "rank" of these matrices. We need to decide how many of these latent features the ALS will create because too little latent features may not be helpful enough, while too many latent features may be meaningless or time-consuming.

4.2.2 Iterations

ALS is an iterative algorithm. In each iteration, the algorithm alternatively fixes one factor-matrix and solves for the other, and this process continues until it converges. First, ALS factorize the original sparse matrix into two different matrices, which can be multiplied back together to produce an approximation of the original matrix. In order to get the closest approximation of the original matrix R, ALS first fills in the factor matrices with random numbers and then makes slight adjustments to the matrices one at a time until it has the best approximation possible. In other words, ALS holds the matrix R and matrix U constant, and make adjustments to the matrix P. It then multiplies the two factors matrices together to see how far the predictions are from the original matrix using root means squared error or RMSE as an error metric. The RMSE basically tells you, on average, how far off your predictions are from the actual values[4]. Keep a mind that when calculating RMSE, only the values that existed in the original matrix are considered. The missing values are not considered. ALS then holds P and R constant and adjust values in the matrix U. The RMSE is calculated again, and ALS switched again and calculates the RMSE again. ALS will continue to iterate until instructed to stop, at which point, ALS has the best possible approximation of the original matrix R. Finally, when the RMSE is fully minimized, ALS simply multiplies the two matrices back together, and the blank cells in the original matrix are filled in with predictions.

4.2.3 Nonnegative

Since ALS will factorize the rating matrix into two factor matrices, which can be multiplied back together to get an approximation of the original matrix, and it would be meaningless if the two factor-matrix contains negative values, so we set the parameter "nonnegative =True" in the ALS model.

4.2.4 ImplicitPrefs

Specifies whether to use the explicit feedback ALS variant or one adapted for implicit feedback data. We set implicitPrefs as False because the data we used is explicit ratings.

4.2.5 Cold-start Strategy

When making predictions using an ALSModel, it is common to encounter users and/or items in the test dataset that were not present during training the model if the dataset is sparse. By default, Spark assigns NaN predictions during ALSModel.transform when a user and/or item factor is not present in the model. However, this is undesirable during cross-validation, since any NaN predicted values will result in NaN results for the evaluation. This makes model selection impossible. So we chose to set the coldstartStradegy parameter to “drop” in order to drop any rows in the DataFrame of predictions that contain NaN values. The evaluation metric will then be computed over the non-NaN data and will be valid.

We selected three most possible values for each rank (10 ,15, 20) and iterations (10, 15, 20) pairs, and then used the grid-search method to automatically find the most appropriate values for each parameter.

4.3 Training Model and Extracting the best model from cross-validation

We trained the ALS model with 5-fold cross-validation on the training set and extracted the best model which has the best performance on the regarding the model accuracy. The result turns out that the best model always the rank with 10 and iterations with 20 on different sample size.

For each step in section 4, we executed it in three environments: local machine, HFSC cluster, and Amazon Web Services. In section 6, we compared the performance of ALS model these three environments.

5 Recommendation Results

After building the ALS model, we wanted to see whether it works or not. Therefore, we took four steps with Spark SQL to query the desired output.

5. 1 Step 1

The ALS model uses the user-based top-N recommendation algorithms to generate the top (n) recommendations for all users. In our case, we choose N equals to 3. The generated data frame has two columns: one is the user Id column with unique user Id in each row, and another is the recommendation column that has all the recommended movie Id and its corresponding movie name to the user Id in that row. However, the output in this step has two challenges that need to be addressed. The first is that the format of the output is perfectly usable in PySpark, but it isn’t very human-readable. To resolve this, we save the data frame as a temporary table and use spark SQL to make it readable.

5. 2 Step 2

Notice that only one movieId and its respective recommendation value for each user is contained on each line, where previously, all recommendations for a given user were contained on one line. It's better to separate each them into different rows. In the new select query, we added the Lateral

view to the explode function to allow us to treat the exploded column as a table, and extract the individual values as separate rows. The new data frame has the user with one recommended movie and the corresponding predicted ratings in one row, while each user has different rows if it has more than one recommended movie. Now the output is readable.

5. 3 Step 3

We join the output of step 2 to the original movie information. It has a better view. We can see what the movie name and genres are that was recommended to that user. Another challenge is that the recommendations include predictions for movies that have been already been watched. This is because the ALS creates two factor-matrix that are multiplied together to produce an approximation of the original rating matrix. The makes the output of ALS include all movies for all users, whether they've seen them or not.

5. 4 Step 4

We filtered the movies that users have already seen by joining the recommendations table to the original movie-ratings table, and then filter the rows the value at rating columns is NOT null. Now, we only have the recommendations for movies that individual users haven't seen. This means we built the ALS model successfully.

6 Performance Measurements

We recorded the runtime and the root of the mean square error of the ALS model for different sample size in different environments. Surprisingly, the HFSC cluster runs Spark slower than

running Spark on our local machine. This may be because the chips on the HFSC cluster is too aged, not fast enough, and we just used one node on the HFSC cluster.

To improve the performance on the cloud, we set up the cluster on AWS using Amazon Elastic MapReduce or called EMR, and S3. The EMR is an open-source framework that uses Apache Spark and Hadoop to quickly & cost-effectively process and analyze vast amounts of data. S3 is an object storage architecture with scalability, high availability, and low latency. Additionally, we chose 5 m4.large nodes on EMR, 1 master node and 4 slave node.

The result of the runtime shows that the ALS model runs much faster than runs on the local machine and HFSC cluster at different scales while running ALS on HFSC cluster cost the most time. Also, the runtime of the ALS model has a positive correlation with scale in all three environments (See plots of comparison in appendix).

Another comparison of the ALS model we interested is the RMSE under scale. The results reflect that the local, HFSC cluster and AWS all reduced the RMSE as the sample size increases. The all reduced the RMSE significantly with the sample size increases when the sample size is small, and the less RMSE is reduced as the sample size gets larger and larger. However, looked at the plots of comparison (see appendix) in detail, we can found that the RMSE of ALS on the AWS deceased faster than the RMSE in the other two environments when the sample size is no large enough, while they nearly have the same RMSE when the sample size is quite large, like 1,000,000.

7 Conclusion

From the evaluation section, we can see that recommendations systems can successfully generate recommended movies with low RMSE. And we found that while the final RMSE are similar, the cloud computing platforms, AWS has much better performance than the local and HFSC cluster regarding the runtime. This definitely a strong proof that the distributed systems are more powerful to handle large scale data than the traditional method. However, there is a tradeoff between the scalability & accuracy, in other words, between runtime & RMSE. So, the appropriate scalability should be determined by business preferences and needs.

8 Future work

In this project, we built a collaborative filtering recommendation system using explicit ratings. It is powerful, but it has limitations. In some time, when a new movie is released, nobody has watched it. It doesn't exist in the ALS matrix, so this movie can't be recommended based on the user's behavior. In this case, another methodology will be helpful. The content-based filtering recommendation, which can generate by the attributes of the movies, no matter it has been seen or not. So, we plan to crawl the movies information from IMDB to add features into our existed ALS model. In other words, we will build a hybrid collaborative fileting recommendation system, in which we can make a balance between collaborative-filtering and content-based filtering.

Another thing we need to pay attention is that sometimes we don't have explicit ratings but only have implicit ratings, for example, when we go to some website to listen to songs, we don't give a ratings on the songs we listen, but it can record which song you have listened and how many times or how long you have listened for each song. In this case, we think we can also build a recommendation system using implicit ratings, and we will try it in the future.

References

[1] Wikipedia. Recommender system.

https://en.wikipedia.org/wiki/Recommender_system#Content-based_filtering

[2]MovieLens. https://grouplens.org/datasets/movielens/
[3] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. Adv. in Artif.

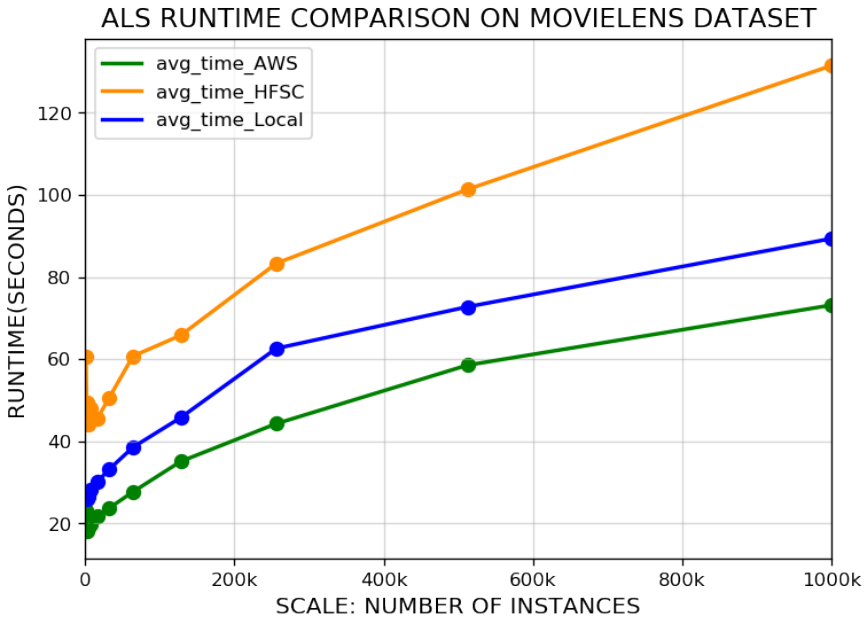
Intell., 2009:4:2–4:2, January 2009

[4] Joonseok Lee, Mingxuan Sun, Guy Lebanon. A Comparative Study of Collaborative Filtering

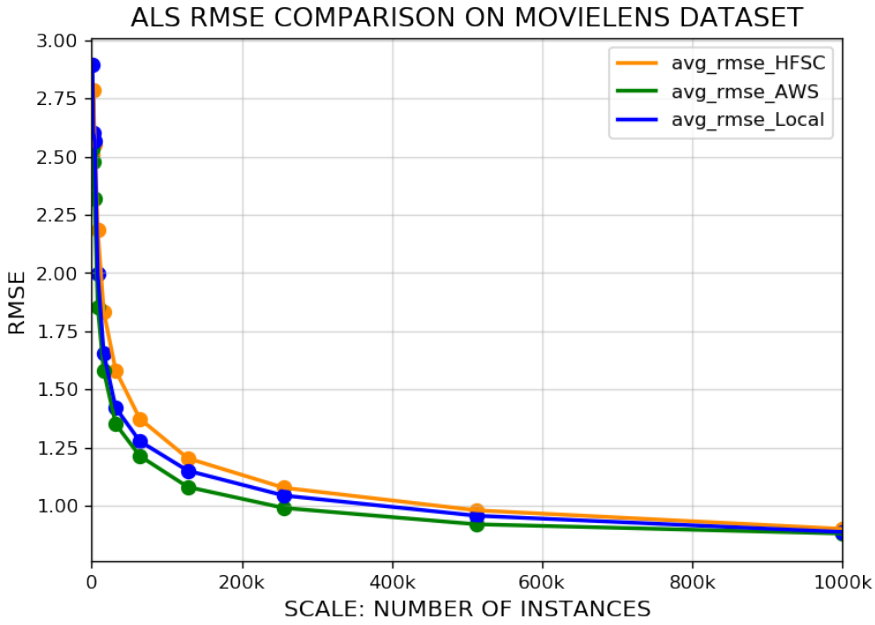
Algorithms, May 14, 2012

Appendix

(a) Execution Time Comparison



(b) RMSE Comparison



(c) Generated recommendations for all users in 5.1

```
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|  1580|[[104074, 11.8679...|
|  4900|[[26395, 8.163957...|
|  7880|[[26395, 5.075081...|
| 14570|[[565, 7.171178],...|
| 15790|[[55253, 4.84224]...|
+------+--------------------+
only showing top 5 rows
```

(d) Normalized Recommendations in 5.2

```
+------+-------+----------+
|userId|movieId|prediction|
+------+-------+----------+
|  1580| 104074| 11.867983|
|  1580|   1783|  9.381773|
|  1580|   7759|  8.470671|
|  4900|  26395|  8.163957|
|  4900|   2357|  7.987455|
+------+-------+----------+
only showing top 5 rows
```

(e) Joined table of the recommender to the movie table in 5.3

```
+-------+------+----------+--------------------+--------------------+
|movieId|userId|prediction|               title|              genres|
+-------+------+----------+--------------------+--------------------+
| 104074|  1580| 11.867983|Percy Jackson: Se...|Adventure|Childre...|
|   1783|  1580|  9.381773|     Palmetto (1998)|Crime|Drama|Myste...|
|   7759|  1580|  8.470671|   Nostalghia (1983)|               Drama|
|  26395|  4900|  8.163957|Rutles: All You N...|              Comedy|
|   2357|  4900|  7.987455|Central Station (...|               Drama|
+-------+------+----------+--------------------+--------------------+
```

(f) Filtered the recommender that only contain the movie for user that had never seen

```
+------+-------+----------+--------------------+--------------------+------+
|userId|movieId|prediction|               title|              genres|rating|
+------+-------+----------+--------------------+--------------------+------+
|   251| 114893|  8.871814|         Sins (1986)|       Drama|Romance|  null|
|   319| 168492|  8.319116|Call Me by Your N...|       Drama|Romance|  null|
|   342|  73000|  5.594062|         Nine (2009)|Drama|Musical|Rom...|  null|
|   467|   4513|  6.996867|House on Carroll ...|            Thriller|  null|
|   549|  82152|   7.58963|      Beastly (2011)|Drama|Fantasy|Rom...|  null|
+------+-------+----------+--------------------+--------------------+------+
```