Applying user based collaborative filtering to develop an efficacious model for the movie recommendation system

GATIK GOLA
Student, King's College, Taunton, UK

ABSTRACT

Recommender systems have become ubiquitous in our lives. Yet, currently, they are far from optimal. In this project, we attempt to understand the different kinds of recommendation systems and compare their performance on the MovieLens dataset. We attempt to build a scalable model to perform this analysis. We start by preparing and comparing the various models on a smaller dataset of 100,000 ratings. Then, we try to scale the algorithm so that it is able to handle 20 million ratings by using Apache Spark. We find that for the smaller dataset, using user-based collaborative filtering results in the lowest Mean Squared Error on our dataset.

INTRODUCTION

A recommendation system is a type of information filtering system which attempts to predict the preferences of a user, and make suggests based on these preferences.

There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. The content of such platforms varies from movies, music, books and videos, to friends and stories on social media platforms, to products on e-commerce websites, to people on professional and dating websites, to search results returned on Google.

Often, these systems are able to collect information about a users choices, and can use this information to improve their suggestions in the future. For example, Facebook can monitor your interaction with various stories on your feed in order to learn what types of stories appeal to you. Sometimes, the recommender systems can make improvements based on the activities of a large number of people. For example, if Amazon observes that a large number of customers who buy the latest Apple Macbook also buy a USB-C-to-USB Adapter, they can recommend the Adapter to a new user who has just added a Macbook to his cart.

Two main approaches are widely used for recommender systems. One is content-based filtering, where we try to profile the users interests using information collected, and recommend items based on that profile. The other is collaborative filtering, where we try to group similar users together and use information about the group to make recommendations to the user. Both approaches are discussed in greater detail in Section 3.

The problem that we address in this project can be formulated as follows. Let *R* be the ratings matrix with dimensions *num_users num_items* . The entry r_{ij} in the ratings matrix *R* contains a non- zero rating value given by the user *i* for the item *j*. The matrix *R* is sparse in nature i.e. most of the entries r_{ij} are missing. We generally have a few ratings or some purchase history for each user and similarly each item will have been rated by a few users, but most of the entries in the ratings matrix are generally missing. The task at hand is to predict the missing entries in the ratings matrix *R*.

The data that constitutes the ratings matrix *R* can be collected either explicitly by asking the users to rate the items or by implicitly deriving the ratings for items based on measures such as whether the user purchased the item, or whether the user clicked a certain page and like wise. We work with the MovieLens dataset, which contains explicit ratings given by users to movies.

We use the mean squared error metric to evaluate the predictions made by our system. The mean squared error is computed as
Due to the advances in recommender systems, users constantly expect good recommendations. They have a low threshold for ser- vices that are not able to make appropriate suggestions. If a music streaming app is not able to predict and play music that the user likes, then the user will simply stop using it. This has led to a high emphasis by tech companies on improving their recommendation systems. However, the problem is more complex than it seems.

Every user has different preferences and likes. In addition, even the taste of a single user can vary depending on a large number of factors, such as mood, season, or type of activity the user is doing. For example, the type of music one would like to hear while exercising differs greatly from the type of music he'd listen to when cooking dinner. to solve is the exploration vs exploitation problem. They must explore new domains to discover more about the user, while still making the most of what is already known about of the user.

$$MSE = \frac{1}{N}\underset{i=1}{\overset{N}{}}(p_{ij} - r_{ij})_2$$

where N is the number of ratings in the test partition, p_{ij} is the predicted rating for user i and movie j and r_{ij} is the actual rating.

For this research paper, I focusedon two main procedures for recommendations: Collaborative filtering & Content-based filtering.

I.

Collaborative Filtering techniques are useful ways of making recommendations based on the likings of the users. A man attribute of this feature is that if a person has liked a movie and the other person has not, then the person who has not liked the movie gets some new recommendations based on those provided by the first one. Some examples of the applications which perform this collaborative filtering are Netflix, Facebook, etc.

We explore two algorithms for Collaborative filtering, the Nearest Neighbors Algorithm and the Latent Factors Algorithm.

1.1.1 Nearest Neighbors Collaborative Filtering: In this approach, those with similar likings, similar tastes and similar rating patterns go forward. Then, the algorithm first measures the similarity between them by assigning a row vector to a specific user. Then the Pearson Correlation is used to as a computational measure. Finally, for the first k given movies j, the average rating value is calculated where each weight corresponds to the similarity values.

1.1.2 Latent Factor Methods: The latent factor algorithm looks to decompose the ratings matrix R into two tall and thin matrices Q and P, with matrix Q having dimensions *num_users k* and P having the dimensions *num$_i$tems k* where k is the number of latent factors. The decomposition of R into Q and P is such that

$$R = Q\,.P_T$$

.

Any rating r_{ij} in the ratings matrix can be computed by taking the dot product of row q_i of matrix Q and p_j of matrix P. The ma- trices Q and P are initialized randomly or by performing SVD on the ratings matrix. Then, the algorithm solves the problem of mini- mizing the error between the actual rating value r_{ij} and the value given by taking the dot product of rows q_i and p_j. The algorithm performs stochastic gradient descent to find the matrices Q and P with minimum error starting from the initial matrices.

Content Based Recommendation algorithm is very similar to the Nearset Neighbours Collaborative Filtering as in this algorithm, the likings of the user are used to make a used profile. Similarly, each user profile is given a vector which describes the characteristics of the user., for example their user rating. Once, the profile is generated, the similarity is calculated by using cosine similarity algorithm between the user and the item profile. Overall, the advantage of this algorithm is that the server doesn't require the private data from the users and the recommendations made to the users doesn't have to be solely based on the current ratings, but the recommender algorithm doesn't recommend the items outside the category of items the user has rated.

We used the MovieLens movie ratings dataset for our experiments[1]. The experiments are conducted on two versions of the dataset. The first version consists of 100004 ratings by 671 users across 9125 movies. The ratings allowed at intervals of 0.5 on a 5-point scale, starting from 0.5 and going to 5. The selected users had to rate tweny movies. None of their private information were taken and each one of them was provided with an id. The dataset has additional information about the movies in the form of genre and tags, however we use only the ratings given by the users to the movies and ignore the other information for the collaborative filtering techniques. For the content filtering portion, information was scrapped from the IMDB website for the corresponding movie. The links to the IMDB page for each movie is provided in a separate file.

The second and bigger version of the dataset consists of 20000263 (20 million) ratings by 138493 users across 27278 movies. Apart from this, the structure of the two datasets also is identical. The movie ids for a particular movie is the same in both datasets, but the user id for the same user is different for the two datasets.

We split each dataset into three partitions: training, validation and test by sampling randomly in the ratios 80%, 10%, 10% respec- tively. The validation partition is used to tune the hyper-parameters for the nearest neighbor and latent factor algorithms. The bigger version of the dataset presents significant scalability challenges.

We try out the following simple baseline methods to give us an idea of the performance to expect from the

1.1.3 Global Average. The global average technique serves as a simple baseline technique. The average rating for all users across all movies is computed. This global average serves as a prediction for all the missing entries in the ratings matrix.

1.1.4 User average. All users exhibit varying rating behaviors. Some users are lenient in their ratings, whereas some are very stringent giving lower ratings to almost all movies. This user bias needs to be incorporated into the rating predictions. We compute the average rating for each user. The average rating of the user is then used as the prediction for each missing rating entry for that particular user. This method can be expected to perform slightly better than the global average since it takes into account the rating behavior of the users into account.

1.1.5 Movie average. Some movies are rated highly by almost all users whereas some movies receive poor ratings from everyone. Another simple baseline which can be expected to perform slightly better than the global average is the movie average method. In this technique, each missing rating entry for a movie *j* is assigned the average rating for the movie *j*.

1.1.6 Adjusted Average. This simple method tries to incorporate some information about the user *i* and the movie *j* when making a prediction for the entry r_{ij} . We predict a missing entry for user i and movie j, by assigning it the global average value adjusted for the user bias and movie bias. The adjusted average rating is given by the formula below.

$$r_{ij} = \partial lobal\_av\partial + (u\_av\partial(i) - \partial_{ij}) + (m\_av\partial(j) - \partial_{ij})$$

The user bias is given by the difference between the average user rating and the global average rating. The movie bias is given by the difference between the average movie rating and the global average rating. Consider the following example which demonstrates the working of the adjusted average method. Let the global average rating be 3.7. The user A has an average rating of 4.1. Thus the bias of the user is (4.1 - 3.7) I.e the user rates 0.4 stars more than the global average. The movie Fast and Furious has an average rating of 3.1 stars. Thus the bias for the movie is -0.6. the adjusted average method will predict that user A will give the movie Fast and Furious a rating of 3.7 + 0.4 - 0.6 = 3.5 .

We implement the nearest neighbors and latent factor methods for Collaborative filtering. The smaller version of the dataset can be processed by using dense matrices and non-vectorized operations. However, if we try to use the simple implementations on the larger 20 million dataset, the code breaks. If we store the ratings matrix for the 20 million dataset in a dense format, it would take up around $140000\ 27000\ 8 = 28GB$ of memory, assuming 8 bytes per matrix entry. Thus we need to use sparse matrix representations to be able to handle the bigger dataset effectively. Also, the matrix operations have to be as vectorized as possible to make efficient use of threads. We observed that even after our best attempts to optimize the code as much as possible, the algorithms still needed a lot of time to be able to process such huge amounts of data. We thus decided to make use of Apache Spark to parallelize the operations and improve the runtime performance of the algorithms by running them on Spark clusters.

We implemented the algorithms such that they could be run on an Apache Spark cluster. We then run the algorithms on a cluster of 20 AWS EC2 instances. we were able to achieve significant improve- ments in the running time by using Spark. The nearest neighbors algorithm took around 30 hrs to run on a single machine, whereas the Spark implementation on the 20 machine cluster was able to run in around 50 minutes.

II.

1.1.7 Data Scrapping. For this algorithm, we needed to obtain movie metadata. While some basic movie metadata, such as release year, tags and genre were provided in the set, we decided to collect further information about the movie, with hopes of incorporating them into our system.

We decided to obtain the required information from the IMDB website. The pages content were accessed using the Beautiful Soup2 python library. Beautiful Soup is a Python library designed for quick turnaround projects like screen-scraping. It allows us to easily search and navigate the pages content. Once the structure of the website was identified, we wrote a script to automatically extract the relevant information about a web-page and create a dump. In python3, Beautiful soup also handle the encoding of the incoming file, further assisting in the automation.

In order to keep the dump size reasonable, we limited the num- ber of fields we extracted. We only extracted the top 10 actors, 2 directors, and 1 writer for each movie. In addition, we extracted the plot summary instead of the synopsis, as extracting the later would have lead to create of a dump of around 1GB.

III.

We also implemented an additional feature that provides a dis- tributed indexer on the movie synopsis as extracted from the IMDB website to allow searching for movies based on keyword searches. It is built on a MapReduce framework to build an inverted index. The framework for the indexer is the same as the one developed earlier in the semester for the assignments. It has been modified to allow searching on the movie information dataset.

First, the dataset is generated by scrapping data from the IMDB pages of the respective movies. The IMDB movie id is provided along with the movie lens dataset. After accessing the webpage, the metadata from the movie is collected and then pickled and stored in a dump. One can also use the same dump and extend the content model.

In the current model, the synopsis refers to the first of the pro- vided plot summaries. The summaries tend to be much shorter than the synopsis and can be extracted much more easily. Also, the size of the dump increased by a factor of almost 100 when using synopses instead of plot summaries. One more reason to select plot summaries is that detailed synopsis are not available for a large portion of the movies, which would result in a bias in the search.

The rest of the search is similar to the one provided in the assign- ments. The reformatter has been adjusted to match the format of the dump. The rest of the search can be run in the identical manner as the assignment. More detailed instructions are provided in a readMe file.

IV.

In addition to searching the movie plot, we also developed a separate search for names of people associated with the movie. These could be the actors, the directors, or the writers. Both searches are independent, in the fact that using the person search, you will not get results for queries appearing in the movie plots, and visa-versa. In order to incorporate the person search for ambiguous queries that could appear in either plot of in a name, such as Hill, we expanded the number of documents that are returned by the index servers, and evaluated all the movies until either 10 appropriate results are obtained or all the movies returned by the index servers have been analysed.

V.

Table 1 represents the performance of various methods on Movie- Lens 100k(small) data

Table 2 represents the performance of User-User Collaborative filtering with the hyperparameter as the number of nearest neigh- bors(k) on MovieLens 100k(small) data. The best result corresponds to k=300, as we increase the k to 500 the performance reduces, as we are taking into account almost all the users (500/670) for predicting the rating. When we increase the number of neighbors to 500, even the users with very small similarity values will be included which may introduce noise and thus the deterioration in performance is observed.

Table 3 represents the performance of Latent Factor Collabora- tive filtering on various hyperparameters: number of latent factors, learning rate for gradient descent and regularization parameter. The best performance corresponds to 500 latents factors, learning rate of 0.01 and 0.1 regularization.

As we increase the number of latent factors, the number of free parameters available for tuning increases. Thus the capacity of the latent factor model increases. This increase in capacity may lead to overfitting if we do not use regularization effectively. In the case of 100k dataset, the number of users and items is less and thus even a high value of k does not result in overfitting. However, in case of the 20 million dataset, since the number of users and items are large, the matrices Q and P are large and the number of tunable paramters are also large. Hence, we observe that a small value of k, around 100 performs well for the larger dataset. A larger value of k results in poor performance for the bigger dataset.

Table 6 represents the performance on Latent Factor Collabora- tive filtering on 20 M Movielens Dataset

The performance of Content Based and User-User Based Collab- orative filtering algorithms are 0.9554 and 0.885 respectively on the 100k dataset. The user based collaborative filtering algorithm gives the best performance.

One of the major challenges in working with the 20 million dataset is memory constraints. The data cannot be stored as a dense ma- trix due to its huge size. We have to make use of sparse matrix representations in order for the program to work without memory issues. Further, intermediate results such as the user-user similarity matrix cannot be computed and stored due to the huge memory footprint. We had to think of ways to compute the similarity values as and when needed. Further, the 20 million datasets also needed a lot of time to run. We were able to overcome the time requirements by writing parallelized implementations of the algorithms using Apache Spark.

A. ***Table 1: Performance of various methods on MovieLens 100k data***

Method/ MSE	**Validation**	**Test**
Global Average	1.1319	1.1368
User Average	0.9281	0.9581
Movie Average	1.3470	1.3672
Adjusted Average	1.2606	1.2911
User - User	***0.8029***	***0.8366***
Latent Factor	0.8486	0.8710

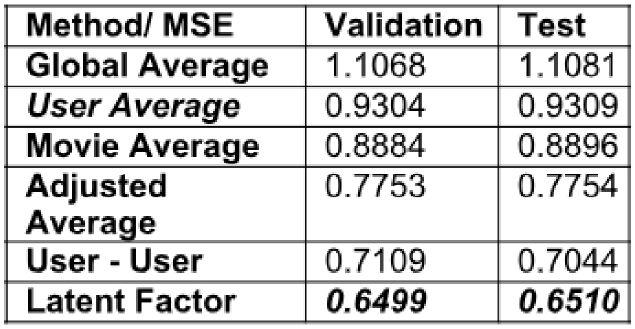
Table 2: User-User Collaborative Filtering for 100k

k nearest users	**Validation MSE**	**Test MSE**
5	0.9794	1.0144
10	0.9704	0.9980
20	0.9414	0.9681
50	0.8767	0.9074
100	0.8365	0.8660
200	0.8088	0.8426
300	***0.8029***	***0.8366***
500	1.0501	1.0701

Table 3: Latent Factor Collaborative Filtering on 100k dataset

k latent factors	**learning rate**	**regulari-zation**	**Validation MSE**	**Test MSE**
30	0.01	1	2.1544	2.1672
30	0.01	0.1	1.0724	1.1091
30	0.001	0.1	1.3548	1.4205
50	0.01	0.1	1.0456	1.0787
100	0.01	0.1	1.0025	1.0350
200	0.01	0.1	0.9335	0.9642
200	0.03	0.1	1.0240	1.0498
200	0.005	0.1	0.9258	0.9602
500	**0.01**	**0.1**	**0.8486**	**0.8710**
500	0.05	0.1	1.032	1.0522

Table 4: Performance of various methods on MovieLens 20M data



Method/ MSE	Validation	Test
Global Average	1.1068	1.1081
User Average	0.9304	0.9309
Movie Average	0.8884	0.8896
Adjusted Average	0.7753	0.7754
User - User	0.7109	0.7044
Latent Factor	***0.6499***	***0.6510***

B.

FUTURE WORK

There are a lot of ways in which this research paper can be improved. For example, the content based approach could have possibly had more ways of classification and it could have had more movie suggestions where you could sort your likings on the basis of actors, directors or producer choices. Moreover, the newly released movies could have he likelihood of being released for recommendations as early as possible. Additionally, the application could internally change the settings as per the user preference for a specific genre/ blockbusters or actors. Nonetheless, this idea could lead to over-fitting, and according to our results, only twenty movies were used in our experimentation process. In addition, we could try to develop hybrid methods that try to combine the advantages of both content based methods and collaborative filtering into one recommendation system.

REFERENCES

[1] A Survey of Collaborative Filtering Techniques; Su et al; https://www.hindawi.com/journals/aai/2009/421425/

[2] Google News Personalization: Scalable Online Collaborative Filtering; Das et al; https://www2007.org/papers/paper570.pdf

[3] Intro to Recommender Systems: Collaborative Filtering; http://blog.ethanrosenthal.com/2015/11/02/intro-to-collaborative -filtering/

[4] Collaborative Filtering Recommender Systems; Stanford Student project; http://cs229.stanford.edu/proj2014/Rahul %20Makhijani,%20Saleh%20Samaneh,%20Megh%20Mehta,%20 Collaborative%20Filtering%20Recommender%20 Systems.pdf

[5] MMDS course slides; Jeffrey Ullman; http://infolab.stanford.edu/ ullman/mmds/ch9.pdf

[6] Recommending items to more than a billion people; Kabiljo et al; https://code.facebook.com/posts/861999383875667/ recommending-items-to-more-than-a-billion-people/