Project 1

Collaborative Filtering Using Netflix Data

Siam Abd Al-Ilah (456968)

Andy Wu (464490)

Charles Yang (456715)

Karen Ye (454998)

CSE 427s Final Project
Washington University in St. Louis
Fall 2019

Introduction

The Netflix Prize

The Netflix Prize was a coding challenge presented by Netflix on October 2006. The mission was to make the company's movie recommendation prediction algorithm more accurate by at least 10%! The algorithm should be able to predict how much a user is going to like a certain recommended movie based off of their other movie preferences. The prize for this challenge was a whopping $1 million!

While the big payday incentive drew attention, it was the enormous data set that left mouths hanging. Netflix provided 100 million ratings for 17,770 movies from 480,189 users. This was before the era of "Big Data" and there wasn't anything quite like it during that time.

Why would any company do this? Most businesses would protect their data like hounds guarding their bones. The prediction algorithm used at the time was called Cinematch, and CEO Reed Hastings was always looking for bright minds to improve the magic formula, but to no avail. He needed fresh ideas and more innovation. Instead of trying to find brilliant minds, he waited for brilliant minds to come to him by opening the Netflix Prize challenge.

Why was this important? Business strategy. A better prediction algorithm leads to customers with higher satisfaction and delight in watching movies they really like. Higher satisfaction leads to using the platform for longer periods of time. Longer service usage leads to more money spent. More money spent by users means more money earned by the company. From a business prospective, a company only has one goal, and that is to make more revenue than expenses incurred, but that doesn't mean users are victims of this trap. "Everyone likes movies!" There is no denying that Hastings knew this when coming up with the Netflix Prize, so both parties are winning. The challenge lasted for 3 years, and the winning group finally collected their $1 million on September 21, 2009.

Our Goal

In our project, we will be tackling this challenge. Our goal is to come up with a prediction algorithm that can accurately predict what movies users will like based on their previous preferences. The data set we are using will not be the original 100 million ratings, but a subset of only 100,000 ratings. We will be using the collaborative filtering algorithm to tackle this challenge.

This application is important in the real world because many companies use these types of algorithms to better their prediction models. They use them to understand their customers which can help improve the products and services that they're offering them.

The data that we're working with applies to big data analysis and cloud computing, because most of the data sets that companies work with are massive. Information gathered from users can be enormous with lots of complexity and bias. Analyzing this data means finding the trends in human behavior, which is not easy because everyone is so different. This relates to cloud computing because it's delivering computing as a service rather than a physical product. Cloud computing refers to the platform for accessing this massive data set and means of getting this information, which is what we need to better predict recommendations. This is why big data and cloud computing are both applicable in the problem that we're trying to solve.

Our Approach

Exploring and Understanding the Data

Our first task was to explore and understand the data set provided to us. Using Spark, we were able to extrapolate the following number of items from the data set:

```
TestingRatings.txt
Movies: 1701
Users: 27555
Ratings: 100478
```

```
TrainingRatings.txt
Movies: 1821
Users: 28978
Ratings: 3255352
```

Given the training set, our goal is to use that data to train a model that can predict the ratings as accurately as possible for all the user-item pairs in the test set. We will be using collaborative filtering as our training algorithm.

With collaborative filtering, there are two approaches we can take. The given data has two features, the users and items (movies), that are unbiased, so we can use that to calculate similarity. Therefore, the two collaborative filtering approaches are finding *similar users* or finding *similar items* when giving a recommendation.

User similarity is found by observing *overlapping items*. This is because similar users will want to watch the same movies. On the other hand, item similarity is found by observing *overlapping users*. This is because similar movies will be watched by the same user. Our next step is to determine which one of these approaches we should use.

Selecting Collaborative Filtering Algorithm

We understand that user similarity is measured by overlapping items and item similarity is measure by overlapping users. The similarity that gives a higher overlap percentage should produce a better rating prediction, resulting in more accurate recommendations. Therefore, our next task is to calculate the level of overlap in *items for users* and *users for items* and then compare the results to determine which collaborative filtering approach to take.

User Similarity

Similar users are considered to be a *user-user model*. We chose a user from the test set and extracted all the movies this user has rated in the training set. Then, we found the total number of others users in the training set who have also rated the same movies. This is the number of overlap of an item for the selected user. We can do this for multiple users in the test set and average over the overlap. This can be done by aggregating the total number of ratings for all movies rated by this user and dividing that by the number of movies this user has rated. We chose 10 users from the testing set for our sample.

From our calculations, we got an average overlap number of 10251. This means on average, a movie has 10251 ratings. From earlier, we know that the total number of unique users in the testing set is 28978. When we divide the number of items for users overlap by the total number of users, we get 10251/28978, which is roughly 35.38%.

Item Similarity

Similar items are considered to be an *item-item model*. Similar to the user-user approach, we chose a movie from the test set and extracted all the users that have rated this movie in the training set. Then, we found the total number of other movies in the training set that have also been rated by these users. For this approach, we also chose to use 10 items to calculate the average overlap on. We did this by aggregating the total number of movies rated by all users for a given movie and dividing it by the number of users that have rated this movie.

From our calculations, we got an average overlap number of 213. This means on average, a user rates 213 movies. From earlier, we know the total number of unique movies in the testing set is 1821. Dividing 213 by 1821 yields 11.70%.

Comparing Overlap Similarity and Final Selection of Approach

As we can see, the user-user model gave a 35.38% overlap and the item-item model gave a 11.70% overlap. Since we said from earlier that the better the similarity the better the rating predictions, we will go ahead and conclude that using a user-user model based collaborative filtering approach will yield better recommendation predictions for this given data set.

Since we are choosing the user similarity approach, movies on average have around ten thousand ratings. This means that efficiency will go down more as compared to choosing the item similarity approach, but we believe it's a fair trade-off given a better recommendation result.

Along with choosing to use the user similarity approach, we have also decided to use the Jaccard similarity measure because it is said to work well in practice. We have also chosen 50 as the k-value to start off with, or number of similar users to predict on. We're using a weighted similarity prediction method and thresholding the ratings to only a value of 3 and above because we are using Jaccard similarity with rounding. This way, we are only looking at relevant users who rated they liked the movie because we want to recommend those same movies to other people if they are similar to that user.

Pseudo Cluster Implementation

Collaborative Filtering Approach

Because the data is so big, we have to work with an even smaller data set locally for the algorithm to finish running. We can use 1% of the test set to sample by using the *.sample()* method on our imported data which can also randomize our user selection from the set.

Our goal will be to use Jaccard similarity to find the most similar users who have already given ratings to movies that the sampled user hasn't yet. Let's call the sampled user we want to give recommendations to user X. We can do this by comparing the movies that the similar users and user X *have* both already rated. Because the users are so similar to user X, we can look at the similar users' ratings for movies that user X hasn't rated yet and predict the same ratings for user X. The goal is to recommend *novel* movies to

this user. If the predicted rating for a new movie tends to be very high for this user, then it's safe to say we should recommend the user this movie to watch.

Implementation

We start off by splitting the training and testing sets and taking only 1% of the training set. We decided to take top 50 most similar users to start predictions on so we set k = 50, but we can tweak this number later.

Next, we threshold the ratings to only keep those that are 3 and above in the training set. We chose a user from the test set and extracted all the movies this user has rated in the training set. Then, we found the total number of others users in the training set who have also rated the same movies. This is the size of the overlapping size. We can do this for all users in the test set and average over the overlap. The implementation is similar to when we calculated the user-user similarity from earlier. From there, we created user pairs for users that are similar, along with their movie counts for the number of movie overlaps. Then, we calculated the Jaccard similarity between each of these similar users using those movie counts and produced a top k list, or the top 50 most similar users for each test user. Since we are testing locally, we used a very small amount of training data, around 1%.

Evaluation

Finally, we got all the user-movie pairs from the test set that we need to predict on and we were able to find the 50 most similar users to the test user and weighted averaged all of the ratings of the similar users together to predict the test user's rating.

We didn't calculate error measure as those values will be insignificant with such a small data size. However, the programs ran without error. After having a list of the predicted results for each user, we can now evaluate our implementation on the cloud with Amazon EMR.

Additionally, we also took some time to add in our own user and had the program predict on that. Here were the following results that is part of the extracted data that we personally added to the training/testing sets:

```
Training Set:
User        Movie           Rating
AW          Rain Man        5.0
AW          Dragon Ball Z   4.0

Testing Set:
User        Movie           Rating
AW          Blade           1.0
```

These were the actual results that AW provided. After running the program with the newly added data, we want the algorithm to predict that AW will rate Blade a 1.0 if he watches it. Here's what it predicted.

```
Prediction:
User        Movie           Predicted Rating
AW          Blade           1.4237
```

The algorithm predicted a 1.4237 for AW, which is within the 0.8 error that we got, so that's good. Since the rating is below a 3, we would probably not recommend this movie to AW.

Cloud Execution

Implementation

By following the same steps as those described in Lab 11, we set up AWS accounts with running Amazon Elastic Compute 2(EC2) instances with Elastic Map Reduce(EMR). We created S3 Buckets for static file storage and observed Amazon Machine Imaging(AMI) for program logistics. After uploading our files and program, we ran the Spark cluster jobs.

We first tested on a small amount of data and then on a larger size. When running the full training data set, it was way too large and the cluster never finished execution. Factors that might have affected the execution time and results were the amount of training and testing data used, the k-value, and number of core nodes.

To better evaluate how different k-values and amount of data used would influence our results, we set up different accounts and clusters to test our program in parallel. We found out that the total execution time relied more heavily on the *amount* of data used instead of the k-value. For example, when running on 10% of the training data, the execution took around 90 minutes on a cluster with 1 master node and 6 core nodes. Running 20% took significantly longer.

We also tried adding more nodes in the cluster but this would only improve the execution speed for a limited percentage of the data. The default is 1 master and 2 core nodes, which ran very very slow so we increased the number of core nodes. With 1 master and 6 core nodes, some of the tests we ran are as follows:

```
Training Set: full
Testing Set: 0.1%
Runtime: ~5 minutes
```

```
Training Set: 10%
Testing Set: full
Runtime: 1hr 25 minutes
```

```
Training Set: 20%
Testing Set: full
Runtime: 5hr 30 minutes
```

Evaluation of Results

After running on EMR, we can predict 100451 movie-user pairs out of 100478 pairs in the testing set. By comparing with the provided ratings in the testing ratings, we calculated the Mean Absolute Error(MAE) and Root Mean Squared Error(RMSE) between the predicted ratings and the true ratings. MAE and RMSE tells us how far away the predicted value is from the true value. For example, if we get a MAE of 0.5, then that means if we choose any random point in our data, we would expect out prediction to be 0.5 away from the true value.

As mentioned before, k-value seems to affect execution time a lot less than amount of data, but nonetheless, we tested out certain k-values to see the difference. When k gets larger, MAE and RMSE both tend to decrease. We noticed that changing a k-value from 50 to 100 yielded a larger change in MAE and RMSE than that from 1000 to 2500. We got the best results with a k-value of something over 5000. See the following results:

Training Set	Testing Set	k-value	MAE	RMSE
full	0.1%	100	1.3347	1.5837
full	0.1%	930	0.9476	1.1712
10%	full	100000	0.8091	1.0114
20%	full	100000	0.8028	1.0015

Our error values are pretty fair. It looks like worst case, our predictions will be 1.0 rating off, but it's fair. The errors are squared before they are averaged in RMSE, meaning it will give relatively higher weight to larger errors. This means that RMSE error measure is usually more useful when large errors are undesirable because it will give notice to them more. RMSE will always be greater or equal to MAE. As shown above, our error results are very similar to each other, but we're working with a very small range of values (1-5) so the difference isn't too significant in this case between the two error measures.

Since we were only using the free-tier account, the compute power seems to be a lot lower. This caused restrictions and took up a lot of time. More powerful instances could probably reduce runtimes even more so we could potentially use more data to train on and get more accurate predictions.

Final Conclusion

Recommendation systems are effective and important, especially in business and sales. There used to be this famous jam experiment conducted in 2000 by psychologists. One table had 24 flavors and another table had 6 flavors. The 24 flavored table attracted 60% of the customers but only 3% of customers actually bought jam, while the 6 flavored table attracted 40% of the customers and 30% of them bought jam. This just proves that a platform can provide million of products, but without a good recommendation system, there is no purpose in variety. Recommendation systems are used everywhere and every day, such as videos on YouTube and songs on Spotify.

Collaborative filtering is a simple and effective prediction algorithm, but it's tedious and time consuming especially when given a large data set. With that being said, it provides a powerful way for recommending new products to users if only the data is *big enough*. In other words, there needs to be a lot of data for the predictions to be accurate and effective.

In our experiment, we chose to go with the user-user model, which inherently will take a longer time than the content-based approach with the item-item model, but we agreed that the more accurate prediction results would be worth the time tradeoff. Other things we could do in the future is actually giving content-based approach a try in recommending items. This will probably be more effective for less popular and more niche movies. We could also try using cosine similarity in the similarity measures instead of using Jaccard.

In the end, we were only able to predict on 20% of the data due to hardware limitations, but if we had more powerful and faster processing power, our results should be even more accurate. In a real world setting, it might be important to analyze the tradeoff between compute power and accuracy as well, although commodity hardware makes brute force computations a lot simpler and less expensive. Overall, recommendation systems are really amazing and it was fun getting to implement our own. Who knows, if we could do this 10 years ago, maybe we could have won that $1 million!