**WASHINGTON UNIVERSITY IN ST. LOUIS**

**St. Louis, Missouri**



**A**

**PROJECT REPORT**

**ON**

**NETFLIX RECOMMENDATION SYSTEM**

**(CSE 427S CLOUD COMPUTING AND BIG DATA APPLICATIONS, FALL 2016)**

**Submitted By:**

**Angelica Tao Zhu**

**Ziyi Liu**

**Jiawei Chen**

**Deeksha Chaudhary**

1. **ABSTRACT**

We have built a system to perform Collaborative Filtering using MapReduce and recommend items, in our case movies, based on preferences of similar users obtained by evaluating the Pearson correlation between each pair of movies and predicting ratings for different movie-user pairs. We used a subset of the Netflix Data to predict movie ratings for users in the testing set, and based on that result, we are able to make personalized movie recommendations to users in the testing set.

To evaluate the accuracy of our prediction system, once we had a working implementation to compute the 200 most similar movies for every movie in the training set, we made rating predictions for (movie, user) pairs in the testing set based on a normalized weighted average of the 200 most similar users, and compared the results with the true ratings provided in the testing set and computed the accuracy of our predictions.

**2. MOTIVATION**

Recommendations are one of the most important ways to achieve customer satisfaction and increase sales. This is not limited to Netflix, Amazon and Walmart. Google does that in Google app, Facebook and LinkedIn do that to suggest potential connections, news companies do that to deliver information that a particular user is interested in. This is why we chose this project. It has immense practical applications, and through the project we are able to gain insight into how this powerful approach is implemented in real life applications.

**3. METHODS**

**3.1 Collaborative Filtering Algorithm**

**Algorithm 1: Finding Similar Movies to All Movies**

When choosing an algorithm for implementing collaborative filtering, we picked algorithm 1 over algorithm 3, since it operates on the entire dataset instead of on individual users or items and would thus take significantly less time than running algorithm 3. We were aware that algorithm 1 runs faster at the cost of being a lot more memory-intensive, but since we are running the algorithm on Amazon EMR and can utilize multiple compute nodes, there is ample space to run the jobs for the dataset.

**3.2 Similarity Measure**

**Pearson Correlation**

When deciding on the similarity measure, we chose Pearson correlation over Jaccard and cosine similarities, taking into account that different users often have a different standard when rating movies, and some overly enthusiastic or overly critical users could skew the statistics and introduce bias in the data. Therefore, subtracting each movie’s average when computing similarities normalizes the ratings and would produce more accurate similarities that removes the bias and produces similarity measures based on actual correlations of features of the movie.

**3.3 Data Model**

**Item-Item model**

While we started out with the user-user model, our final implementation was constructed using the item-item model. As we found out in milestones 2, there are 28978 unique users and 1821 unique items in the training set. Since our implementation computes similarities between all pairs of users or items, there would be a total of pairs for the user-user model. In comparison, the item-item model would produce a total of pairs of items. Since each pair of users or items generates one output of their similarity, the item-item model produces a lot less output than the user-user model and would therefore run much faster and take a lot less space in practice.

**4. IMPLEMENTATION DOCUMENTATION**

**4.1 Implementation Changes**

* **Final approach: Algorithm 1: Finding Similar Items to All Items**
* **Initial Approach: Algorithm 3: Finding Similar Users to User A.**

This section documents the changes in our approaches since Milestone 1 and 2 and the justifications for those choices.

**Justification for the switch:** Our initial approach was to use Algorithm 3: Finding Similar Users to User A, as discussed in class. Since we decided on the user-user model, algorithm 3 would consist of 6 jobs per user: 2 pre-processing jobs, 2 for computing the similarity between user A and all other users, and 2 jobs to find the K most similar users and predict movie ratings for movies user A has watched in the testing set based on ratings in the training set of those similar users.

However, in the process of writing the MapReduce programs, we realized that this particular implementation would not be very efficient in computing predictions for the purpose of this project, since it requires that we run a separate sequence of jobs for each user in the testing set, and each run would have 6 jobs requiring different command line inputs for user-id, adding files to distributed cache, and some jobs operating on outputs of multiple previous jobs. Therefore, the job is not very streamlined and since there are 27555 total different users in the testing set, it would not be very efficient in calculating predictions for all users in the set. In fact, if we were to continue using algorithm 3, a better choice would be to use the item-item model and to find the K most similar items to each particular item, requiring only 1701 different runs and computing more predictions each run.

After realizing that the above approach is not a very efficient prediction model, we decided to instead implement algorithm 1, a more memory-intensive algorithm that computes predictions at a faster rate, since it operates on the entire dataset and computes Pearson correlation for all pairs of users at the same time instead of on a per-user basis. Since Algorithm 1 is a memory-intensive algorithm, when we tried executing the job on our pseudo-cluster and later on EMR using User-User model, the program terminated before completion because it ran out of memory. Therefore, for reasons we previously stated in *Section 3.3: Data Model*, we decided to make the switch from the user-user model to the item-item model, which ran successfully and gave useful predictions that we will analyze in *Section 5: Results*.

**4.2 Workflow**

The recommendation system can be divided into two parts. In the first part, we used algorithm 1 on the item-item model to compute the Pearson correlation between all pairs of items. In the second part, we found the 200 most similar items for all items and predict ratings for (movie, user) pairs in the testing set based on a normalized weighted average of the similar items.

**4.2.1 Part One:**

**Step 1: Transforming the input data from (movie, user, rating) into (user, movie, rating).**

This job is necessary because our algorithm below was originally implemented for the user-user model. Therefore, performing this job on TrainingRatings.txt and TestingRatings.txt adapts the algorithm for the item-item model.

**Step 2: Preprocessing**

We preprocess the transformed TrainingRatings.txt so that each line contains two additional values, numRatings, and sumRatings for the given movie.

**Step 3: Job 1 and 2**

In these two jobs, we use the preprocessed data from the previous steps to compute the Pearson correlation between each pair of movies.

**4.2.2 Part Two:**

**Step 4: Job 3 - Top K similar items**

In this step, we used the similarity measure output from the previous step to obtain the top K most similar movies for each movie. K can be modified through the command line, so we tried different values of K to find an ideal balance between the quantity and accuracy of the predictions. For our final algorithm, we used .

**Step 5: Job 4 - Testing Ratings Predictions**

In this step, we used the K most similar movies obtained from the previous step to compute predictions for all (movie, user) pairs in TestingRatings.txt based on the similar movies’ ratings in TrainingRatings.txt. We outputted four differently calculated predictions in our first few runs of the job. The four prediction methods we used are: average, normalized average, weighted average, and normalized weighted average. We then calculated the errors to decide which formula results in the most accurate predictions. In our final implementation, we used normalized weighted average as our rating prediction method.

**5. RESULTS**

The final configurations we used are: Calculating similarities using algorithm 1 with the Item-Item model computing Pearson correlation; Predicting ratings using the normalized weighted average of the most similar users.

**5.1 Number of predictions**

Taking , the number of predictions we made is 47490.

**5.2 Final Error measures**

* **Mean Absolute Error: 0.8709**
* **Root Mean Squared Error: 1.0922**

The table below shows the number of predictions we computed for different values of K.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Value of K** | **RMSE for average** | **RMSE for normalized average** | **RMSE for weighted average** | **for weighted normalized average** | **Number of Predictions** |
| 20 | 1.1965845 | 1.1050222 | 1.2023280 | 1.1053764 | 6387 |
| 30 | 1.1946710 | 1.1082386 | 1.2015711 | 1.1081908 | 9435 |
| 40 | 1.2168934 | 1.1290085 | 1.2233923 | 1.1294329 | 11476 |
| 50 | 1.2299647 | 1.1413677 | 1.2374206 | 1.1417752 | 13970 |
| 75 | 1.2160885 | 1.1349084 | 1.2231239 | 1.1346461 | 20162 |
| 100 | 1.2091354 | 1.1282242 | 1.2178414 | 1.1275800 | 26835 |
| 125 | 1.1894015 | 1.1128756 | 1.1976570 | 1.1117801 | 33238 |
| 150 | 1.1823946 | 1.1037017 | 1.1904837 | 1.1027523 | 38188 |
| 175 | 1.1749232 | 1.0998294 | 1.1853436 | 1.0992391 | 43139 |
| 200 | 1.1685047 | 1.0933768 | 1.1780616 | 1.0922302 | 47490 |

**Table 2: Errors and predictions for different values of K**

|  |  |  |  |
| --- | --- | --- | --- |
| Average | Normalized Average | Weighted Average | Normalized Weighted Average |
| 0.9052734 | 0.8720527 | 0.9098094 | 0.8708808 |

**Table 3: Mean Absolute Error (MAE) for**

As the table above shows, gives the lowest RMSE and produces the most predictions. In the four prediction methods, normalized weighted average gives the lowest RMSE as well as the lowest mean absolute error (MAE), and therefore is our final choice of prediction method.

**5.3 Results for our own preferences**

We rated 93 movies in total amongst the ones we have watched and then looked at the predictions for the movies we didn’t rate but have watched. The prediction results were pretty accurate. Following is the description for a few movies we didn’t rate but have watched:

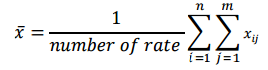
* WWE: John Cena: Word Life was rated 2 by the system and I didn’t like it much.
* Doctor Who: The Power of Kroll was rated 5 by our system and I indeed liked it very much.
* Final Destination 2 was rated 2 by the system and I didn’t like it much.

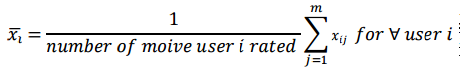
So the predictions made by our system matches with our choices of the movies.

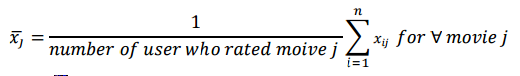
**6. DISCUSSION AND FUTURE WORK**

One really efficient improvement to our system is to come up with a better normalization algorithm, since the current normalization algorithm does not fully consider a user or item’s preferences. For example, a user who has only rated very few movies but has liked all of them may give them high ratings, but after this normalization, his inclinations are no longer accurately reflected.

In the future, one approach we can try is to have a preprocessing program for the whole data along with 3 statistics to get the new dataset. If **𝑥𝑖𝑗** represents the rate user **i** gave to the movie **j** and we have **n** users and **m** movies, then the 3 statistics would be:







Then we want to make the dataset to just have the specific interaction between user i and movie j, say 𝑒𝑖𝑗.



So now we would compute similarity between the e values. The computation wouldl be the same, we just have to compute the new 𝑒𝑖𝑗. And when we get 𝑒𝑖𝑗, we can get 𝑥𝑖𝑗 back using the formula



**7. APPENDIX: CLOUD EXECUTION**

We executed our MapReduce program on the full data on two platforms, Amazon EMR and Microsoft Azure.

**7.1 Amazon EMR**

We used the cluster that Amazon EMR provides and used S3 storage to save our results. We input our jar file, training, and testing dataset.

The method to execute our job on EMR is just adding steps in the cluster. We selected the jar file we wanted to execute, and wrote down the arguments like the way we would in the pseudo-cluster.

We first implemented the user-user model in the Amazon EMR, and then we found out that since, the number of users is much larger than the number of items, the result becomes so huge. We needed to use 16 m3.xlarge cores to calculate the step3: job1 for almost 1 hour, and our result was over 200GB big! And we couldn’t get the result for step4 as the Group and Sort phase for this step resulted in system running out of memory.

So, we tried to use the item-item model, and with 4 m3.xlarge core, the run time was much shorter.

|  |  |
| --- | --- |
| **Steps** | **Running Time** |
| Step 1: Transfer the user-to-user job to the item-to-item job | 40 seconds for training, 40 seconds for testing |
| Step 2: Preprocess for the movie, pre\_movie | 56 seconds |
| Step 3: job1 | 1 minute |
| Step 4: job2 | 3 minutes |
| Step 5: TopK | 1 minute |
| Step 6: moviestats | 40 seconds |
| Step 7: computation | 5 minutes |

**Table 4: Running time for different steps on Amazon EMR**

**7.2 Microsoft Azure**

In fact, as we performed too many wrong steps with the user-user method in the Amazon EMR, we spent quite a lot of money on this, so we decided to change the platform to Microsoft Azure, which also has a Hadoop platform called HDInsight Cluster.

This platform is a little less intuitive than Amazon EMR. Firstly, we needed to create a Resource groups with an HDInsight Cluster and a Storage account. This time we used D3 (2 nodes, 8 cores) as both worker nodes and header nodes. After that, we got a hadoop cluster. We can look through the dashboard to see all the information for the cluster and can use the Storage account to upload and download the data we want.

Then we can use ssh to connect to the cluster, and it’s much like what we do in the pseudo-cluster. We use command line to connect the local file system with HDFS. And then we can do all the work again, and use this model to find out our own preferences.

|  |  |
| --- | --- |
| **Steps** | **Running Time** |
| Step 1: Transfer the user-to-user job to the item-to-item job | 36 seconds for training, 22 seconds for testing |
| Step 2: Preprocess for the movie, pre\_movie | 44 seconds |
| Step 3: job1 | 5 minutes |
| Step 4: job2 | 11.5 minutes |
| Step 5: TopK | 51 seconds |
| Step 6: moviestats | 39 seconds |
| Step 7: computation | 52 minutes, 41 seconds |

**Table 5: Running time for different steps on Microsoft Azure**