**Collaborative Filtering of Netflix Data**

Amit Jha ([ajha@wustl.edu)](mailto:ajha@wustl.edu)), Vinaya Sheni([vinaya.sheni@wustl.edu)](mailto:vinaya.sheni@wustl.edu))

David Kwon ([dkwon@wustl.edu)](mailto:dkwon@wustl.edu)) , Andy Kim ([dohoon.kim@wustl.edu](mailto:dohoon.kim@wustl.edu))

1. **Motivation**

Recommendation systems are one of the key application areas where big data tools and technologies play a big role these days. As part of final project, we wanted to experiment with Netflix sample data to do prediction of movie-user ratings and see how well we can implement this algorithm. The Collaborative filtering approach lets us predict ratings of products based on item-item or user-user similarity of rating and knowledge gained from this approach can be applied to other domain also where such kind of user/product related relationship exist (such as product based merchandise website like amazon).

1. **Approach**

We had several options regarding solution approach to this problem. Specifically, we thought of and implemented approach as mentioned below –

Item-item vs User-User comparison – In the training dataset given to us there are 28,978 distinct users and 1821 distinct movies. Given this data, we realized it will be much more efficient to do an item to item comparison vs use to user comparison of ratings data. This will require much lesser number of looping of data comparison compared to user-user comparison. We also realized that chances of getting overlapping ratings in item-item comparison is much higher than user-user comparison.

Normalization of ratings data – We performed normalization of ratings data to remove any bias in the rating. For this we implemented two approaches –

1. Normalize the ratings based on average ratings grouped by movie
2. Normalize the ratings based on average ratings grouped by user

The observed result for both the approach is provided further in final section of this report.

Similarity Calculation -- For calculating similarity, we considered all three options (jaccard, cosine, pearson) and then settled on using Pearson Correlation.

Weighted vs un-weighted ratings – For calculating final prediction, we used both weighted and un-weighted factor to calculate the final rating for the user-movie pair.

1. **Implementation detail**

**Implementation of Pre-Processing step** -- We used Pig to implement pre-processing step for this project. This required implementing following steps to prepare the data for Map Reduce Phase –

* + 1. Load the data from TrainingRatings
    2. Group it by movie and user separately
    3. For each grouping calculate the average rating of movies by the group and subsequently normalize the rating by subtracting the rating by average rating.
    4. Output the final data in format – movieid, userid, rating, normalized\_rating, average.

This phase eventually produced two files which were used separately as input for the

Map Reduce phase of the algorithm.

**Map Reduce Phase of algorithm**

For this solution; we implemented a variance of Algorithm 1 discussed in the class. Apart from the preprocessing we have implemented two map reduce jobs. The detail of Map Reduce Jobs is as given below –

**Map Reduce JOB 1** –

In the Map phase of Job 1 the input is movieid, userid and stats (rating, average rating, normalized rating).

The output of the map phase is user-id as key and (movieid, stats) as value.

The reducer gets userid as key and list of (movieid, stats) as value. The output generated is movieid, movieid combination along with their stats.

**Map Reduce JOB 2 –**

The Mapper of job 2 is an IdentityMapper and generates movie, movie pair along with their stats as output.

The Reducer phase of job 2 gets (movie, movie) pair as input key and ratings pair as value. It calculates the Pearson correlation between those ratings to arrive at the final similarity factor between the movies. It outputs (movie, movie) pair as key and their similarity factor as value.

**The predictor application**

For predicting the rating of given user, movie combination we implemented a Java program to process the three files. It takes as its input –

1. TestingRatings.txt
2. TrainingRatings.txt
3. Output of Map Reduce which has movie, movie pair and their similarity combination.

The output is generated in the format of userid, movieid, expected rating, calculated rating.

**Options** – There are several ways the final ratings of user movie combination are being calculated –

1. Based on average rating of all found movies combination and their corresponding ratings.
2. Based on Top-K matches and their corresponding ratings
3. Based on weighted average of all found movies combination and their corresponding ratings.
4. Based on Top-K matches and their corresponding weighted average.
5. **Execution of Programs in various modes**

Execution in eclipse in virtual machine –

To test the correctness of program we stripped down input file to just 1000 lines and ran the program locally in Eclipse in the class provided virtual machine. Once we were certain the program logic is correct, we attempted execution on full dataset of 3.2million records using Eclipse. Our first Map Reduce Job failed due to machine running out of disk space. As the first map reduce job failed, we dint try running the second map reduce job using eclipse in this mode.

Execution in pseudo distributed cluster on virtual machine

We tried similar execution with full data set in pseudo distributed cluster which is provided with the virtual machine. We ran into similar out of disk space issue related job failure in first Map Reduce task.

Upon investigating failure closely, we realized that the file being generated by Map Reduce of Job 1 was generating file of size more than 10GB. We observed this by watching the size of output directory being generated by the program continuously while the program was running. The temporary file created there kept growing till it eventually ran out of space when the output folder size stood somewhere around 8GB. We concluded (as was mentioned in project instructions) that it won’t be feasible to run the program on virtual machine with full dataset.

So instead of using full dataset, we stripped down to smaller data set and executed program once. We did not use result to do any analysis.

Execution in Eclipse in local laptop

The next logical step for us was to take the program and whole data set to Amazon EMR and try the execution there. However; we wanted to verify the correctness of out Algorithm locally end to end before moving to EMR. As disk space was the only known issue in the virtual machine that we came across we thought of way to eliminate it by using our local laptop were the disk size is significantly larger compared to virtual machine. To experiment in this way, we set up Eclipse and Hadoop locally in our laptops. Please note that we dint do any installation of Hadoop apart from just unzipping the package and using the jar files to set up the eclipse project. We took the list of required jar files from existing eclipse project in virtual machine to build the new project in local eclipse.

When we executed the program in this mode the first map reduce job finished **in 4 minutes and 2 seconds** and produced output of **10.95GB**.

The second Map Reduce job took longer. In the local eclipse mode, it took **19 minutes 23 seconds** to process the **10.95GB** of data produced by first Map Reduce Job. Its output turned out to be **49.3 MB** in size with **1.64 million** records in it which are basically movie-movie pair with their similarity score between 1 and -1 in value.

Execution in the cluster

As part of this course, we have set up our own Hadoop cluster with 2 node server configurations. As an alternative to EMR, we wanted to test our jobs on this small cluster to verify if we can execute the whole algorithm end to end in a time sufficient manner. The key configuration of the servers is as given below –

Node count – 2.

Server Model – IBM x3650

Processor – Intel Xeon 3.0GHz quad core

Number of processor per server- 2

RAM – 16GB each server

HDD – 2 TB each server

Ethernet Switch – Standard Ethernet router.

Hadoop 2.8.1 was installed with yarn in cluster mode on these servers. There are 2 data nodes in all.

The jobs were executed with map-reduce mode set as Yarn on the cluster. We executed the jobs in default mode which created only one reducer for both jobs. We also executed in mode where we set the **mapred.reduce.tasks=10**. The execution time in both modes are as given below.

Default Mode

|  |  |  |  |
| --- | --- | --- | --- |
| Job | Number of Maps | Number of Reducers | Time Taken |
| Job1 | 2 | 1 | 16 min 17 secs |
| Job2 | 82 | 1 | 29 min 26 secs |

Mode with **mapred.reduce.tasks=10**

|  |  |  |  |
| --- | --- | --- | --- |
| Job | Number of Maps | Number of Reducers | Time Taken |
| Job1 | 2 | 10 | 8 min 40 secs |
| Job2 | 82 | 10 | 12 min 25 secs |

As can be seen above, increasing the number of reducers results in more efficiency of the systems. In future, we would like to experiment more with this parameter to arrive at optimum value to achieve best possible execution outcome.

1. **Analysis of quality of prediction compared to actual user given ratings**

To estimate the quality of results of prediction we used Mean Absolute Error as well as Root Mean Square Error.

Normalization of ratings grouped by movie vs by user

The quality of predictions turned out to be much better when we normalized the ratings grouped by movies compared to grouping by user. The MAE and RMSE in these cases were –

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Normalized by Movie rating | | | | |
|  | Average | Average Top K | Weighted All | Weighted Top K |
| MAE | 0.79 | 1.1 | 0.75 | 0.97 |
| RMSE | 0.99 | 1.41 | 0.97 | 0.98 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Normalized by user rating | | | | |
|  | Average | Average Top K | Weighted | Weighted Top K |
| MAE | 0.79 | 1.18 | 0.82 | 1.19 |
| RMSE | 0.98 | 1.42 | 0.99 | 1.4 |

Overall; we conclude that to predict better results we should be using grouping by movie, weighted average and all matches.

* **Spark Implementation**

Using the help of Amazon EMR, we implemented the algorithm now converted into a Spark script. For the input data, we tested with normalized ratings by movies, which was done by running the same pig script mentioned earlier, and the MAE and RMSE were calculated using ‘weighted’ only with varying Top K and nearest neighbor parameters. We introduced and experimented with nearest neighbors as we wanted to see how the results differ if I take into consider only a certain number of top most similar neighbors based on similarity. Along with the MAE and RMSE value, we included the total runtime of the algorithm to see how the parameter affects how long the program takes. Our pyspark script consists of the full algorithm (jobs 1 +2 and predicting) and takes in the preprocessed data files as an input, so the run time adds up to a considerable amount. The results are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Spark Error Data | | | |
|  | 5 Neighbors, K=5 | 50 Neighbors, K=50 | All Neighbors, K=5 |
| MAE | 1.356277 | 0.955427 | 1.365981 |
| RMSE | 1.732745 | 1.250229 | 4.106787 |
| Total Running Time | 85 minutes | 72 minutes | 126 minutes |

From our data, we were able to conclude that the run time and error results depend highly on the threshold parameter for neighbors and top k recommendations. We were surprised by the abnormally high RMSE for the ‘All Neighbors, K=5’ – one possible reason why this might be is because errors in RMSE are weighted higher than MAE and for those particular parameters the predictions may have been inaccurate. Note how for ’50 Neighbors, K=50’ has a much shorter run time but still has a better error rating than the formerly discussed and ‘5 Neighbors, K=5’ – from this, we were able to conclude that run time is independent of algorithm accuracy. Overall, the spark experiment gave us a much better understanding of how we could use RDD transformations and actions instead of the traditional MR method through Java.

1. **Future Work / Lesson Learnt**

While it’s a good start, we are not satisfied with the Mean Absolute Error of xx for our predictions. This means we are off by almost rating of 1 on the scale of 1 to 5 when predicting the ratings. We would like to improve on it by considering other factors. Some of the factors that can be further considered are –

Gathering more details about the movie from external sources (e.g. IMDB) to fine tune the ratings that it gets normally.

Factor in a user’s behavior more closely (rates something very high or low) than we have so far done in this algorithm implementation.

We would also like to work on fine tuning cluster parameter further to achieve yet better results in terms of execution timings on the cluster.

Overall, through this final project, we were able to experience first hand how we can benefit from using clusters when dealing with big data – our local implementations always returned errors like buffer overflow when trying to run the algorithm on full datasets, and these were avoided when run in the cluster. For testing our algorithm, we used the procedure explained in class so we have a careful, thought-out approach in how to design and test our algorithm (testing locally on small set of data, testing it in local cluster, etc…). This approach was great and effective in the sense that we were able to keep track of our process and quality check iterations as we go.