MapReduce Approach to Collaborative Filtering for the Netflix Challenge

Wenmei Bo, Yuanqin Jin, Jonathan Li, Yachen Qin

**Background and Motivation:**

Since the advent of large scale online shopping, retailers have necessitated methods to generate increased revenue by offering suggestions to shoppers for items that a particular user might find appealing. Two large online corporations, Netflix and Amazon, that dominate the online video streaming and online retail respectively, implement their own recommendation systems for users based on previously watched and rated movies/television series and previously bought items. Old school brick and mortar stores also attempt to employ recommendation systems for their customers; Barnes and Nobles, Powell’s etc. offer a more simplified approach, recommendations for the most popular books or movies or based off of “best” lists accumulated from critics or opinion sections of newspapers.

The approach used in more traditional brick-and-mortar stores called the Top-N List, takes the most popular N items and recommends them to everyone without regard to preference. However, online companies such as Netflix or Amazon have more powerful tools at their hands, namely the buying preference data gathered from millions of users over millions of products. The “Long-Tail”, items with usage frequency on the lower majority of the spectrum, generate such massive amounts of additional business for these companies that they dedicate billions of dollars to improve their recommendation systems to accommodate these less popular items. The best example being Netflix’s competition with a $1 million-dollar prize to improve its recommendation system.



Looking past the Top-N approach of traditional brick and mortar stores, recommendations for the long-tail can take place through various strategies; one primarily used is collaborative filtering. Two common models to collaborative filtering are item-item similarity and user-user similarity. Item-item similarity attempts to find out the similarity between items within the system and recommends an item to a user that has been determined to be similar to what the user already likes. User-user models attempts to find similar users to a person and recommends items based off of the activity of similar users.

The long-tail’s importance to retailers such as Amazon or services such as Netflix cannot be overstated. As one Amazon employee put it, “We sold more books today that didn't sell at all yesterday than we sold today of all the books that did sell yesterday." With Amazon revenue at $136 billion in 2016 and Netflix at $8.83 billion in 2016, the long-tail does not just represent a concept of overall popularity for those companies, it represents billions of dollars in revenue per year.

Therefore, high-quality recommendation systems prove essential in a modern day online business. Large mega-corporations such as Netflix have offered hefty rewards to improve their internal recommendation systems, and the billions of dollars in revenue at stake ensure that it will remain of vital importance for companies moving forward in the digital age.

**Data:**

For the big data application and cloud computing, the real world data we found on the internet for the Netflix Prize containing 17,770 movies gets used. The data is downloaded from the Netflix Prize website. The metadata contained 17,770 text files, with one movie in a file and a text file named “movie\_titles” with movie names matched to movie-ids.

|  |  |  |  |
| --- | --- | --- | --- |
| File | Record Position | Element Name | Describe and Remarks |
| movie\_titles | 1st column | movie id | The movie id match the mvxxxxx.txt file |
| movie\_titles | 2nd column | movie year |  |
| movie\_titles | 3rd column | movie name | The name of the movie. |

The data in this file are a combination of metadata from the Netflix website of user’s behavior and the movie’s rank.

For the small data/pseudo cluster run, the dataset used is the Netflix dataset that was provided by the class which includes a TestingRatings.txt file and a TrainingRatings.txt file. These two datasets will be used to test the final collaborative filtering model on. These two datasets were officially provided by Netflix for the Netflix prize challenge. The training ratings dataset contains 3.25 million ratings and the test set has 100,000 ratings. The datasets are newline delimited and are of the format MovieID,UserID,Rating.

Example:

4,476,4

6,2345,4

34,543,1

…

…

**Approach:**

The approach used was the item-item similarity model to implementing the collaborative filtering. The user-user based model has some limitations: one is the difficulty in measuring the similarities between users, and the other is an issue of scalability. As the number of customers and products increases, the computation time of the algorithm grows exponentially. The item-based model was proposed to overcome the scalability problem as it calculates item similarities in an offline basis. It assumes that a user will be more likely to purchase items that are similar or related to the items that he or she has already purchased.

Therefore, the approach taken was the item-item based similarity model. The similarity measure being used will be the cosine similarity to determine the similarity between two items.

and are the vectors of overlapping ratings between movie *x* and movie *y*.

The tool used was Spark for big data analysis.

**Small Data on Pseudo Cluster**

**Results and Discussion:**

The small data used a very small subset of the Netflix data provided by the class. The reason this was done was because the run-time of the approach even on the dataset provided by the course, even though it was not as large as a “big data set,” still took un

**Big Data Application**

**Implementation Description:**

In this instance, the items being analyzed will be the movies. First, the cosine similarities are calculated between all pairs of movies. This will initially be calculated into a pairs format with [(movie1, movie2), similarity]. This will be done for all pairs of movies so there will be a total of n-squared number of movie-movie, similarity measures. After this, the pairs format will then get broken up into stripes format in which there will be a RDD of the form [movie1, (movie2, similarity1\_2), (movie3, similarity1\_3)….] until there are stripes for every single movie 1 through n.

The next step will be to take a portion of the data as a test set, a much smaller subset. The data comes in the form of (movie, user, rating) and gets transformed into (user, (movie, rating)). For each user in the test set, the set of movies each user has rated in the training set gets found. The RDD that gets created is of the form [(test\_movie1, true\_rating1), {(training\_movie1, training\_rating1), (training\_movie2, training\_rating2),….}]. The first pair (test\_movie1, rating) may appear multiple times because these RDD’s are created for each movie that each user has rated in the test set, but the user\_id is not present in the RDD; since each user has a unique set of training movie ratings, each RDD will still be unique.

Then the test movie is compared to all the training movies for similarity, and this is done by looking into the RDD created from the stripe RDD’s created from the training set containing all the similarity measures as stated above. The top-K most similar movies from the stripes similarity RDD are chosen and the predicted rating is calculated based on  
 s = similarity, r = rating from training set, k = # for top k  
All the movie-movie similarities created in the similarity stripes RDD from the training set should contain all the movies from the test set as well because the training set is much larger than the test set. If not, the RDD leaves a blank [(test\_movie, r), {}].

Then the results are stored into a new RDD of the format (test\_movie, true\_rating, predicted\_rating). This allows for simple calculation of error.

**Results:**

**Discussion:**

\*\*Include Runtimes

**Conclusion:**

Lessons Learned: Although test/training are both smaller than the actual big dataset, they are still too large to test the basic functions/get output, therefore we created mini datasets.

Overall lesson: Code must be error resilient because testing on the smaller dataset created many key errors that would not occur on the full dataset because RDD not filled out.