The Netflix Prize: Exploration of High Density in Recommender Systems in a Large Dataset

Timothy Ozimek
North Carolina State University
teozimek@ncsu.edu

1 Background

The movie service, Netflix, offers customers the choice of watching several thousand movies on demand. While the service now offers a highly popular online, on-demand, streaming delivery method, in the past, operations were run through a mail DVD system. A user of the service could request a certain number of movies at any given time. After watching and returning the DVDs, a customer could receive new movies as quickly as the mail service could deliver them. Along with the movies, Netflix provided a list of recommended movies based on a user's viewing preferences. A viewer can voluntarily rate movies they've seen with the understanding that their rating would guide Netflix in suggesting new movies. The more ratings they gave, the better the recommendations became.

To provide this service, Netflix wrote the software tool, *Cinematch* [1]. This tool used a combination of machine learning and data mining algorithms to perform its duties. It uses the aptly named recommender systems, which have received serious study over the years. The intuition behind these systems is based on how word-of-mouth conversations spread news of a product. These conversations occur between friends and acquaintances who tend to share common opinions over a variety of topics. Likes and dislikes accordingly tend to correlate.

The field of data mining and machine learning has observed this relationship and formalized the task in a recommender system. Such systems tend to employ the technique of collaborative filtering. In short, this method scours data, looking for similarities in rating patterns, much like word-of-mouth works among similar friends, but at a far larger scale. The result of this similarity search will filter a group of similar users. This filtered group provides numerical information to compute a recommendation[3].

A variety of algorithms have been adopted. Two stand large: item-based collaborative filtering (IBCF) and user-based collaborative filtering (UBCF). The names of each imply their method of search. IBCF finds similarity of items, in this case movies, and makes recommendations based on the ratings of a target user. Likewise UBCF establishes similarity through the intuitive word-of-mouth approach by comparing similar user ratings[2].

To stimulate research in improving their recommender system, Netflix offered the Netflix prize in 2006. The objective seemed tantalizingly simple: be the first team to improve recommendations by 10% over *Cinematch* and receive $1 million. Without delving too far into details, contestants were given a dataset of over 100 million user ratings of 17700 movies. Against this dataset algorithms could be devised, tested, and submitted to Netflix. Netflix would run the submission against a hidden testset. A simple root-mean-square-error would be computed per tested rating to arrive at a final score. This score was the metric compared against *Cinematch* to determine the winner[1].

The prize was captured in September 2009 by the team "BellKor's Pragmatic Chaos", a group of professional statisticians and machine learning experts. Their methods were sophisticated, entirely new, and were comprised of 107 predictor algorithms run as an ensemble. At their core, however, they were recommender systems that had advanced ways of producing better similarity metrics[3].

The objective of this paper is less ambitious than the Netflix Prize. The combined efforts of the winning team included well over 2000 hours of work for their preliminary solution where even their simplest algorithms took 45 minutes to run. Several searches spanned over many hours[4]. The scope is beyond what can be provided in less than a semester's time.

Instead this paper will examine a high density subset of the dataset. Initial exploration of dataset included the use of the *recommenderlab* tool suite, a set of tools and algorithms in R "which provides the infrastructure to develop and test recommender algorithms for rating data and 0-1 data in a unified framework"[5]. While the tool was useful for exploring small datasets it did not scale with high memory / density matrices and therefore was abandoned. Instead a custom, optimized C++11 library of tools was developed from scratch to measure: UBCF, IBCF, similarity, normalization, nearest neighbor optimizations, and finding where high density based predictions begin to lose effectiveness.

High density simply means that the user-item ratings matrix contains enough entries to fill the matrix past a pre-determined percentage. The study [3] explores performance for different collaborative filtering algorithms especially towards the lower end of the density spectrum. It notes that depending on density different approaches should be taken to get collaborative filtering to work.

The objective of this paper is to explore the high density region ($> 20\ \%$) with user-based collaborative filtering, item-based collaborative filtering, different methods of similarity comparison, finding 'k' nearest neighbor parameters, comparing the effects of normalization, finding if UBCF or IBCF trends better, and establishing a relationship between accuracy and various levels of density in this high density region.

2 Method

The approach to the project is essentially a survey of different recommender systems. The kernel of these approaches are the two main collaborative filtering techniques: user and item based. Two main programs run the algorithms of each while several others will construct appropriate input datasets. Parameters for normalization, similarity, dataset, and nearest neighbors will be supplied to each. Below is a discussion of each method.

2.1 UBCF

This technique is given a set of test users that request ratings for a randomly selected movie of which they have rated. Against this training set, all users who have rated the test movie are extracted. In this subset of users a similarity comparison is done against the test user. A sorted list is returned. From the most similar users in this list, as set by a parameter k, a rating can be made by averaging results of their scores. The search is expensive because there is no easy way of visiting the entire user space, especially with a dense subset. Because this technique compares user-to-user directly it receives the UBCF designation.

2.2 IBCF

Similar to UBCF, IBCF instead seeks items that are most similar to each other. In most traditional uses of IBCF, a list of most similar items are returned based on a similarity score. The algorithm has been modified to instead return a user's estimated rating from a similarity matrix.

The first step is to construct a user / movie ratings matrix where all ratings of the training set are filled accordingly. Then, systematically, each item's ratings are compared to all other items. A similarity computation is performed that yields a similarity number. This number is stored in a user / movie similarity matrix. Note that this matrix is triangular and will be roughly half the size of the ratings matrix.

When a test user requests a rating, the test movie's row index is used. Along this row are returned the similarity values for all of the test user's known ratings. A weighted sum of the user's own ratings is used via the formula to construct the final result:

$$\text{Rating} = \frac{\sum_{i=1}^{k}(r_i * sim_i)}{\sum_{i=1}^{k}(sim_i)}$$

Another weighing explored was the squared sum. The highest similarity is always 1 therefore the lower the similarity the more the squared rating is penalized:

$$\text{Rating} = \frac{\sum_{i=1}^{k}(r_i * sim_i^2)}{\sum_{i=1}^{k}(sim_i^2)}$$

2.3 Similarity Metrics: Cosine and Pearson

Similarity metrics in this paper follow two main formulations. Cosine distance measures similarity via the formula:

$$cos(\theta) = similarity(A, B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{N} A_i B_i}{\sqrt{\sum_{i=1}^{N} A_i^2}\sqrt{\sum_{i=1}^{N} B_i^2}}$$

Pearson similarity is computed as:

$$r = similarity(A, B) = \frac{\sum_{i=1}^{N}(A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_{i=1}^{N}(A_i - \bar{A})^2}\sqrt{\sum_{i=1}^{N}(B_i - \bar{B})^2}}$$

Both forms will be examined across UBCF and IBCF. The Pearson is expensive because it requires two passes of the data, one for the average, and the second for the similarity. However, this cost may be justified because of an inherent tendency to normalize.

2.4 Normalization

Users can be optimists or pessimists on all their ratings or for portions of their ratings. To see the effect of this bias, similarity measurements will be conducted under normalized and non-normalized data. In UBCF two main biases will be normalized. The first will be for computation of similarity. The second will be on the returned score to the user, to shift back into the biased domain.

For IBCF, the similarity matrix is constructed with normalized ratings. Because only a similarity is returned, a test user's bias need not be computed.

The normalization formula used is simply a shifted bias from the average rating a user gave on all movies that they rated:

$$Rating_i = Rating_{ui} - \overline{Rating_u}$$

2.5 Blending of Global Data for Cold Starts or Sparse Matches

In cases where similarity yields undefined results, for example, floating point overflow or underflow, the average rating for a movie is used instead. Care is taken to check for such conditions.

2.6 Other Approaches Considered

There are many other areas to explore with recommender systems. Some mentioned in [3], [4], [6], and [7] include Restricted Boltzmann Machines (RBM), Matrix Factorization / Single Value Decomposition (SVD), and Slope-One estimation. Even more exotic approaches described in [2] include simulated annealing. Indeed, the space of applications is very large and continually growing. Computation and limited research time prevented exploration into these areas. [1] indicates that the Netflix Prize intermediate solution took 2000 hours and involved 107 algorithms for RBM and SVD,

which exceeded the time alloted for the assignment. The Slope-One estimation was more appropriate for linear regressions but not with the implemented system in this paper. Finally, only the high density region of the dataset was explored. The paper [7] notes that different approaches might be considered depending on the data density with higher density producing more reliable results. The results of this project might show an upper bound expected for these types of algorithms.

3 Plan and Experiment

3.1 Dataset

The dataset is comprised of over 100 million ratings of 17700 movies from 480,000 users. 17700 text files, one per movie, contain a user id, rating, and date per line in ASCII comma-separated format. A movie ID value was given on the first line of the file as an integer number. A separate file gives the movie title for a given ID. User IDs were arbitrary integers assigned by Netflix that stripped away personally identifying information. The same ID is used by a user for all movies that they rate. Ratings are ordinal integer values from 1 to 5, with 1 indicating the worst rating and 5 the best. Dates are also given in month-day-year format.

The overall experiment will follow the guidelines for evaluation as set by Netflix. A test set (not provided) will query the recommender system which will give a decimal rating for the user based on that user's prior movie ratings. Netflix secretly held the actual rating given by the user. Errors will be tabulated using the root-mean-square-error formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}}$$

– Dataset overall description, explain how subsample density sets are generated

– RMSE evaluation, chosen by Netflix over linear error methods, such as MAE, to make the penalty of errors be more pronounced. Emphasizes accuracy.

– Baseline is average ratings for each movie over the entire training set. This is the standard used by the papers written for CF and for the Netflix Prize. It is reasonable because these papers note at how well it works, even when some methods begin failing.

3.2 Hypotheses / Questions

– Normalization will offer some benefit to both algorithms. However, the effect should be relatively small. The ratings space only occupies 5 ordinal values; there is not much to normalize.

– High density data will yield subperb results for UBCF or IBCF. These are the best conditions for such data. However, a cross-over point will be observed where average RMSE will beat these metrics. A simple test against a low density sample showed that RMSE scored 1.05 while the other methods ranged from 1.20 to 1.32 RMSE. A cross-over point might occur in the 30 % density region.

– User based collaborative filtering will work better than item based collaborative filtering. This is just a guess. It will be interesting to see which method produces better results.

– Pearson similarity will beat Cosine similarity. If any normalization effects occur, Pearson gives some treatment to them because it computes similarity from the average. Further, Pearson will have little effect between normalization and non-normalized data.

– A relationship for a fixed number of movies and fixed number of users in a subsample will demonstrate which trend might hit a cross-over point sooner.

– Low k values in the k-nearest-neighbors parameters will perform poorly. [7] notes that these metrics are somewhat arbitrary.

3.3 Experimental Design

– Random number generator, repeatability

Table 1: Movie vs. User Count High Density Subsamples

Movie / User	500	1500	2500	3500	4500
1000	0.851	0.794	0.763	0.741	0.723
3000	0.624	0.541	0.502	0.476	0.456
5000	0.491	0.405	0.368	0.344	0.327
7000	0.401	0.320	0.287	0.267	0.252
9000	0.338	0.263	0.234	0.217	0.204

– 10-fold cross validation

– Tests generated in a fold by randomly removing a rating for a test user. The rating was withheld from the CF algorithms and used in the RMSE calculations.

– Subsampled dataset picked movies and users by number of ratings counts.

– Density matrix established: Show chart of Top User x Top Movie with density filled in

– The first experiment will determine which k-NN is the best setting for each combination of input. Results will be computed against the M1500_U500 subsample with further subsamples depending on results. The k parameter will vary from 1 to 20 (or more or less) to find a maximized setting. The results of these experiments will then set the k-value for all subsequent tests.

– The remaining experiments will use the 5x5 set of high density subsamples found in table (entry here). Cosine and Pearson will be applied across all IBCF / UBCF for RMSE evaluation.

– Normalization settings will be added to the prior settings to their effects. Normalization occurs in the similarity matrix construction and similarity evaluation in both CF settings. It is also used to denormalize a user's bias un UBCF.

– U and M parameters held while other parameter varies to uncover where IBCF and UBCF excel or underperform.

– Across all experiments IBCF and UBCF compared.

4 Results

– knn graph of results with values chosen in the experiments, also shows IBCF better than UBCF

– combined graphs for Pearson/Cosine/Denormalized/Normalized/Baseline per IBCF and UBCF

– overall density vs. RMSE (demonstrates no cross-over point in the high density region)

– UBCF and IBCF parametric graphs (hold one constant while other varies)

5 Conclusion

The dataset is large. A good portion of time was spent on exploring IBCF and UBCF features in *recommenderlab* by using the built-in MovieLense database. The routines did a great job in explaining the overall programming approach, but they fail to run against extraordinarily large amounts of data. The remedy is to sample the full dataset with a collection of C++ tools. To this end, much of the development work has gone into creating these tools and processing the data for use in R Studio.

The toolsuite might be extended to perform the calculations described in this report. Raw C++ can handle the large amounts of data and run much quicker than R's interpreted environment. However, the interpreted environment offers the advantage of operating on data without reloading it for each experiment.

A day or two might be spent looking for APIs to enhance the approach.

KNN IMPORTANCE NORMALIZATION IMPORTANCE OPTIMIZATION GPU IBCF BETTER THAN UBCF DENSITY AS LOW AS 20% WORK WELL POSSIBLY MUCH LOWER INTERPOLATION METHODS

Lessons learned: Was overwhelmed by algorithms at first. Explore a bit more and be patient with results. (SEE SLIDES)

GIT Repository

Code for this project can be found under the 'c++' subdirectory at the Github address `https://github.com/AquaPi271/alda_project`.

References

[1] Xavier Amatriain, Justin Basilico. *Netflix Recommendations: Beyond the 5 Stars (Two Parts)*. The Netflix Tech Blog, 2012. https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429. 11/5/2017.

[2] Linyuan Lu, Matus Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, Tao Zhou. (2012) *Recommender systems*. In Physics Reports pp. 1-49, Elsevier B. V. 0370-1573.

[3] Yehuda Koren. (2009). The BellKor Solution to the Netflix Grand Prize.

[4] Robert M. Bell, Yehuda Koren. (2007) *Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights*. Seventh IEEE International Conference on Data Mining. pp. 43-52. IEEE.

[5] Michael Hahsler (2017). recommenderlab: Lab for Developing and Testing Recommender Algorithms. R package version 0.2-2. http://lyle.smu.edu/IDA/recommenderlab/

[6] Daniel Lemire, Anna Maclachlan. (2007). *Slope One Predictors for Online Rating-Based Collaborative Filtering*. In CoRR, Volume abs/cs/0702144.

[7] Joonseok Lee, Mingxuan Sun, Guy Lebanon. (2012). A Comparative Study of Collaborative Filtering Algorithms.