CSCI 5123 Replication Assignment

Bergstein, Ryan
bergs643@umn.edu

Newman, Camden
newma674@umn.edu

Abstract

In this study, a replication of a preexisting 2010 RecSys paper was conducted that focused on the performance of recommendation algorithms in predicting the top-N items for a user. The students conducting this replication followed the paper's methodology for testing the performance of all the algorithms listed in the paper as closely as was outlined, as well as introduced two new datasets into the procedure. The purpose of the replication aimed not only to test the validity of original paper's findings, but verify that their results held true on unexplored datasets.

1 Introduction

The paper under review, "Performance of Recommender Algorithms on Top-N Recommendation Tasks", written by Paolo Cremonesi, Yehuda Koren, and Roberto Turrin[1] desired to look closer at the nature of top-N recommendations. In their study, the authors proposed that there are no trivial relationships between error and accuracy metrics, that there is a different way to introduce a non-biased test set for evaluation, and that there are new variants of preexisting recommendation algorithms that lead to improved top-N performance. The two datasets they used were the MovieLens-25M and Netflix-100M datasets, which contain information about how users rate certain movies. It was found that SVD class of algorithms achieve higher recall and precision metrics.

1.1 Datasets

In this replication two extra datasets were used as well as the MovieLens-25M and Netflix-100M prize datasets, the public FilmTrust and CiaoDVD movie datasets[2]. The sizes of the datasets were as follows: MovieLens contained 25 million user ratings, Netflix contained 100 million, FilmTrust contained 35,497, and CiaoDVD contained 72,700 ratings. The goal was to use not only new, but smaller datasets to compare the replicated paper to.

1.2 Algorithms

The algorithms used within this study are various collaborative filtering algorithms such as asymmetric SVD and Item-Item collaborative filtering, as well as non-personalized methods like to use as a baseline. Specifically the paper uses seven algorithms. (PureSVD can change depending on parameters so technically there are eight algorithms used).

- MovieAvg. MovieAvg returns a sorted list of the movies in the dataset based on total average rating. This algorithm is non-personalized and is used as a baseline for comparison.
- TopPop. TopPop returns a sorted list of the movies in the dataset by the frequency of how often each movie is rated. This algorithm is also non-personalized and is used as a baseline.
- K Nearest Neighbors (kNN). A neighborhood is constructed by taking the pearson correlation between movies in the dataset and then making recommendations to a user based on that calculated neighborhood. The ratings are also normalized by each movies average rating.
- Non-Normalized kNN. This is another neighborhood model that instead uses the cosine similarity between movies to devise the neighborhood. It is also non-normalzied so there is no dampening.
- Asymmetric SVD. Aysmmetric SVD follows the classic SVD algorithm where the user x item rating matrix is decomposed into two vectors representing the users tastes and the movies features. Ratings are then calculated by taking the dot product of both vectors.
- SVD++. SVD++ is very similar to SVD, but also considers the implicit information present in the data to make recommendations. In adds more entries to the two feature vectors to account for this new information.
- PureSVD. PureSVD another variant of the SVD algorithm and differs in how it computes the users predicted rating. It focuses solely on the matrix factorization aspect of SVD without incorporating additional user or item features. The version of PureSVD utilized in this paper allows for a parameter to be inputted which specifies the length of the feature vectors.

2 Methodology

The methodology followed was the exact same as described in [1]. Each dataset was broken down into 2 subsets, a training set called M and a testing set called T. The test set T was obtained by taking a 1.4% random sample from the main set containing all ratings and then filtered so that only 5 star ratings were present. Another set of data was created to observe the long-tail distribution

of items where the majority of ratings are condensed into a small percentage of the most popular items. The training and testing splits were performed on this dataset as well, containing 70% of the total data. To measure the recall and precision, the procedure in the paper was followed exactly. First, train the model over the ratings in M. Then for each item i rated 5 stars by user u in T:

1. Randomly select 1000 additional items unrated by user u.
2. Predict the ratings of the test item i and for the additional 1000 items.
3. Form a ranked list of all 1001 items according to their predicted ratings.
4. Form a top-N recommendation list by taking the top N items of the ranked list.
5. If the test item i is in the top-N list then we have a hit.

The overall recall and precision is denoted as the following formulas:

$$recall(N) = hits/\left|T\right|$$

$$precision(N) = recall(N)/N$$

The recall and precision were calculated for each datasets *long tail* items as well as all of its items. Each of the eight models described earlier were used from a python scikit[3] to train on the M dataset and were tested on the T dataset.

3 Results

After following the procedure described in Section 2, we gathered and graphed each model's performance in regards to precision and recall. Just as in the original paper, a range of N=20 was used for the top-N list of recommended items. Similar to the original study's findings, the SVD class of algorithms, namely PureSVD50, performed the best of the tested algorithms. PureSVD150 and SVD++, while usually be

3.1 CiaoDVD dataset

We can see from figures 1 and 2, which depict the performance of our algorithms on the Ciao dataset with all items, that different algorithms do in fact have differing top-N capabilities. As expected, the SVD class of algorithms performed above the other tested algorithms. Surprisingly, PureSVD150 performed significantly worse than expected, which most likely is a result of having too many latent factors with only 72,700 ratings. Notably, PureSVD50 performed the best on all items and the long-tail, aligning with the original paper's findings.

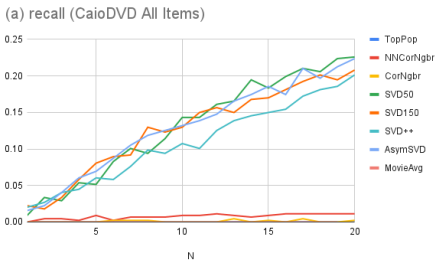

Figure 1: recall-at-N on all items

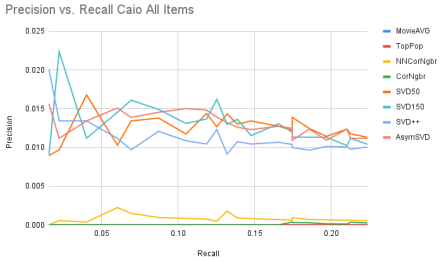

Figure 2: precision-versus-recall on all items

The following figures 3 and 4 represent our findings on the Ciao long-tail dataset. Again we see the SVD class perform above the other algorithms, and with less items in the set, PureSVD50's performance improved even more. We did find it surprising that in taking the long-tail of an already small dataset, that the disparity between the SVD class of algorithms and the others we tested was as large as it was, especially the consistency of the resulting precision as N was increased.

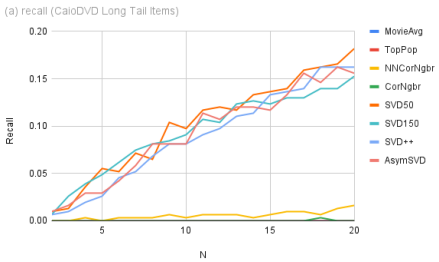

Figure 3: recall-at-N long-tail

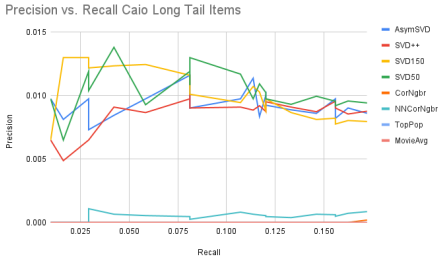

Figure 4: precision-versus-recall long-tail

3.2 FilmTrust dataset

Figures 5 and 6 depict our study on our FilmTrust dataset, our smallest dataset of only 35, 497 ratings. As expected with such a limited number of ratings, the disparity between our algorithm's performances was larger with less data to train on. Against our expectations, the three strongest SVD algorithms *far* outperformed our other tested algorithms, which we believed would contest the SVD class as a result of limited ratings at hand.

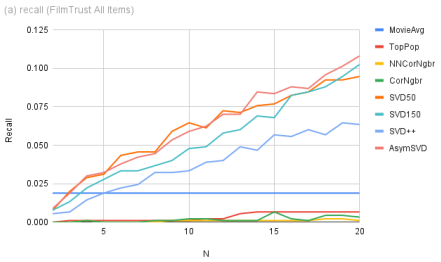

Figure 5: recall-at-N on all items

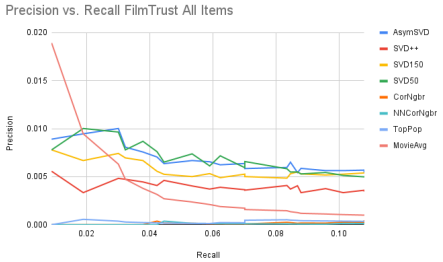

Figure 6: precision-versus-recall on all items

Our study of the FilmTrust longtail, shown in figures 7 and 8, behaved as expected, with results dropping in value with even less items available to train on. While the disparity between the performances of our algorithms narrowed, the position of which performed the best relatively remained the same, with PureSVD50 narrowly beating out AsySVD.

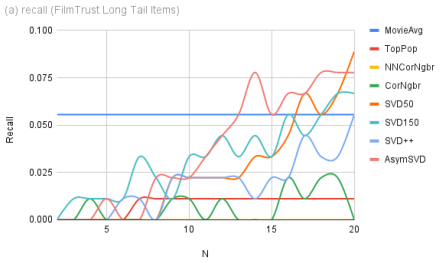

Figure 7: recall-at-N long-tail

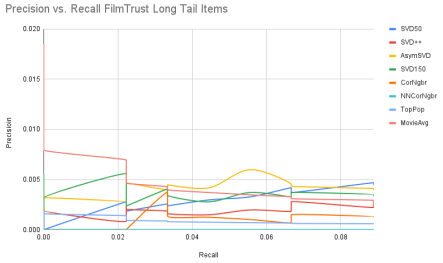

Figure 8: precision-versus-recall long-tail

3.3 MovieLens dataset

When looking at figures 9 and 10, we can see that our results on the MovieLens dataset line up quite well with the results of the original paper, with the SVD class of algorithms at the top. The CorNgbr and NNCorNgbr permformed less than expected when compared to the paper. This is most likely due to the reduction of the dataset to account for low hits.

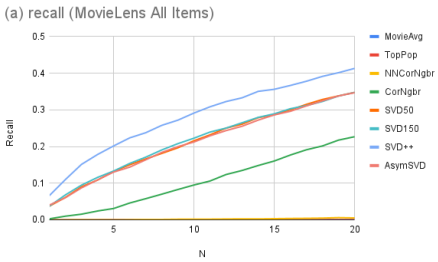

Figure 9: recall-at-N on all items

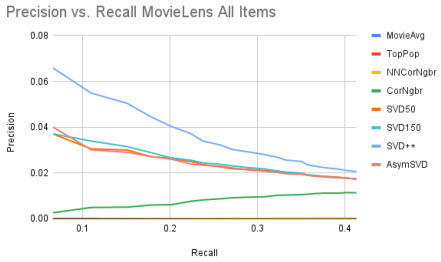

Figure 10: precision-versus-recall on all items

For figures 11 and 12, which represent the long-tail items for the MovieLens dataset, shows that SVD class algorithms are still the top performers and the Item-Item CF algorithms are lacking in higher recall over a higher N. In figure 12 for CorNgbr, the precision actually increases when recall increases which does not follow the papers results.

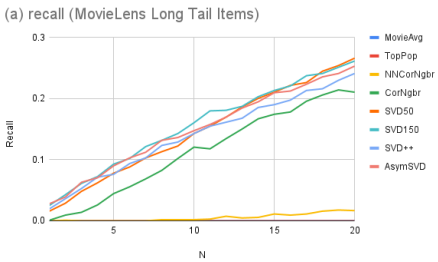

Figure 11: recall-at-N long-tail

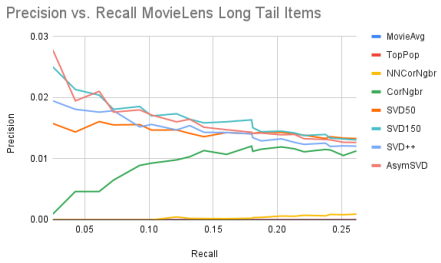

Figure 12: precision-versus-recall long-tail

3.4 Netflix dataset

Looking at figures 13 and 14, we can see that our results on the Netflix dataset line up quite well with the results of the original paper, with the SVD class of algorithms at the top. However, unexpectedly our results for CorNgbr and NNCorNgbr had low recall and precision, which may be a result of the clamping effects in the Surprise algorithms. PureSVD50 was the highest performing algorithm in both studies, with MovieAvg as the worst in both. In our study there more of a gap between the better performing algorithms and the bottom algorithms, but we attribute that to our use of less ratings.

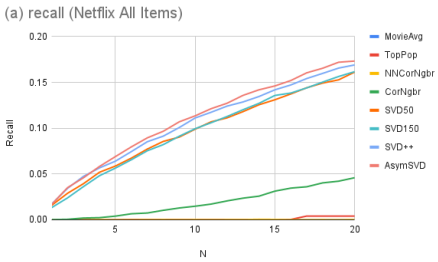

Figure 13: recall-at-N on all items

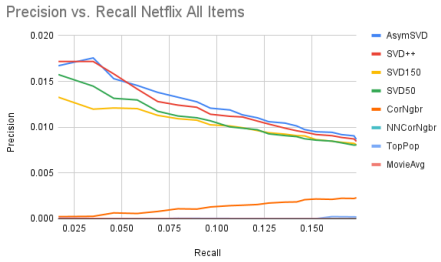

Figure 14: precision-versus-recall on all items

The long-tail Netflix dataset, as seen in Figure 15 and 16, also closely resembled the original paper's findings, but again our CorNgbr and NNCorNgbr performed under expectations.

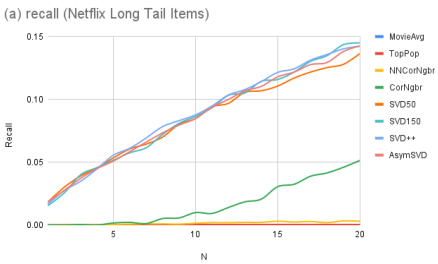

Figure 15: recall-at-N long-tail

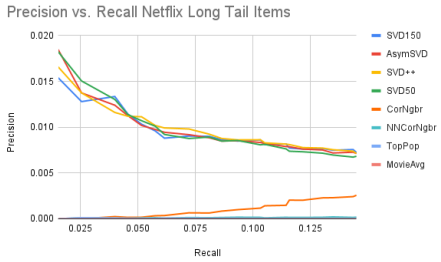

Figure 16: precision-versus-recall long-tail

4 Discussion

To conduct our replication, we wrote and ran our code on personal devices that didn't have the computing power of anything close to industrial size. As a result, we were forced to slim down the size of the datasets to a size that cooperated with our limited storage capacity and could be processed in reasonable time. While the original paper conducted their study with all 25 million MovieLens ratings, we took a random sample of 2 million ratings from that 25M dataset to then split into training and test sets as outlined in previous sections. Similarly, we slimmed the Netflix dataset down from 100M to 10 million ratings and processed it accordingly.

During the process of replication we also found that the original paper left some topics ambiguous in how they approached them. One such topic was how they went about testing their datasets, specifically how many folds were used to test upon. We assumed that because there was no mention of any folds or cross-validation that they only used a single test set, so we did the same in our

study.

One final caveat that we encountered in our replication surrounds the recall and precision metrics. While we wrote our recall and precision functions ourselves as the original paper did not provide their code, we believe them to function correctly and had them double checked by peers, all reporting them to be correct as well. The scale of our results match up with the scale of the original paper, but our actual numbers are much lower than expected. We believe that the algorithms we used from the Surprise science toolkit exhibit "clamping", a tactic used to confine results to a certain range of numbers after computing the predicted ratings. If an algorithm were to have created a prediction for a movie with a higher value than the max rating on that scale, it would be "clamped" down to that highest value. As a result, our top-N list of items to be recommended is no longer a true top-N sorted list, which in turn lowered our final metrics.

Regarding the accuracy of RMSE in predicting a true top-N recommendation list, our final beliefs align with the original paper. It is evident that error metrics like RMSE and MAE provide a good estimate of a top-N list, but we have shown their relationship to be an estimate at best. In all tested datasets both small and large, at least one of the non-personalized "baseline" algorithms outperformed more complex RMSE optimized algorithms, which further argues the idea that RMSE may not be the best metric to define a top-N list on.

References

[1] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 39–46, New York, NY, USA, 2010. Association for Computing Machinery.

[2] Librec github repository.

[3] Nicolas Hug. Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174, 2020.