Evaluating Recommender Behavior For New Users

Daniel Kluver
GroupLens Research
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455 USA
kluver@cs.umn.edu

Joseph A. Konstan
GroupLens Research
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455 USA
konstan@cs.umn.edu

ABSTRACT

The new user experience is one of the important problems in recommender systems. Past work on recommending for new users has focused on the process of gathering information from the user. Our work focuses on how different algorithms behave for new users. We describe a methodology that we use to compare representatives of three common families of algorithms along eleven different metrics. We find that for the first few ratings a baseline algorithm performs better than three common collaborative filtering algorithms. Once we have a few ratings, we find that Funk's SVD algorithm has the best overall performance. We also find that ItemItem, a very commonly deployed algorithm, performs very poorly for new users. Our results can inform the design of interfaces and algorithms for new users.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering

General Terms

Algorithms, Measurement

Keywords

Recommender Systems, Evaluation, Profile Size, New User Experience, New User Problem, User Cold Start

1. INTRODUCTION

One of the important problems in recommender systems is the new user experience. If the system cannot provide a good user experience, then new users are likely to leave and not return. Even if the system can make accurate recommendations, accuracy alone is not enough to make a good first impression. If the system only recommends items the user has seen, then the user may see no value in the system and leave. Alternatively, if the system only recommends obscure items, then the user will have no way to know if the recommendations are worth taking. This problem is made more complex by how little information we have about new users.

Past work on new user experience in recommender systems has tended to focus on the 'cold start problem'. Cold start refers to the problem that no algorithm can make highly personal recommendations with very limited amounts of information. Unfortunately this means that regardless of the algorithm, no recommender system can make high quality recommendations for users who are still joining the system.

The most studied strategy for gathering more information for new users is to include a new user rating survey. Systems that utilize this practice do not allow their users to receive recommendations until some fixed number of ratings have been entered. The idea is that after enough ratings have been entered, the predictions will be 'good enough'. This idea has been explored by many authors; most consider a system in which the user is presented with a list of items and asked to rate items until they have enough ratings. For a good discussion of prior work in this direction see [8].

While past work has carefully covered the new user rating survey, it has not carefully evaluated which algorithm is best at recommending to new users. The new user survey work sidesteps this issue by assuming that the user will receive no personalization until the survey is complete. Even for systems with a new user survey, carefully choosing algorithms for new users may be very important. Users fresh out of the new user survey will still have far fewer ratings than the average user of the recommender system. Different algorithms may provide very different experiences for these users, even if they give very similar results for the average user.

The ideal way to study this issue would be a live study conducted on people who are new to a recommender system. Users could be randomly assigned an algorithm and then surveyed after a random number of ratings to establish what they thought of the new user experience. Unfortunately, there are many algorithms and new users are a scarce resource.

Fortunately, there has been a growing trend in recommender system research to perform multiple metric offline evaluations of recommender systems. This work comes out of a pushback against early recommender evaluations which only evaluated recommenders in terms of their predictive accuracy [13]. The response of this pushback has been the generation of many different metrics which allow us to more carefully understand the separate aspects of a recommender's output [11, 3, 18]. This strategy allows us to dig much deeper into the differences between algorithms, looking past simple measures of accuracy to more meaningful dimensions such

RecSys'14, October 6–10, 2014, Foster City, Silicon Valley, CA, USA.

ACM 978-1-4503-2668-1/14/10 ...$15.00.
http://dx.doi.org/10.1145/2645710.2645742 .

as how novel the recommendations are, or how frequently the recommendations change for any user. This allows us to capture the qualitative tradeoffs between algorithms which are otherwise quite similar, which helps us choose algorithms that are better suited for a particular domain.

In this work we combine these two streams of research: new user experience research and multiple metric algorithm evaluation. Our goal is to understand the differences between algorithms for new users, especially how these algorithms respond to different amounts of information. By using multiple metrics we can understand not just what algorithm is best in terms of absolute performance, but also gain a more nuanced understanding of the qualities of different algorithms' recommendations.

We structure our work around the following research question:

- RQ1: How do algorithms behave for users with few ratings?

Algorithm behavior is a very broad subject. We will look at three distinct sub-questions to help capture algorithm behavior.

- RQ1.a: How well can different algorithms predict the future ratings of new users?
- RQ1.b: How well can different algorithms rank and recommend good items to new users?
- RQ1.c: How do algorithms behave as measured by other metrics, such as popularity and diversity, for new users?

We will answer these questions by developing an evaluation framework that can be used to understand how algorithms perform for new users. This framework will compare algorithms on eleven different metrics to allow a more complete understanding of the differences between algorithms. We will then use this framework to study how three common algorithms behave for new users. Finally, we will draw conclusions from our evaluation and outline directions for future work.

2. METHODOLOGY

Ideally we would measure new user experience with a live survey on users who are just joining a recommender service. Because new users are a scarce resource we opt to first explore the new user experience in an offline setting. This allows us to test many more profile sizes and algorithms than an online study. We will adapt the standard train and test methodology so that test users have only a specified number of training ratings. By running this approach for various profile sizes we can generate plots of performance against profile size. This lets us understand both how the algorithms compare on these metrics, and how the algorithm performance changes with the number of ratings provided.

This, or a similar technique, has been used by previous authors to make plots of metric values over profile size [4, 2, 9, 11]. Past work used these plots to establish that more ratings lead to better results. We take this technique farther and develop a much more nuanced understanding of the different algorithms, and how they compare for different profile sizes.

Our approach is related to the temporal evaluation methodology, in which a recommender is repeatedly trained on all ratings up to a point in time, and then tested using the next ratings to be made for each user. Temporal evaluations have been used in the past to evaluate how the list of recommended items change as users enter more ratings into the system [11]. While the output of a temporal evaluation can be very similar to our own, the methodologies have some key differences. Where a temporal evaluation is designed specifically to consider the order in which users make ratings, our methodology is designed to try to eliminate this ordering effect from our consideration of the algorithm's behavior.

As mentioned earlier, many recommender systems use a new user survey to collect the first user ratings. These ratings are different than normal user ratings. As we explain shortly, our methodology uses randomization in key places to remove this ordering effect. This allows us to better capture the underlying algorithm performance on "normal" user data. While this makes our methodology unsuited for applying temporal metrics, it allows us to separate an algorithm's behavior from how ratings are collected for new users. We feel that this is better given the limitations of our dataset, and our focus on the algorithm, rather than how ratings are collected.

Algorithms

We seek to develop an understanding of how a range of different standard algorithms perform for new users. Rather than picking advanced algorithms that are tuned for new user performance, we decided to first understand how representatives of three common types of algorithms perform. Therefore we will compare UserUser[14], ItemItem[15], and Simon Funk's SVD[7] against two simple baseline algorithms.

Due to the low profile sizes involved in our evaluation, not every algorithm was able to make predictions for every user-item pair. In these cases, we use the UserItemBaseline prediction as a fallback. This ensured that our prediction based metrics were always evaluated over the same number of items. We found this unnecessary for recommendations as each algorithm was able to make a complete recommendation with only one rating. Therefore we form recommendations as the top 20 items with predictions from the algorithm which were not in the training set. More information about these algorithms can be found in table 1. Code for our evaluation, using the Lenskit evaluation framework [6] is available at `https://bitbucket.org/kluver/coldstartrecommendation`.

ItemBaseline	Item's average rating, with mild bayesian damping towards the global mean [7].
UserItem-Baseline	ItemBaseline adjusted by the user's average offset from the ItemBaseline. Mild bayesian damping is applied [7].
UserUser	User based nearest neighbor collaborative filtering [14] with a neighborhood size of 30 and ratings normalized by subtracting the UserItemBaseline score.
ItemItem	Item based nearest neighbor collaborative filtering [15] with a neighborhood size of 30 and ratings normalized by subtracting the ItemBaseline score.
SVD	Simon Funk's SVD based collaborative filtering approach [7] with 30 features and 150 training iterations per feature.

Table 1: Summary of algorithms.

Metric	Description	Range	Preferred direction
Accuracy and Ranking Metrics			
RMSE	Standard prediction accuracy metric [16].	0 to 5 stars	Smaller is better.
nDCG	Standard ranking quality metric [16].	0 to 1	Larger is better.
Monotonic Recommendation Metrics			
Precision@20	Count of recommended items in the test set and rated 4.0 or higher[1] [16].	0 to 1	Larger is better.
MAP@20	Average of precision at each rank one through twenty [12].	0 to 1	Larger is better.
Fallout@20	Count of recommended items in the test set and rated 2.0 or lower[1] [2].	0 to 1	Smaller is better.
MeanRating@20	Average rating of recommended items that are in the test set.	0 to 5 stars	Larger is better.
RMSE@20	RMSE computed only over those items that were recommended.	0 to 5 stars	Smaller is better.
Non-monotonic Recommender Metrics			
SeenItems@20	Count of recommended items in the test set.	0 to 20 items	Moderate values are better.
Average-Popularity@20	The average number of users in the training set that have seen recommended items [18].	1 to 6000 users	Moderate values are better.
AILS@20	The average pairwise similarity between recommended items [18].	-1 to 1	Moderate values are better.
Spread@20	The Shannon's entropy of the distribution of recommended items for users in the test set.	0 to 12 bits	Moderate values are better.

Table 2: Summary of metrics

Each algorithm was tuned based off of Ekstrand's 2012 evaluation [5]. Since this evaluation is on a different dataset (MovieLens 10M vs. MovieLens 1M), we performed minor tuning using Ekstrand's values as a starting point. We tuned based on RMSE performance. While tuning, we found that the previous damping parameter used for the baselines to be too large. We found that a value of 5 gave much better results for new user recommendation. Our key results are not sensitive to minor changes of algorithm parameters.

Metrics

We present results from eleven metrics. As has been seen in previous studies [10] we find that some of are metrics are redundant. Future work should be able to use a smaller list of metrics. The metrics are described in table 2. We split our metrics into three groups roughly along our three sub research questions.

Accuracy and Ranking Metrics.

RMSE and **nDCG** evaluate the recommender based on its prediction for each item. These evaluate the accuracy of the predictions and the ability of the recommender to rank the list of all items. We find that these two metrics perform quite similarly in our evaluation.

Monotonic Recommendation Metrics.

These metrics evaluate the top 20 recommendations produced by the recommender. The choice of 20 items for the evaluations was arbitrary, testing with other values showed similar results, except on one metric. This issue will be addressed later in the results section. We refer to these as monotonic metrics as there is a clear good direction for the metric.

[1]We also tried 5.0 and 1.0 as cutoffs respectively, and found no meaningful difference.

Precision@20, **MAP@20**, and **Fallout@20** are standard information retrieval metrics which measure the quality of the recommendations. As we show later, these metrics were not useful for our evaluation. Therefore, to help understand the quality of our recommendations, we also include MeanRating@20, and RMSE@20.

MeanRating@20 tries to establish how much users like their recommendations by measuring users average satisfaction with recommended items based on withheld ratings. **RMSE@20** accompanies this metric and lets us know if the recommended items are likely to be incorrectly predicted. This can help us understand why an algorithm scores well on MeanRatings@20.

Non-monotonic Recommendation Metrics.

SeenItems@20, **AveragePopularity@20**, **AILS@20**, and **Spread@20** are all non-monotonic measures of the user's recommendations. These metrics are harder to apply than the monotonic measures of recommendation quality. Unlike the previous metrics, we expect that users will be most satisfied by moderate values of these metrics, and that both extremes can lead to a negative user experience.

We use these metrics as they can help us understand important properties of how users perceive their recommendations. As mentioned in the introduction the number of recommended items that the user has seen (estimated by **SeenItems@20**) can have a major impact on how much users trust the recommendations. If an algorithm recommends too few items that the user has seen then we expect the user will have trouble evaluating if the recommendations are trustworthy. Too many, and the user may feel that the recommendations are not interesting or useful. Unfortunately, the optimal value for these metrics will vary by domain, user, and context. That said, we feel it is useful to understand how these metrics vary across algorithms for new users, as

this knowledge can help inform default algorithm selection within a particular domain and context.

AveragePopularity@20 is a crude metric of how novel the recommendations are to the user [18]. Broadly speaking, we expect users will prefer recommendation lists containing more novel (less popular) items. However, if the recommended items are too novel the user is unlikely to have heard of the items and will have no point of reference in understanding their recommendations.

Past work has shown that users like their recommendations to be more diverse [18, 17]. The belief is that by providing more diverse lists we are giving the user a larger range of items to choose from. This can make choosing an item easier as the user is less likely to have to compare very similar options. It has also been shown, however, that too much diversity can lead to less user satisfaction [18]. This is probably due to a tradeoff between how diverse a list is and how well it captures a user's tastes. At some point, a list cannot become more diverse without including items that do not interest the user. We will measure diversity with the **AILS@20** metric.

The **AILS@20** metric is based on the Intra-List Similarity (ILS) metric introduced by Ziegler et al. [18]. AILS@N is simply a rescaling of ILS to be scale free with regards to N, this was done to make the metric easier to interpret, by putting it on the same scale as similarity values. We used the same item similarity metric for AILS@20 as we used in the ItemItem recommender.

The **Spread@20** metric is designed to measure how well the recommender spreads its attention across many items. We expect that algorithms with a good understanding of its users will be able to recommend different items to different users. Like diversity, however, we expect that it is not possible to reach complete spread (recommending each item an equal number of times) without making avoidably bad recommendations. When Spread is large we know that the recommender is recommending many items with a relatively even distribution. When Spread is small, the recommender is focusing on a small set of items which it recommends to every user. In this way spread is similar to an item coverage metric, but provides a more nuanced measure of item coverage as it considers how often each item is recommended.

Spread is computed by computing recommendations for each test user. Let $C(I)$ be the count of how many times item I showed up in the recommendations. Let $P(I) = C(I)/\sum_I C(I)$ be the probability that a random recommendation is for item I. We define spread as the Shannon's entropy of $P(I)$, $Spread = -\sum_I P(I)\log(P(I))$.

Expectations about algorithm performance.

We expect that the ItemBaseline algorithm will have approximately constant performance on all metrics. Since the ItemBaseline is such a simple algorithm we expect that any reasonable algorithm should be able to do better than it. That said, we expect the improvement over the ItemBaseline to be gradual, with larger improvements only happening with more data.

The UserItemBaseline should only differ from the ItemBaseline for metrics that use the prediction value. For all other metrics these two provide the same responses, and we will only report results for the ItemBaseline. As the UserItemBaseline is still quite a simple baseline, we expect that the non-baseline algorithm will beat this baseline as well given enough ratings.

We expect that the baseline algorithms will provide the most popular recommendations, with the least diversity, and the least spread. Any personalized algorithm should be able to make more nuanced recommendations than simply recommending the best rated items, which should translate as gains on these metrics. Furthermore, we expect that as an algorithm learns more about the user it should decrease in popularity, and increase in diversity, and spread, as it becomes able to make more personal and nuanced recommendations.

Dataset

For this evaluation we will use the MovieLens 1M dataset[2]. This dataset contains 1 million ratings from 6000 users on 4000 movies.

These ratings were collected from the MovieLens system, which requires 15 ratings in a new user survey before users can enter the system. To the best of our knowledge, at the time this data was collected the new user survey preferred items that were very popular. We will randomize the order of the ratings to try to avoid measuring the effect of these initial ratings, and instead focus on the algorithm's performance on normal ratings.

Each user in the dataset has at least 20 ratings. Therefore we will evaluate user profiles with up to 19 ratings. This will allow us to ensure that each user in the dataset has at least one test rating.

Evaluation Procedure

The natural way to perform this evaluation is with a modified user crossfold strategy. The normal five-fold cross validation strategy splits the users into five groups. In each of five folds all ratings from four of the groups will be kept in the training set, and then some subset of the ratings from each user in the fifth group will be set aside for the test set. The selection of a test set can be random to avoid any possible ordering effects in the data due to a new user rating strategy. To understand how metrics change with a different number of users we can simply generate separate random crossfolds for each simulated profile size. At each crossfold we include only a constant number of training ratings for each test user, ensuring that all test users have the specified profiles size.

Unfortunately, we found that this strategy causes biases in our evaluation. In particular, this strategy led to the size of the test sets changing as a function of the simulated profile size. As we kept more ratings for the training set, we would have fewer ratings in the test set. This biased our results; some metrics such as nDCG proved to be sensitive to the size of the test set. As we compared different profile sizes we would see trends that were solely a property of the changing test set size.

Our solution to this bias is to subsample from an existing crossfold. To generate training and test sets for different number of ratings from 1 to 19, we first generate a set of test and training splits so that test users have 19 training items using the crossfold strategy outlined above. Then, to generate any other number of ratings, we randomly downsample the 19 item training set. This allows us to generate training sets with a given number of ratings for test users, such that each profile size is evaluated with the same test set.

[2] `http://grouplens.org/datasets/movielens/`

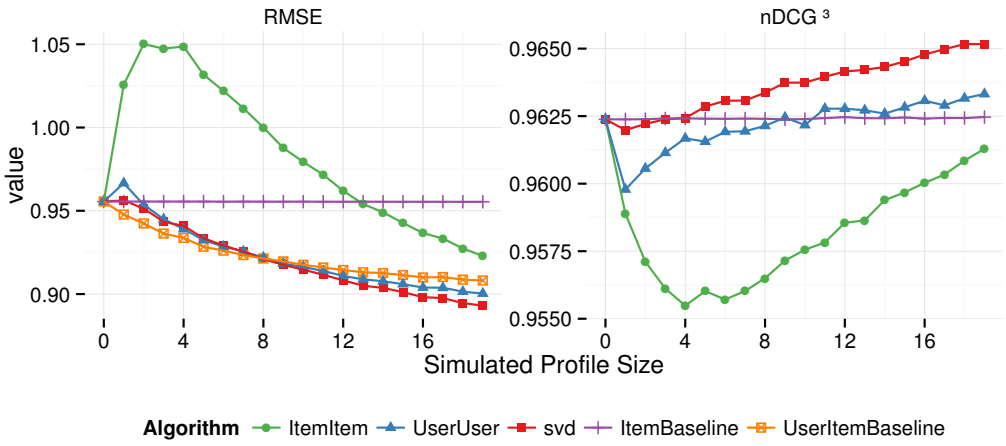

Figure 1: Accuracy and Ranking Algorithms

3. RESULTS

Using the methodology described in section 2 we generate plots for each metric showing how the five algorithms perform versus simulated profile size.

Accuracy and Ranking Metrics

RMSE.

Figure 1 shows how accurate our algorithms are as measured by RMSE. UserUser, and SVD both behaved essentially as expected. SVD shows no improvement with only one rating, and UserUser actually gets worse with the first rating. Other than that, these algorithms outperform ItemBaseline. Interestingly, UserItemBaseline outperforms both algorithms until around 8 ratings, at which point both algorithms start to perform better. Given the size of the differences between UserItemBaseline, UserUser, and SVD we conclude that all three are equally accurate for new users. That such a simple algorithm can outperform more complicated algorithms for small profile sizes suggests that recommending for users with very few ratings is a hard problem.

ItemItem performs quite poorly for new users. Until we have 13 ratings, ItemItem is less accurate than the ItemBaseline. In fact, for the first two ratings ItemItem appears to perform *worse* as more ratings are added. This trend is a result of the limited coverage of the ItemItem algorithm. Considering the accuracy of RMSE on only those items where it makes a personalized prediction, it shows a monotonic increase in accuracy similar to the other algorithms. However, for small rating counts ItemItem can only predict for some percent of the items (around 60% for users with two ratings, and around 75% for users with four ratings). Because of this, many of the predictions come from the better performing UserItemBaseline. As we get our first few ratings, the predictions become less likely to be from the UserItemBaseline, leading to an artificial upward trend in RMSE.

nDCG.

Figure 1 shows the algorithms performance on the nDCG metric. The results for nDCG are very similar to the RMSE results. Both UserUser and SVD take an initial hit to accuracy at the first rating. This time UserUser takes much longer to recover and only beats the baseline after around 12 ratings. Again ItemItem performs quite poorly. This leaves SVD performing the best, passing the baseline after about 3 ratings. Again we find that our best algorithm for the first few ratings is our baseline.

[3]We do not plot UserItemBaseline here as it is equivalent to ItemBaseline on this metric.

Monotonic Recommendation Quality Metrics

MAP@20, Precision@20, Fallout@20.

Figure 2 shows MAP@20, Precision@20, and Fallout@20. All three metrics show essentially the same trend, UserUser gets the lowest score, the baseline gets the highest score. On both MAP@20 and Precision@20, SVD gets a higher score than ItemItem throughout, performing almost as well as the baseline. On Fallout@20 ItemItem and SVD perform about as well as each other, with SVD showing a very slight lead. We also tested recall and mean reciprocal rank metrics, but found them to exhibit the same trend, so they were not explored further.

This result is rather strange as MAP and Precision are metrics where larger scores imply better performance, whereas Fallout is a metric where larger scores imply poor performance. Therefore we would not expect to see rank equivalent behavior from these metrics. Unfortunately this means that these metrics contradict each other. One possible reason for this is that these topN metrics are known to be biased toward algorithms that recommend more popular items [1]. Looking forward to the average popularity (figure 4) of recommended items, we see that the average popularity of the recommendations closely replicates the previous three plots. As a result of this, we discard these metrics, and consider less biased metrics to help us analyze the quality of the recommendations.

SeenItems@20.

We address this non-monotonic metric out of turn here as it provides important context for the next two metrics. The first graph in figure 3 shows the average number of items in the top 20 recommendations for each algorithm that were also in the test set. The average number of test set items in recommendations shows a similar trend to the MAP and precision metrics, with UserUser getting by far the fewest seen items, and the baseline getting the most. As UserUser gets less than one seen item on average we found that UserUser

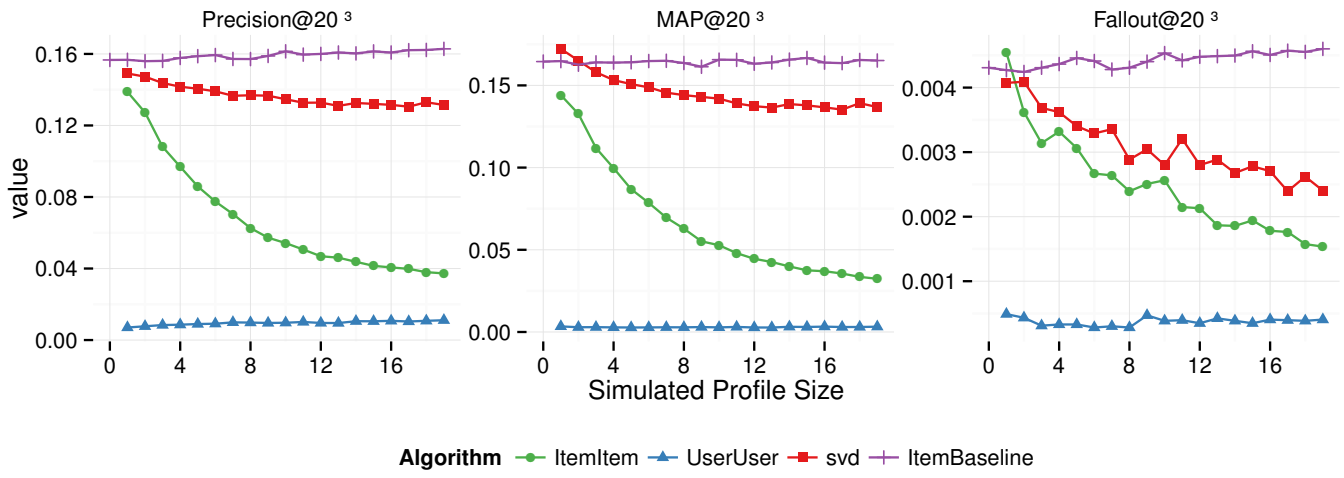

Figure 2: Information Retrieval Recommendation Quality Metrics

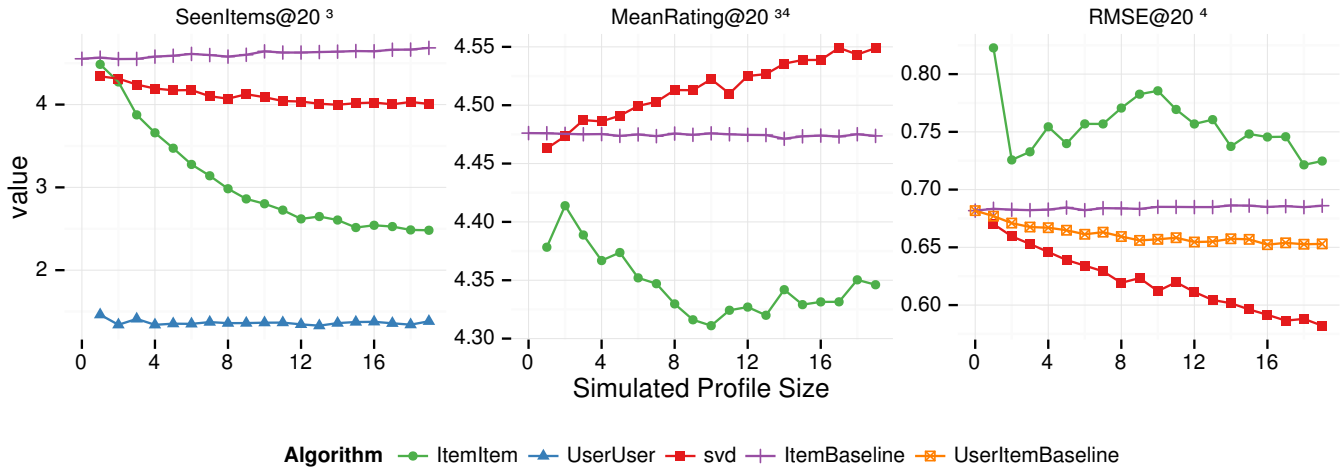

Figure 3: Recommendation Quality Metrics

was not appropriate for the MeanRating@20 and RMSE@20 metrics. In particular, we found that the UserUser results for these metrics are highly sensitive to recommendation list size. Because of this issue we will not include UserUser in our next two metrics and consider the next two metrics inconclusive for UserUser. That said, we consider the small number of seen items to be a bad result for UserUser. Put simply, if we cannot reliably estimate if the recommendations are reasonable, how can we expect a user to do so?

MeanRating@20.

The second graph in figure 3 shows the average rating for those items in the top 20 recommendations for each algorithm. The first thing to note about this graph is that all of these algorithms are doing a reasonable job of recommending items the user might like to watch, with all algorithms averaging above 4 stars (implying that the recommended items are, at least on average, enjoyable). SVD performs as expected, starting around the baseline for a few ratings, and then steadily improving to be the best algorithm after 3 ratings. Again, we find that ItemItem performs poorly until quite a few ratings are added, and does worse than baseline throughout.

RMSE@20.

The third graph in figure 3 shows the accuracy of our algorithms on items in the top 20 recommendations. Not surprisingly, this figure shows the same trends as the last one, meaning that the algorithms whose recommendations were (on average) most likable, were those whose recommendations were most accurate. Again, SVD performs quite well, easily beating the UserItemBaseline after only 1 rating.

Non-monotonic Recommender Quality Metrics

AveragePopularity@20.

The first graph in figure 4 shows the average popularity of items recommended by each algorithm. As expected the baseline algorithms makes the most popular recommenda-

[4]We do not plot UserUser here as it cannot be reliably measured for this metric.

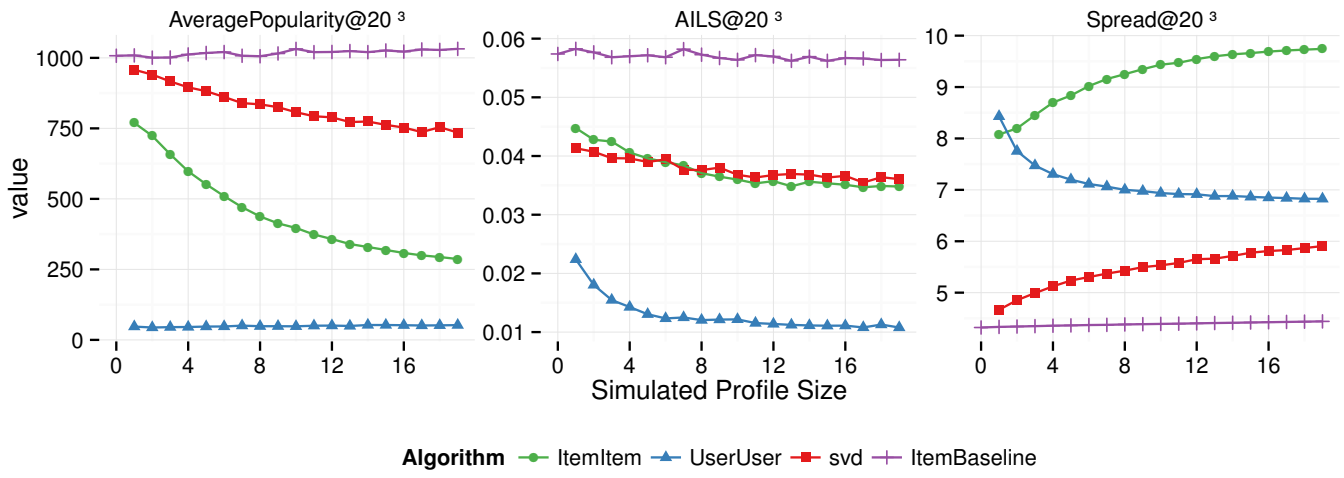

Figure 4: Non-monotonic Recommendation Quality Metrics

tions, followed by SVD, ItemItem, and then UserUser in a distant last. As stated earlier, We expect that users will feel the baseline's popularity (around 1000) is too high. We also expect that users will perceive UserUser's popularity (around 50) as too low. SVD and ItemItem both seem to perform well, with SVD generating more popular recommendations than Item-Item. SVD and ItemItem both show a trend of decreasing popularity as more ratings are entered, suggesting that these algorithms will make more nuanced, less popular recommendations for users they know more about.

AILS@20.

The second graph in 4 shows a similar trend to the popularity trend. Again, we expect that the baseline, which provides the least diverse recommendations, would be perceived as not very diverse by users. Quite possibly due to the preference for obscure movies, UserUser seems to produce the most diverse lists of any algorithm. We expect that the UserUser recommendation lists might be too diverse. Unlike popularity, SVD and ItemItem seem to perform equivalently in terms of diversity. We also see that all three collaborative filtering algorithms show a trend of increasing diversity (decreasing inter-list similarity) as more ratings are entered.

Spread@20.

The third graph in figure 4 shows the spread of the recommendations from each algorithm. As expected the baseline performs worst on spread, as it recommends essentially the same items to each user. SVD performs mildly better than the baseline, and shows definite improvement as more ratings are added. UserUser and ItemItem start at around the same value as each other, however, after only two ratings they have crossed, as ItemItem quickly increases to have the highest spread of all the algorithms, and UserUser quickly decreases into second place. Interestingly, only UserUser shows a trend to have less spread as it learns more about the user, ItemItem and SVD both show the expected trend of increasing Spread as they learn more. This could be some form of a regression towards the mean, where as it learns more about users its recommendations become less personalized to the users individual quirks. More research will be needed to understand this trend.

4. DISCUSSION

Algorithm	Prediction accuracy	Recommendation quality	Other properties
ItemItem	Poor	Poor	Good
UserUser	Good	Inconclusive	Poor
SVD	Good	Good	Good

Table 3: Summary of algorithm behavior for new users

The high level results of our evaluation are summarized in table 3. Overall we see that SVD seems to be the best performing algorithm for cold start use. SVD was the only algorithm that consistently outperformed the baseline algorithms on prediction and recommendation tasks. ItemItem had arguably better results on the average popularity and spread metrics, indicating that it was able to find more novel items, and make a larger range of recommendations to users. Unfortunately, ItemItem also performed quite poorly at both prediction and recommendation, which makes it less suitable for new users.

UserUser performed quite well at prediction, but tended to favor unpopular items in its recommendations. While some novelty is good, we believe that UserUser goes too far with obscure movies in its recommendations, and therefore does not make good recommendations. Therefore we conclude that UserUser alone is not well suited for new users. It is possible that careful tuning or hybridizing with another algorithm may help UserUser perform well for new users. Exploring this issue further is left as future work.

Interestingly, we repeatedly see that for very small numbers of ratings the UserItemBaseline is our strongest algorithm, both in terms of accuracy, and recommendation quality. This suggests a switching strategy for live systems, in which the UserItemBaseline is deployed until a user has enough ratings, after which they are switched to a better recommender. The switching point between algorithms can be determined with our methodology. The only downside of this approach is that the baseline performed poorly on our non-monotonic metrics, which may effect users as they first join the system.

One possible reason that the SVD algorithm performs as well as it does is the regularization built into the algorithm. UserUser and ItemItem algorithms have no built in method for damping their predictions towards the mean when they do not have much data. The regularization in the cost function of the SVD model makes it so its predictions will gradually shift away from the baseline predictions as more relevant ratings are added. The downside of this, however, could be the higher popularity and lower spread seen by the SVD algorithm. The same regularization that helps avoid making mistakes due to too little data may also prevent the algorithm from taking occasional risks on more obscure movies.

One of the limitations of our work is that we only evaluated on one dataset. Future work should replicate our experiment on other datasets from different domains, or where ratings are collected in different ways. This will establish which of our results are true properties of the algorithms, and which results are specific to our dataset.

One important issue that our methodology cannot address is that we can only compare users who have made some minimum number of ratings (20 for our dataset) Unfortunately, this means that we do not know if there are differences between users who do and do not survive in the system. This is an unfortunate side effect of insisting on a constant test set throughout the evaluation. More work needs to be done to understand users who only make one or two ratings to understand how they are different and discover why they leave the system.

Nonetheless, we feel that our methodology will allow offline comparisons of many different cold start experiences. Our work has only scratched the surface of recommender algorithms. Future work can be done comparing these standard algorithms against newer algorithms such as those designed for cold start, or those designed to leverage extra non-rating information. More work should be done to understand how extra information about a user can effect cold start ratings. Our methodology can be used to measure this effect, and even establish a tradeoff between the extra information or extra initial ratings.

5. CONCLUSION

We find that recommending for new users is hard and none of our algorithms can reliably beat a simple baseline for users with very few ratings. Once the user has a small number of ratings we find that the SVD algorithm performs best. This result can be useful both in designing new recommender systems, and in designing novel recommendation algorithms for new users. We also present a methodology and list of metrics for understanding recommender behavior that can be used to extend our results to other algorithms and datasets.

6. ACKNOWLEDGEMENTS

The authors would like to express their gratitude towards our colleagues at GroupLens research for their feedback and support while developing this work. We would like to specially thank Michael Ekstrand for his important work on the Lenskit recommendation framework. This research was supported by the National Science foundation under grants IIS 08-08692 and IIS 10-17697,

7. REFERENCES

[1] A. Bellogín. *Performance prediction and evaluation in Recommender Systems: an Information Retrieval perspective*. PhD thesis, Universidad Autónoma de Madrid, Madrid, Spain, Oct. 2012.

[2] P. Cremonesi, F. Garzottto, and R. Turrin. User effort vs. accuracy in rating-based elicitation. In *RecSys '12*. ACM, 2012.

[3] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys '10*. ACM, 2010.

[4] S. Drenner, S. Sen, and L. Terveen. Crafting the initial user experience to achieve community goals. In *RecSys '08*. ACM, 2008.

[5] M. Ekstrand and J. Riedl. When recommenders fail: predicting recommender failure for algorithm selection and combination. In *RecSys '12*. ACM, 2012.

[6] M. D. Ekstrand, M. Ludwig, J. A. Konstan, and J. T. Riedl. Rethinking the recommender research ecosystem: Reproducibility, openness, and lenskit. In *RecSys '11*. ACM, 2011.

[7] S. Funk. Netflix update: Try this at home. http://sifter.org/~simon/journal/20061211.html, Dec. 2006.

[8] N. Golbandi, Y. Koren, and R. Lempel. Adaptive bootstrapping of recommender systems using decision trees. In *WSDM '11*. ACM, 2011.

[9] N. Golbandi, Y. Koren, and R. Lempel. Adaptive bootstrapping of recommender systems using decision trees. In *WSDM '11*. ACM, 2011.

[10] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, Jan. 2004.

[11] N. Lathia, S. Hailes, L. Capra, and X. Amatriain. Temporal diversity in recommender systems. In *SIGIR '10*. ACM, 2010.

[12] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[13] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *CHI EA '06*. ACM, 2006.

[14] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94*. ACM, 1994.

[15] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW '01*. ACM, 2001.

[16] G. Shani and A. Gunawardana. Evaluating recommendation systems. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 257–297. Springer US, 2011.

[17] M. C. Willemsen, B. P. Knijnenburg, M. P. Graus, L. C. Velter-Bremmers, and K. Fu. Using latent features diversification to reduce choice difficulty in recommendation lists. *RecSys '11*, 2011.

[18] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW '05*. ACM, 2005.