When Recommenders Fail: Predicting Recommender Failure for Algorithm Selection and Combination*

Michael Ekstrand and John Riedl
GroupLens Research
Dept. of Computer Science, University of Minnesota
{ekstrand,riedl}@cs.umn.edu

ABSTRACT

Hybrid recommender systems — systems using multiple algorithms together to improve recommendation quality — have been well-known for many years and have shown good performance in recent demonstrations such as the NetFlix Prize. Modern hybridization techniques, such as feature-weighted linear stacking, take advantage of the hypothesis that the relative performance of recommenders varies by circumstance and attempt to optimize each item score to maximize the strengths of the component recommenders. Less attention, however, has been paid to understanding what these strengths and failure modes are. Understanding what causes particular recommenders to fail will facilitate better selection of the component recommenders for future hybrid systems and a better understanding of how individual recommender personalities can be harnessed to improve the recommender user experience. We present an analysis of the predictions made by several well-known recommender algorithms on the MovieLens 10M data set, showing that for many cases in which one algorithm fails, there is another that will correctly predict the rating.

Categories and Subject Descriptors

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

Keywords

Recommender systems, evaluation, hybrid recommenders

1. INTRODUCTION

Hybrid recommender systems [1] are a well-known technique for harnessing the strengths of multiple recommenders to produce results that are more accurate or useful than those achieved by individual constituent recommenders. They have proven to be powerful means of trimming error; both top-placing systems in the NetFlix prize used hybrids of many algorithms [11].

One key development in the course of the NetFlix prize was *feature-weighted linear stacking* (FWLS) [11], a method of computing a linear combination of individual scoring functions where the blending coefficients are functions of user or item metadata rather than constants.

To win the NetFlix prize, however, contestants created large ensembles of over 100 recommenders. Such large algorithms are impractical to deploy in practice. Therefore, we need to know how to select the algorithms to include in ensembles for production deployments.

Additionally, while FWLS and other hierarchical methods [13] are important advances, they have to date provided little insight into *why* different recommenders are good or bad in particular circumstances. We seek to understand the particular strengths and weaknesses of various recommender algorithms. By understanding when individual algorithms perform well or poorly, we can select algorithms with complementary strengths and combine them in ways that maximize these strengths.

This work also has the potential to provide insight into what "personalities" various algorithms possess, a key prerequisite for selecting and optimizing algorithms for specific user needs. While past work has provided ways of reasoning about user needs in recommendation [7] and using offline evaluations [5] and user studies [12, 2] to evaluate algorithms in the light of user needs, there remains to be gained an understanding of the particular characteristics of the various algorithms in use in a general sense. Developing this understanding will enable future systems to be built more easily and with less costly testing, as algorithms can be selected based on general properties and then validated and tuned with user testing rather than involving users in every step.

To further these aims, we raise and attempt to address the following research questions:

RQ1 Do different recommender algorithms make different errors?

RQ2 Can we identify tractable features of users or items that make them easier or harder for individual algorithms to predict for?

RQ3 Does using these differences in recommendation errors to drive a hybrid recommender improve recommender performance?

The success of hierarchical hybridization strategies such as FWLS suggests that these hypotheses are true, but we

*The scripts to re-run the evaluations in this paper are available at `http://www-users.cs.umn.edu/~ekstrand/recsys2012/recsys-scripts.tgz`.

RecSys'12, September 9–13, 2012, Dublin, Ireland.

seek to demonstrate this more concretely and develop a transparent model for combining recommenders. Our goal is not to immediately produce a more accurate hybrid, but to gain insight into the relative performance of algorithms that can inform future recommender designs and deployments.

To address these questions, we analyzed prediction error of several algorithms using the LensKit recommender toolkit [3] on the MovieLens 10M data set in a cross-validation setup. While prediction accuracy is just one piece of the broader picture of recommender usefulness and suitability, it provides a tractable way to inspect the differing errors made by recommender algoritms.

2. RELATED WORK

Burke [1] provides an overview and taxonomy of hybrid recommender systems, outlining a variety of methods (including *switching* and *weighting*) that can be used to combine individual recommenders into a composite recommender. In this work, we focus primarily on switching hybrids, which pick which recommender to use in each situation and report its result alone; this framing provides the simplest way to study what causes individual algorithms to succeed or fail. More recent developments in hybridization include hierarchical methods like feature-weighted linear stacking [11].

McNee [7] argued for designing and evaluating recommender systems in the context of user needs, picking recommenders that provide specific characteristics that support user information needs. Later work [12] demonstrated that different algorithms that perform similarly on aggregate numeric measures of accuracy exhibit differing user-visible behaviors.

3. METHODOLOGY

We used the MovieLens 10M data set[1] (ML10M) and the LensKit recommender toolkit [3] for our experiments. We used LensKit's evaluation framework to partition the data set in a 5-fold cross-validation configuration. Users were partitioned into 5 sets; for each user in each partition, we randomly selected 20% of their ratings to be the test ratings for that data set, with the remaining ratings plus all ratings from users in the other partitions forming the training set.

We then ran five recommender algorithms on the data, and captured the predictions each algorithm made for each test rating. We used the following algorithms, choosing parameters based on prior results in the research literature and experience tuning LensKit for the MovieLens data sets [3]:

- Item-user mean, the item's average rating plus the user's mean offset with mild Bayesian damping to push means based on few ratings towards the global mean [4]. This algorithm was also the baseline for all others — if they could not make a prediction, the item-user mean was used.
- Item-item collaborative filtering [10] with a neighborhood size of 30 and ratings normalized by subtracting the item-user mean.
- User-user collaborative filtering [9] with a neighborhood size of 30, using cosine similarity over user-mean-normalized ratings [3]. In the predict stage, ratings were normalized by z-score [6, 3].
- FunkSVD [4, 8] with 30 features and 100 training iterations per feature.
- Lucene as a tag-based recommender. Since the ML10M data set contains tags for movies, we created a document for each movie containing its title, genres, and tags (repeating each tag as many times as it was applied). Recommendation were then computed as in item-item collaborative filtering, with item neighborhoods and scores computed by a Lucene `MoreLikeThis` query.

After running the recommenders, we processed each test set to discard all users with fewer than 10 test ratings (ultimately using 44,614 of the 69,878 users in ML10M) and splitting their test ratings into two sets: 5 ratings from each user went into a tuning set, and the remaining ratings stayed in the test set.

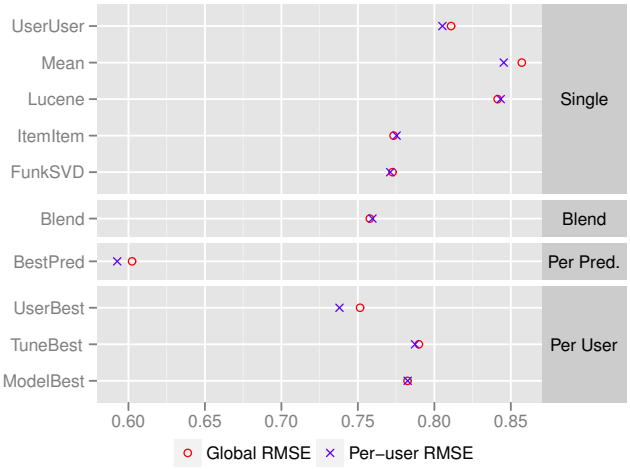

Figure 1: Algorithm accuracy

The "Single" section of Figure 1 shows the overall RMSE achieved by each of the recommender algorithms. "Blend" is the predictions produced by a linear model of the individual algorithms trained against the tuning set.

4. RESULTS

To evaluate and compare algorithm performance, we considered various types of measurements: getting a prediction "right" (within some threshold, such as 1/2 star), being better in absolute error than another algorithm, and providing better accuracy overall for all of a user's predictions. We also tested per-item accuracy and have not yet found any particularly interesting effects; therefore, we only present the per-prediction and per-user results.

4.1 Getting Predictions Right

Algorithm	# Good	% Good	Cum. % Good
ItemItem	1044371	52.23	52.23
UserUser	166008	8.30	60.53
Lucene	90018	4.50	65.03
FunkSVD	53313	2.67	67.70
Mean	21617	1.08	68.78
Unexplained	624291	31.22	100.00

Table 1: Cumulative good predictions (0.5 star)

[1] `http://grouplens.org/node/73`

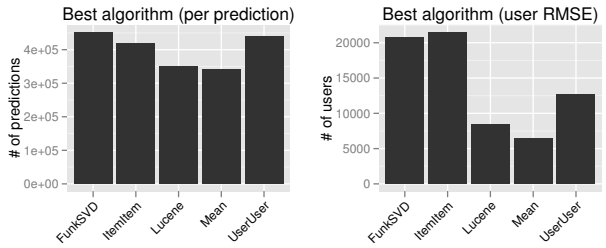

Figure 2: Distribution of best algorithms

Table 1 shows the cumulative "good" predictions for the algorithms we tested. A prediction is considered "good" if it is within 0.5 stars; the table is computed by first picking the algorithm that has the most good predictions. The remaining algorithms are selected and computed by picking the algorithm which has the most good predictions that no prior algorithm has correctly made and adding it to the table. So ItemItem predicts 52% of ratings correctly, UserUser an additional 8%, and so on.

This result provides initial confirmation of H1: algorithms differ in which predictions they get right or wrong. ItemItem gets the most predictions right (52%), but the other algorithms correctly make various predictions until 69% of all test ratings are predicted correctly by at least one algorithm.

This result is robust to higher thresholds; using a threshold of 1.0 stars for good prediction scales the ItemItem hit count up and the other hit counts correspondingly down, but does not change the relative ordering of algorithms.

The existence of differences in the errors made by individual algorithms is further substantiated by the results shown for "BestPred" in Figure 1. This is the RMSE achieved by switching hybrid recommender that uses an oracle to select the best predictor for each individual prediction; the left side of Figure 2 shows how often each algorithm provided the best prediction. This shows that, if we can perfectly predict the best predictor to use, there is room for substantial improvement in error. It therefore provides a lower bound on the error of a switching hybrid comprised of the algorithms in our experiment.

When selecting algorithms to deploy in an ensemble recommender, it is not necessarily desirable just to pick the ones that perform the best. If two algorithms are highly correlated in the errors they make, failing in the same cases, then including both of them will likely not provide much benefit. In selecting algorithms, we look for the following criteria:

- Unique benefit — individual algorithms should contribute unique benefit with respect to the other algorithms in the ensemble.
- Distinguishability — it should be possible to figure out how to blend the algorithms or to select which one to use.
- Tractability — given two algorithms with similar benefit, prefer algorithms that are less expensive to operate.

In general, we found FunkSVD and ItemItem to be highly correlated; the absolute error produced by each has a correlation of $\rho = 0.888$, and ignoring one accrued the greatest benefit to the other. This corroborates their showing in Table 1; if we regenerate the table using FunkSVD first, ItemItem is the last recommender to be picked before Mean. FunkSVD and ItemItem had the two highest coefficients in the Blend model, however, suggesting that they are contributing unique signal beyond what can be picked up in a threshold analysis.

4.2 Comparing by User

For each user, we determined the algorithm that gave the best RMSE on their test ratings. The right half of Figure 2 shows the distribution of best algorithms. As with optimizing individual predictions, no one algorithm is the winner, suggesting room for intelligent selection of algorithms that will perform better for different users and providing further confirmation that different algorithms do make different mistakes.

To test H2, we tried several regressions to predict relative algorithm performance. Using the log of the user's rating count, their average rating, and the variance of their ratings, we were able to build a logistic model that showed significance and some predictive power for ItemItem being the best algorithm if we ignore FunkSVD (since they have such similar behavior, most users for whom FunkSVD was best ItemItem as their second best algorithm). Table 2 shows the coefficients of this regression; the probability of ItemItem being the best predictor increases with the number of ratings the user has provided and their rating variance, and decreases as their mean rating increases. Relative to the other predictors, therefore, ItemItem does better when more ratings are available (not surprisingly), and is also mildly boosted by low average ratings and high variance in user ratings. Thus H2 is also confirmed — these features correlate with one algorithm beating the others.

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-2.8966	0.1045	-27.73	0.0000
log10(count)	1.6090	0.0227	70.80	0.0000
mean	-0.1047	0.0224	-4.66	0.0000
var	0.2818	0.0203	13.85	0.0000

Table 2: "Item-item best" prediction model

The ROC curve, computed by holding out 20% of the users as a test set, is shown in Figure 3; its area is 0.68, suggesting that this regression may also be useful from a practical standpoint for identifying when ItemItem is a good choice of recommender.

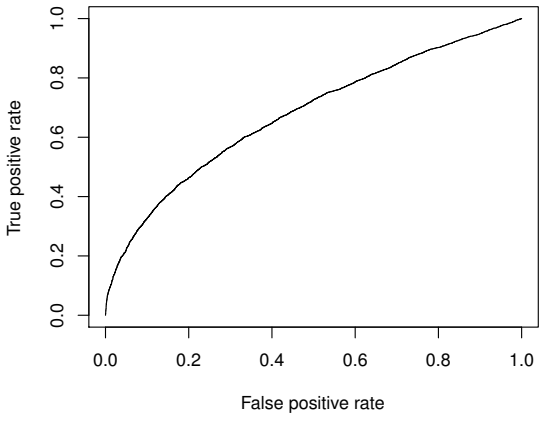

Figure 3: ROC curve for predicting "ItemItem is best"

Building an immediately useful hybrid from these findings has so far been challenging. In Figure 1, the "Per User"

section shows the RMSE achieved by three different per-user algorithm switching strategies. *UserBest* uses the algorithm with the best RMSE for that user; it shows a lower bound on the error of per-user switching hybrids optimizing for user RMSE. It shows some improvement over all single algorithms and over the linear blend, particularly when looking at per-user error. It therefore seems that it should be possible to achieve accuracy improvements through intelligent selection of algorithms on a per-user basis.

TuneBest attempts to pick the best algorithm using the predictions for each user's 5 tuning ratings, then apply that algorithm to the remaining test ratings. Using this approach — which does not provide any insight into why the algorithm is selected — achieves an accuracy a bit worse than the best single algorithms (ItemItem and FunkSVD). Its per-user RMSE is slightly better than any single algorithm.

ModelBest uses the 5 probe ratings to train a logistic regression predicting whether either UserUser or Mean will be better than ItemItem for that user based on the log of the number of items the user has rated and an interaction term between that and the variance of their ratings. In turning the modeling from Table 2 around and using it to decide which recommender to use, we found these features to be most useful; variance on its own was not significant and did not make a noticeable contribution in addition to the item count and interaction terms, and mean was similarly unhelpful in improving the regression's power. The regression is then thresholded to decide whether ItemItem or UserUser is used; the threshold was chosen to produce a ratio of UserUser to ItemItem choices that kept in line with the other two models. The resulting predictor beats UserUser alone, but does not improve upon ItemItem. It does, however, beat *TuneBest*; we infer from this that user features are useful for hybridization, but there is still work to do to make this algorithm an actual improvement over the current state of the art.

5. CONCLUSION AND FUTURE WORK

In our experiment, we found that recommenders do indeed fail on different users and items, thus confirming H1. We have also identified user features predicting relative algorithm performance and achieved some success building a hybrid around them, providing preliminary confirmation of H2 and H3, but more work needs to be done to make the algorithm an improvement over the state of the art and to develop a deeper understanding of what makes the various recommenders succeed or fail.

Immediate future work involves continuing to look for features that will help us to select the appropriate recommender to use, as well as investigating blending approaches using hierarchical regressions (similar to FWLS), with the goal of understanding what it is that makes particular algorithms work well, where their individual weaknesses are, and how to combine them into an effective ensemble.

User studies and qualitative investigation of the items and users themselves will likely be helpful in further elucidating the specific behavior of each algorithm. So far, our work has focused only on generic statistics of users and items in the rating set; seeing what actual items are being mispredicted and collecting user feedback on erroneous predictions or bad recommendations will hopefully provide further insight into how the algorithms behave.

Systematic investigation of recommender failures has potential to improve both our understanding of the workings and characteristics of recommender algorithms and our ability to successfully deploy them. By understanding when recommenders fail, we can know better what ones are most likely to work well in particular situations and be more effective at designing and deploying novel recommender systems finely tuned to the particular demands of their domains and users.

Acknowledgements

Our colleagues in GroupLens Research, particularly Tony Lam, Shilad Sen, and Aaron Halfaker, have provided invaluable assistance in this work. We also gratefully acknowledge the support of the National Science Foundation under grants IIS 10-17697 and 08-08692.

6. REFERENCES

[1] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, Nov. 2002.

[2] M. D. Ekstrand, P. Kannan, J. A. Stemper, J. T. Butler, J. A. Konstan, and J. T. Riedl. Automatically building research reading lists. In *RecSys '10*, pages 159–166. ACM, 2010.

[3] M. D. Ekstrand, M. Ludwig, J. A. Konstan, and J. T. Riedl. Rethinking the recommender research ecosystem: reproducibility, openness, and LensKit. In *RecSys '11*, pages 133–140. ACM, 2011.

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

[5] A. Gunawardana and G. Shani. A survey of accuracy evaluation metrics of recommendation tasks. *J. Mach. Learn. Res.*, 10:2935–2962, 2009.

[6] J. Herlocker, J. A. Konstan, and J. Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr.*, 5(4):287–310, 2002.

[7] S. M. McNee, J. Riedl, and J. A. Konstan. Making recommendations better: an analytic model for human-recommender interaction. In *CHI '06 Extended Abstracts*, pages 1103–1108. ACM, 2006.

[8] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDD Cup and Workshop 2007*, Aug. 2007.

[9] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *ACM CSCW '94*, pages 175–186. ACM, 1994.

[10] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *ACM WWW '01*, pages 285–295. ACM, 2001.

[11] J. Sill, G. Takacs, L. Mackey, and D. Lin. Feature-Weighted linear stacking. *arXiv:0911.0460*, Nov. 2009.

[12] R. Torres, S. M. McNee, M. Abel, J. A. Konstan, and J. Riedl. Enhancing digital libraries with TechLens+. In *ACM/IEEE JCDL '04*, pages 228–236. ACM, 2004.

[13] A. Umyarov and A. Tuzhilin. Improving rating estimation in recommender systems using aggregation- and variance-based hierarchical models. In *RecSys '09*, pages 37–44, New York, New York, USA, 2009. ACM.