Network-based recommendation: Using graph structure in user-product rating networks to generate product recommendations

David Cummings <davidjc>
Ningxuan (Jason) Wang <nwang6>

1 Introduction

1.1 Abstract

Given a set of users and their reviews of items, recommendation systems generate ranked lists of items to recommend to individual users. In this paper, we demonstrate two such systems: one that uses projections of the bipartite user-item network to generate recommendations, and, as a basis for comparison, a straightforward probabilistic model that does not take advantage of graph structure. Testing by holding out certain reviews from the data, we find that the graph-based method outperforms the model-based method in predicting the held-out reviews.

1.2 Previous work

Over the years, recommendation systems have become a highly-developed field, with various algorithms to generate product recommendations based on customers' similarities to one another. Some previous work uses explicit trust networks to augment existing recommendation systems[2][3]; however, for this project, we focus on generating an implicit trust network as a projection of a bipartite graph of users' ratings of items. This process is most closely paralleled by work presented in *Bipartite network projection and personal recommendation* (Tao Zhou et al.)[1]. This paper creates a directed, weighted graph among users based on products they have purchased in common, then uses the graph to create lists of products to recommend to them. Previous work has found that recommendations from this graph-based approach can outperform both a global ranking of most-popular products and a model-based algorithm.

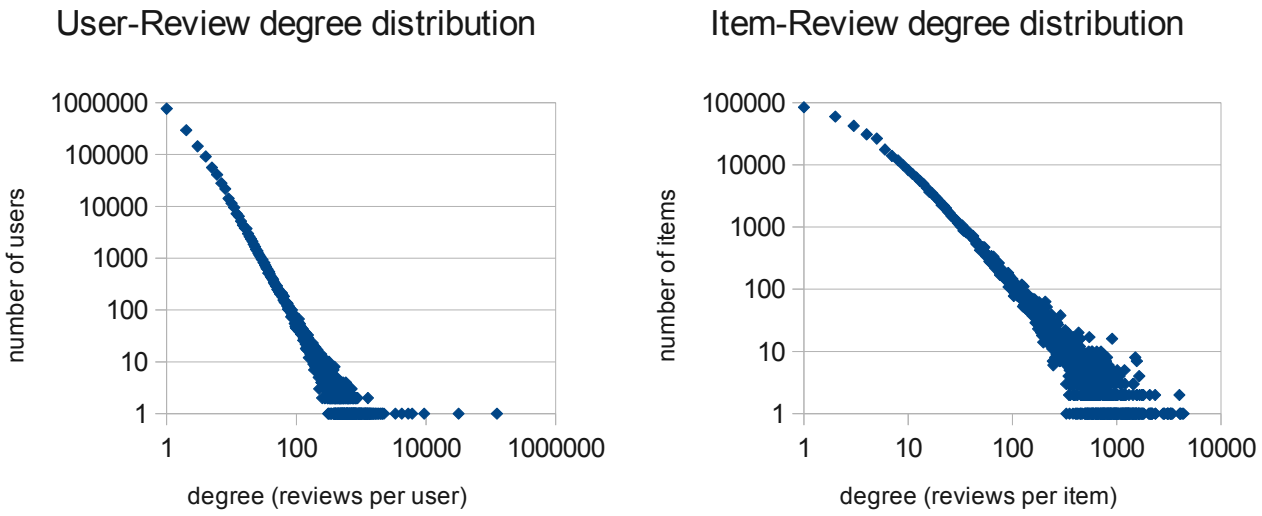
The Zhou et al. *Bipartite* paper is our major inspiration for this project; however, its experiments are significantly limited. For example, the user base is comprised of only 943 people. We aim to explore the use of similar algorithms on a much larger data set, in order to provide more conclusive proof of whether this graph-based approach can improve upon existing recommendation systems. Also, Zhou et al. treated connections between users and products in a binary fashion, not allowing for users to express some range of like or dislike for a product. Since we can take into account a range of ratings from 1 to 5, we can approach this problem with greater granularity. Finally, we can also characterize the nature of the network that is generated through this process.

2 Data

In some ways, our approach is most strongly constrained by the data that is available to us. Two of the papers we reviewed focused on the use of explicit trust networks, as seen in Epinions, where users can explicitly form networks among themselves that represent how much they trust each others' opinions. [2][3] However, since such data sets are generally small, and larger real-world data sets of social networks are generally not available, we focus on generating an implicit network instead.

Our data set is the Amazon product co-purchasing metadata as hosted on the SNAP website. <http://snap.stanford.edu/data/amazon-meta.html>

This data contains almost 400,000 products with at least one review, and contains nearly 8 million reviews on these products. For each product entry, we get a list of reviews, which contains unique user IDs as well as the ratings that users gave to the product. It is exactly from this kind of bipartite graph, with users on one side and products on the other, that we can generate a graph to show the implicit connections between users who rate the same item similarly to one another.



Analysis of the data shows that both the degree distribution over users and degree distribution over items follow power-law distributions. Using the methods described in [4], the empirical estimate for the scaling parameter is approximately 1.64 for users, and 2.46 for items. The average degree for users (that is, reviews per user) is 15.79, and average degree for items (that is, reviews per item) is 4.089. Average degree for users is biased high due to the existence of a few users with artificially high degree: this is further explored in section 4.

3. Model-Based Recommendation

3.1 Algorithm

As a simple type of collaborative filtering, the model-based recommendation system treats each user as a sparse vector, with separate features representing their reviews/purchases of each item. In this representation of the problem, one approach is to generate the odds that the input user will buy or positively review each item. Given a user u, for each item I, we can think of two classes or sets of users, U^{I+} and U^{I-}. U^{I+} represents the set of users who liked I (that is, they bought or reviewed it highly), and U^{I-} represents the rest of users. Since we cannot generate probabilities directly, we can instead calculate the odds of a user U being among the sets U^{I+} and U^{I-}:

$$\frac{P(u \in U^{I+} \mid u)}{P(u \in U^{I-} \mid u)}$$, which by Bayes' Rule is equivalent to $$\frac{P(u \mid u \in U^{I+})P(u \in U^{I+})}{P(u \mid u \in U^{I-})P(u \in U^{I-})}$$

If we use log-odds instead of simply odds, and let u be a vector in R^n representing purchases or reviews of all *n* items, where each review can be considered independently, then we generate the following probabilistic interpretation:

$$\text{log-odds} = \log\left(P\left(u \in U^{I+}\right)\right) - \log\left(P\left(u \in U^{I-}\right)\right) + \sum_{j=1}^{n} \log\left(P\left(u_j \mid u \in U^{I+}\right)\right) - \log\left(P\left(u_j \mid u \in U^{I-}\right)\right)$$

The prior probabilities $P\left(u \in U^{I+}\right)$ and $P\left(u \in U^{I-}\right)$ are given simply by $|U^{I+}|/|U|$ and $|U^{I-}|/|U|$, respectively. The probabilities for each index of the vector u are given by:

$$P\left(u_j \mid u \in U^{I+}\right) = \frac{\sum_{v \in U^{I+}} v_j}{|U^{I+}|}$$, or, using Laplace smoothing, $$P\left(u_j \mid u \in U^{I+}\right) = \frac{1 + \sum_{v \in U^{I+}} v_j}{2 + |U^{I+}|}$$

Since usually $|U^{I+}|$ is orders of magnitude smaller than $|U^{I-}|$, we can pre-calculate $\text{sumv}(j) = \sum_{v} v_j$ and let $\sum_{v \in U^{I-}} v_j = \text{sumv}(j) - \sum_{v \in U^{I+}} v_j$, for a significant speed-up of performance.

The result is a standard Bayesian model, which, given a user, generates odds scores as a consistent metric to compare between all items. To generate a list of recommended items for a user, we simply order all items by their odds scores. In practice, items that the user has already purchased or reviewed always appear at the top of this list, but since we do not want to re-recommend redundant items, we will remove such items from the list.

3.2 Incorporating ratings

To bring ratings into this model, we introduce a set of weights to double- or triple-count reviews with different numbers of stars. For instance, one mapping was $\{1 \rightarrow 0, 2 \rightarrow 0, 3 \rightarrow 1, 4 \rightarrow 2, 5 \rightarrow 3\}$. This allows the u_j values to range from -2 to 2, instead of 0 or 1. We tested various different mappings, but none significantly improved upon the binary 0/1 case, for reasons further explored in section 6.2.

4 Projections from the Bipartite Graph

4.1 Graph projection

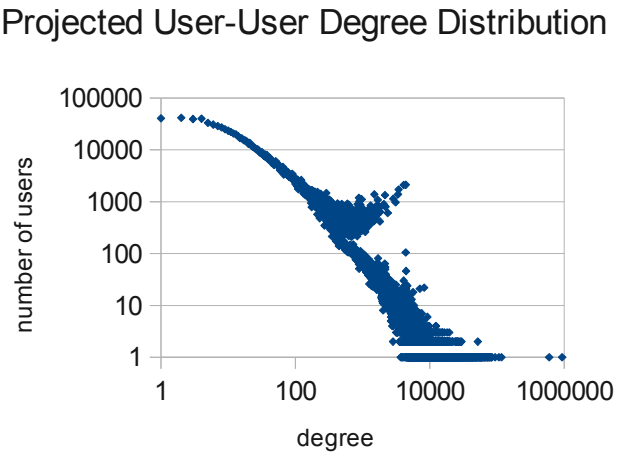
In a bipartite network of two sets of nodes where only nodes from different sets are allowed to be connected, a network projection may be constructed for one set of nodes through their connections with nodes from the other set. To prevent loss of information, the projection contains weighted edges that reflect the degree of connections between two nodes. In the data, we have two sets of nodes: one for items and one for users. Each review from a user to an item is represented as a weighted edge between them. For the case of creating an unweighted projection, we can simply generate edges between any two users that have a reviewed item in common, and edges between any two items that have a reviewer user in common. However, this projection does not carry much information. We would prefer that edges between nodes tell us how similar the nodes are-- for instance, if two users both reviewed the same five items, we would want a higher-weight edge between them than the case that two users reviewed five items each but only one item in common.

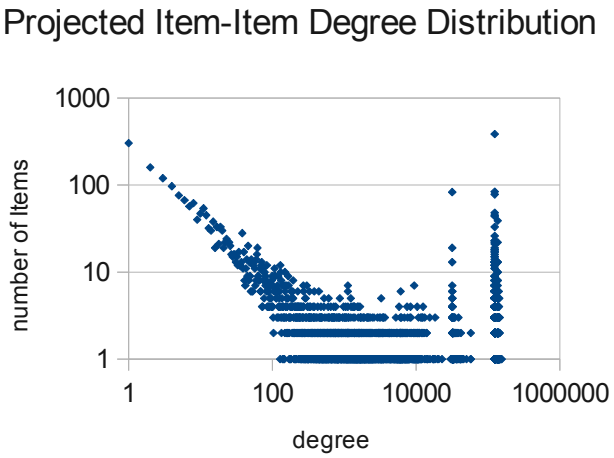
To achieve this, we use the following algorithm. Consider two nodes, u and w, both among the same set in the bipartite graph-- either users or items. Let the set of nodes $v_1 ... v_n$ be all the neighbors of u in the original graph-- the v_i's are from the opposite set from u and w. The weight of the directed edge uw is given by the following formula:

$$\text{weight}(uw) = \frac{1}{degree(u)} \sum_{i=1}^{n} \frac{1\{\text{edges } uv_i \text{ and } wv_i \text{ exist}\}}{degree(v_i)}$$

If the resulting sum is zero-- that is, the intersection of the sets of neighbors of u and w is empty-- then we do not create an edge. If we think of u and w as users, and v_i's as items, we can describe many useful features of this sum. As users u and w have more items in common, the sum increases; this is very nearly what we want, but it requires normalization. We first normalize by dividing by the degree of u. If u has reviewed many items, of which the intersecting v_i's are only a small subset, then we want to have a lower-weight edge; conversely if u has only reviewed exactly those v_i's which are in common with w, then we want to have a higher-weight edge. Then, we normalize by the degrees of the v_i items themselves. Items which have reviews from many users (ex. national bestseller books) are not as strong a signal for predicting similarity as less-popular ones (ex. books from niche genres).

Below is the degree distribution for our projected user-user graph and projected item-item graph. These projected graphs generally follow the expected power-law distribution; however, there are some significant irregularities. Upon investigation, these are mainly caused by a small subset of users who appear to have reviewed incredible numbers of items-- the most, upwards of 140,000. These users appear to be Amazon's own product ratings. Although we considered removing these users, we decided to leave them in the data set. This way, using the graph-based recommendation methods described below, we could be assured of discovering more items within a few hops of the network.





5 Graph-Based Recommendation

5.1 Projections in practice

In practice, we found that projections from the bipartite graph are much too large to hold in memory. Our best efforts still resulted in an estimated 16GB memory footprint for the user-user projected graph, and an estimated 500GB memory footprint for the item-item projected graph, neither of which could be handled by our machines.

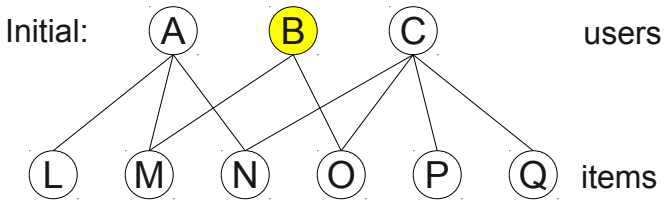
To circumvent this, rather than ever generating a full projection on either subgraph, we rely on generating only the local projected network structure when needed. For instance, given a particular user, we can find their neighbors in the bipartite projected graph by a simple two-step process, iterating through the items the user has reviewed, then the other users who also reviewed the same items. Since we must also iterate through the items that the user's neighbors have reviewed, the entire process can be thought of as a three-step iteration through the graph, originating at the input user.

5.2 Algorithm

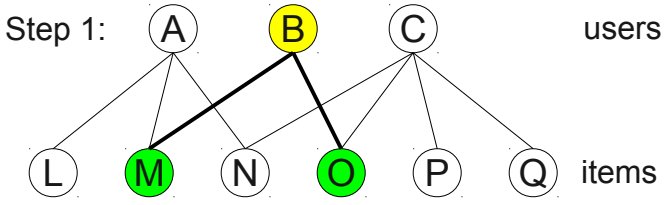
Given a user as input, we generate the local projected user-user graph as described above, by iterating over all items a user has already reviewed, and then finding all 'neighbor' users that also reviewed the same items. Then, for each of the items reviewed by these neighbor users, we have chosen to generate a score based on the sum of their edges, normalized by the degree of each neighbor node. In other words, given an input user u, for each item v_i reviewed by each neighbor user w, we compute a ranking score:

$$\text{score}(u, v_i) = \sum_{w \in \text{neighbors}(v_i)} \text{weight}(uw) \cdot \frac{1}{\text{degree}(w)}$$

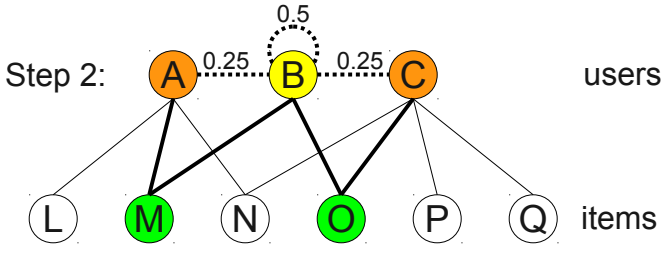
This choice makes the graph-based recommendation algorithm essentially equivalent to a three-step random walk through the graph, originating at the input user. As an example, consider the following, in which we have users A, B, and C, and items L, M, N, O. P, and Q:



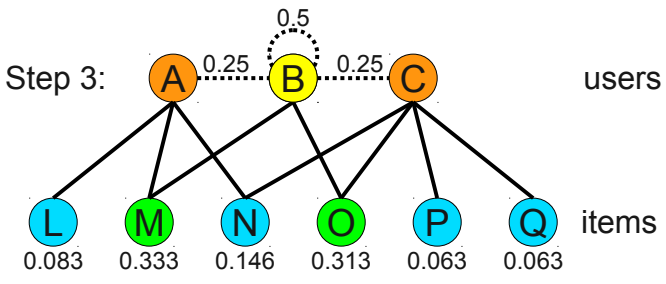
In the initial state, we consider user B to be the input on which we want to predict.



In step 1, we expand out to the items that user B has reviewed.



In step 2, we generate the weights for each edge in the projected user-user graph, based on the items in common. (B has a self-edge of weight 0.5 here.)



In step 3, we generate scores for each item according to the formula above. The score for N, for example, is (0.25 / 3 + 0.25 / 4), representing the contributions from A and C, respectively.

~~score(B, M): 0.333~~
~~score(B, O): 0.313~~
score(B, N): 0.146
score(B, L): 0.083
score(B, P): 0.063
score(B, Q): 0.063

Having generated scores for each of the discoverable items; that is, those within three hops of the input user, we can then rank them in descending order. (The resulting list for the example on the previous page is shown at left.) The items already reviewed by the input user always appear at the top of this list, so to avoid re-recommending items the user has already reviewed, we remove them from the recommendations.

5.3 Incorporating ratings

We also tried to use a user's rating of an item to adjust the ranking of the item among all the recommended candidates. We hypothesized that ratings might provide extra information regarding the quality of an item, as the fact that a user simply reviews item doesn't necessarily mean that the user likes it or will recommend it to other users. To add in specific rating scores, while the rest of the method remains the same, the variation we've conducted to the existing graph-based algorithm is to multiply the ranking score in the above calculation by rating scores in the range of 1 to 5, i.e.,

$$\text{score}(u, v_i) = \sum_{w \in \text{neighbors}(v_i)} \text{weight}(uw) \cdot \frac{1}{\text{degree}(w)} \cdot \text{rating}(uw)$$

5.4 Explainability

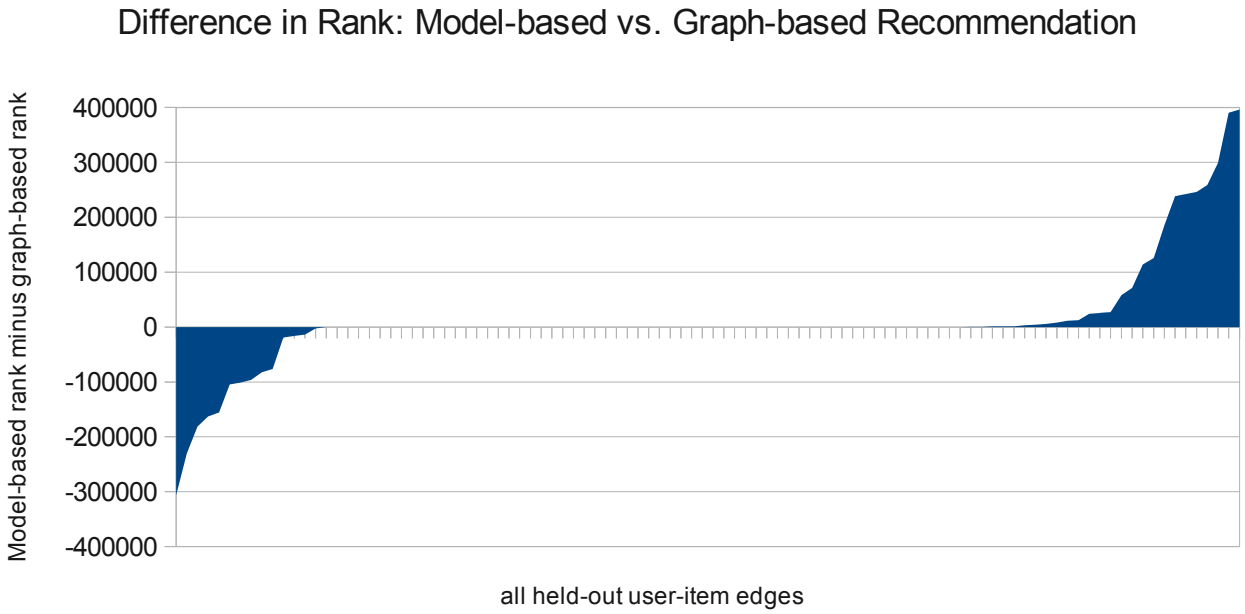
This graph-based algorithm has an advantage over the model-based algorithm in that the reasons for its results are more intuitively understood and explained. Given a particular user-item recommendation, one can trace exactly which neighbor users and other items contributed to the ranking score. In contrast, since the model-based algorithm necessarily uses all items and all users, it is more difficult to determine which of the users or items most influenced the ranking score. Since the ability to understand and debug recommendation systems is an important priority in real-world applications, this gives some advantage to the graph-based algorithm independent of its results.

6 Testing and Results

6.1 Testing Procedure

To test the performance of both methods, we randomly picked 100 users who have rated no less than 2 and no more than 10 items. For every user, we randomly took an item off the list of items rated by the user as held-out data. Then, we entered the user into each recommendation algorithm. In the resulting list of candidate items, we looked for where the held-out item appeared on the list of recommended items. The higher the item appeared on the recommendation list (that is, the higher its score and the lower its rank on the list), the better the method performs on this particular user-item pair.

6.2 Comparison of Results



This figure shows the difference in rank over all held-out user-item pairs we tested. Area above the zero line represents cases where graph-based outperformed model-based recommendation; area below represents cases where model-based outperformed graph-based, and the majority of cases (the middle 59%) show that both found the held-out item in the top position.

Our results showed that our method of graph-based recommendation consistently outperforms our method of model-based recommendation. 75% of the time, the graph-based method got the held-out item in the top position of the generated candidate list for a user, while the model-based method only achieved this in 59% of cases. In every case that the model-based method correctly predicted the held-out item in the top position, the graph-based method did as well.

In 15% of cases, the model-based method gave better results than the graph-based method; that is, the held-out item appeared higher in the recommendation list for model-based than for graph-based. These cases were almost entirely due to the graph-based method not reaching the held-out item in its local, three-step exploration from the test user, thus failing to place the held-out item on the recommendation list at all.

In 26% of cases, the graph-based method gave better results than the model-based method. The length of the recommended-items list for the graph-based method was consistently shorter than the total number of items, so when the graph-based method *did* find the held-out item in its local exploration of the graph, it tended to be in a better position.

We tried a few different approaches for incorporating ratings information: in the graph-based approach, reweighting edges based on number-of-stars; in the model-based approach, double-counting or triple-counting reviews with high ranking. However, none of these modifications resulted in significant improvement to the resulting recommendations.

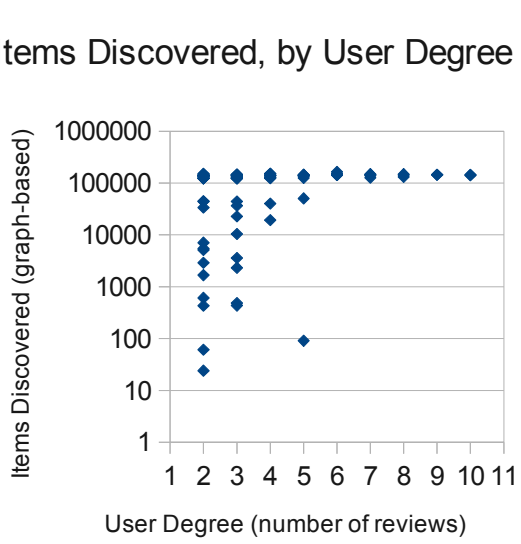
6.3 Insights

We were surprised that either method succeeded in having such a high rate of perfect success, placing the held-out item in the top position on the recommended-items list. Upon examination of the data, this appears due to two factors. First, many items appear in grouped sets, for example the three Lord of the Rings Extended Edition DVD's. Most users who have bought any two of these DVD's are highly likely to have bought the third, so our held-out-item approach will very often predict the 'missing' DVD.

Second, and more commonly, Amazon often duplicates user ratings among equivalent products. For instance, if a book is available in both hardcover and paperback editions, and a user rates the paperback edition at 5 stars, then Amazon also creates a 5-star rating for the hardcover edition as well. In effect, it appears that the user has bought both the paperback and hardcover editions, and rated them separately as 5 stars. We considered cleaning the data to remove this effect, but it appears that Amazon's rating duplications are not consistent: not every rating is always duplicated among equivalent products. (This might be caused, for instance by only duplicating once the paperback edition of a book is released, but leaving the original hardcover reviews as they were.) Thus, we left the data unmodified, with the expectation that in practice, a real recommendation system would be able to remove such duplicates from recommendation lists based on items a user had already bought.

Further inspecting the few cases in which the model-based method performed better than the graph-based method, we noticed the comparative biases of the two recommendation systems. For an example, one user had bought two documentaries about dancing, and one CD of classical music, and we held out one of the dancing documentaries. The graph-based algorithm, in its three-step exploration of the local graph, found many more users and many more items in the genre of classical music, and recommended other classical music CDs higher than dancing documentaries. The model-based algorithm, noting that the dancing documentary was a rarer and thus stronger signal upon which to classify the user, placed the held-out dancing documentary higher in its recommendation list. Thus, in cases where a user has both niche and popular interests, the graph-based method tends to place more generally popular items higher on the list, while the model-based method tends to place more niche items higher on the list.

For graph-based recommendation, we noted that both the number of reachable items and the quality of the ranking scores were largely a function of degree of the input user; that is, the number of items they had already reviewed. Thus, as a user buys and reviews more items, the algorithm both finds a greater number of items overall, and also has more information with which to generate accurate recommendations.



Though we had hoped that the incorporation of weights for different numbers of stars would improve our recommendations, in retrospect it is logical that a binary reviewed/not reviewed approach is sufficient. If we are trying to predict whether a user will buy or review some item in the future, the strongest signal is simply whether they have bought or reviewed similar items in the past. Star-ratings provide insight into users' opinions of items within their sphere of interest, but to discern the sphere of interest itself, what is most useful is simply their purchase history.

7 Conclusion

Although there do exist more sophisticated model-based collaborative filtering techniques, we were encouraged that a graph-based approach could outperform at least one reasonable method of model-based recommendation. Further work could definitely be used in comparing other recommendation systems to our graph-based method; for instance using some measure of vector similarity rather than the Bayesian probabilistic interpretation, or using other machine learning techniques to find clusters of users rather than restricting our view only to neighbors in the graph.

One particular limitation of our graph-based method bears further discussion. In some cases, the graph-based method failed to find the held-out item in its local graph exploration. As shown in the graph on the previous page, most cases found acceptably large numbers of items, but some were restricted to less than 100. One possible fix for this would be to simply append all the items not already on the recommendation list, perhaps in descending order of general popularity, in order to at least list all items using the graph-based algorithm. Another promising, though computationally expensive, approach would be to allow the graph-based method more than three steps of exploration. With a sufficient number of steps, the algorithm would incorporate nearly all items in its list. That said, either of these approaches is largely an academic exercise, since real recommendation systems will likely only use the top few results in the recommended-items list.

8 References

[1] Tao Zhou, Jie Ren, Matus Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. The American Physical Society, 2007.

[2] Cai-Nicolas Ziegler, Jennifer Golbeck. Investigating interactions of trust and interest similarity. Decision Support Systems, 2006.

[3] Hao Ma, Dengyong Zhou, Chao Liu, Michal R. Lyu, Irwin King. Recommender systems with social regularization. Web Search and Data Mining (WSDM), 2011.

[4] Aaron Clauset, Cosma Rohilla Shalizi, M. E. J. Newman. Power-law distributions in empirical data. SIAM, Rev. 51.