On the Value of Random Opinions in Decentralized Recommendation

Elth Ogston, Arno Bakker, and Maarten van Steen

Department of Computer Science
Vrije Universiteit
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
{elth, arno, steen}@cs.vu.nl

Abstract. As the amount of information available to users continues to grow, filtering wanted items from unwanted ones becomes a dominant task. To this end, various collaborative-filtering techniques have been developed in which the ratings of items by other users form the basis for recommending items that could be of interest for a specific person. These techniques are based on the assumption that having ratings from similar users improves the quality of recommendation. For decentralized systems, such as peer-to-peer networks, it is generally impossible to get ratings from all users. For this reason, research has focused on finding the best set of peers for recommending items for a specific person. In this paper, we analyze to what extent the selection of such a set influences the quality of recommendation. Our findings are based on an extensive experimental evaluation of the MovieLens data set applied to recommending movies. We find that, in general, a random selection of peers gives surprisingly good recommendations in comparison to very similar peers that must be discovered using expensive search techniques. Our study suggests that simple decentralized recommendation techniques can do sufficiently well in comparison to these expensive solutions.

1 Introduction

Many successful recommendation systems are based on the idea of *collaborative filtering (CF)* [6]. In collaborative filtering, two users who have liked the same things in the past are assumed to like similar things in the future. A user's preference for a new item, such as a movie or a book, can therefore be predicted by examining ratings of that item made by users that previously had similar opinions. Traditionally, CF algorithms operate on complete knowledge, that is, the ratings of all users are known in one location. This makes it easy to discover the similar users needed to make the predictions. This, however, also makes these algorithms hard to employ in a decentralized context, where not all users' ratings can be available at all locations. In this paper we investigate how well CF algorithms operate on partial knowledge; that is, how many similar users does an algorithm actually need to produce good recommendations for a given user, and how similar must those users be.

We consider the example of a network of millions of interconnected personal video recorders. In the near future, these devices will not just be able to receive and record programs from satellite or the ether but also over the Internet. As a result, they will make more content available to the user than ever before, creating the need for a recommendation system that helps the user to decide what to watch. To build a decentralized recommendation system for these recorders we need to answer the question: what knowledge they need to achieve good quality recommendations for their users and how to obtain it?

In the context of personal video recorders, there are a number of related tasks for which recommendation systems can be used. Recommendation information can be used to augment an electronic program guide by adding a predicted rating to each item. Alternatively, it can be used to rank the items currently available for a user to watch (i.e., present the user with a Top-N of programs to watch). Both of these tasks require calculating accurate recommendations for an entire set of items. We conjecture that in a network that offers large amounts of content a simpler recommendation task might be sufficient. In this situation users are more likely to be interested in a list of some programs which they are certain to enjoy, rather than knowing ratings for all programs, or identifying the absolute best of the currently accessible programs. Simply discovering *some* good programs creates an easier recommendation task. Firstly, it requires an algorithm that need only accurately rate programs the user will find good, rather than having to accurately predict ratings in the entire rating range. Secondly, an absolute ordering of programs is not required. Finally, only a sufficiently large number of good programs must be identified, it does not matter if some are missed. This task is more suitable for a decentralized setting in which a Top-N recommendation can never be fully correct due to the fact that not all programs or ratings data are available to each user.

The main contribution of this paper is that we show for the well-known MovieLens data set [10] that sufficiently good recommendations can be made based on the ratings of a relatively small number of random users. We believe this to be an important result in light of the various attempts to port CF solutions to decentralized systems. Based on our experiments we conjecture that simple solutions are good enough.

The remainder of the paper is organized as follows. In Section 2 we present background on collaborative filtering algorithms and our system model. Section 3 describes our experiments studying the effects of the number and type of users on recommendation quality for the MovieLens data set. We present conclusions in Section 4.

2 Background and System Model

The amount of information made available through computer networks often means that people need to be selective about what content they spend their time on. This is especially true in future video-on-demand systems where so many videos are available that it is infeasible to even browse through them all. Given such an overabundance of options, recommendation systems can help people make choices by aggregating opinions on what others have found, in their experience, to be valuable. In the simplest case such recommendations can take the form of a single joint rating which is given to all users. A group of people can, however, have very different opinions about the value of an item. More advanced algorithms thus provide personalized predictions by filtering the opinions upon which a recommendation is made. This is done on the principle that users that have exhibited similar opinions on items in the past are likely to continue to have similar opinions on new items [7, 1].

At an abstract level the problem of collaborative-filtering considers a set of N users, $U = \{u_1, \ldots, u_N\}$ and a set of M items $X = \{x_1, \ldots, x_M\}$. Each user provides ratings, taken from a set of possible values, V, the rating scale, for a subset of the items in X. These ratings form an $N \times M$ user-item matrix, R, where the entry $r_{i,j}$ is the rating of user u_i for item x_j, or empty if that rating is unknown. The basic recommendation task is to predict a rating value for a given empty element $r_{i,j}$ based on the known values in R. This is done by means of a prediction function, f, where $f(R, i, j) \mapsto V$.

The prediction function usually performs two tasks. First, it *selects* rows from the matrix which correspond to data which is most likely to accurately predict $r_{i,j}$. Second, it *aggregates* the information in these rows to calculate an actual value for $r_{i,j}$. When the user-item matrix is used as the input to f, the rows selected correspond to users that are similar to user u_i. This is called *user-based* collaborative filtering. The item-user matrix, $R^\top$, can also be used as the input to the prediction function, thus calculating $f(R^\top, j, i)$. In this case the rows selected correspond to data items that have received similar ratings to the item x_j. This is called *item-based* collaborative filtering [8]. Exactly how f performs the selection and aggregation tasks is the subject of many studies on which heuristics lead to the best recommendations [2, 1, 8].

In our study, we assume an architecture in which each user has a personal networked video recorder by which he or she rates content. These personal devices can exchange gathered ratings with the devices of other users via the network, and use them to make personal predictions to their respective users using a given prediction function f. As the network grows, it becomes infeasible to distribute all ratings, i.e. the full matrix R, to all recorders. The video recorder for user u_i must therefore base its predictions on a submatrix of R denoted R_i. In this paper, this submatrix R_i will consist of u_i's own ratings and the ratings of a specific set of other users, called u_i's *peer group*, as described in Section 3.1.

Following the above, there are five factors that can influence the quality of the predictions in decentralized algorithms. In addition to (1) the size and (2) composition of the peer group of each user, the quality of prediction will be affected by the properties of function f itself. In particular, it depends on f's (3) selectiveness in choosing rows of the ratings matrix to consider, (4) the sophistication of the method by which aggregation is performed, and (5) whether the function considers user-based or item-based correlations. We study the effects of these five factors. As we shall see, the differences between simple and sophisticated approaches are small enough to raise the question of whether we need sophisticated algorithms at all.

3 Experiments

We present an analysis study in which we examine the effects of the five factors identified in the previous section on the quality of decentralized peer-to-peer recommendation algorithms. We first introduce our methodology and the data set we consider in Section 3.1. Next, we study the effect of peer-group size and composition in isolation from other factors, by using rudimentary prediction functions in Section 3.2. The analysis is repeated with sophisticated prediction functions from the well-known CF algorithms in Section 3.3. Section 3.4 analyzes the suitability of the well-known algorithms for the task of identifying just some good items to recommend to a user (as opposed to the absolute best available). Section 3.5 repeats the analysis of Section 3.4 for the PocketLens peer-to-peer recommendation algorithm proposed in [4].

3.1 Experiment Methodology

For our experiments we organize the users' personal video recorders into a peer-to-peer overlay. The personal recorder for user u_i will make its predictions on the submatrix R_i, consisting of the ratings of the peers it is connected to in the peer-to-peer overlay and its own ratings. To test the influence of the five factors, we vary the number and type of peers u_i is connected to and the prediction functions used.

The users' personal devices are organized into overlays as follows. Each of the nodes in the overlay stores the ratings data for a single user, that is, the node for user u_i stores the ith row of R. Nodes are connected by directed *links* to other nodes, called their *neighbors*, thus forming a peer-to-peer overlay network. The set of links of each node is called its *neighbor cache* which has a size c. Only the ratings data stored at a node's neighbors is available as input to the prediction function f. Note that because links are directed, more than c nodes can use the ratings of any particular user.

In addition to varying c, we consider two contrasting peer-to-peer overlay topologies. In the first, neighbor caches contain links to random nodes, creating a *random overlay*. In the second, neighbor caches contain links to the nodes to which a node is most similar, given a similarity function d for rating data, creating a *best-neighbors overlay*. Given the base assumption of collaborative filtering that ratings from similar users provide the best quality recommendations, these two cases represent a worst-case and a best-case scenario, respectively.

A best-neighbors overlay can be constructed in a decentralized fashion, for instance by using a gossiping protocol such as Cyclon/Vicinity [11]. Nodes exchange their rating data and compute the similarity to the other peers using the given similarity function d. By remembering the best candidates so far, while continuing to exchange preferences with other peers, each node will eventually fill its neighbor cache with the nodes most similar to it. As running such a protocol is more expensive in terms of network usage than discovering random peers, the random and best-neighbor overlays also represent the cheap and expensive solution respectively. We use Pearson's correlation using significance weighting [2] as the similarity function to define best-neighbors overlays in all our experiments. Following Herlocker *et al.*'s conclusions for the MovieLens data set we set the significance weighting parameters to *minCommonItems*=2 and *maxCommonItems*=100 for all experiments. Negative correlations are not considered.

We evaluate the performance of each algorithm for differing values of c and the two topologies using the MovieLens data set [10]. This data set consists of 100,000 ratings, on a scale of 1 to 5 stars, of 1,682 movies made by 943 users. Each user rates at least 20 items, but the data set is still sparse: 94% of the user-item space has no rating. For evaluating the performance we partition this data into a training set and a test set. The training set forms the matrix R, constituting the users' ratings used to populate the

Table 1. Summary of the ratings in the training and test set

Ratings	Count in Training Set	Count in Test Set
1*	5568	542
2*	10375	995
3*	24721	2424
4*	30858	3316
5*	19048	2153
Total	90570	9430

nodes in the overlay. The test set consists of 10 randomly chosen movies per user as summarized in Table 1.

Each experiment consists of constructing the peer-to-peer overlay using the training set and then attempting to make predictions for the 9430 withheld (user,movie) pairs. Our experiments thus measure algorithm performance in making 9,430 predictions based on 90,570 ratings. In particular, each node will attempt to predict the rating its user u_i would give to the 10 withheld items based on its R_i matrix, consisting of the ratings of the user and those of its neighbors. The resulting predictions are compared to the 9430 actual ratings in the test set using several metrics. The experiments are conducted using the CoFE collaborative filtering engine [5] that implements centralized user-based collaborative filtering. We extended CoFE to support item-based recommendation and the rudimentary recommendation algorithms.

We use the following metrics to evaluate predictions. Initially, we consider the *mean absolute error (MAE)* metric [9]. Given a list L of H user-item pairs (u_{i_1}, x_{j_1}), ..., (u_{i_H}, x_{j_H}), a corresponding list A of actual user ratings for these user-item pairs $r_{i_1 j_1}$, ..., $r_{i_H j_H}$ with $r_{i_k j_k} \in V$, and a corresponding list P of *unrounded*[1] predictions of the ratings for the user-item pairs $r^*_{i_1 j_1}$, ..., $r^*_{i_H j_H}$ with $r^*_{i_k j_k} \in V$, the mean absolute error is given by:

$$MAE = \frac{\sum_{k=1}^{H} |r^*_{i_k j_k} - r_{i_k j_k}|}{H}$$

Associated with MAE is the *coverage* metric which measures what fraction of the predictions attempted actually returned a result. Predictions for user u_i and movie x_j may fail because, for example, none of the user's neighbors actually rated x_j.

Mean absolute error is a rough estimation of the overall accuracy of an algorithm. It considers errors in any part of the ratings scale to be equal. For our stated purpose of identifying a set of some good items, however, errors at the top end of the scale become more important than errors elsewhere in the scale. In order to measure recommendation accuracy more precisely we will use the standard information-retrieval metrics *recall* and *precision* in Section 3.4. Recall and precision compare, for a particular query q, the set of selected items S_q, which were returned in reply to q, and the set of relevant items T_q, which contains all items that are correct replies to q. Recall measures the fraction

[1] Users rate and see predictions as integer values, but for the calculation of prediction-performance metrics the unrounded predictions returned by the recommendation algorithms are used.

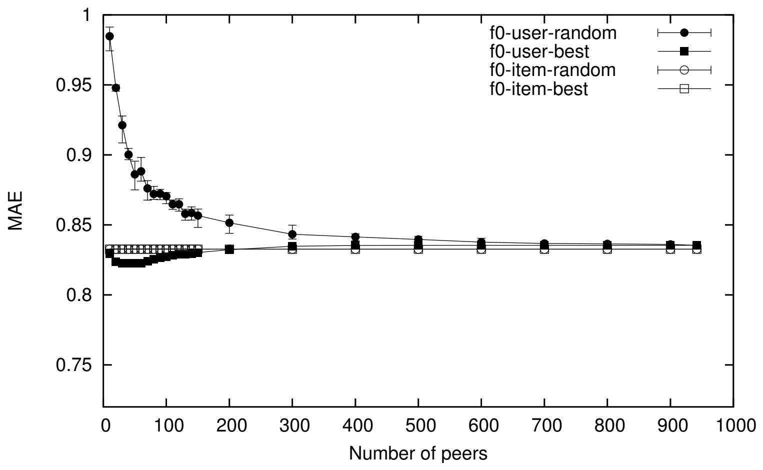

Fig. 1. Recommendation quality for the four rudimentary algorithms. For the random overlays, values are averaged over three runs. The vertical errorbars show the minimum and maximum value obtained in these three runs. Note that the y-axis starts at 0.72.

of correct replies to q that actually appeared in the selected set: $|S_q \cap T_q|/|T_q|$. Precision measures, for the selected set, the fraction of correct replies it contains: $|S_q \cap T_q|/|S_q|$.

3.2 Rudimentary Recommendation Algorithms

In this section we establish a baseline for the effect of varying peer-group size and composition. We also look at the underlying differences between user-based and item-based algorithms. We consider a rudimentary prediction function, f_0. This function performs no selection on its input matrix. To predict a rating for user u_i of an item x_j it simply computes the mean value of the relevant column in the input matrix. More specifically, when given a user-item matrix R, f_0 calculates a user-based prediction by computing the average of the rating for item x_j as given by the nonempty entries in column j (that is, from users who have rated x_j). When given an item-user matrix, f_0 calculates an item-based prediction by computing the average over the values for items rated by user u_i.

Figure 1 shows recommendation quality in terms of MAE versus the size c of a user's peer group (i.e., its neighbors). We consider four different inputs to f_0. In the user-based cases, the input consists of the submatrix R_i as constructed from the peer group. In the item-based cases, the input is $R_i^\top$, the transposition of R_i. The x-axis shows the effect of having more or less ratings data available to f_0; the random and best-neighbors variants show the effect of the quality of the information available.

The first thing to note when analyzing Figure 1 is the scale of the y-axis. MAE values range from 0.82 to 0.99. An MAE of 1 means that predictions are, on average, one star off from the actual ratings given by users. From this perspective, a difference in MAE 0.17 is fairly insignificant, and we could say that all four algorithms perform fairly well. It is interesting to note that the best reported MAE for an algorithm on this data set is 0.72 [8].

Because item-based predictions are based on a user's own ratings, and f_0 does not select among these ratings, the item-based algorithms simply recommend the average of a user's ratings for all predictions. Therefore, the results are independent of the network type and peer-group sizes. For group sizes of over 200 these item-based algorithms actually produce the lowest MAE of all algorithms, 0.83. Such a small MAE indicates that users tend to give similar ratings to all the movies they rate. Table 1 shows that, in general, this is fairly true for this data set: the average rating over the whole data set (training+test) is 3.53 and using this value for all predictions gives an MAE of 0.94.

For the user-based best-neighbors algorithm, in which the peer-to-peer network performs user selection, we see that smaller group sizes, in which less information is available, produce better results. This indicates that some users are better predictors of each other than others, and therefore selection can have a positive effect. It also shows the disadvantage of using averaging, by which mediocre opinions can drown out good ones, in the aggregation function. The fact that this algorithm performs better than the others only for groups sizes under 200 indicates that the number of very similar neighbors per peer is fairly small. The small difference between good performance and bad again indicates that all peers are similar enough to provide acceptable predictions.

We also calculated the coverage values for the four algorithms. The item-based algorithms are always able to make predictions for all items. For the user-based best-neighbors algorithm coverage was about 1.0 for all group sizes showing that nodes' best neighbors practically always had at least one rating for their movies in the test set. This could indicate that nodes with a large numbers of ratings tend to be chosen more often as best neighbors. For random groups coverage was as low as 0.69 for a group size of 10 but rose quickly to 0.98 or higher for group sizes over 100.

Given the overall small differences in MAE it could be said that even small groups of randomly chosen neighbors produce sufficiently good recommendations. This leads to the interesting conclusion that we need only consider small groups of users, and their exact composition may not be that critical. Note, however, that the difference in terms of MAE between the best performing algorithm to date and the trivial algorithm that always predicts the average rating is only two tenths of a star. This makes MAE an unintuitive metric for measuring recommendation performance. The trivial algorithm does not provide user-specific recommendations nor does it accurately predict which movies are very good or very bad. Therefore, a metric judging this algorithm's performance should clearly indicate that it performs poorly. For now, we continue to use MAE as it is a standard metric that, although subtly, gives a decent indication of the general performance difference between algorithms. We return to the issue of performance metrics in Section 3.4.

3.3 Sophisticated Recommendation Algorithms

The experiments with a simple prediction function, f_0 provided some initial insight into the effect of decentralization on recommendation quality. By removing the selection task from the prediction function we were able to examine the situation where all peer selection (if any) is done by the peer-to-peer protocol. In this section we reintroduce (additional) selection by the prediction function. Letting the prediction function make the selection is to be preferred provided quality does not suffer much, as it is a local operation on the matrix R_i rather than a search operation on the network.

Advanced prediction functions use two types of selection: (1) choosing from the matrix those ratings of a user that are of interest, and (2) judging the relative importance of the ratings chosen. This second selection is accomplished by assigning weights to the input ratings. For this experiment we use a prediction function f_1 that was designed by Herlocker *et al.* [2] to optimize user-based prediction accuracy. This prediction function performs both types of additional selection. First, when asked to make a prediction for item x_j for user u_i, it selects from the matrix R_i supplied by the peer-to-peer overlay the ratings of x_j as made by the z users most similar to u_i, thus creating a rating vector $\vec{r}$. To calculate the similarity it uses the same function as the best-neighbors overlay (Pearson's correlation with significance weighting).

Second, when making the actual prediction for item x_j it weights the rating of the z users most similar to u_i with their similarity value. In short, in addition to any selection by the peer-to-peer network, f_1 limits the set of opinions to consider to z and weights those opinions based on just how similar they are in absolute terms. In this experiment we use a parameter set shown by Herlocker to be optimal for user-based predictions on this data set [2], in particular, z is set to 60 and the Pearson significance weighting parameters are set to *minCommonItems*=2 and *maxCommonItems*=100, as before.

Figure 2 shows data for the experiment from the previous section repeated with prediction function f_1. The first two plots in Figure 2 (using the dark symbols) show results for user-based prediction using f_1 on random and best-neighbors overlays. For comparison, the f_0 results for user-based prediction on the best-neighbors overlay are also shown. For both overlays, f_1 improves predictions over f_0. For the random overlay, f_1 bases predictions only on the more similar users in the random input set. For small

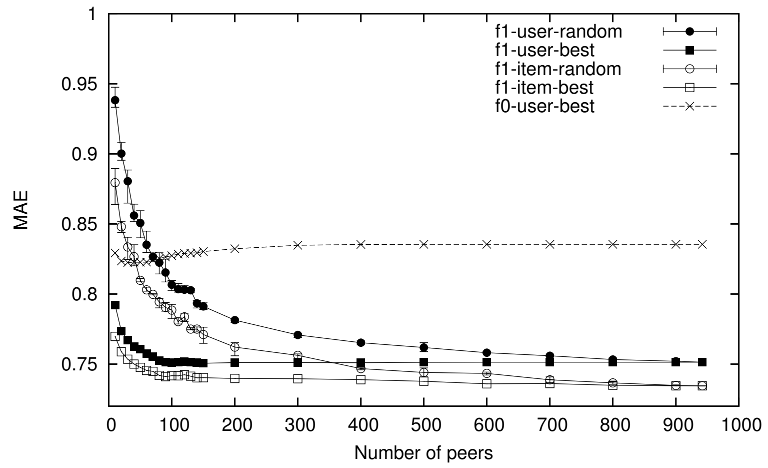

Fig. 2. Recommendation quality for the sophisticated algorithms. For the random overlays, values are averaged over three runs. The vertical errorbars show the minimum and maximum value obtained in these three runs. Note that the y-axis starts at 0.72.

group sizes this results in very little data with which to make predictions, but it is very effective for larger group sizes. The additional ability to weight these inputs relative to each other allows f_1 on a random overlay to outperform f_0 on a best-neighbors overlay for group sizes over 100. Using a best-neighbors overlay to provide f_1 with higher quality input improves recommendation, especially for smaller groups. The difference between the f_1 user-based best-neighbors and the f_0 user-based best-neighbors curves shows how valuable weighting input results can be.

The second two plots in Figure 2 (using the open symbols) show results for item-based prediction using f_1 on random and best-neighbors overlays. Interestingly, even though f_1 was not designed as an item-based prediction function, the results improve slightly on those for user-based predictions. This indicates that there may be more similarity between items than between users in the data sets, though the imprecision of the MAE metric precludes hard conclusions. It should be noted that the item-based best-neighbors algorithm is in fact a hybrid item-based/user-based approach, with user-based selection taking place within the peer-to-peer network and item-based selection taking place within the prediction function.

We also examined the coverage of the algorithms using f_1. This was slightly lower than the coverage using f_0, especially for predictions based on random groups of users, but still above 0.93 for all algorithms for a group size of 100 or more.

Overall, this experiment shows that for a more sophisticated prediction function making item-based predictions in a best-neighbors network produces the best MAE values. The differences between the algorithms are, however, fairly small. In general, it appears that performing selection within the prediction function, even out of a small amount of random input, is more effective than performing selection within the peer-to-peer network. This again indicates that peer-to-peer networks that provide users with small amounts of random ratings information from other users might be a sufficient basis for decentralized-recommendation algorithms.

3.4 Identifying Good Programs

Our measurements of mean absolute error in Section 3.3 give an indication of the relative quality of our recommendation methods. MAE, however, provides only a general measure of overall quality. As described in the introduction, in the context of a personal video recorder we are most interested in being able to produce accurate recommendations for movies at the five-star end of the ratings scale. To investigate recommendation behavior in more detail, we employ the standard information-retrieval metrics precision and recall (see Section 3.1), as follows.

For the user-item pairs in the test set, we separate the list of returned predictions P, according to prediction value, into the sublists P_{1*}, P_{2*}, P_{3*}, P_{4*} and P_{5*}. We also divide the actual ratings of the test set in a similar manner into A_{1*}, A_{2*}, A_{3*}, A_{4*} and A_{5*}. Thus, P_{5*}, for instance, contains all of the five-star predictions ($r^*_{i_k j_k} = 5$) and A_{5*} contains all the actual five-star ratings in the test set ($r_{i_k j_k} = 5$). The user-item pairs (u_{i_k}, x_{j_k}) that correspond to the predictions and ratings in P_{5*} and A_{5*} can be viewed as a selected-items set S_{5*} and a relevant-items set T_{5*}, respectively, for the query "find all five-star movies for each user". This allows us to calculate precision and recall per

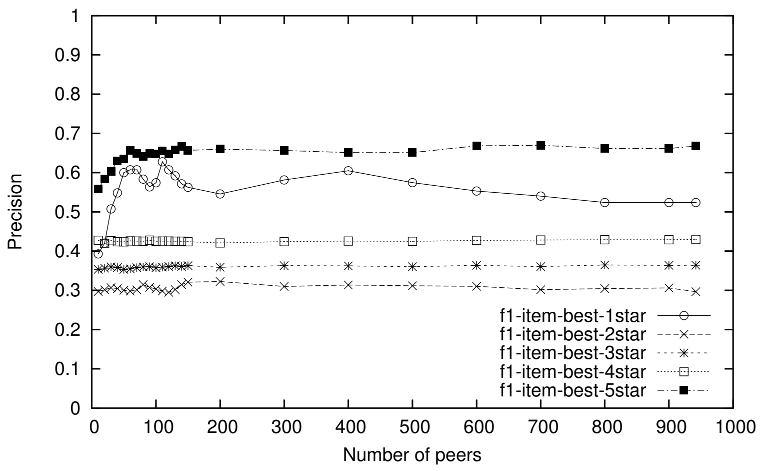

Fig. 3. The 1–5 star precision of f_1 item-based for differing numbers of similar peers

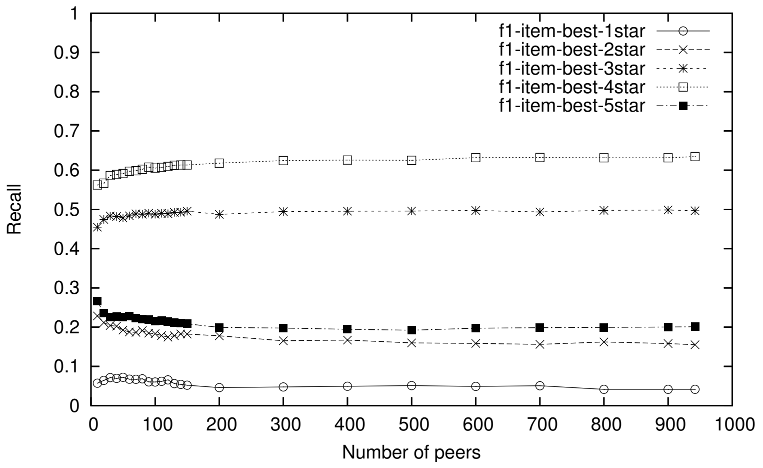

Fig. 4. The 1–5 star recall of f_1 item-based for differing numbers of similar peers

rating value. Note that precision and recall are computed only over the predictions that could actually be made (see the discussion of the coverage metric in Section 3.1).

We use these new metrics to analyze the best performing according to MAE, the f_1 item-based best-neighbors algorithm, in Figures 3 and 4. The figures establish that items with different values in the ratings scale are not, in fact, treated equally by the algorithm. In general, precision is higher for items at the extremes for the rating scale, while recall is higher for items in the middle of the rating scale. This tradeoff between recall and precision is not unusual, increasing precision requires an algorithm to be more picky about the replies it chooses, which tends to decrease recall.

Figures 3 and 4 indicate that this algorithm tends to predict extreme ratings values only when the rating is fairly clear, and otherwise chooses a safer prediction in the middle of the scale. This is in line with the fact that the algorithm does aggregation by taking a weighted average of ratings. Items with mixed reviews should thus tend to be given mediocre predicted ratings, while items which everyone liked or disliked can be given extreme ratings.

Fortunately, for the task of recommending some good items, we are most interested in having a high precision for five-star items, as is the case. Five-star recall is less important, as long as it is high enough for a query for five-star items to produce some answers. Five-star recall for the test set is 20 percent for this algorithm at a group size of 200. An average user rates about 21% of movies with five stars, so in the collection of 1682 movies there are about 357 movies he will like. If we use the recall for the test set as an estimate for recall on the whole data set, the algorithm recalls 20% of these 357 enjoyable movies, yielding roughly 71 movies to watch. At 1.5 hours per movie, this translates to 107 hours of viewing pleasure. A video-on-demand system is likely to give access to even more content.

Comparing the performance of the four f_1 algorithms from Section 3.3, we find that using a best-neighbors overlay instead of a random one results in higher precision values, especially for one-star and five-star items and groups smaller than 200. In particular, one-star precision is up to 24 percentage points higher and five-star precision is 11 percentage points higher for the item-based best-neighbors algorithm. For the user-based best neighbors algorithm these values are 28 percentage points and 8 percentage points, respectively. This higher precision does not come at the cost of a lower recall, which remains practically the same for these extremes. Recall for the other values increases up to 6 percentage points. Figure 5 and 6 show precision, respectively recall for f_1 item-based using a random overlay.

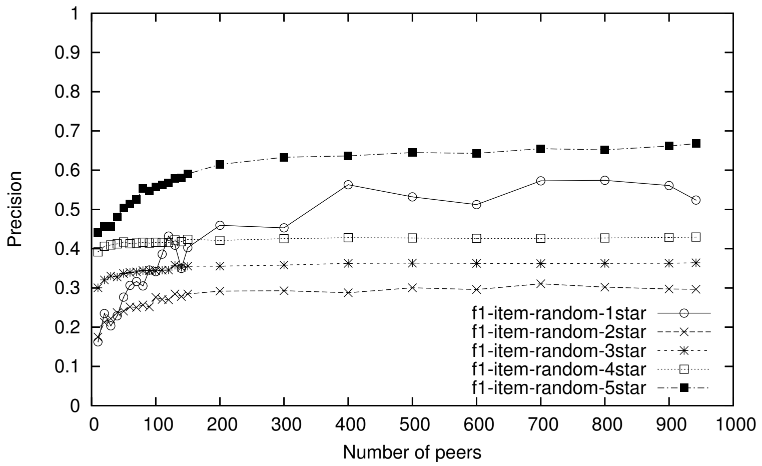

Fig. 5. The 1–5 star precision of f_1 item-based for differing numbers of random peers. Values are averaged over three runs.

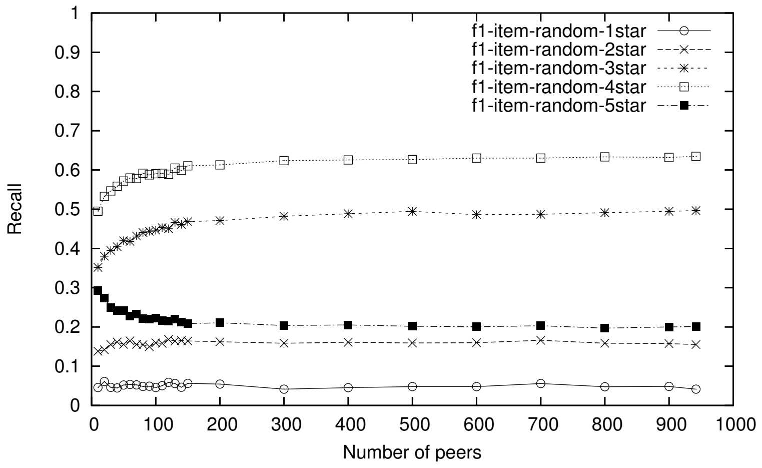

Fig. 6. The 1–5 star recall of f_1 item-based for differing numbers of random peers. Values are averaged over three runs.

In general, making item-based in place of user-based predictions results in higher precision for five-star ratings. The item-based best-neighbors algorithm improves this precision by an average of 8 percentage points over the user-based best-neighbors algorithm for all group sizes. This is, however, at the cost of recall, which we found to be up to 10 percentage points lower for five-star predictions. Meanwhile recall on four-star and three-star items improves slightly, indicating that item-based prediction gives higher five-star precision because it has a greater tendency to give predictions from the middle of the scale. One-star precision decreases around 8 percentage points, its recall went down by 3 percentage points on average.

Precision results may be affected by the fact that there are a very small number of one-star items in the data set, due to the way user opinions were gathered. A data set with a more even distribution of ratings might result in slightly worse precision results. On the other hand, in a video-on-demand network, users are also likely to watch and rate a majority of items at the upper end of the scale.

For the task of recommending good items, predictions that are slightly off will probably not be noticed, while predictions that are very wrong could undermine a user's faith in the recommendation system. A list of good items to watch should ideally contain only five-star items. A user will probably also be glad to watch four-star items, but will be annoyed to find one-star or two-star items in the list. We thus introduce a further metric, *adapted top precision (ATP)*, which measures precision for the query "find five-star movies for each user" but also considers a four-star prediction a valid answer. Formally, $ATP = |S_{5^*} \cap (T_{4^*} \cup T_{5^*})|/|S_{5^*}|$. Figure 7 shows ATP for each of the four f_1 algorithms. All four algorithms perform well on this metric. Even the worst performing algorithm at the smallest group size still returns 77% four- or five-star items when asked for five-star items.

Overall, the experiments in this section confirm the conclusions we made in Section 3.3. The item-based best-neighbors algorithm generally produces the best

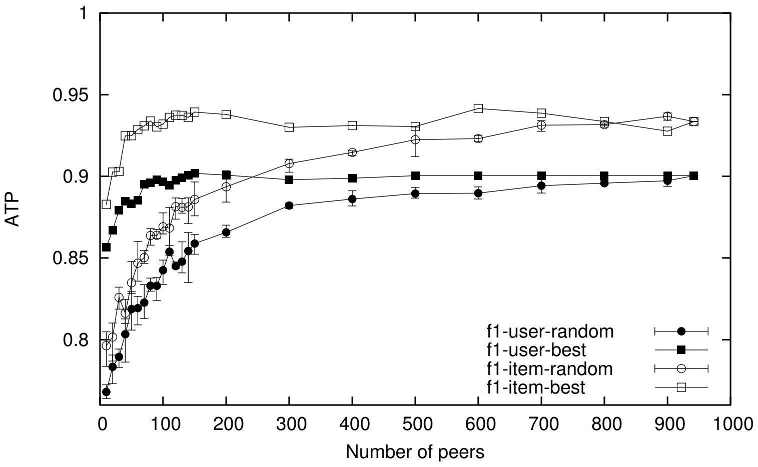

Fig. 7. Adapted top precision for differing numbers of peers. For the random overlays, values are averaged over three runs. The vertical errorbars show the minimum and maximum value obtained in these three runs. Note that the y-axis starts at 0.76.

recommendations, especially from the perspective of simply finding good items. But again, results for item-based prediction on a random network are not that much worse. We also still find that increasing group size improves results, but that small groups can still produce good recommendations. As an example, when given the task of finding a set of items which are predicted to have five-star ratings, for a group size of 200, the item-based best-neighbors algorithm returns 425 five-star items, 179 four-star items, 26 three-star items, 9 two-star items, and 5 one-star items. The item-based random algorithm returns 444 five-star items, 202 four-star items, 53 three-star items, 14 two-star items and 10 one-star items.

3.5 Comparison to PocketLens

PocketLens is an item-based prediction algorithm designed specifically for a peer-to-peer setting. In [4], Miller *et al.* evaluated the performance of PocketLens using several different underlying overlays: a Gnutella-based random overlay, a best-neighbors overlay, and two Distributed-Hash Table-based overlays. The performance of each overlay was tested using a non-standard version of the MovieLens data set with twice as many items. They found the best MAE performance was achieved by the random overlay and with sufficient coverage (already 90% for groups of just 65 peers). Their measurements thus support our conclusion that random overlays can be used for decentralized CF algorithms. We show it holds for the standard MovieLens data set and for user-based algorithms, and when measured using more expressive metrics. In addition, we provide a detailed examination of why random overlays can be used.

To examine how PocketLens performs on the new task of recommending some good items we repeat the analysis from the previous section for this algorithm. Figure 8 compares the adapted top precision measure for the PocketLens prediction function on a

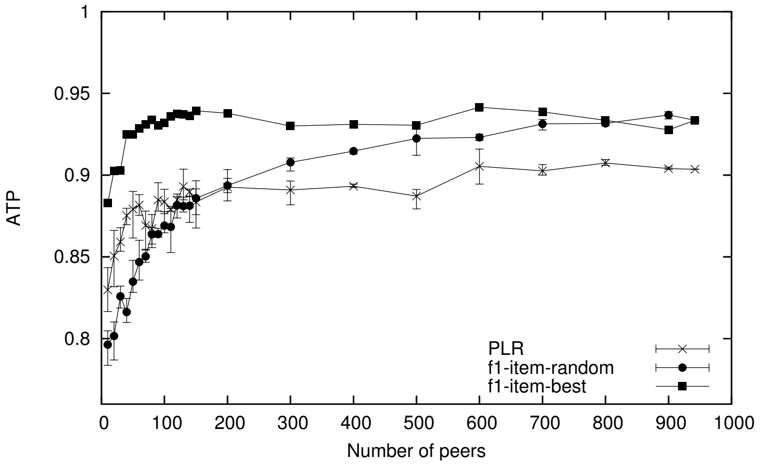

Fig. 8. Adapted top precision for PocketLens compared to f_1 item-based predictions. For the random overlays, values are averaged over three runs. The vertical errorbars show the minimum and maximum value obtained in these three runs. Note that the y-axis starts at 0.76.

random overlay with the most similar algorithm we studied, the item-based f_1 prediction function on a random overlay. We also plot the results for item-based f_1 on a best-neighbors overlay, the algorithm which produced the best ATP above.

The PocketLens prediction function performs better than f_1 on the random overlay for group sizes smaller than 300, but fails to improve its predictions for larger group sizes. In a more detailed exploration of this behavior we found that PocketLens produces high five-star precision values in this range, but that this is at the cost of low five-star recall. For a group size of 100, for instance, PocketLens has a five-star recall value of 0.07 while the item-based f_1 algorithm has a recall of 0.22. Overall, we found that PocketLens produces lower recall for all ratings except for four-stars, indicating that it has a much greater tendency to guess that items will be rated four-stars, which is the rounded average ratings value for the data set (see Section 3.2).

4 Conclusions

Our experiments with the MovieLens data set bring us to the conclusion that the neighbors from which a peer receives ratings data may not be critical to the quality of peer-to-peer recommendations. That is, neither the number of neighbors nor selecting the most similar really matters. If a peer has access to ratings from a few hundred, randomly chosen other nodes, we see that reasonable recommendations can be obtained. This is a notable result in light of the various attempts to port existing centralized collaborative-filtering algorithms to peer-to-peer networks. We conjecture that there may be no need to incur the added costs of structuring a network in order to improve recommendations.

Whether these results can be generalized remains to be seen. The quality of recommendations provided by any algorithm is highly dependent on the quality of the input ratings data, which in turn, strongly depends on the rating behavior of users [3]. To this end, we plan to extend our experiments to other data sets. There may be circumstances in which selecting best neighbors is worth the trouble. For example, our current experiments show that quality of recommendation does improve if neighbors are *not* selected randomly, albeit by a small amount. Thus, although the results presented in this paper are promising, further research is needed in order to truly substantiate our claims.

References

1. BREESE, J., HECKERMAN, D., AND KADIE, C. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. Tech. Rep. MSR-TR-98-12, Microsoft Research, Redmond, WA, USA, May 1998.
2. HERLOCKER, J., KONSTAN, J., AND RIEDL, J. An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms. *Information Retrieval 5*, 4 (Oct. 2002), 287–310.
3. HERLOCKER, J., KONSTAN, J., TERVEEN, L., AND RIEDL, J. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems 22*, 1 (Jan. 2004), 5–53.
4. MILLER, B., KONSTAN, J., AND RIEDL, J. PocketLens: Toward a Personal Recommender System. *ACM Transcations on Information Systems 22*, 3 (July 2004), 437–476.
5. OREGON STATE UNIVERSITY. COllaborative Filtering Engine version 0.4. http://eecs.oregonstate.edu/iis/CoFE/, Sept. 2005.
6. RESNICK, P., IACOVOU, N., SUCHAK, M., BERGSTROM, P., AND RIEDL, J. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings 1994 ACM Conference on Computer Supported Cooperative Work* (Chapel Hill, NC, United States, Oct. 1994), pp. 175–186.
7. RESNICK, P., AND VARIAN, H. Recommender systems. *Communications of the ACM 40*, 3 (1997), 56–58.
8. SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. Item-Based Collaborative Filtering Recommendation Algorithms. In *Proceedings 10th International Conference on the World Wide Web (WWW10)* (Hong Kong, Hong Kong, May 2001), pp. 285 –295.
9. SHARDANAND, U., AND MAES, P. Social Information Filtering: Algorithms for Automating "Word of Mouth". In *Proceedings 1995 ACM SIGCHI Conference on Human Factors in Computing Systems* (Denver, CO, USA, May 1995), pp. 210–217.
10. UNIVERSITY OF MINNESOTA. GroupLens Home Page. http://www.grouplens.org/, Sept. 2005.
11. VOULGARIS, S., AND VAN STEEN, M. Epidemic-style Management of Semantic Overlays for Content-Based Searching. In *Proceedings 11th International Euro-Par Conference* (Lisbon, Portugal, Aug. 2005), pp. 1143–1152.