Horting Hatches an Egg: A New Graph-Theoretic Approach to Collaborative Filtering

Charu C. Aggarwal Joel L. Wolf Kun-Lung Wu Philip S. Yu

IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
{charu, jlwolf, klwu, psyu}@us.ibm.com

Abstract

This paper introduces a new and novel approach to rating-based collaborative filtering. The new technique is most appropriate for e-commerce merchants offering one or more groups of relatively homogeneous items such as compact disks, videos, books, software and the like. In contrast with other known collaborative filtering techniques, the new algorithm is graph-theoretic, based on the twin new concepts of *horting* and *predictability*. As is demonstrated in this paper, the technique is fast, scalable, accurate, and requires only a modest learning curve. It makes use of a hierarchical classification scheme in order to introduce context into the rating process, and uses so-called creative links in order to find surprising and atypical items to recommend, perhaps even items which cross the group boundaries. The new technique is one of the key engines of the Intelligent Recommendation Algorithm (IRA) project, now being developed at IBM Research. In addition to several other recommendation engines, IRA contains a situation analyzer to determine the most appropriate mix of engines for a particular e-commerce merchant, as well as an engine for optimizing the placement of advertisements.

1 Introduction

Collaborative filtering refers to the notion of multiple users "sharing" recommendations, in the form of ratings, for various items. The key idea is that the collaborating users incur the *cost* (in time and effort) of rating various subsets of the items, and in turn receive the *benefit* of sharing in the collective group knowledge. For example, they can view predicted ratings of other items that they identify, see ordered lists of those items whose predicted ratings are the highest, and so on. Collaborative filtering is generally computer based, and is typically most useful on e-commerce web sites selling homogeneous items. Examples of such items include compact disks, videos, books, software and the like. However, many other applications are possible. For example, collaborative filtering could be used to rate web pages themselves. Moreover, the ratings themselves need not necessarily be explicitly provided by the user. In the case of web pages, for instance, the (suitably bucketized) time a user spends on a web page could serve as a surrogate for the rating. Personalization processes are in their infancy, but represent an extremely important e-commerce opportunity. See [8] for the introductory survey article of a Communications of the ACM journal issue devoted to recommender systems in general, and collaborative filtering techniques in particular. Another article in that issue describes GroupLens [7], collaborative filtering which led to the founding of Net Perceptions.

In some e-commerce implementations the various possible queries are explicitly visible to the user, while in others the system generates appropriate recommendations implicitly and automatically. In all cases the most fundamental query is to rate a specific item for a specific user. This is in effect the atomic query upon which all other rating-oriented queries are based.

Naturally the goal of a collaborative filtering algorithm should be to make accurate rating predictions, and yet to do so in a manner that is fast and scalable. Furthermore, it is essential that the collaborative filtering technique be able to deal with the inherent sparsity of the rating data. It is unreasonable to expect users to rate a large number of items, but the total number of items available at the e-commerce site is likely to be huge. Said differently, the collaborative filtering algorithm should have a small learning curve.

We believe that collaborative filtering represents a different sort of data mining problem. Traditional clustering and nearest neighbor algorithms are frequently employed in data mining in order to identify peer groups, but they do not seem appropriate in this context due to high dimensionality, incomplete specification, the closeness measure, and so on. Thus other approaches must be examined.

1.1 Background

Two early and well-established approaches to collaborative filtering algorithms are those of LikeMinds [6] and Firefly [9]. It will be useful from the perspective of motivating our own work to describe these techniques briefly here. For more details see the papers themselves.

We begin by introducing a little notation. Suppose that there are N collaborative filtering users and there are K items which can be rated. In general the number of items rated by user i will be much less than K. Define a 0/1 matrix M by setting $M_{i,j}$ equal to 1 if user i has rated item j, and 0 otherwise. Define the set $R_i = \{1 \leq j \leq K | M_{i,j} = 1\}$, in other words the items rated by user i. Thus $card(R_i)$ is the row sum for the ith row, while the column sum for column j represents the number of users who have rated item j. For any matrix element $M_{i,j} = 1$ we denote by $r_{i,j}$ the actual rating of item j by user i. We will assume that ratings are always integers on a scale from 1 to some fixed value v. For a so-called 7 point scale, which is commonly used, we have $v = 7$, and $r_{i,j}$, if it is defined, will be an integer between 1 and 7. The value $v = 13$ is also commonly used, though there is some evidence to support the notion that a value of 7 is large enough from the perspective of a user's ability to accurately differentiate ratings [9]. Odd values of v allow the user to express neutrality, while even values force the user off the fence one way or the other.

In LikeMinds the estimated rating of a particular item j for a specific user i (who hasn't already rated that item) is computed as follows. Consider any other user k who *has* rated item j. LikeMinds calls user k a *mentor* for user i and item j, and we shall adopt this convention as well. Examine all the items l rated by both users i and k. LikeMinds defines a so-called *closeness* function $C(|r_{i,l} - r_{k,l}|)$ based on the absolute value of the difference between two ratings for the item l. Specifically, let us assume a 13 point rating scale, as is employed in [6]. Then the minimum distance between two ratings is 0 and the maximum distance is 12. LikeMinds employs the closeness function indicated in Figure 1. Now the *closeness value total* $CVT_{i,k}$ is defined as the sum over all items l rated in common of the closeness function values. Thus

$$CVT_{i,k} = \sum_{l \in R_i \cap R_k} C(|r_{i,l} - r_{k,l}|). \tag{1}$$

The so-called *agreement scalar* $AS_{i,k}$ is then computed as

$$AS_{i,k} = \frac{CVT_{i,k} \cdot log_2(card(R_i \cap R_k))}{card(R_i \cap R_k)}. \tag{2}$$

This can be viewed as the logarithmically scaled average value of the closeness function values. Now different mentors of user i and item j will have varying agreement scalars. (Presumably, though this is not explicitly noted in [6], only those mentors with agreement scalars greater than zero will figure into the actual rating process.) Two possible implementations of the algorithm are discussed. In one which we shall ignore hereafter, only the highest valued mentor k is used for predictive purposes. In the other, competing predictions from different mentors are weighted in proportion to the relative weights of their agreement scalars. The original rating $r_{k,j}$ of item j by mentor k is then transformed into a rating prediction $s_{k,j}$ by 'scaling' it to fit the range of user i. We understand this to mean that if min_i denotes the smallest rating given by user i and max_i denotes the largest such rating, then the rating is computed as

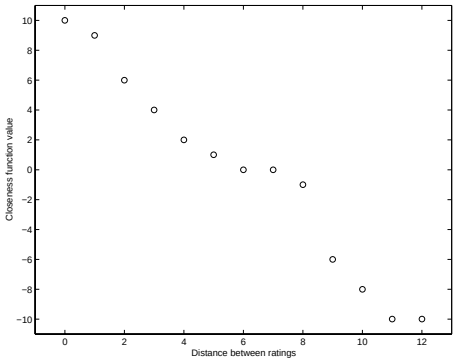

Figure 1: LikeMinds Closeness Function

$$s_{k,j} = min_i + \frac{(r_{k,j} - min_k) \cdot (max_i - min_i)}{max_k - min_k}. \tag{3}$$

So the final LikeMinds predicted rating $\mathcal{L}_{i,j}$ of item j for user i can be computed as

$$\mathcal{L}_{i,j} = \frac{\sum_{AS_{i,k}>0} AS_{i,k} \cdot s_{k,j}}{\sum_{AS_{i,k}>0} AS_{i,k}}. \tag{4}$$

In Firefly the ratings are computed somewhat similarly, at least in spirit. Of course, the details are different. Specifically, they make use of a *constrained Pearson r coefficient* to form a measure of closeness between two users. Consider again the rating of an item j by a user i, and let k be another user who has rated item j. Examining all the items l rated by both users i and j, and assuming a 13 point scale as above, the constrained Pearson r coefficient $\beta_{i,k}$ is given by

$$\beta_{i,k} = \frac{\sum_{l \in R_i \cap R_k} (r_{i,l} - 7)(r_{k,l} - 7)}{\sqrt{\sum_{l \in R_i \cap R_k} (r_{i,l} - 7)^2 \sum_{l \in R_i \cap R_k} (r_{k,l} - 7)^2}}. \tag{5}$$

(In [9] the authors use a 7 point rather than 13 point scale, with the *midpoint* 4 instead of 7.) As noted there, the standard *Pearson r coefficient*, defined using the relevant *means* instead of the midpoints, is constrained to be between -1 and +1. The former implies perfect negative correlation between users i and l, while the latter implies perfect positive correlation. The authors use the *constrained* coefficient to mimic this while ensuring that its value increases only when both users have rated an item positively or both have rated an item negatively. They then consider all such mentors k of user i and item j whose constrained Pearson r coefficient is greater than a predetermined threshold L, and form the weighted average of the ratings $r_{k,j}$, the weighted being proportional to the coefficient. (In [9] the value $L = 0.6$ is used, though for a 7 point scale.) So the final Firefly predicted rating $\mathcal{R}_{i,j}$ of item j for user i can be computed as

$$\mathcal{R}_{i,j} = \frac{\sum_{\beta_{i,k}>L} \beta_{i,k} \cdot r_{k,j}}{\sum_{\beta_{i,k}>0} \beta_{i,k}}. \qquad (6)$$

Note that the raw rating of user k is used in Firefly. There is no notion of scaling, as exists in LikeMinds.

1.2 Motivation

These two collaborative filtering techniques are quite reasonable in design and have apparently been successful. We do believe, however, that they can be improved upon in various ways. Although details of the data structures used to implement the algorithms are not provided in [6] or [9], we certainly believe the techniques can and have been made sufficiently fast and scalable. Instead, our concerns pertain to the accuracy of the predicted ratings given the inherently sparse data. Equivalently, we believe that these algorithms require more user ratings than absolutely necessary to make accurate predictions, thus phrasing our concerns in the *dual* context of a learning curve instead. Note that this ability to make accurate ratings predictions in the presence of limited data is by no means a trivial issue: Collaborative filtering algorithms are not deemed universally acceptable precisely because users are *not* willing to invest much time or effort in rating the items. Unquestionably this issue is the Achilles heal of collaborative filtering.

Specifically, the accuracy / data sparsity argument against the LikeMinds and Firefly algorithms is as follows. Consider again trying to find a rating of item j for user i. The ratings are calculated directly from mentors k of user i and item j. The number of users who have rated item j is small to begin with. The rating is based in both cases on some sort of closeness measure computed *on the set* $R_i \cap R_k$. But both sets R_i and R_k are likely to be small. Each is probably not much more than some predetermined minimum threshold R, typically on the order of 10 or 20 items, which allows the user to participate in the collaborative filtering. (Such thresholds are universally employed in collaborative filtering systems – otherwise users might be tempted to reap the benefits without paying the costs.) The number of items is presumably very large. Thus the intersection $R_i \cap R_k$ is likely to be extremely small in cardinality. Adding to this, only a fraction of the mentors will qualify for inclusion in the weighted average. So one may wind up basing the predictions on a very few mentors k, each of which has very little overlap in items rated with user i. Note, for example, that the agreement scalar defined for LikeMinds will be positive (provided the closeness value total is) when there are just 2 items in the intersection. If user k's ratings of those two items happen to agree with user i's ratings of those two items, the agreement scalar will equal 10 – not a terrible score in this context. The rating of item j may be based on such minimal commonality.

1.3 New Approach

To counter this fundamental sparsity problem, we propose in this paper a collaborative filtering algorithm based on a new and different graph-theoretic approach. While we shall give the full details in a subsequent section, the basic idea is to form and maintain a directed graph whose nodes are the users and whose directed edges correspond to a new notion called *predictability*. This predictability notion is in one sense stronger than the various notions of closeness of LikeMinds and Firefly, and yet in another sense more generic (and thus more likely to occur). It is stronger in the sense that more common items are required to be rated in common by the source and target users. It is more generic in the sense that it accommodates not only pairs of users whose ratings are genuinely close, but also user pairs whose ratings are similar except that one user is more effusive with his or her ratings than the other user. It also accommodates user pairs who rate items more or less oppositely, as well as combinations of the above two scenarios. Whenever one user predicts another user, a linear transformation is constructed which *translates* ratings from one user to the other. The ultimate idea is that predicted rating of item j for user i can be computed as weighted averages computed via a few reasonably short *directed paths* joining multiple users. Each of these directed paths will connect user i at one end with another user k who has rated item j at the other end. No other users along the directed path will have rated item j. However, each directed arc in the graph will have the property that there is some real rating predictability from one user to the other. The overall rating for this path is computed by starting with the rating $r_{k,j}$ and translating it via the composite of the various transformations corresponding

to the directed path.

In comparison, the techniques of LikeMinds and Firefly might be considered as arising from a single arc joining one user to the other. This makes the predicted rating a direct process. Our algorithm is more indirect. However, there are tradeoffs. Given the same ratings data, the link in LikeMinds or Firefly may far weaker than ours. Our links automatically handle scaling, which LikeMinds does to a degree but Firefly does not. In addition to their relative strength, our links are relatively plentiful because predictability generalizes closeness. It is our contention that examples of user pairs whose ratings are similar except that one is *left-shifted* from the other will occur frequently. (A similar thought is expressed in [5].) And note that although LikeMinds would scale the predicted ratings arising from such a pair properly, they may very well not qualify as close in the first place, and thus not be included. We also believe that occasional examples of *opposite* pairs will occur. Think of a politician whose views are the reverse of yours: In order to vote without actually thinking, simply vote oppositely and be assured that you are voting correctly.

Our new collaborative filtering approach forms the basis of an engine which is currently being demonstrated within IBM Research. The mythical e-commerce site in the demonstration system offers items from four major groups, namely compact disks, videos, books and software. (These are much the same groups of items offered amazon.com.) The engine incorporates several novel query features as well. For example, it makes use of a hierarchical classification scheme in order to introduce *context* into the rating process. The idea is that a user who is examining a web page in the classical music category might be happier at that moment to see a recommendation for Handel's *Messiah* than a jazz cd. On the other hand, there is built-in code to find so-called *creative links* between item pairs which may span category or even group boundaries. Thus a user who is examining Doyle's *Sherlock Holmes* book may see a recommendation for the *Seven Percent Solution* video. The notion here is to encourage additional sales by 'pushing' the user towards novel items which might not have been in the original buying plans.

The collaborative filtering technique introduced here is itself one of several recommendation engines of the *Intelligent Recommendation Algorithm* or *IRA* project at IBM Research. For example, there are content-based and product-based engines as well. These are also being demonstrated at the moment, using a mythical computer store as an e-commerce site. (IBM has, in fact, such a site.) While there are no papers yet describing these engines, we refer the reader to [1] for details on the projected clustering techniques employed in them. IRA also contains a so-called *situation analyzer* to determine the most appropriate mix of engines for a particular e-commerce merchant. Additionally IRA contains an engine for personalizing and optimizing the placement of web advertisements. The underlying mathematical details of our advertising engine are described in [2].

1.4 Organization

The remainder of this introductory paper is organized as follows. In Section 2 we briefly list some of the sample queries which our new collaborative filtering algorithm can handle. Section 3 contains the mathematical details of our technique, which is the focus of this paper. We include there the related definitions of *horting*, and *predictability*, the formal rating algorithm, as well as techniques for handling dynamic additions and/or changes to the ratings, and so on. In Section 4 we describe experiments to compare the accuracy, speed and scalability of the various collaborative filtering algorithms. We point out that these experiments are based on artificial data taken from our demonstration system, in the absence of a full-scale trial of IRA with real users. It is therefore quite arguable that our experimental results do not realistically portray the true accuracy of our algorithm. On the other hand we can show with some confidence results about the data sparsity described for motivational purposes above. In any case, we shall report on the accuracy results of our trial in a subsequent paper. In Section 5 we describe very briefly some of the additional key features of our collaborative filtering engine. Section 6 contains a conclusion.

2 Sample Queries

The following is a partial list of queries which can be answered by using techniques described in Section 3. We assume that all users have rated at least some predetermined minimum number R of items. We first list some rating-based queries:

- Query Q_1: Show user i the projected rating for item j, assuming the item has not already been rated by that user and assuming the algorithm is able to meaningfully estimate the item's rating. If the user has already rated the item then that rating is shown instead. If the algorithm is unable to determine the projected rating, then the *average* rating for that item is shown instead. The status, that is, which of these three alternative scenarios has occurred, is also returned. This is the basic atomic query for all of the three rating oriented queries below. That is, it is employed as a subroutine by each of them. But it could presumably also be invoked explicitly by user i, with an item j of his or her choosing.

- Query Q_2: Show user i an ordered list of up to M (as yet unrated) items from a subset $\mathcal{I}$ which are projected to be liked the most, subject to the constraint that each of them has a projected rating of at least some minimum value r. This subset can be chosen explicitly or implicitly in a variety of ways. For example, it may consist of a set of promotional items. Alternatively it can be defined as one or more categories in a hierarchical classification scheme, or the ancestors up through some predetermined level of those categories. The category may be that category associated with the item or web page last examined by the user. It may also consist of those items which are creatively linked to an existing subset. Or it may consist of combinations of any or all of the above. If the query finds no items to recommend it can automatically consider the category one higher in the classification hierarchy, and so on. Figure 2 shows this sample query from our demonstration system.

- Query Q_3: Show user i an ordered list of up to M (as yet unrated) items from a subset $\mathcal{I}$ which are projected to be liked the least, subject to the constraint that each of them has a projected rating of at most some maximum value r. Surprisingly this kind of recommendation can also result in sales for (presumably somewhat perverse) reasons: Consider, for example, the popularity of videos such as the *Rocky Horror Picture Show*. (The same comments apply to this query as those for the most liked query above.)

- Query Q_4: Select an ordered list of up to M users from a subset $\mathcal{J}$ each of which of which has projected rating of at least r that are projected to like item j the most. This is effectively the inverse of the second query, and could be used by the e-commerce merchant to generate mailings (electronic or otherwise) for specific promotion items.

It should be reasonably obvious that queries Q_2 through Q_4 can be readily answered based on repeated calls to the query Q_1 subroutine.

The following additional queries can be answered based on the underlying data structures of the collaborative filtering algorithm. All the queries below are aimed at forming chat groups and the like, and so must deal with anonymity issues. They could also each be made specific to a particular category.

- Query Q_5: Show user i an ordered list of up to M other users (made suitably anonymous) with whom he or she has rated the most items in common.

- Query Q_6: Show user i an ordered list of up to M other users with whom his or her ratings are most similar.

- Query Q_7: Show user i an ordered list of up to M other users with whom his or her ratings are least similar.

All of the key queries described in this section are available in our demonstration system, as is the personalized and optimized web advertisement engine.

3 Algorithmic Details

Our approach to generating good quality collaborative filtering predictions involves a pair of new concepts which we will call *horting* and *predictability*. We have already described the notion of predictability at a high level. This section contains the formal definitions and the mathematical details of the algorithm.

We say that user i_1 *horts* user i_2 provided either (1) $card(R_{i_1} \cap R_{i_2})/card(R_{i_1}) \geq F$, where $F \leq 1$ is some predetermined fraction, or else (2) $card(R_{i_1} \cap R_{i_2}) \geq G$, where G is some predetermined threshold. Notice that horting is reflexive (by virtue of the first condition in the definition): User i_1 always horts user i_1. But it is not symmetric: If user i_1 horts user i_2 it does not follow that user i_2 horts user i_1. (That is why we do not use the term *cohorts*. Horting is only symmetric for special cases, such as where the predetermined fraction F is 0 or 1.) Horting is also not transitive: If user i_1 horts user i_2 and user i_2 horts user i_3 it does not follow that user i_1 horts user i_3. The intent here is that if user i_1 horts user i_2 then there is enough commonality among the jointly rated items (from user i_1's perspective) to *decide* if user i_2 predicts user i_1 in some fashion or not. Notice that the two conditions in the definition of horting can be replaced by a single condition $card(R_{i_1} \cap R_{i_2}) \geq min(Fcard(R_{i_1}), G)$, and this value is in turn greater than or equal the constant $min(FR, G)$ – so we simply choose parameters such that this constant is sufficiently large.

We now make the notion of predictability more precise. For a given pair $s \in \{-1, +1\}$ and $t \in \{t_{s,0}, ..., t_{s,1}\}$, there is a natural linear transformation of ratings $T_{s,t}$ defined by $T_{s,t}(r) = sr + t$. (Here, $t_{-1,0} = 2$, $t_{-1,1} = 2v$ are chosen so that $T_{-1,t}$ keeps at least one value within $\{1, ...v\}$ within that range. The value of t will typically be close to the average of 2 and $2v$, namely $v + 1$. Similarly, $t_{1,0} = 1 - v$, $t_{1,1} = v - 1$ are chosen so that $T_{1,t}$ keeps at least one value within $\{1, ...v\}$ within that range. The value of t will typically be close to the average of $1 - v$ and $v - 1$, namely 0.) We say that user i_2 *predicts* user i_1 provided i_1 horts i_2 (note the reversed order!), and provided there exist $s \in \{-1, +1\}$ and $t \in \{-2v + 1, ..., 2v - 1\}$ such that $[\sum_{j \in R_{i_1} \cap R_{i_2}} |r_{i_1,j} - T_{s,t}(r_{i_2,j})|]/card(R_{i_1} \cap R_{i_2}) < U$, where U is some fixed threshold. The term $D_{s,t} = D(i_1, T_{s,t}(i_2))$ on the left represents the *manhattan* or L_1 distance between the ratings of user i_1 and the

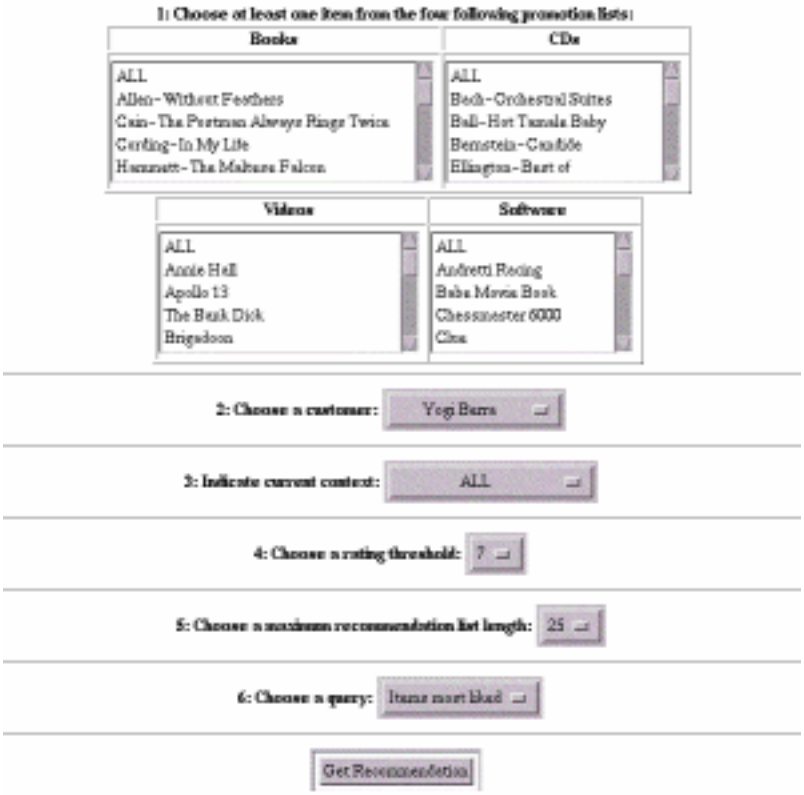

Figure 2: Sample Query

transformed ratings of user i_2 on the set of items they both rate, normalized by the number of those items. (We call this the *manhattan segmental* distance. Clearly other metrics could be employed instead.) Note that if $s = 1$ and $t = 0$, user i_2 behaves more or less like user i_1. (This would correspond to our interpretation of closeness, similar to that in [2] and [9].) In the idealized situation that $D_{s,t} = 0$ for $s = 1$ and $t = 0$, user i_2 behaves *exactly* like user i_1. If $D_{s,t} = 0$ for $s = 1$ and $t = 1$, user i_1 is simply more effusive with praise (by one rating unit) than user i_2. On the other hand, if $D_{s,t} = 0$ for $s = 1$ and $t = -1$, user i_2 is more effusive than user i_1. Finally, if $D_{s,t} = 0$ for $s = -1$ and $t = v + 1$, user i_2 behaves in a manner precisely opposite to user i_1. Nevertheless, user i_2 predicts user i_1 perfectly. (Again, one might think of a politician from some mythical state such as East Carolina.)

It should be clear from this discussion that (1) predictability is a more general notion than mere closeness, but (2) the horting requirement in the definition imposes a relatively rigid requirement that the predictability be based on sufficient data to be legitimate. As with horting, predictability is reflexive, but not symmetric or transitive. This is primarily because of the horting component of the definition.

If user i_2 predicts user i_1, we define $s\star$ and $t\star$ to be the values of s and t, respectively, which minimize $D_{s,t}$. (There are only $4v - 2$ possible pairs (s, t), so this optimization is performed quickly via exhaustive search.) The pair $(s\star, t\star)$ will be called the *predictor* values. We will use the linear transformation $T_{s\star,t\star}$ to translate ratings from user i_2 to user i_1.

Let H_i denote the set of users who are horted by user i. Let P_i denote the set of users who predict user i. Thus $P_i \subseteq H_i$.

In order to implement our prediction process we will need to employ three distinct data structures. These are as follows:

- For each item j we will need to maintain an *inverted index* of all the users who rate item j. For details on inverted indexes, see [4]. We will insist that the inverted indexes be sorted in increasing order of users i. (By this we mean that we will equate the index of user i with a unique identification given to that user, and sort in terms of that id.) We store the actual ratings in this inverted index as well.

- We will also need to maintain a *directed graph* in which the nodes are the users and there is a directed arc from user i_1 to user i_2 provided user i_2 predicts user i_1. We will store the corresponding predictor values in the directed graph, as well as the items rated by the various users and the ratings themselves. (In our implementation of

this algorithm we actually adopt a reasonable alternative: we maintain a somewhat larger directed graph in which the directed arcs between nodes correspond to horting itself – that is, we can relax the predictability portion of the directed arc condition. Thus the number of nodes is the same, but only a subset of the directed arcs correspond to predictability. Because we do this we also have to store the predictive values of the predictors with the directed arcs.)

- Finally, we maintain an *update list* consisting of those new and modified ratings of items for the various users which have not yet been incorporated into the inverted indexes and the directed graph. This is because that update operation is modestly labor intensive, and we do not wish to always perform it in real time. (The timing of this update is under the control of a separate scheduler module.) When the update operation does occur we simultaneously flush the list.

Now, consider the atomic rating query Q_1 from Section 2: We wish to predict the rating of item j for user i. First we check the inverted index and the update list to determine if user i has already rated item j. What happens next depends on whether or not user i has any (other) items remaining in the update list. Equivalently, the question is whether or not user i has recently added to or modified his or her ratings.

If so, we wish to incorporate these ratings into the prediction process, but we are willing to employ a slightly stale view of the other users by ignoring the remaining items in the update list. The prediction algorithm now proceeds in three stages: First, the inverted indexes for all items in the revised R_i are examined via a merge and count operation, so that the set H_i of users who are horted by user i can be quickly computed. See Figure 3 for an illustration of this process. Notice that success and/or failure of the horting question can often be determined without checking all the inverted indexes. Thus, since the inverted indexes are arranged in increasing order of size, the largest inverted indexes will frequently not need checking. Second, those members of H_i which are also in P_i are found via the appropriate calculations. Third, a shortest path in the directed graph from any user in P_i to a user who rates item j is computed. See Figure 4 to visualize the shortest path process. (In Figure 4 the dotted directed arcs correspond to horting but *not* predictability. The solid directed arcs correspond to both horting and predictability.) The shortest path algorithm can be implemented via breadth first search. If the overall collaborative filtering process is going to be effective, the length of such a directed path will typically be small, and can be found effectively by the breadth

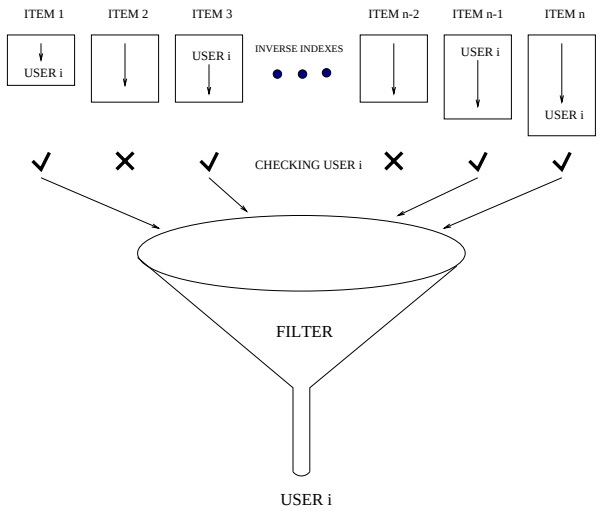

Figure 3: Inverse Indexes

first search algorithm. In fact, if a directed path is not found within some very small distance D (we use $D = 4$), the algorithm terminates with the failure: Item j cannot be rated for user i. For details on shortest path algorithms see, for example, [3]. This directed path allows a rating computation based on the composition of transformations of a predicted value for item j by user i. So, for instance, given a directed path $i \rightarrow i_1 \rightarrow ... \rightarrow i_l$, with predictor values $(s_1\star, t_1\star), ...(s_l\star, t_l\star)$, the predicted value for the rating of item j will be $T_{s_1\star,t_1\star} \circ ... \circ T_{s_l\star,t_l\star}(r_{i_l,j})$. In our algorithm the average of these predicted values computed from all such directed paths of minimal length is taken as the final predicted rating. Figure 4 shows two such directed paths having length 2.

If user i has not changed or added to his ratings since the update list was last flushed, then the first two steps are not necessary. Only the third step need be performed.

Note that because we are willing to accept slightly stale data associated with users other than user i, only those directed arcs emanating from that user need be computed on the fly. The rest are prestored and do not change during the prediction process. (Allowing this staleness seems to us to be in the proper spirit of the algorithm, since we are keeping user i's ratings current and all other user's ratings *nearly* so. But an implementation not willing to accept this would simply keep the update process running in real time, thus maintaining current versions of the inverted indexes and the directed graph.)

The above algorithm for query Q_1 allows answers to queries Q_2 through Q_4 as well, as already noted. Query Q_5 can easily be answered via a lookup in the directed graph, and that is in fact the reason for maintaining the horting data in that graph as well. Similarly, queries Q_6

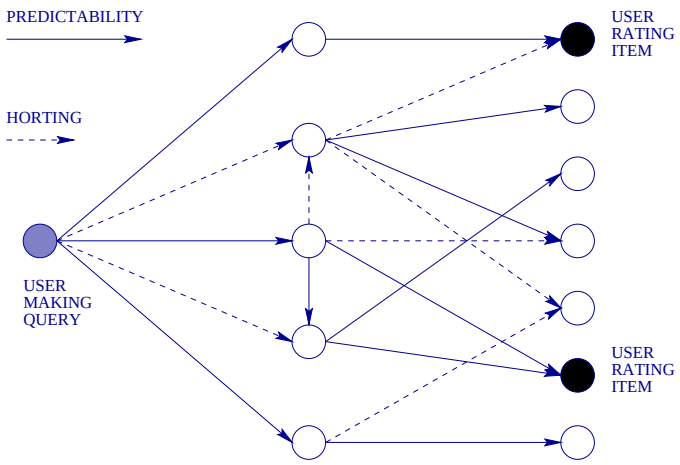

Figure 4: Horting and Predictability

and Q_7 can easily be answered via the directed graph.

We do not address the issue of computational complexity explicitly here, but note that the time required to perform the shortest path search in step 3 is small, since the distances involved are not large and the directed graph is relatively sparse. (In the next section we will present data which supports these assertions.) As the number of users increases one can tighten the various parameters F, G and U, so that the system will be quite scalable.

We should point out that our collaborative filtering algorithm also contains techniques for presenting the user with those items which the system would most like to be rated. The idea is to thereby reduce the learning curve time for the collaborative filtering process. While we will not have space to go into detail here, fundamentally the approach is to partition the collection of items into a small *hot* set and a large *cold* set. (Typically the cardinality of the hot set is two orders of magnitude smaller than the cardinality of the cold set.) Items in the hot set are chosen based on their popularity, and some of these are presented to each user in order to increase *commonality* of rated items. Items in the large cold set are presented to the users in order to increase *coverage* of the remaining rated items. Partitions into more than two sets are also possible in our algorithm. We have not seen a discussion of this issue in other collaborative filtering papers, though we presume something similar is typically done.

4 Experimental Results

In this section we describe results of synthetic simulation experiments used to evaluate our collaborative filtering algorithm. We have done this as part of our demonstration system, and of course we are relying on these experiments until our full trial can be deployed. We state up front that this is not particularly fair to the other algorithms, because we are evaluating the accuracy of our collaborative filtering tool on data which happens to more or less fit our model. So the fact that our algorithm does best in these tests should certainly be taken with a grain of salt. However, we can probably place somewhat more stock in the key derived data showing data sparsity and such. And this data is the motivating force behind our algorithm in the first place.

We briefly describe the experimental methodology employed in our simulations. We chose a $v = 13$ point rating scale. In all cases we used a total of 5,000 items, and these were partitioned into a hot set consisting of 50 commonly rated items and a cold set of 4,950 items which were infrequently rated. Again, the hot set items are hot in part because the user is 'pushed' to rate these items. We assumed that each user would rate an average of 20 items, 10 from each of the hot and cold sets. Thus we adjusted the probabilities accordingly ($p_{hot} = 0.2$, $p_{cold} = 0.00202$) and picked randomly for each user and item to determine if a rating was given. Next we describe the ratings themselves. The average rating m_i for each item i was chosen from a uniform distribution on the set $\{1, ..., 13\}$. Each user j was then randomly assigned an *offset* factor o_j, chosen from a normal distribution about a mean of 0, to account for his or her level of 'effusiveness'. (Positive means more effusive than average, negative means less so.) Finally, a small fraction of the users were chosen to be 'contrarians'. In other words these users like items that other users dislike, and vice versa. Now, given that item i was determined to be rated by user j, a random rating $RR_{i,j}$ was chosen from a normal distribution with mean m_i, the offset factor o_j was added, the overall rating was reversed if user j was a contrarian, and the final rating $r_{i,j}$ was constrained to lie within 1 and 13. In formulas, this yields $r_{i,j} = min(13, max(1, RR_{i,j} + o_j))$ for normal users, and $r_{i,j} = min(13, max(1, 14 - RR_{i,j} - o_j))$ for contrarians.

Dealing with the accuracy of the various algorithms first, consider Tables 1 through 3. These present, for each of the three collaborative filtering techniques discussed in this paper, results of experiments to show actual versus predicted ratings. The experiments were for 5,000 items and 10,000 users, but other similar experiments yielded comparable results. Each table is 13 by 13, with the rows indicating actual ratings and the columns indicating predicted ratings. Table 1 is for LikeMinds, Table 2 for Firefly, and Table 3 for our algorithm, here labeled IRA. The procedure was to choose a random pair of one item item and one user for which a rating had been given, mask that rating, perform the collaborative filtering algorithm, and finally compare the predicted and actual results. Only experiments in which the algorithms

were able to compute a rating were considered. The others were discarded in favor of a new random pair. (Table 4 shows the percentage of successful predictions.) Approximately 100 successful experiments were performed for each actual rating between 1 and 13, so that there were roughly 1300 experiments in all. Then the totals were normalized by dividing by the number of experiments for each actual rating. Turning them into percentages, each row sums to approximately 100 percent. Clearly it is 'goodness' to be as close as possible to a diagonal matrix here. From that perspective, LikeMinds does quite well, with the diagonal values averaging over 61 percent. It is true, however, that LikeMinds does seem prone to making large errors on occasion, which we attribute to its inability to properly distinguish the reversal of some users relative to others. Firefly does less well, having a diagonal value average of approximately 31 percent. It is possible that varying some of the parameters associated with Firefly could improve this, but we were not able to achieve any better results. On the other hand, Firefly is probably less prone to having outliers, perhaps because scaling is not done. Clearly IRA performs best here. The average value along the diagonal is over 81 percent. and there are very few examples of truly bad performance. But again, we are assuming a model of data which precisely fits our technique in terms of effusiveness, contrarians and so on. So this level of performance is to be expected. The true test of our collaborative filtering algorithm will come in the user trial.

Now consider Table 4, which shows certain key data about experiments under three different scenarios. In each case 1,000 random experiments were made, again via the technique of masking existing ratings for later comparison. In each case there were 5,000 items, as before. Scenario one had 1,000 users. Scenario two had 10,000 users, similar to the experiments of Tables 1 through 3. Scenario three had 100,000 users, which we believe is adequately large for any collaborative filtering test.

Considering LikeMinds first, note that the number of successful runs of the algorithm went from 68 with 1,000 users, up to nearly universal success thereafter. The average number of mentors grows rapidly with the number of users, but only about three quarters of the mentors actually are *qualifying* mentors. This holds, more or less, through the three scenarios. And those qualifying mentors have little intersection, which we believe is a key observation. Notice that on average only about 3 items are rated in common for the two relevant users. And the agreement scalar, as noted before, is not very high. We have also computed both the average error and the error for those items actually rated 11 and above. (As noted in [9], the latter is probably more important than the former in a recommendation system.) Both of these are relatively stable, though the error for well-liked items appears to drop with the number of users.

Firefly lags somewhat behind LikeMinds in Table 4. The number of successful predictions grows somewhat more slowly, though eventually nearly every attempt yields success. The average number of mentors is, of course, identical to that of LikeMinds, but the average number of qualifying mentors is lower throughout. And again, the cardinality of the intersection is small, similar to that of LikeMinds. The value of β is close to 1, but this is not as impressive because of the cardinality factor. The average errors shrink as the number of users grows, but the errors are fairly large.

IRA does best of all, with a large percentage of successful predictions. The average number of directed paths grows with the number of users, but the path length does not. And here the intersection between adjacent nodes in the directed path is quite a bit higher. We should point out that we chose the minimum intersection cardinality G to be 5, 8 and 11, respectively, for the three different user size scenarios, the minimum number of ratings R to be 20, and the fraction F to be 1 (causing the first constraint in the definition of horting to be irrelevant). The average error figures are best among all the algorithms considered.

In all cases the ratings were computed with great speed. These algorithms all scale well.

5 Additional Features

In this section we briefly list several additional features which are part of our collaborative filtering algorithm and are included in the current code. While we feel that these features are important, we do not have sufficient space to go into details in this paper.

- The algorithm can prestore single or multiple recommendations per user. If the user has not changed his or her ratings recently this feature allows for essentially immediate and accurate query responses. Otherwise, the updating event itself can be used to trigger a new prestore computation.

- The algorithm can keep track of recommendations previously made to the user. This can be used to avoid or limit the number of repetitive recommendations, as desired.

- The algorithm can use a weighted keyword frequency distribution analysis of browsed and purchased items as an alternative to actual user ratings. The keyword data is collected automatically from the web pages describing the items. This surrogate approach is significant in that ratings are optional,

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	64	20	9	1	2	2	0	1	0	0	0	0	0
2	4	59	27	7	1	0	1	0	0	0	0	0	0
3	0	3	70	22	4	0	0	0	0	0	0	0	0
4	1	0	5	69	19	3	1	1	0	0	0	0	0
5	0	0	8	10	58	19	5	0	0	0	0	0	0
6	0	0	3	3	10	56	28	1	0	0	0	0	0
7	0	0	0	1	8	12	58	16	3	1	1	0	0
8	0	0	0	0	5	3	16	60	13	1	0	0	0
9	0	0	0	0	6	4	8	16	57	4	4	0	0
10	0	0	0	0	0	0	1	8	20	65	5	1	0
11	0	1	1	1	1	1	1	0	4	17	56	13	3
12	0	0	0	0	0	0	1	1	0	9	23	55	10
13	0	0	0	0	0	0	0	1	1	4	8	17	68

Table 1: LikeMinds Accuracy: Rows=Actual, Columns=Predicted

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	22	49	12	12	5	0	0	0	0	0	0	0	0
2	8	53	21	8	3	3	2	0	0	0	2	0	0
3	10	14	29	29	10	6	2	0	0	0	0	0	0
4	2	8	15	32	20	12	5	2	3	2	0	0	0
5	0	3	5	21	26	21	16	0	6	0	2	0	0
6	0	0	2	14	14	25	27	8	6	0	3	2	0
7	0	1	6	3	3	12	47	15	7	4	0	0	1
8	0	0	0	6	10	12	22	16	12	16	4	4	0
9	0	0	5	5	0	11	13	11	24	24	8	0	0
10	0	0	0	0	0	0	6	9	17	45	9	12	2
11	0	0	0	0	2	2	3	7	9	19	29	21	9
12	0	0	0	0	0	0	0	2	6	14	25	37	16
13	0	0	0	0	0	0	3	2	2	5	18	47	23

Table 2: Firefly Accuracy: Rows=Actual, Columns=Predicted

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	75	16	5	1	1	1	0	0	0	0	0	0	0
2	7	73	15	3	1	1	0	0	0	0	0	0	0
3	3	12	80	5	0	1	0	0	0	0	0	0	0
4	0	2	10	68	12	6	1	0	0	0	0	0	0
5	0	0	1	9	78	11	0	1	0	0	0	0	0
6	0	0	2	0	13	74	10	1	1	0	0	0	0
7	0	0	1	2	2	10	79	4	1	0	1	0	0
8	0	0	1	0	3	0	15	69	8	2	1	0	0
9	0	0	0	0	0	1	1	15	77	5	1	1	0
10	0	0	0	0	0	0	2	0	8	80	5	5	1
11	0	0	0	0	0	1	1	2	5	11	71	9	1
12	0	0	0	0	0	0	0	1	1	5	10	74	10
13	0	0	0	0	0	0	0	1	2	2	2	16	77

Table 3: IRA Accuracy: Rows=Actual, Columns=Predicted

Algorithm	Description	1,000 Users	10,000 Users	100,000 Users
LikeMinds	Percent Successful Predictions	68	98	100
	Average Number of Mentors	4.49	36.22	102.85
	Average Number of Qualifying Mentors	3.09	22.88	78.01
	Average Intersection Cardinality Of Qualifying Mentors	3.02	2.98	3.11
	Average Agreement Scalar Value	10.05	9.91	10.12
	Average Error	0.71	0.56	0.61
	Average Error for Well-liked Items	0.70	0.64	0.53
Firefly	Percent Successful Predictions	61	95	99
	Average Number of Mentors	4.49	36.22	102.85
	Average Number of Qualifying Mentors	2.89	15.79	66.94
	Average Intersection Cardinality of Qualifying Mentors	3.04	2.98	3.01
	Average β Value	0.90	0.91	0.90
	Average Error	1.30	1.24	0.99
	Average Error for Well-liked Items	1.21	1.20	0.89
IRA	Percent Successful Predictions	83	99	100
	Average Number of Directed Paths	1.46	2.51	4.08
	Average Directed Path Length	2.48	2.68	2.17
	Average Intersection Cardinality of Directed Path Arcs	7.29	9.25	12.13
	Average Manhattan Segmental Distance of Directed Path Arcs	0.69	0.68	0.68
	Average Error	0.35	0.33	0.31
	Average Error for Well-liked Items	0.36	0.42	0.32

Table 4: Key Comparative Results

and depend entirely on user cooperation. Additionally, a new item arrives without a rating history, but with its web page. The automatic nature of the data collection is much like browse time data. As such, this approach to personalization can serve to eliminate or greatly reduce the time period in which recommendations are not available for a new system or user. Our process takes advantage of new projected clustering techniques. These techniques dynamically select the relevant dimensions during cluster formation, which is far more powerful than prefiltering before clustering. See [1] for more details.

- The algorithm can employ a feedback learning process in the personalization process itself. After presenting a recommendation the process will give the user the opportunity to say what rating the user would have given to that item. Although this will be optional, it helps generate continually changing recommendations if the user chooses to respond.

- The algorithm can optimize its recommendations by revenue or other criteria, if desired, with constraints on the number of times each of the various items are recommended.

- As noted before, the algorithm interfaces with a target advertising optimization process. The goal is to personalize the web advertisements to the individual users, and thereby optimize the exposures, click rates and/or revenues associated with web-based advertising. See [2] for details.

- As noted before, the algorithm includes a scheduling module which updates the key data structures associated with user recommendations during periods of relative inactivity of the collaborative filtering system.

- As noted before, the algorithm contains techniques for presenting the user with those items which the system would most like to be rated.

- Finally, as noted before, the algorithm employs a keyword analysis in order to create a list of creative links, joining pairs of items which may in fact come from different groups. This module has a user interface so that an administrator can choose to accept or reject these links.

6 Conclusion

In this introductory paper we have described a new type of collaborative filtering algorithm, based on twin new notions of horting and predictability. The algorithm described performs quite well on artificial data, and will be tested in a real user trial in the near future. Put in a larger context, our new collaborative filtering fits as one of the key engines of the Intelligent Recommendation Algorithm project under development at IBM Research. We will have more to say about the various components of IRA in the future.

References

[1] C. Aggarwal, C. Procopiuc, J. Wolf, P. Yu and J. Park, "Fast Algorithms for Projected Clustering", *Proceedings of ACM SIGMOD Conference*, Philadelphia PA, pp. 61-72, 1999.

[2] C. Aggarwal, J. Wolf and P. Yu, "A Framework for the Optimizing of WWW Advertising", *International IFIP Working Conference on Electronic Commerce*, Hamburg, Germany, 1998. Published in *Trends in Distributed Systems for Electronic Commerce*, W. Lamersdorf and M. Merz, editors, Springer-Verlag Lecture Notes in Computer Science, Vol. 1402, pp. 1-10, 1998.

[3] T. Cormen, C. Leiserson and R. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge MA, 1992.

[4] C. Faloutsos, Access Methods for Text, *ACM Computing Surveys*, Vol. 17, pp. 50-74, 1985.

[5] Y. Freund, R. Iyer, R. Shapire, Y. Singer, "An Efficient Boosting Algorithm for Combining Preferences", *International Conference on Machine Learning*, Madison WI, 1998.

[6] D. Greening, "Building Consumer Trust with Accurate Product Recommendations", LikeMinds White Paper LMWSWP-210-6966, 1997.

[7] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordan and J. Riedl, "GroupLens: Applying Collaborative Filtering to Usenet News", *Communications of the ACM*, Vol. 40, No. 3, pp. 77-87, 1997.

[8] P. Resnick and H. Varian, "Recommender Systems", *Communications of the ACM*, Vol. 40, No. 3, pp. 56-58, 1997.

[9] U. Shardanand and P. Maes, "Social Information Filtering: Algorithms for Automating Word of Mouth" *Proceedings of CHI '95*, Denver CO, pp. 210-217, 1995.