Challenges and Solutions for Synthesis of Knowledge Regarding Collaborative Filtering Algorithms

Daniel Lowd[†], Olivier Godde[‡], Matthew McLaughlin, Shuzhen Nong, Yun Wang, and Jonathan L. Herlocker.

School of Electrical Engineering and Computer Science

Oregon State University

102 Dearborn Hall

Corvallis, OR 97331

{[†], [‡], mclaughm, nong, wangyun, herlock}@cs.oregonstate.edu

Abstract

Collaborative filtering (CF)-based recommender systems predict what items a user will like or find useful based on the recommendations (active or implicit) of other members of a networked community. In spite of more than ten years of research, there is little consensus on state-of-the-art knowledge regarding CF predictive algorithms. There are many barriers to synthesis of the significant quantity of available published research on CF algorithms. We present results from an empirical study that attempts synthesis on popular CF algorithms and use this study to illustrate some key challenges to synthesis in CF algorithm research. In response to these challenges we propose the development of publicly maintained reference implementations of proposed CF algorithms and empirical evaluation procedures and we introduce CoFE, a public software framework with the goal of jumpstarting the building of these reference implementations. Finally, we demonstrate how CoFE was used to implement a high-performance nearest-neighbor-based algorithm that scales to arbitrary numbers of users.

1 Introduction

"Read any good books lately?" Every day, people ask each other questions such as this in an attempt to sort through the plethora of options that both enrich and afflict modern living. In a world with vastly more books available than any of us has time to look at, much less read, we all need some way to decide among them. By passing along recommendations to friends with similar taste, people distribute the work of finding good books in order to spend more time reading books they enjoy. Unfortunately, not all of our friends share our tastes, limiting the number of useful counselors available. Furthermore, those friends who do share our tastes probably haven't read every book we might like, and can only furnish recommendations for the few they know. Of course, the problem is more general than finding good books – people must make decisions about a great many things, including movies, restaurants, web sites, house plants, tropical resorts, and so on. How can we get better recommendations on such an ever-widening assortment of options?

[†] Daniel Lowd is now at the Department of Computer Science and Engineering, University of Washington, Seattle, WA, 98195, lowd@cs.washington.edu.

[‡] Olivier Godde can be reached at ogodde@yahoo.com.

1.1 Brief Introduction to Collaborative Filtering & Recommender Systems

Collaborative filtering-based recommender systems address precisely this problem by drawing upon the experiences of thousands or even millions of people. For example, a book recommendation web site using collaborative filtering technology could combine the ratings of millions of online users to give better and broader book recommendations than any of the users could get from friends. The work of finding good books is thus distributed, as in a circle of friends, but on a much larger scale. Amazon.com is one well-recognized example of a site that uses collaborative filtering in this manner. In addition to helping users find items of interest, collaborative filtering has proven to benefit e-commerce retailers as well. Amazon.com reported that many more sales result from items recommended by collaborative filtering than from those shown on bestseller or featured items lists [15]. Another success story found that collaborative-filtering based recommendation e-mails generated twice as many purchases as manual recommendations [27]. These systems are just two of the many recommender systems, commercial and academic, that have been developed using collaborative filtering. Other systems have included MovieLens and NetFlix.com for recommending movies, PHOAKS for recommending websites, Ringo for recommending music, Jester for recommending jokes, and many more [7,9,25,26].

1.2 Important Terminology Used in this Paper

Recommender systems are a relatively new area of study and standardized vocabulary is only beginning to emerge. Here we briefly provide our definitions for the terminology used in this paper.

Resnick et al. describe a *recommender system* as follows: "In a typical recommender system people provide recommendations as inputs which the system then aggregates and directs to appropriate recipients." [20] The most common technology used to implement recommender systems is *collaborative filtering*, which we may refer to as *CF* for short. Recommender systems may incorporate non-CF technology. However, in this article, we focus exclusively on CF technology.

We refer to anything recommended by a recommender system as an *item*. This term could refer not only to books and movies, but also to restaurants, web pages, New Year's resolutions, and so on. The type of items being recommended and the context in which they are recommended is known as the content *domain* of recommendation (e.g. web-based book recommendation domain).

A *user* is an individual who interacts with a recommender system, providing the system with ratings in order to receive recommendations or predictions.

Ratings are statements of preference by users for items. At a minimum, a rating consists of three elements: a user, an item, and a *rating value*. The rating value may be binary, integer valued, real-valued, or even unary (a single positive rating value, but no negative or ambivalent rating values). For integer- and real-valued ratings, low numbers generally indicate negative preference (the item was bad), middle numbers indicate ambivalence (the item was neither good nor bad), and high number indicate positive preference (the item was great!).

A *prediction* or *predicted rating* is a recommender system's estimate of the rating value that a user would assign to an item. We refer to a *recommendation* as an item with a high predicted

rating for a user that is "recommended" to the user. Recommendations are often called "best bets."

A *collaborative filtering algorithm* is a procedure that examines ratings data from users, infers preference patterns among many users and many items, and computes a predicted rating for a given user (termed the *active user*) on a given item (termed the *active item*). A ranked list of recommended items can be generated as well, by predicting ratings for all items and listing the *N* items whose predicted ratings are the highest[1]. The *accuracy* of a CF algorithm is defined as how close an algorithm's predicted ratings are to the true ratings supplied by users.

Explicit ratings are evaluations that are directly entered by users; for example, a user's rating of 1-5 stars for an item at Amazon.com is an explicit rating. *Implicit ratings*, in contrast, are indications of preference that are derived from other user behavior, such as purchasing certain items from a catalog or visiting specific web sites. Algorithms that work with implicit user ratings generally operate differently to take into account a very different quality of information. For this study, we chose to focus solely on explicit ratings and algorithms designed to operate on them.

A *dataset* is a collection of preference ratings data from a community of users on a set of items in a particular target domain. The most popular publicly available datasets involve ratings for movies and videos: the EachMovie dataset [16], and the MovieLens dataset [1].

A *metric* is a computation applied to the output of a collaborative filtering algorithm to provide an evaluation of the quality of the collaborative filtering algorithm.

1.3 The Challenge of Finding the "Best" CF Algorithm

There have been many different collaborative filtering algorithms proposed to compute predictions of users' ratings. One algorithm will be more effective than the others, given specific circumstances. Given a choice, you would always want to use the most effective algorithm possible, since that ought to result in a better user experience, more e-commerce sales, less time wasted browsing through irrelevant items, and so on. For just this reason, a more effective algorithm has become the most popular research goal among collaborative filtering researchers in recent years.

How one best determines if an algorithm is "more effective" is still open to debate, but most researchers use an algorithm's average accuracy when tested on an existing database of user ratings. Others may look at execution time and memory requirements as well.

So what are the "best" CF algorithms? With almost ten years of published scientific research on the development and evaluation of CF algorithms, we would expect to have solid recorded knowledge about which algorithms are best for which content domains. An examination of the published research indicates that we are far from that goal. What we find are many reports of empirical studies that are hard to generalize beyond the context of their published study. We find many published articles introducing new CF algorithms that follow the same template. The template begins by proposing an algorithm and then claiming experimental results showing its

[1] Since computing the predicted rating of every item is usually too computationally intensive, scientists have developed algorithms that approximate the process, generating a set of items that have predicted ratings above a threshold (e.g., predict items that are likely to be "good" but may not necessarily be the "best" recommendations).

empirical superiority over one or two baseline algorithms. Unfortunately, we find it hard to evaluate the strength of such results due to variations in the experimental procedures, different datasets, and different algorithm implementations used to evaluate those algorithms.

New work and new methodology is required in CF algorithms research to bring us closer to the goal of understanding what algorithms are "best" for which domains. Rather than continue to propose new algorithms using methodologies that inhibit cross-comparison, we need research that seeks to synthesize the diversity of work that has been done before into a coherent picture that can be directly applied by practitioners seeking to implement or employ recommender systems using collaborative filtering.

1.4 Contributions of this Article

This article presents some initial results of our attempt to unify the knowledge regarding the accuracy of collaborative filtering recommendation algorithms. In particular, our contributions are:

1. A specification of challenges faced by scientists attempting to synthesize the existing published research on collaborative filtering algorithms. These challenges are presented in Section 2.
2. A case study representing an attempt to synthesize existing published work on CF algorithm accuracy. This study consists of an empirical comparison of a collection of proposed collaborative filtering algorithms that previously have only been examined individually against non-comparable baseline algorithms. This case study in Section 3 is used to illustrate the challenges that we introduce. The case study itself has several key contributions to further research on CF algorithms:
    a. Evidence that contradicts previous accuracy claims for certain algorithms.
    b. Evidence showing that nearest neighbor algorithms (in particular, the Item-Item algorithm) are most accurate at predicting rating values on multi-valued rating data of entertainment.
    c. Enhancements to several existing algorithms that significantly improved accuracy in our experiments. Most notable is an adaptation of the Bayesian network approach suggested by [4] that uses normalized user ratings rather than discrete rating classes.
3. Proposals for specific research methodologies and research infrastructure that would enable future research to better face the previously described challenges, enabling more generically usable scientific results. These proposals are discussed in Section 4 of this article.
4. An infrastructure that we have developed and made freely available to the public as a first step towards facing the challenges, including
    a. A portable and highly extensible software framework for investigating collaborative filtering algorithms, enabling rapid design and evaluation of new algorithms. This software framework represents the first step towards collaborative filtering research infrastructure that enables more effective future CF research.

b. Source code for reference implementations of a collection of collaborative filtering algorithms that have been proposed by prominent CF researchers. Most of the algorithms have been tuned to ensure that they can perform at least as well as their original creators claimed.

This infrastructure is introduced in Section 5.

5. Finally, a high performance, production capable recommendation engine, built on top of the CF software framework. Supporting well-known nearest-neighbor methods, this engine can generate hundreds of recommendation lists per second, given millions of user ratings. The source code for this engine is freely available. This software, introduced in Section 5.1, should greatly increase the availability of collaborative filtering technology to all software developers.

2 Challenges to Synthesis

Greater scientific advances are possible when individual research contributions can be synthesized into a broader understanding through objective comparison and third-party evaluation. In this section, we introduce characteristics of collaborative filtering algorithms research that present challenges to achieving such an understanding. The list of challenges that we have itemized in this section is not intended to be a complete list. Rather they are the challenges that we have found frequently obstruct our attempts to synthesis. In the following section, Section 3, we illustrate examples of these challenges in an empirical case study.

2.1 Challenge 1: Different Datasets

Different datasets have different characteristics that can significantly affect the outcome of empirical analysis of collaborative filtering algorithms. For example, datasets may have different numbers of ratings per user and thus different amounts of training data or they may have different granularity of ratings – one dataset may include ten levels of preference, while the other may only include five.

When two groups of researchers use entirely different datasets, it is difficult to synthesize their results into any form of greater understanding. Proprietary datasets – ones that have not been released to the public – present additional challenges because scientists not affiliated with the original researchers are unable to reproduce or extend results without access to the data.

Even when researchers use the same, publicly available dataset, their results may be dangerous to compare. CF researchers often want to run hundreds or thousands of different CF algorithm variants against the dataset. Yet with very large datasets, this can take an unacceptable amount of time, so they create smaller subsets of the whole dataset. Scientists have taken a variety of approaches (or lack of) to ensure that the subsets are "representative" of the whole. The additional variance creates additional uncertainty when trying to synthesize results achieved on different subsets of the same dataset. For example, some researchers sample only users with a minimum number of rated items to ensure that learning algorithms have enough information to work with for each user [5,9]. Others may randomly sample users without regard for number of ratings.

2.2 Challenge 2: Different Evaluation Metrics

There are many different metrics that can be applied (for a more complete discussion of CF metrics, see [10]), however for the purposes of this article, we are considering collaborative filtering accuracy metrics. Examples of accuracy metrics include mean absolute error [9], precision and recall [23], and the rank half-life metric [4].

When two experiments use different empirical evaluation metrics, then the results of those metrics are very challenging to synthesize. For example, one experiment may report that algorithm A has a mean absolute error of 0.7, while the other experiment may report a precision of 70%. How do these two algorithms compare? We cannot say based on this information from the two experiments; synthesis is not possible.

In particular, we can see this problem with ranked list evaluation of CF algorithms, where there is no emerging standard evaluation metric. Ranked list evaluation metrics attempt to measure the effectiveness of an algorithm at producing a useful list of top recommendations ranked by likely relevance or interest to the user. This contrasts with mean absolute error which measures overall prediction error. As an example, different ranked list metrics have been used by Breese et al. [4], Karypis et al. [13], Sarwar et al. [21], and Schein [24]. In this study, we limited our experiments to measuring prediction error; we leave ranked list evaluation for future work.

2.3 Challenge 3: Different Experimental Protocols

An experimental protocol includes the procedures used to train an algorithm to learn preferences, and the exact procedure to apply the evaluation metric. At a high level, experimental protocols for analysis of offline data (data previously collected) all follow a common procedure. This procedure can roughly be described as "withhold-and-predict". A ratings dataset is broken into two subsets, the learning set and the test set. The learning set is fed into the collaborative filtering algorithm as training data, which then predicts ratings or makes recommendations for items not in the training set. The accuracy of those predictions or ratings is evaluated based on the available ratings in the test set. Aside from this basic organization, there are many ways that experimental protocol can vary that can inhibit synthesis. These variances include:

a. **Treatment of recommendations for which test ratings are not available**. If the test protocol involves having an algorithm generate a "top-N" best recommendations list for each test user, then there is the situation that the algorithm may recommend an item for which we have no rating in the test set. This issue can be handled in several ways. Most commonly the experimental protocol will simply evaluate the top N recommended items for which there are test ratings. However, another approach is to assume a default negative rating.

b. **Treatment of missing or low confidence predictions**. Some algorithms are unable to produce predictions or recommendations for items when insufficient ratings data for those items is available. Several methods have been applied. The most obvious method is to ignore failed predictions, as done by [3,9]. Alternately, the set of ratings testing can be restricted to be only those that all algorithms could predict. Or a less accurate, less personalized algorithm, such as the average rating for an item, can be used to predict in situations where the primary algorithm fails to generate a prediction.

2.4 Challenge 4: Variance in algorithm implementation

The final of the four challenges is that when different scientists implement what they believe to be the same algorithm, the implementations commonly provide different results. This variance can occur for many reasons, including:

a. **Different interpretation of algorithm details**. In certain research publications, such as conference proceedings, the need for brevity almost guarantees that there will be insufficient space to describe all the details required to completely specify how to implement a particular algorithm. Thus, scientists trying to re-implement a previously published algorithm often will apply their own interpretation of how details should be handled.

b. **Algorithm tuning**. Some algorithms have many parameters; adjusting the parameters can cause the algorithm to respond differently to particularly inputs. Each scientist may tune the algorithm to meet a different need – using a different set of values for controlling parameters. Furthermore, scientists rarely publish in detail how algorithm parameters were tuned.

c. **Errors in the code implementing the algorithm**. It is very hard to detect errors in implementations of collaborative filtering algorithms unless the errors cause the algorithm to generate results that are highly improbable.

d. **Application of algorithm "enhancements".** Clean and simple abstract representations of algorithms communicate the best and are more readily accepted under peer review. Yet real world success often requires that these clean and simple algorithms be embellished, often with heuristics that cannot be justified theoretically. These enhancements are often not discussed in published work, yet certain enhancements are required to produce the optimal accuracy.

Variance in algorithm implementation is a considerable problem due to the dynamics of peer review. Scientific peer review culture rewards researchers who present new algorithms that outperform existing algorithm by some criteria. In order to gain acceptance for their new algorithm, researchers must implement one or more of the previously existing algorithms, and then show that their new algorithm out-performs them. These researchers are highly motivated to ensure that their new algorithm has no errors, and that it has the best enhancements applied. However, they have less incentive to ensure that the implementations of the competing algorithms are optimally implemented.

3 Case Study: An Empirical Comparison of Popular Algorithms

In spite of the aforementioned challenges (and at first, in some ignorance of them), we set out to synthesize 10 years of recorded knowledge about collaborative filtering (CF) algorithms through empirical experimentation. This consisted of comparing the accuracy of many proposed CF algorithms in a common controlled experimental setup. The primary research questions of this activity were as follows:

- Could we replicate the good performance claimed by the authors of each algorithm?
- Could we replicate the claims of relative performance made by authors of each algorithm?

- Could we establish a global ranking of algorithm quality, with respect to mean absolute error?

We evaluated a set of algorithms on two subsets of the EachMovie dataset (one with increased sparsity) as well as the Jester joke dataset. Our metric of evaluation was mean absolute error. In this section, we describe our experiment, reporting both our empirical results and the examples of the challenges we observed.

3.1 Descriptions of Algorithms Evaluated

We chose to evaluate primarily algorithms that are frequently cited in the literature, specifically algorithms the work on explicit ratings data, as well as a few additional algorithms we constructed. In some cases, we made minor modifications to existing algorithms in order to improve performance. For the purposes of repeatability, we describe all the tested algorithms in some detail here. A summary is provided in .

Code	Name	Implementation Reference	Modified
MIR	Mean Item Rating	[9] (as "average")	No
AMUR	Adjusted Mean User Rating	[9] (as "bias-from-mean average")	No
AMIR	Adjusted Mean Item Rating	*New*	-
CORR	Pearson r Correlation	[9]	No
VSIM	Vector Similarity	[4]	Yes
HORT	Horting	[3]	Yes
ITEM	Item-Item	[23]	No
BC	Bayesian Clustering	[4]	No
BN	Bayesian Network	[4]	No
CBN	Continuous Bayesian Network	*New*	-
PD	Personality Diagnosis	[18]	No

Table 1: Summary of all algorithms included in study.

3.1.1 Notation

In order to more precisely describe the operation of some of the algorithms, we define here a certain amount of mathematical notation for use in this section.

In this paper, U represents the set of all users and I represents the set of all items. Let U_i be the subset of U consisting of all users who have rated item i. Analogously, let I_u be the subset of I consisting of the items rated by user u. Let $r_{u,i}$ be the rating of user u on item i, if known. In this

document, the variables *i* and *j* refer to items and *u* and *v* refer to users. Let $\bar{r}_u$ be user *u*'s mean rating, and let $\bar{r}_i$ be the mean rating for item *i*.

The algorithms we investigated all compute a predicted rating for a user *u* on an item *i*, which we refer to as $p_{u,i}$. In this context, *u* and *i* may be termed the "active user" and "active item," respectively.

3.1.2 Non-personalized Algorithms

One of the simplest recommendation techniques is to recommend those items that are most popular. We refer to such algorithms as "non-personalized," since their ratings reflect the preferences of the entire user set more than those of the active user. The recommendations of these algorithms are analogous to the New York Times bestsellers list or weekly box office statistics. Since these algorithms are so simple and straightforward, we expect that more complicated algorithms should at least match their performance in predicting individual ratings.

3.1.2.1 Mean Item Rating (MIR)

The simplest algorithm we implemented uses the mean rating of the active item for its prediction, independent of which user is the active user: $p_{u,i} = \bar{r}_i$. This algorithm has been used as a baseline by Breese et al. [1998], Herlocker et al. [1999], Goldberg et al. [2000], and others, sometimes under the name "POP," short for "popularity." We refer to this algorithm as Mean Item Rating to distinguish it from other non-personalized algorithms.

3.1.2.2 Adjusted Mean User Rating (AMUR)

In examining different CF rating data sets, we have found that each user has a different distribution of ratings across possible rating values. For example, 80% of one user's ratings may have the value "4", while 75% of another user's ratings may have the value "3". One possible explanation for these variations in rating distribution is that the two users described may have had different perceptions of the rating scale. For example, one user's rating of "4" may indicate the same underlying preference as another user's rating of "3". We can account for this by using the offsets from each user's mean rating rather than their raw ratings. Herlocker et al. [1999] found that averaging these offsets and adding the active user's mean rating produced more accurate predictions than Mean Item Rating:

$$p_{u,i} = \bar{r}_u + \frac{\sum_{v \in U_i} (r_{v,i} - \bar{r}_v)}{|U_i|}$$

We refer to this as a normalization of ratings even though it is not a true normalization in the statistical sense[2]. Algorithms that use this kind of normalization assume that each user's mean rating represents a neutral preference, and that set amounts above or below that mean represent the same preference for all users. One can think of examples where this is not the case: for

[2] That would require dividing by the standard deviation of each user's rating distribution – thus creating a "normal" distribution. However, adjusting for differences in the width of a distribution (the std. dev.) has not shown to significantly improve prediction accuracy [Herlocker et al. 1999].

example, if some users only rate items they like, then a user's mean rating could be a poor indication of neutral preference.

3.1.2.3 Adjusted Mean Item Rating (AMIR)

An alternate normalization technique is to use mean item ratings rather than mean user ratings. By taking the Adjusted Mean User Rating algorithm and swapping users for items, we obtain a new algorithm with predictions generated by the following formula:

$$p_{u,i} = \overline{r_i} + \frac{\sum_{j \in I_u} (r_{u,j} - \overline{r_j})}{|I_u|}$$

This algorithm assumes that each user rates all items some constant amount above or below those items' mean ratings. For example, in this model, one user might rate every item 1 higher than its average, while another might rate every item 1 below its average. Thus, all users are still assumed to have the same overall preferences, though their individual rating scales may differ.

3.1.3 Nearest Neighbor Algorithms

The first algorithms used in collaborative filtering systems were nearest neighbor algorithms. With one exception, algorithms of this class generate predictions by first computing the similarity of the active user to each potential "neighbor" and then doing a weighted average of the most similar neighbors' ratings for the active item. The underlying theory is that users who have rated items similarly in the past are likely to do so in the future. These algorithms are all analogous to asking like-minded friends for item recommendations. The one exception to this is the Item-Item algorithm, which forms a neighborhood of items rather than users, but is otherwise quite similar to the user-based algorithms [23].

Some researchers have referred to this class of algorithms as "memory-based," because many of them require that all ratings be kept in memory in order to compute predictions [4,12,18]. However, as we show in Section 6, this is not always the case – variants of the Pearson r Correlation algorithm using sampling can be shown to be almost as effective as the original while using only a fraction of the memory.

3.1.3.1 Pearson r Correlation (CORR)

The Pearson r Correlation algorithm was used in some of the earliest collaborative filtering systems [19,25], yet it remains a popular baseline algorithm today, since it is easy to implement and fairly effective. In this algorithm, Pearson's r correlation coefficient is used to define the similarity of two users based on their ratings for common items:

$$sim(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \overline{r_u})(r_{v,i} - \overline{r_v})}{\sigma_u \sigma_v}$$

σ_u and σ_v represent the standard deviations of the ratings of users *u* and *v*, respectively. Both the rating averages ($\overline{r_u}$, $\overline{r_v}$) and standard deviations are taken over just the common items rated

by both users. In order to achieve the best possible implementation, we have used the modification suggested by Herlocker et al. [1999], which weights similarities by the number of item ratings in common between *u* and *v* when less than some threshold parameter γ:

$$sim'(u,v) = \frac{\max(|I_u \cap I_v|, \gamma)}{\gamma} sim(u,v)$$

This adjustment avoids overestimating the similarity of users who happen to have rated a few items identically, but may not have similar overall preferences. Such correlations may be high, but due to the limited amount of data, we have little confidence in them.

The adjusted similarity weights are used to select a neighborhood $V \subset U_i$, consisting of the *k* users most similar to *u* who have rated item *i*. If fewer than *k* users have positive similarity to *u*, then only those users with positive similarity are used. The ratings of these "neighbors" are combined into a prediction as follows:

$$p_{u,i} = \overline{r_u} + \frac{\sum_{v \in V} sim'(u,v) \times (r_{v,i} - \overline{r_v})}{\sum_{v \in V} |sim'(u,v)|}$$

3.1.3.2 Vector Similarity (VSIM)

The Vector Similarity algorithm considers each user's set of ratings as a vector and uses the cosine of the angle between two users' ratings vectors as a measure of their similarity [4]. More precisely,

$$sim(u,v) = \frac{\sum_{i \in I_u \cap I_v} r_{u,i} \times r_{v,i}}{\sqrt{\sum_{i \in I_u} r_{u,i}} \sqrt{\sum_{i \in I_v} r_{v,i}}}$$

As in Pearson r Correlation, a neighborhood *V* is formed consisting of the *k* users most similar to the active user that have rated the active item. (Breese et al. [1998] did not limit the number of neighbors, but we found this step to be very helpful.) A prediction is then computed as follows:

$$p(u,i) = \frac{\sum_{v \in V} sim(u,v) \times r_{v,i}}{\sum_{v \in V} |sim(u,v)|}$$

Breese et al. [1998] also proposed adjusting the similarity weight computation so that agreement about infrequently rated items would contribute more to two users' similarity than agreement about frequently rated items. However, we did not find this modification to be helpful in our experiments, so we did not include it in our experiments.

3.1.3.3 Horting (HORT)

One weakness Pearson r Correlation and Vector Similarity share is that in order for two users to be considered similar, they must have rated items in common. If only a few users have rated a given item, none of whom has much in common with the active user, then Pearson r Correlation and Vector Similarity might both be unable to produce a prediction. In theory, two users who rate items similarly to a third are likely to have similar taste, even though they may have rated no items in common. The Horting algorithm recognizes that indirect similarity by allowing neighbors to be acquired transitively [3].

In the Horting algorithm, each user is represented by a node in a directed graph, where a link from user *u* to user *v* means that user *v* "predicts" user *u*. Also stored with each link are two integers, $s \in \{-1,+1\}$ and $t \in Z$. These variables specify a linear transformation $L_{s,t}(r) = sr + t$ to normalize the target user's ratings with respect to the originating user's ratings. This allows users who rate items consistently higher, lower, or opposite of each other to predict each other. (Note that on a 0 to *n* rating scale, all "useful" values of *t* will actually lie between $-2n$ and $+2n$, since those offsets are sufficient to convert a minimal rating to a maximal rating, or vice versa, even when $s = -1$.) In practice, we did not find negative transformations ($s = -1$) to be helpful, so we did not use them.

Adjacencies in this directed graph are determined by two threshold requirements. The first establishes that the target user has rated a representative sample of the items rated by the originating user. The original authors called this requirement "horting," a new word derived from "cohorts," specific to this algorithm. User *u* is said to *hort* another user *v* if either *v* has rated some fraction α of the items rated by *u*, or if *v* has rated at least β of the items rated by *u*. α and β are both algorithm parameters. Mathematically, user *u* horts user *v* if $|I_u| \leq \alpha |I_u \cup I_v| / |I_v|$ or $|I_u \cup I_v| \geq \beta$. Note that this is not symmetric: if user *u* has rated 10 items, user *v* has rated those same 10 items plus 100 more, $\alpha = 0.2$ and $\beta = 20$, then *u* horts *v* but *v* does not hort *u*, since *u* has not rated a sufficient sample of the items rated by *v*.

The second threshold establishes that the target and originating user tend to rate items similarly, after taking into account different rating scales via the linear transformation $L_{s,t}(r)$. The prediction error *e* between two users is the average absolute difference of their common ratings. More precisely,

$$e(u,v,s,t) = \frac{\sum_{i \in I_u \cap I_v} |r_{u,i} - L_{s,t}(r_{v,i})|}{|I_u \cap I_v|}$$

If there exist *s* and *t* such that $e(u,v,s,t) < \delta$ for some prediction error parameter δ and user *u* horts user *v*, then user *v* is said to predict user *u*. This means that there is a link from user *u*'s node to user *v*'s with the variables *s* and *t* set to minimize $e(u,v,s,t)$. These optimal values for *s* and *t* can be found by calculating the prediction error *e* for each possible value of *s* and *t*.

The predicted rating for user *u* on item *i* is computed by searching through the graph at each distance level $l = 1 \ldots k$ and determining if there is at least one user in the graph within distance *l* of the user *u* that has rated item *i*. The predicted rating, $p_{u,i}$ is the average transformed rating given by all users at distance *l* who have rated item *i*, for minimum distance *l*. For users more

than one step away, transforms are composed. If no user of distance less than or equal to *k* from the active user has rated item *i*, then no prediction can be computed. This method should tend to use ratings from better predictors if possible, but will use worse ones as necessary.

On some datasets, we found that accuracy could be improved by adding two additional parameters: *m*, the minimum number of neighbors required, and *M*, the maximum number of neighbors allowed. Here, neighbors are those users whose ratings are aggregated to compute a prediction for a given item. While traversing the graph to make a prediction, if fewer than *m* neighbors have been found at a distance level of *l* or less, the algorithm will continue searching at the next level. Note that in this case, the neighbors whose transformed ratings are averaged to produce a prediction could come from two or more different levels. Once *M* neighbors have been found, the algorithm will average the transformed ratings of those *M* neighbors and terminate. Neighbors of lower prediction errors were considered first, to ensure that the best *M* predictors were used. If *m* is greater than 1, then these *M* predictors could be distributed over 2 or more distance levels. Note that the modified algorithm is equivalent to the original when $m = 1$ and $M = \infty$.

3.1.3.4 Item-Item (ITEM)

Each of the nearest-neighbor algorithms discussed so far finds users who have rated the active item and are similar to the active user. An alternate approach is to find items rated by the active user that are similar to the active item. Sarwar et al. [23] proposed several different algorithms that used similarities between items, rather than users, to compute predictions. These algorithms all assume that the active user's ratings for items related to the active item are a good indication of the active user's preference for the active item. Of the algorithms proposed by Sarwar et al., we only implemented adjusted cosine similarity, the algorithm Sarwar et al. [23] found to be most accurate; here were refer to it as Item-Item, since it is the only algorithm we tested that computes similarities between items. In this algorithm, the cosine of the angle between the item rating vectors is computed, after adjusting each rating by subtracting the rating user's mean rating. Specifically,

$$sim(i,j) = \frac{\sum_{u \in U_i \cap U_j} (r_{u,i} - \overline{r_u})(r_{u,j} - \overline{r_u})}{\sqrt{\sum_{v \in U_i} (r_{v,i} - \overline{r_v})^2} \sqrt{\sum_{w \in U_j} (r_{w,j} - \overline{r_w})^2}}$$

Note that unlike Pearson r Correlation, means are taken over all ratings for a user or item, not a subset of ratings shared with any other user or item. We found it helpful to adjust similarity weights based on the number of users in common, if the number of common users was below a certain threshold:

$$sim'(i,j) = \frac{\max(\gamma, |U_i \cap U_j|)}{\gamma} sim(i,j)$$

The predicted rating for a given user *u* and item *i* is computed using a neighborhood of items $J \subset I_u$ consisting of the *k* items rated by *u* that are most similar to *i*. If there are fewer than *k* items with positive similarity to *i*, then just those are used.

$$p_{u,i} = \overline{r_i} + \frac{\sum_{j \in J} sim'(i,j)(r_{u,j} - \overline{r_j})}{\sum_{j \in J} sim'(i,j)}$$

3.1.4 Probabilistic Algorithms

An alternate approach to the nearest neighbor methods is to learn a probabilistic model of the data, and use this model to predict ratings. Probabilistic algorithms tend to have more direct mathematical justification than nearest neighbor methods, given their assumptions of user behavior. Probabilistic algorithms have also been referred to as "model-based algorithms" [4,12,18], but we prefer the term "probabilistic algorithms". Nearest neighbor methods, such as the Horting algorithm, may build models as well, if only to represent neighbors.

3.1.4.1 Bayesian Clustering (BC)

Breese et al. [1998] proposed a simple probabilistic model for collaborative filtering, based on the assumption that there are distinct groups of users, each with fairly homogeneous taste throughout. For example, types of users who watch movies might include those who love action movies, those who love romantic comedies, those who love art films, and so on. Using machine learning methods, these different user groups can be learned automatically from the data. Then, in order to predict the rating for a particular user on a particular movie, we could simply average each user group's mean rating for that movie, weighted by the probability that this particular user is a member of that group.

We implemented the proposed Bayesian clustering algorithm as a naïve Bayes classifier, where each item rating is conditionally independent given user class, a hidden variable representing the user's preference type[3]. In this model, we store each probability that a user of a given class will assign a given rating to a given item. With the application of Bayes' rule, these probabilities are also sufficient to determine the probability that a user is a member of a given class. Note that this is only one of several probabilistic clustering models that have been proposed for collaborative filtering; for other models, see [27] and [11,12].

These probabilities are learned from the training data using a gradient ascent approach with a fixed number of iterations. First, the model is randomly initialized. Then in each iteration, each user is assigned to the most probable class based on previously rated items. Since the membership of each user class may have changed, user class probability distributions must be recomputed. Of course, once the user class probability distributions have changed, some users may no longer be in their most probable class, so the process repeats. The predicted rating for an item is an average of the expected values for each preference class multiplied by the probability that the active user is a member of that class.

---

[3] See [Mitchell 1997] for a more thorough explanation of Naïve Bayes classifiers and training through gradient ascent.

3.1.4.2 Bayesian Network (BN)

Breese et al. [1998] proposed using Bayesian networks for collaborative filtering. Each node in this model is a categorical variable representing an item, whose states cover every legal rating and "No Rating." The inclusion of a "No Rating" state allows the model to be learned with complete data even if no user has rated every item. The probability distribution for each item is modeled by a decision tree. We used the Microsoft Research's WinMine Toolkit to build all of our Bayesian networks [6].

For generating a prediction, all nodes in the network are instantiated with the ratings or lack thereof for the active user. The probability of the "No Rating" state is clamped to zero for the item in question and the probability distribution over all legal ratings is generated using Markov-blanket inference [4]. In Markov-blanket inference, the probability that a given variable has a given state is dependent on its parents (variables in the given variable's decision tree), its children (variables in whose decision trees the given variable appears), and its children's parents. The predicted rating is the resulting expected value.

Unlike all previously discussed algorithms, the Bayesian Network algorithm directly assumes that a missing rating (marked by the "No Rating" state) is an indication of preference (negative preference in this case). This is an interesting approach, with some logic behind it – the fact that you haven't watched a certain movie, for example, could indicate that you wouldn't be interested in watching other, similar movies.

This algorithm makes some additional assumptions regarding how user preferences may be effectively modeled: it assumes that each different rating for an item represents a distinct preference class (e.g. the algorithm doesn't know that 4 is closer to 5 than 1), that a user's rating for any given item depends only on that user's ratings of a few specific items in the dataset, and that this dependence can be effectively represented using decision trees.

3.1.4.3 Continuous Bayesian Network (CBN)

One of the weaknesses of the Bayesian Network algorithm used by Breese et al. [1998] is that it treats the training data as classification examples rather than numerical ratings. The decision trees it builds depend on having many training examples with identical ratings of several items in order to build the probability distribution at each leaf. It is difficult, however, to find many users who have given three or four items identical rating values, and thus most of the splits in the decision trees are on the "No Rating" state (about 97% in models built from our EachMovie dataset, described in Section 3.2.1.1). In other words, users' predictions are based largely on what they choose to rate, ignoring most of the actual ratings given. While the resulting model may be interesting to analyze, since it shows many simple relationships between items, it fails to take advantage of much of the information in the original data.

An alternative approach is to represent each item rating as a numerical, not categorical, variable in the network. To do this, we represented each item's rating as a binary Gaussian variable, either having the value "No Rating," or a real number representing an offset from a user's mean rating. The model is trained not on the raw ratings themselves (i.e., $r_{u,i}$), but on each rating's offset from its user's mean rating (i.e., $r_{u,i} - \overline{r_u}$). This algorithm has two advantages: first, it works with normalized ratings rather than raw ratings, to take into account differences between users' rating distributions; second, it treats ratings as interrelated numbers, rather than distinct classes. We call this modified algorithm Continuous Bayesian Network, since it closely

resembles the Bayesian Network algorithm in assumptions and implementation, but represents user ratings as continuous variables. Using this revised algorithm on the EachMovie dataset, we found that fewer than 55% of the decision tree splits were on "No Rating."

3.1.4.4 Personality Diagnosis (PD)

The Personality Diagnosis algorithm works on the assumption that the active user has the same true preferences as some other user, though the observed ratings may differ by Gaussian noise [18]. Unique to this algorithm is the idea that users have "true" ratings for each movie with differing observed ratings due to temporary moods and impulses. This algorithm is also unique in that it uses both a probabilistic approach and a nearest-neighbor framework: though it never computes a neighborhood directly, it does compute similarities and perform a weighted average over all ratings for the item. We include it among other probabilistic algorithms because of the methods it uses for computing the similarity.

The similarity between the active user *u* and some neighbor *v* is the probability that *u*'s "true" ratings are identical to *v*'s observed ratings. This is fairly straightforward to compute given the assumption that observed ratings differ from true ratings according to Gaussian noise with some variance σ^2. To predict a rating for the active user on the active item, the probability of each valid rating value is computed by summing the probabilities of all users who have given that rating value to the active item. The predicted rating value is the one with the highest probability, not the expected value, as in other probabilistic approaches.

3.1.5 Other Algorithms

There are algorithms we did not fully implement and investigate. Some were omitted due to time restraints, others because they claimed no improvement in predictive accuracy on explicit ratings data. These algorithms include RecTree[5], Dependency Networks[8], Eigentaste [7], Singular Value Decomposition [22], and Probabilistic Latent Semantic Analysis [12].

3.2 Experimental Methods Used in the Empirical Study

3.2.1 Datasets

We performed our experiments on three different datasets in order to cover some of the variation present in different collaborative filtering systems. The datasets we selected were those that were most available and most commonly used by other researchers. They are summarized in Table 2, and described in further detail in the subsections that follow.

Name	Users	Items	Ratings	Ratings/User	Density	Rating Scale
EachMovie	6,185	1,580	258,351	41.8	0.0264	0 to 5, integer
Sparse EachMovie	12,144	1,550	261,459	21.5	0.0139	0 to 5, integer
Jester	17,998	100	908,312	50.5	0.5048	-10.00 to 10.00, real

Table 2: Summary of datasets included in our investigation.

3.2.1.1 EachMovie

One of the largest datasets of explicit user preferences is EachMovie, a movie rating database collected over a period of 18 months by the Compaq corporation [16]. EachMovie contains the ratings of approximately 60,000 users for a set of 1,800 movies, 2.8 million ratings in all, or an average of 46 ratings per user. The rating scale ranges from 0 to 5.

Each rating also has a weight associated with it, which indicates whether the user saw the movie or merely clicked a button reading, "That looks awful." In our analysis, we only used ratings for movies the user actually saw, reducing the average number of ratings per user to 41.8. Finally, we used a subset consisting of all ratings for a random 10% of the users. We used a representative sample of the entire dataset to enable us to analyze and cross-validate more algorithm variants in the timeline of our project with the computation power available.

3.2.1.2 Sparse EachMovie

To better study the effects of sparsity on each algorithm, we artificially increased the sparsity of a subset of EachMovie data by selecting a random 50% of the ratings of a random 20% of the users in the complete EachMovie dataset. The resulting dataset has approximately the same number of ratings as the first, but spread out over twice as many users, resulting in half the density. As before, only explicit ratings were included.

3.2.1.3 Jester

The Jester dataset consists of ratings on 100 jokes by almost 18,000 users, over 900,000 ratings in all [7]. The ratings scale goes from –10.00 to +10.00, with increments of 0.01, yielding 200 distinct possible ratings in all. Each joke was rated immediately after being read by the user, using an image map with one extreme representing strong liking and the other strong dislike. Before receiving any recommendations, all users were required to first rate a "gauge set" of 10 jokes.

For a collaborative filtering dataset, this dataset is exceptionally dense. Since it takes fairly little time to read and rate a joke, and since only 100 jokes are available, a user could rate every joke in less than an hour. The average user actually rated about 50 jokes, but 50% density is much higher than the 2.6% density for EachMovie.

While importing the ratings into our database, we discovered that over 900 of the ratings were outside the allowed range of –10.00 to +10.00. On the advice of Goldberg [2003], these ratings were removed and not considered in our experiments.

The almost-continuous nature of these ratings could present difficulties for some algorithms. Bayesian Clustering and Bayesian Network all build models that compute the probabilities of a sub-population of users assigning each discrete rating to a given movie. For a dataset with 200 discrete ratings this is impractical: each probability would be very low, and many would be zero. The best way to use these algorithms would be to group the ratings into a smaller number of rating ranges, and then use the algorithms on the ranges rather than the raw ratings. Due to the time required to adapt these algorithms and select the optimal rating ranges, as well as their poor performance on the movie datasets, we chose not to test these algorithms on the Jester dataset.

The Horting and Personality Diagnosis algorithms were also designed to work on discrete data, but were included in the experiment anyway. Specifically, the Horting algorithm uses an integer offset t for normalizing one user's ratings with respect to another's; if we were to let t be

fractional instead, perhaps increased fineness would improve this algorithms performance on the dataset. The Personality Diagnosis computes the probability that a user's rating is each integer, and recommends the integer rating with the highest probability. This method thus introduces artificial coarseness when used on almost continuous data, and might perform better if it returned the expected value instead.

3.2.2 Metrics

Here we describe briefly our choice of evaluation metric. A complete discussion on appropriate metrics for collaborative filtering is beyond the scope of this article. We refer readers to [10], for a substantial discussion on the topic of evaluation of collaborative filtering systems. In this experiment, we applied two variants of the most popular accuracy metric – mean absolute error (MAE). MAE measures how close predicted ratings are to the true ratings. For a set of ratings in the test set, *T*, MAE is defined as follows:

$$\text{MAE} = \frac{\sum_{(u,i,r_{u,i})\in T} | p_{u,i} - r_{u,i} |}{| T |}$$

There are usually some ratings in *T* for which a given algorithm is unable to furnish a prediction. For example, when using the Pearson r Correlation predictive algorithm, if none of the users who rated the active item had rated any items in common with the active user, no prediction can be computed. In this situation, most scientists choose to simply remove that prediction from consideration, so that it doesn't affect the MAE of that algorithm; however in the extremes, this can lead to algorithms that avoid making errors by never predicting unless evidence is overwhelming. We have assumed in this study that high coverage – the percentage of ratings for which an algorithm can supply a prediction – is important. Thus we require that every algorithm provide a prediction for every item. To achieve this goal, whenever an algorithm cannot produce a prediction (usually because the computation does not consider all the data in the dataset), we instead supply the population average – the Adjusted Mean User Rating. To distinguish this evaluation approach from the traditional one, we refer to it as Augmented MAE, since it computes the Mean Absolute Error of a given algorithm after extending its coverage with an alternate algorithm. We also implemented the more traditional approach of omitting failed predictions from the average which we continue to reference as MAE. In the extreme case, if an algorithm refused to produce any predictions, its Augmented MAE would be equal to that of the Adjust Mean User Rating algorithm, while its MAE would be undefined.

Another consideration is that we can only evaluate the accuracy for the active user on items that the active user has provided a rating. Thus, it is possible that the item with the highest predicted rating may not be considered in the evaluation because we do not have the active user's "true" rating to compare against. This weakness will exist in any offline experiment where users have not rated all items.

3.2.3 Tuning Algorithm Parameters

One of the challenges of comparing so many different algorithms was ensuring that they were performing as well as reasonably possible. This required both correct implementation and careful tuning of parameters for each algorithm. Ensuring correct implementation can be very challenging, since a minor bug or oversight could easily disadvantage an algorithm in a slight,

but measurable way. A proper implementation also requires a complete understanding of the original algorithm.

We approached algorithm tuning with one simple goal: achieve for each algorithm implementation, at a minimum, a level of accuracy comparable to that reported by the initial authors of the algorithm. To accomplish this, we attempted to replicate other researchers' experiments whenever possible. For the Horting algorithm, this was not possible, since we did not have access to the synthetic dataset on which it was originally tested. For the other algorithms, we achieved close agreement between the accuracy of our implementations and the results reported by the algorithms' original authors.

To achieve well-tuned implementations of the different algorithms, extensive experimentation was done with each algorithm on a variety of test datasets to gain an understanding of the sensitivity of various parameters. Since an algorithm's optimal parameters may depend on the dataset, it is usually necessary to tune each algorithm to the given dataset. Tuning an algorithm's n parameters is equivalent to finding a global minimum in an n-dimensional space. To exhaustively test just 10 different values for each parameter would thus take 10^n different experiments. This is reasonable for algorithms with one or two parameters, but not practical for those with more, such as the Horting algorithm. We attempted to automatically search the parameter space using simulated annealing, but hand-tuning proved to be a faster and more reliable method. In practice, algorithms are not very sensitive to all of their parameters at once, and reasonable values may be obtained by educated experimentation, applying knowledge of the algorithm and dataset.

Figure 1 demonstrates our efforts at tuning the Personality Diagnosis algorithm for the EachMovie dataset. Since Personality Diagnosis only has one parameter, variance, this is both straightforward to do and easy to display. From our experiments, this algorithm appears to have minimum error on EachMovie at a variance of 2.5, the same value used by Pennock et al. [2000]. Tuning algorithms with more parameters is both more difficult to do and to display.

Algorithms were tuned using the first of the 10 cross-validation splitts used in the empirical evaluation of the algorithms. The parameters that yielded the lowest MAE were then used for the actual experiments. While we could not test every possible combination of parameters, we applied our best efforts to ensure each algorithm had the best representation.

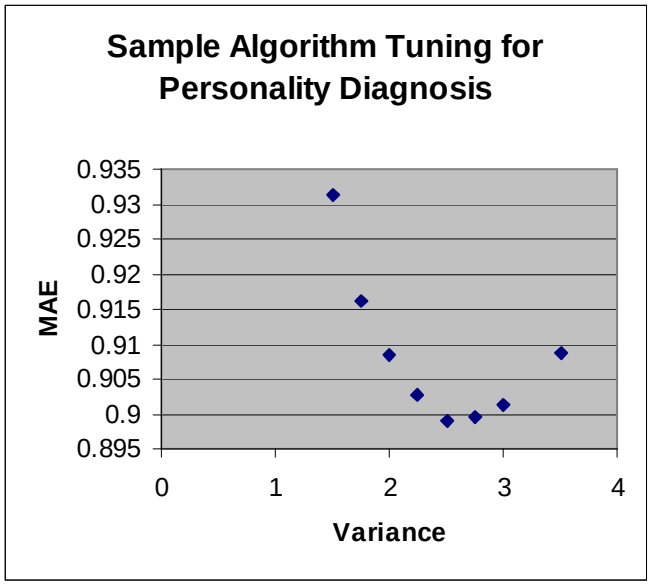

Figure 1: Effect of variance on the MAE for the Personality Diagnosis algorithm. Efforts were made to make each algorithm perform as well as possible on each dataset. This shows the different values of the variance parameter tried while tuning the Personality Diagnosis algorithm on the first split of the EachMovie data.

3.2.4 Experimental Protocol

The first stage of the protocol was to determine the best parameters for a given algorithm and dataset as described in Section 3.2.3. The next stage involved the comparison of the algorithms in an consistent experimental manner. We computed the Mean Absolute Error (MAE) and Augmented MAE (described in Section 3.2.2) using a ten-way cross-validation method. Each rating in the original dataset was randomly assigned to one of 10 subsets, so that each subset represented a random 10% of the overall dataset. For each experiment, nine of the subsets (90% of the ratings) were used for training and the remaining subset (10% of the ratings) was used for testing. By alternating which set was the test set, we generated 10 independent measurements of each algorithm on each dataset. We then sorted algorithms by their average subtest MAE scores and used a paired t-test on the 10 subtests to determine if differences between adjacent algorithms were statistically significant.

3.3 Results from the Empirical Study

The results of the empirical study are summarized in Figure 2, Figure 3, and Figure 4. Except for the Bayesian Clustering algorithm, MAE and Augmented MAE produced approximately the same results. Complete numerical results are listed in Table 3.

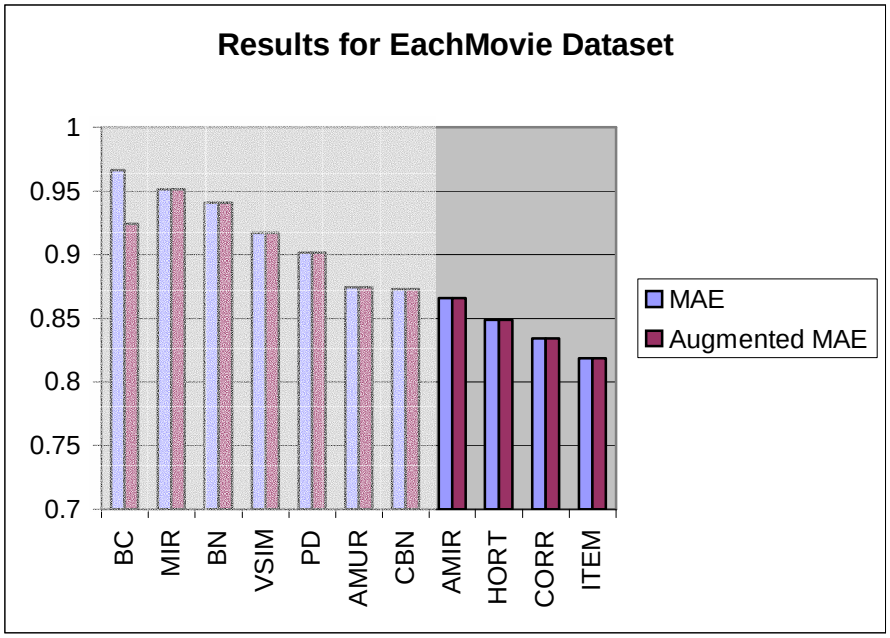

Figure 2: Algorithm accuracy using a random 10% subset of the EachMovie dataset. Lower is better. See Table 1 for the full names of the algorithms tested. All differences between adjacent algorithms in the graph are statistically significant at $p < 0.01$ except between CBN and AMUR. All algorithm results that are worse than a non-personalized algorithm (e.g., AMIR) are grayed out to indicate their poor performance.

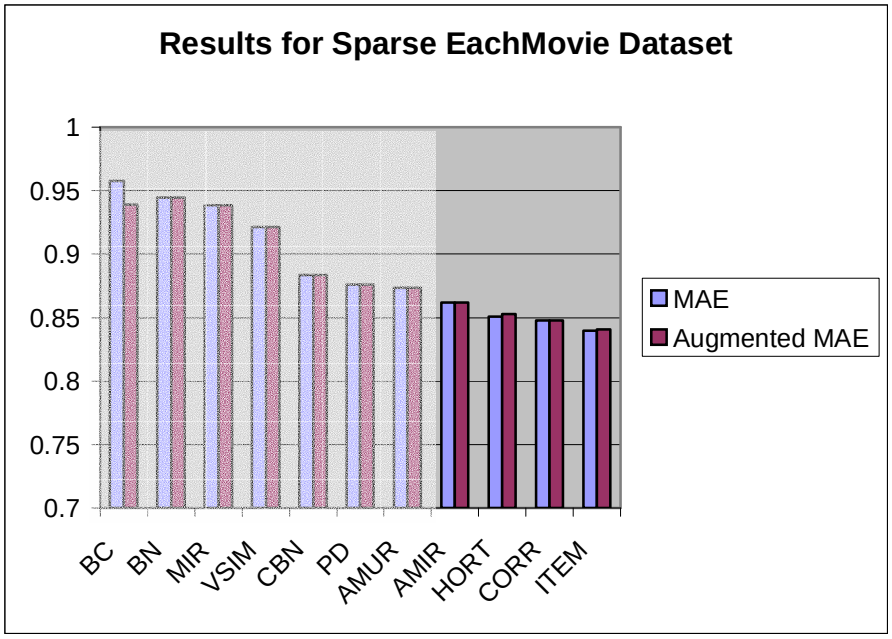

Figure 3: Algorithm accuracy on 50% of the ratings from 20% of the users in the EachMovie dataset. Lower is better. See Table 1 for the full names of the algorithms tested. All differences between adjacent algorithms in the graph are statistically significant at $p < 0.01$ except between AMUR and PD. All algorithm results that are worse than a non-personalized algorithm (e.g., AMIR) are grayed out to indicate their poor performance.

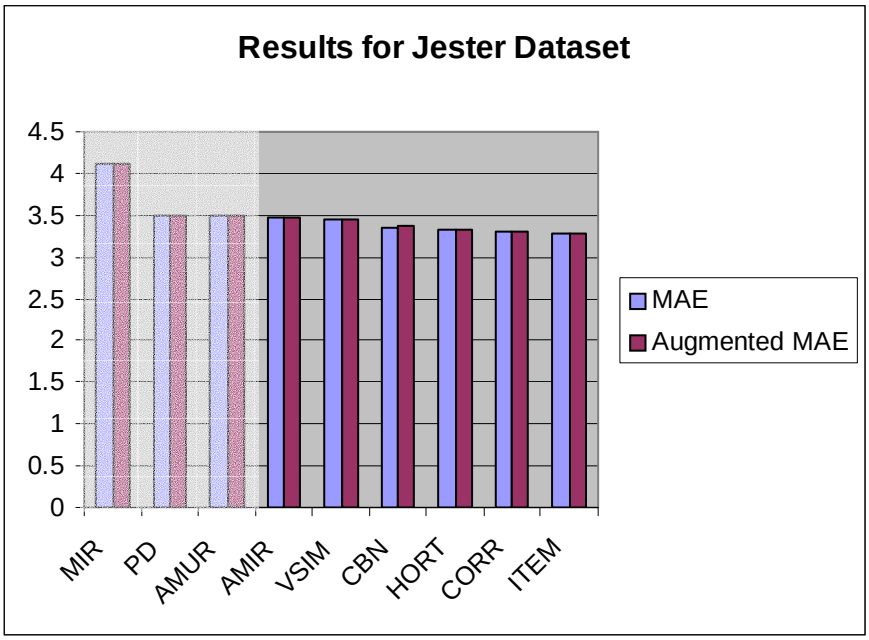

Figure 4: Algorithm accuracy on Jester dataset. Lower is better. See Table 1 for the full names of the algorithms tested. All differences between adjacent algorithms in the graph are statistically significant at $p < 0.01$, except for between AMUR and PD. All algorithm results that are worse than a non-personalized algorithm (e.g., AMIR) are grayed out to indicate their poor performance.

Algorithm	EachMovie Accuracy		SparseMovie Accuracy		Jester Accuracy	
	MAE	Aug. MAE	MAE	Aug. MAE	MAE	Aug. MAE
ITEM	0.8196	0.8200	0.8397	0.8406	3.274	3.275
CORR	0.8340	0.8342	0.8475	0.8479	3.301	3.301
HORT	0.8498	0.8499	0.8508	0.8526	3.324	3.331
AMIR	0.8663	0.8663	0.8619	0.8619	3.481	3.481
CBN	0.8752	0.8753	0.8834	0.8834	3.363	3.380
AMUR	0.8759	0.8759	0.8734	0.8734	3.488	3.488
PD	0.8971	0.8972	0.8759	0.8759	3.497	3.497
VSIM	0.9150	0.9150	0.9211	0.9212	3.438	3.438
BN	0.9400	0.9400	0.9446	0.9446	-	-
POP	0.9513	0.9513	0.9385	0.9385	4.106	4.106
BC	0.9756	0.9263	0.9577	0.9389	-	-

Table 3: Algorithm accuracy results over all datasets. Lower is better. See Table 1 for the full names of the algorithms tested.

3.4 Analysis of Empirical Results

On all of our datasets, Item-Item outperformed all other algorithms, though in some cases by a very small margin. Pearson r Correlation and Horting were consistently just behind. On a different dataset or with a different metric, we might observe a different ordering; nevertheless, the consistency across our datasets indicates that Item-Item is very effective at predicting individual ratings in these datasets.

The MAE and Augmented MAE metrics ranked the algorithms identically with one exception. Bayesian Clustering was unable to produce predictions of significant confidence on a significant number of test examples. Since the more accurate Adjusted Mean User Rating algorithm's predictions were used by Augmented MAE in these cases, Augmented MAE was lower than MAE. For all other algorithms, there were very few failed predictions, and thus Augmented MAE and MAE were very close.

Algorithm performance was fairly consistent, with relatively few changes in rank order among the three different datasets. Most algorithms did worse on the sparser version of EachMovie, but there were several exceptions. Notably, all non-personalized algorithms did better, perhaps because there were more total ratings in the sparse version. Personality Diagnosis and Bayesian Clustering also did better, though we are not sure of the causes. Since Personality Diagnosis shares certain similarities with probabilistic clustering methods such as Bayesian Clustering, this may suggest that clustering techniques depend more on the number of users than the number of ratings from each.

In all three datasets, the top performers were nearest-neighbor algorithms. There are several possible reasons for this. Since nearest-neighbor algorithms were introduced first, they have received more attention, while probabilistic algorithms are a more recent approach. Further study may lead to more competitive probabilistic algorithms. It's also possible that nearest-neighbor methods are simply are more natural approach to collaborative filtering, since they base their predictions on all user ratings (unlike Bayesian networks) and have simple analogs in the real world.

3.4.1 The Success of Item-Item

Why did Item-Item work so well in our experiments? That's hard to say; it is difficult to analyze in detail the recommendation biases of an algorithm that works by combining thousands or millions of ratings. We think that it may have something to do with the way people's preferences tend to relate to each other. It is unlikely that even two people with similar tastes will agree on all topics. For example, you might trust a friend's advice about movies, but not about music. Or maybe you only trust their recommendations for comedy movies. User-based algorithms, such as Pearson r Correlation and Horting, assume that users with common preferences about a few items have common preferences about all items. That may be true within specific domains, but is less likely to hold over a more diverse set of items. For example, if you rate several Star Trek movies highly, then that may be a good indication of certain science fiction tastes, but doesn't say very much about what movies you may enjoy from other genres. Unfortunately with user-based nearest-neighbor algorithms, once you've rated those movies, the average preferences of Star Trek fans will influence your ratings for all genres. Item-Item avoids much of this problem by looking for patterns among items, rather than users. It might observe, for example, that Star Trek movies are closely related to each other, but that they aren't at all

related to Marx Brothers movies. Your rating for a new movie is then computed based only on your ratings for related movies.

This approach has potential disadvantages, as well. An example best illustrates this. Suppose, hypothetically, that the movie Galaxy Quest, a self-aware parody of Star Trek, is loved by two main groups of people: those who love Star Trek movies and those who hate Star Trek movies. The "Trekkies" love Galaxy Quest because they know the original, and thus understand the references present in the parody. The "anti-Trekkies" love Galaxy Quest because it shows the absurdity present in something they detest. We suppose that all other people are indifferent to the movie, or dislike it. A user-based algorithm could match up Trekkies or anti-Trekkies and recommend Galaxy Quest to a member of either group. The Item-Item algorithm, however, would detect little relationship between Galaxy Quest and the Star Trek movies, since of the people who like Galaxy Quest, some love and some hate Star Trek. Thus, the Item-Item algorithm would fail to usefully recommend Galaxy Quest in this circumstance (we term this the "parody problem," though it could appear in a variety of domains.) However, since Item-Item does better than Pearson r Correlation, this situation does not appear to be a big factor in the datasets we used. The commonality and effect of this issue in other domains is an open question.

3.4.2 Issues with Horting

The Horting algorithm did slightly worse than Pearson r Correlation on all datasets. While tuning the algorithm, we found that the transitive neighbor property was of limited help on these datasets: we never found an advantage to using a path length greater than two. For datasets with much greater sparsity than any we tested, the transitive neighbor feature might give this algorithm an edge. In general, having a large number of neighbors seemed to be more important than having only the best neighbors – we achieved the best results by making the threshold requirements for neighbors fairly lax.

Overall, the Horting algorithm was a fairly challenging algorithm to tune due to its many parameters. In addition to the four parameters in the original algorithm [3], we found two additional parameters necessary for achieving the best possible accuracy. While the many parameters might seem to give one a great deal of power to select the best neighbors, this algorithm does worse than Pearson r Correlation, which is more easily tuned. Our hypothesis is Pearson r Correlation works better because it has a better measure of user similarity, statistical correlation, rather than a complicated set of thresholds. In addition, Pearson r Correlation features continuous normalization of user ratings, while Horting only selects the best integer offset. The Horting algorithm is also as susceptible to the same generalization issues that we suspect hinder Pearson r Correlation, namely that neighbors tend to share only a subset of a user's tastes but are used to predict a user's entire tastes.

3.4.3 Comparison of Probabilistic Methods

Of the probabilistic methods, Personality Diagnosis performed best in Sparse EachMovie and Continuous Bayes Net performed best on the more dense datasets. This suggests that Continuous Bayes Net may be more sensitive to sparsity than Personality Diagnosis, and perhaps more sensitive than nearest neighbor methods in general. This is actually not surprising – the decision trees utilized by the Bayesian network algorithms require that enough users have rated several items similarly. When each user has fewer ratings, the probability of this is much lower, resulting in a less accurate model. A possible solution to this might be to include estimated

ratings, generated using another algorithm, before training the Bayesian network. It remains to be seen if artificially increasing density in this manner would actually make the algorithm more effective. Of course, an alternate explanation is that Personality Diagnosis benefits from more users, since Sparse EachMovie had almost twice as many users as EachMovie.

The original Bayes Net algorithm proposed by Breese et al. [1998] did consistently worse than the Continuous Bayes Net algorithm. The original Bayes Net did a little bit better or a little bit worse than Mean Item Rating, while Continuous Bayes Net was nearer in performance to Adjusted Mean Item Rating, suggesting that normalization could have a consistently beneficial effect across algorithms. Perhaps if other algorithms, such as Bayesian Clustering and Personality Diagnosis were also normalized, they would be more competitive. The clustering algorithm recently proposed by Hofmann [2003] does include normalization, and it would be interesting to learn how it compares to the best of these algorithms.

3.5 Discussion of Case Study Challenges

Of the algorithms tested, Item-Item, Horting, Bayes Net, and Personality Diagnosis all claimed greater accuracy at predicting individual user ratings than some form of the Pearson r Correlation algorithm when they were originally published [3,4,18,23]. Of these, only Item-Item actually demonstrated greater accuracy in our experiments. Why did these algorithms work better then, yet perform worse than non-personalized algorithms in our experiments? Our belief is that either differences in dataset or testing methods make their results non-comparable to ours, or that differences in implementation make the comparison invalid. The situation is troubling either way: collaborative filtering experiments can easily be inconsistent or invalid! This observation led us to examine the causes of these discrepancies and anything else that got in the way of unifying current knowledge on collaborative filtering. We detail our findings from the case study in the remainder of this section.

3.5.1 Case Study Challenge 1: Different Datasets

One of the first algorithms we chose to implement was the Horting algorithm [3]. The algorithm featured several new, interesting ideas and reported very promising results: better coverage and half the error when compared to other commercial algorithms on a synthetic data set. When we tested this algorithm ourselves on public, real-world datasets, we found that its performance tended to be worse than that of Pearson r Correlation. Without access to the original authors' synthetic data, we could not replicate their experiments to verify the correctness of our implementation or understand specifically why the synthetic data worked so much better. All we could do is describe our experiences tuning and testing the algorithm, and the inferior results we observed.

As we researched other algorithms, we were pleased to find that most algorithms were tested using some sort of movie dataset. Of the algorithms we implemented, only Horting was never tested on movies. Unfortunately, not all algorithms were tested on the same movie dataset: experiments on Item-Item [23] and some of the work on Pearson r Correlation [9] were done using the MovieLens dataset, while most other researchers have been using the EachMovie dataset. Since MovieLens ratings are on a 1-5 scale and EachMovie ratings are on a 0-5 scale, error rates on the former dataset will tend to be much lower and cannot be compared directly. We chose EachMovie for our experiments since it seems to be more widely used, but tested our

Item-Item implementation on the MovieLens dataset as well, to verify the correctness of our implementation relative to the original authors'.

The EachMovie dataset is fairly large: 2.5 million ratings from tens of thousands of users. This is both an advantage and a disadvantage: a large number of ratings more closely represents the environments faced by some collaborative filtering systems, but it can also overwhelm some of the slower algorithms. The longer any individual test takes, the fewer tests can be run and the less research can be accomplished. Slower experiments are especially detrimental when tuning algorithms with many parameters. To combat this, many researchers [4,5,17,18] used subsets of EachMovie in their experiments. [5] used an EachMovie subset to focus on users who had rated a certain number of ratings to make predictions useful. This diversity forces the additional question: which subset?

We tried two different subsets of EachMovie in an effort to test these algorithms under a variety of conditions. What we found was that raw results, optimal tuning parameters, and even the experimental ranking of the algorithms depended on the subset used. For example, the algorithm Personality Diagnosis did better than Continuous Bayesian Network on the first EachMovie subset but worse on the second. This means that the choice of subset is a very important one, almost as important as dataset in regard to its effect on results.

Of course, experimental results that only apply to movie datasets are not very interesting. The only other public dataset of explicit ratings that we found was the Jester dataset. Since it covers jokes, not movies, and has almost 20 times the density of the EachMovie dataset, we figured it would add some diversity to our study. However, jokes and movies are only two of the many possible applications of collaborative filtering. How would these algorithms fare on an e-commerce dataset, perhaps the most common use of collaborative filtering? We would have liked to test these algorithms on half a dozen datasets or more, but they simply weren't available. A canon of standardized datasets (including smaller subsets for slower algorithms) would have helped us a lot.

Standardized datasets could also reduce the issue of non-comparable ratings: we found a fair number of ratings in the Jester dataset that were outside of the specified bounds. We chose to remove these ratings from consideration entirely; an alternate approach, used by [14], is to replace these ratings with the closest legal rating value. These different methods are not likely to have a large effect on the results, but they would make direct comparisons less certain. On the EachMovie dataset, we removed all ratings for which the weight was not 1, indicating that the user hadn't actually seen the movie but merely indicated a strong disinterest in it. Though these ratings contain useful information, they seem to be of separate class, and thus not quite equivalent to the rest of the ratings. While we chose to remove them, another researcher might not, again skewing the results slightly.

3.5.2 Case Study Challenge 2: Different Evaluation Metrics

For studies evaluating the accuracy of a given collaborative filtering system, two types of metrics are commonly used: predictive accuracy metrics, and ranked list evaluation metrics. Predictive accuracy metrics include MAE, mean squared error, 0-1 error, and other measurements of a system's effectiveness at predicting the rating a given user would assign to a given item. While a number of predictive accuracy metrics have been suggested, we were pleased to find a common denominator among almost all studies: the use of MAE. This commonality can also be misleading: the MAEs found by different studies may be incomparable due to other reasons,

such as differences in the dataset used, experimental methods, or algorithm implementations. Nevertheless, it is encouraging that at least one tool for CF research is already fairly standardized.

Unfortunately, MAE does not necessarily represent the total predictive utility of an algorithm. First of all, the metric has no measure of novelty or usefulness to users. It has been suggested that predictions of extreme interest or disinterest matter more than predictions of indifference, and should be weighted more highly. Of course, the extent to which this is true may depend on the domain, or even the specific user. This is also a weakness that most metrics will share.

More specific to MAE is the issue that, for discrete domains, a rating's most likely value is considered more accurate than the expected value. Suppose that given available evidence, the active user has a 70% probability of rating a given movie 5 and a 30% probability of rating it 4. The expected value is thus 4.7, with an expected MAE of $0.7 \cdot (5 - 4.7) + 0.3 \cdot (4.7 - 4) = 0.42$. The expected MAE of the most likely value, 5, is significantly lower: $0.7 \cdot (5 - 5) + 0.3 \cdot (5 - 4) = 0.30$. The MAE metric thus penalizes algorithms for returning expected values rather than most likely values, even though the expected value may contain more useful information for a user. We found this to hold true in practice as well as in theory on the EachMovie dataset. This is particularly disturbing since most collaborative filtering algorithms return some sort of average or expected value; of the algorithms we tested, only Personality Diagnosis did not.

If MAE is not the complete answer, what metric or combination of metrics is? While many have been proposed, we know of no study that evaluates a wide selection of metrics relative to user satisfaction, perhaps the ultimate test of an algorithm.

3.5.3 Case Study Challenge 3: Different Experiment Protocols

Whenever possible, we attempted to replicate the results of other researchers to ensure that our implementations were equivalent to theirs. The many different datasets used and the complexity of some algorithms certainly complicated this, but a surprising obstacle was differences in experimental methods. From the outset, we chose to train all algorithms using training sets that consisted of 90% of each user's data and test them on the remaining 10%. Other researchers use all ratings from a set of training users and test predictive accuracy on a separate set of training users with some or most of their ratings withheld.

Unfortunately, these two approaches induce somewhat different testing architectures. Consider the requirements of the typical nearest-neighbor algorithm: all training data must be present in memory, and we can only make predictions for users with at least one rating in the training set. This works fine for a train/test split, but complicates the second approach considerably: if the set of test users is distinct, then users must be added and removed from the training set as we test an algorithm. Furthermore, for model-based algorithms, do we relearn the model with the one extra user? By assuming that an extra test user would have a negligible impact on the model or by allowing the system to compute predictions for test users not in the training set, we can work around these differences and make either approach work. However, even simple workarounds take time.

This was the biggest methodological variation we observed, but more are possible. If researchers use different testing methodologies, then in order to reproduce previous results, any researcher will have to implement all of them, wasting valuable time. More likely, if reproducing

experiments is too inconvenient, then their results will never be verified, only cited. Either outcome is detrimental.

3.5.4 Case Study Challenge 4: Variance in Algorithm Implementation

Differences in data, metrics, and methodology would not be so great of an issue if those proposing new algorithms also tested all leading algorithms in the same scenario. Instead, each new algorithm is typically tested with only a couple other algorithms as baselines. When the testing scenario is different in one or more of the ways previously described, numerical results are not comparable to those of any previous work. While the new algorithm's performance relative to the selected baselines may be clear, it is hard to know how it would compare to other algorithms of interest. In the earlier example of Horting and Item-Item, the difference in dataset would not have been an issue if the Horting algorithm had been included in the experiments for Item-Item.

Unfortunately, researchers must invest a considerable amount of time on each additional algorithm, especially those that are more complicated or subtle. It is also easy to miss details of the algorithm that are essential to good performance. The only effective way to test an algorithm is to measure its performance and verify that it performs as well as previously published. For example, our first implementation of the Item-Item algorithm always used the best *k* similarity weights, even if some of them were negative, implying dissimilarity. This error was only uncovered when we attempted to replicate the original researchers' experiments and achieved much worse results. A careful rereading of the algorithm description suggested that perhaps only positive similarities were to be used, and additional experimentation verified that this was helpful. Had the experiment not used publicly available data and been fairly straightforward to reproduce, we might never have caught that bug.

We found indications of implementation differences among the works of other researchers as well. For example, [4] and [18] report significantly different results for Vector Similarity using exactly the same dataset and metric, suggesting a possible difference in the algorithm. Most significant and worrisome were implementations of the Pearson r Correlation algorithm, used by almost every researcher as a baseline. Common differences include failing to restrict the number of neighbors used, using neighbors with negative as well as positive similarity weights, and not weighting based on the number of common items. We have found these details to be essential for the best performance. Variance in implementations of this algorithm are most dangerous because it is commonly used as a standard. Non-standard implementations may even yield misleading results: several algorithms reported to outperform Pearson r Correlation did much worse against a well-tuned version in similar experiments.

4 Recommended Improvements to Methodology and Infrastructure

We've identified four challenges that are roadblocks to synthesizing published literature on collaborative filtering algorithms and presented a case study that illustrates examples of those challenges that we've encountered. In this section we discuss how the collaborative filtering scientific community could address these challenges in a manner that maximizes the effect of each published research result on the useful collected knowledge on CF algorithms. We will consider each challenge in turn.

4.1 Addressing the Challenge of Different Datasets

In a perfect world, collaborative filtering researchers would evaluate their new proposed algorithms on a wide variety of datasets. However, CF researchers are traditionally idea-rich and dataset-poor. They have many innovative ideas to improve CF algorithms, but do not have access to many datasets. For several years, the majority of published CF algorithm research papers used movie rating data. Every couple of years, one or two new high quality datasets become available, so there is a slow convergence towards the point where a sufficient variety of datasets are available, but we are still far from that point.

Testing on a Variety of Datasets. In the initial days of CF, it was sufficient to use one dataset to prove that an algorithm provided respectable results. We were proving the basic fact that CF can be effective, a fact that had consequences for a broad population of scientists and practitioners. Now much of the necessary work on CF algorithms now is much more incremental. In order for published reports of this work to be valuable to a broad population, and thus the greater collection of knowledge on CF, scientists must demonstrate that the incremental improvements demonstrated are generally applicable. The greater the differences between attributes of the different datasets tested, the more evidence that the improvements are generally applicable.

Standardized Data Subsets and Splits. As we described in Section 2.1, the availability of a common dataset doesn't always eliminate variability. Scientists must split the data into learning and test data sets, and often they will not use all available ratings to enable more rapid empirical evaluation. We suggest that when a dataset is released to the public, it should include several subsamples at varying sizes that researchers, and a variety of predefined splits between training data and test data. In this manner, we enable scientists to evaluate their algorithm on the **exact** same collection of ratings.

Releasing Previously Proprietary Datasets. One of the reasons that high quality CF datasets are so rare is that they require extended involvement of a large community of users. Scientists can't afford to use the traditional model of attracting participants via compensation. However, a substantial number of organizations provide web-based services that collect data that would be valuable to CF researchers. We need to convince these organizations to release this data to the scientific community.

4.2 Addressing the Challenge of Different Evaluation Metrics

Selecting a single standardized evaluation metric is probably not the answer, as the appropriate metric may differ depending on the context of recommendation [10]. However, we believe that a greater concentration of usage can be focused on a few well-chosen metrics by providing easy to use software for applying those metrics and generating easy-to-use accuracy reports. If this software was publicly available, and validated by scientists, it could increase the efficiency of CF algorithm analysis while serving to reduce the variety of metrics appearing in published literature. It would also enable scientists to more easily record and report results from a wider variety of metrics in the publications.

4.3 Addressing the Challenge of Different Experimental Protocols

At a high level, experimental protocols for evaluating CF algorithms are reasonably standardized. As described in Section 2, the withhold-and-predict approach is common. The variations in experimental protocol are found in the details of the experiments. One approach to

eliminating these variations is to create a more detailed standard testing procedure and document it very well. Unfortunately standards are rarely perfect; the standards committees frequently fail to reach consensus on the details of an approach, or the standard is not perfectly documented.

To address these failures in standardization efforts, some standards organizations will create a *reference implementation* of the standard. The reference implementation is a completely functional implementation of the standard that is made available to all parties. When an ambiguity is discovered within the standard, the ambiguity can be resolved by using the approach taken by the reference implementation.

We propose that reference implementations of *standard testing procedures* should be developed and made freely available to all. These standard testing procedures would consist of implementations of the metrics and protocols most commonly used in evaluating CF algorithms. The source code of the reference implementations would be also be publicly available and would be collaboratively maintained by all researchers working with collaborative filtering. Since different datasets and metrics test algorithms under different conditions, several different standard testing procedures could coexist, emphasizing different CF environments. For example, an e-commerce based standard testing procedure could include e-commerce data; a media recommendation standard testing procedure could include movie data and a ranked list metric. Researchers could then apply one or more metrics to proposed algorithms without having to re-implement every previous algorithm. System developers could select the best algorithms for their systems with confidence, based on the testing procedures that are closest to simulating their actual environments.

Such standard testing procedures would be fully automated, and any scientist could apply the procedure to any algorithm and achieve a numerical result that would be comparable to other results produced using the same procedure. This would solve the consistency problems entirely by replacing researcher-specific experimental protocols with community-standardized experimental protocols.

4.4 Addressing the Challenge of Variance in Algorithm Implementation

There are a variety of different actions that a researcher can take to reduce the probability of significant variations in results due to variations in implementation of the algorithm. Many of them are good procedures that should be generally followed. For example, researchers should ensure that their implementation of others' algorithms can achieve previously published results whenever possible by replicating published experiments. In the end, the only way to completely remove variation is when all researchers are using the same implementations of the algorithms.

Once again we propose that creating reference implementations is the best solution. Publicly available reference implementations of popular algorithms would not only reduce the barrier to performing high quality CF algorithm research, they would greatly increase the ability to synthesize results reported by different researchers on the same algorithm.

5 Shared Infrastructure for CF Algorithm Research

We have argued that many aspects of the challenges we have discussed can be surmounted by the creation of reference implementations. In order to efficiently make this a reality, there is a need for shared software infrastructure. At the core, this software infrastructure would include reference implementations of software for automated experimental protocols. In addition, this

software infrastructure would serve as a repository for reference algorithm implementations, where each algorithm implementation represents the best known implementation of that algorithm. As researchers develop new algorithms, they can contribute their implementations of those algorithms to the shared collection. Other researchers can then evaluate their new ideas against those algorithms in the collection, knowing that they are comparing against the best known implementations.

In addition to addressing the challenge of variability in algorithm implementation, a shared collection of algorithm implementations would provide substantial shared infrastructure to the research community. The collection of algorithm implementations would reduce barriers towards more exhaustive comparisons among different algorithms. Currently, when a researcher proposes a new algorithm, they will only compare it against a small number of competing algorithms, because the time necessary to implement and debug each of those competing algorithms makes such comparisons expensive. With freely available implementation algorithms, and a well documented software framework for interacting with those implementations, researchers would be able to run empirical experiments on many different algorithms in the time it would have taken them to implement one by themselves.

The sharing of algorithm implementations represents more repeatable science as well. Scientists will be more able to exactly repeat the experiments of others, and examine much closer the exact context of a previous experiment and discover more of the assumptions that were made in those experiments. This will also provide a forum for the recording of more incremental improvements that have been discovered, which are not usually recorded due to the academia's distaste for publication of incremental results. Improvements that are often seen as incremental, particularly those that are runtime performance related, are often substantially important to the practitioner.

5.1 CoFE – A Shared Framework for Collaborative Filtering Algorithm Analysis

As a seed effort to create a shared collection of collaborative filtering algorithms, we have created a software package, currently known as CoFE, which is short for **Collaborative Filtering Recommendation Engine**. CoFE was designed with three key goals in mind:

1. Create a software toolkit that allows easy creation of new collaborative filtering algorithms and a framework that enables easy and repeatable empirical analysis of such algorithms.
2. Create a collection of well-tuned reference implementations of proposed algorithms to be distributed with the software toolkit.
3. Create a collaborative filtering recommendation engine that has sufficient stability and performance to support small to medium web sites and other personalization applications.

CoFE is now a freely available shared resource for researchers and practitioners involved or interested in collaborative filtering-based recommender systems [2]. We have written the system entirely in Java to ensure that it is highly portable and requires no commercial software to run.

The CoFE software provides a software framework that simplifies the creation of new algorithms. CoFE is structured in object-oriented fashion and provides code to support functionality common to all algorithms, primarily data access and manipulation commands. To

create a new algorithm only requires implementing an interface containing a small number of well-defined methods. The effort of programming a new algorithm is greatly reduced.

CoFE provides a simple protocol for requesting recommendations or predictions from the currently active algorithm. This allows a common set of analysis tools to be developed independently of the algorithm implementations. The goal is to distribute these tools with CoFE to enable their use as shared infrastructure.

Towards the goal of creating a collection of well-tuned reference implementations of CF algorithms, we have implemented and distributed implementations of all the algorithms discussed in the case study from Section 3. In each case, we have tuned the algorithm until we were able to recreate the results in the original publications that presented the algorithms. We hope that the proponents of those algorithms will continue to contribute new versions of the algorithms with the latest improvements.

CoFE is implemented entirely in Java and should run on any platform that fully supports Java. It provides a simple API that can be used from Java via Remote Method Invocation (RMI) or from most other languages via the CORBA distributed object framework. The current API of CoFE is shown in Figure 5, while a diagram of the CoFE architecture is shown in Figure 6. New algorithms can be easily created in Java by implementing the Algorithm Java interface which is shown in Figure 7. Support for data access is provided to algorithms through a DataManager class, which handles all loading and indexing of ratings data.

```
public interface CoFE {
   public void setRating(int user, int item, float rating)
   public int setRatingList(ItemRating[] newRatings)

   public void removeRating(int user, int item)
   public void removeRatingList(int user, int[] itemIDs)

   ItemRating getRating(int userID, int itemID)
   public ItemRating[] getRatingList(int userID, int[] itemIDs)

   public ItemPrediction getPredictedRating(int userID, int itemID)
   public ItemPrediction[] getPredictedRatingList(int userID, int[] itemID)

   public ItemPrediction[] getRecommendations(int curUser, int number, int offset)
   public ItemPrediction[] getRecommendationsByType(int curUser, int number, int offset, int type)

   public ItemRating[] getUserRatingList(int userID)
   public ItemRating[] getItemRatingList(int itemID)

   public int getNextUserId()
}
```

Figure 5: The CoFE application programming interface, slightly simplified for readability (some minor informational methods like getMaxRating() not depicted; throws clauses removed for readability)

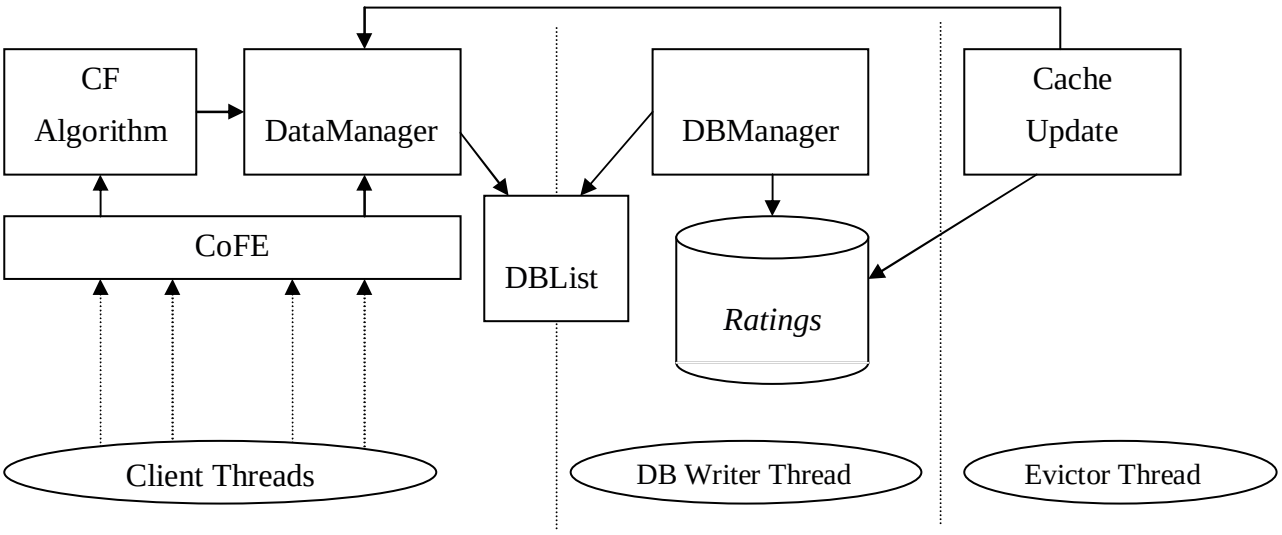

Figure 6: The architecture of the CoFE Collaborative Filtering Recommender System. One thread is created for each client request. Those threads perform the computation and perform write only to the internal cache, performing no I/O. One server thread is responsible writing all cache updates to the database, and one server thread is responsible for evicting cache entries when the cache fills up.

```
public interface CFAlgorithm {
  public ItemPrediction predictRating(int userID, int itemID)
  public ItemPrediction[] getRecommendations(int activeUser, int n)
  public ItemPrediction[] getRecommendationsByType(int activeUser, int n, int type)
  public void updateUser(int userID)
}
```

Figure 7: The Java interface that new algorithms must implement in CoFE. The actual interface has been simplified for readability (exceptions thrown are not listed).

6 A High-Performance, Open-Source, CF Recommender System

One of the three goals of the CoFE recommender engine framework was to support a production-worthy, high performance implementation of a collaborative filtering-based recommendation engine. We have iterated over the architecture and data structures of CoFE until we met that goal. The result is that CoFE can support a sustained load of hundreds of recommendation requests per second given an off-the-shelf desktop computer and a dataset where all the ratings can be loaded into main memory. For datasets that are larger than the main memory of a computer, we have implemented sampling algorithms that select a subset of the data that will fit in memory, while maintaining approximately the same level of accuracy as using the entire data set.

We chose to support the popular nearest-neighbor-based algorithm with similarities computed by Pearson correlation [9] as the high performance CF prediction algorithm. This algorithm is described in Section 3.1.3.1. At the time we started building CoFE, this particular algorithm had been shown as having the best performance in multi-valued rating datasets when evaluated using mean absolute error [4,9,9] and performance comparable to the best performance with unary or

binary datasets. Given the results in this paper, we will likely implement a high-performance version of the Item-Item algorithm in the future.

The nearest neighbor is what Breese et al. refer to as a memory-based algorithm [1998]. For every prediction, the algorithm requires access to every rating; it must “remember” all ratings. To support high performance (100+ predictions/second) with memory-based algorithms, CoFE stores all ratings in RAM. This approach is necessary because any regular disk access latency will prevent the algorithm from maintaining the high throughput of predictions.

Theoretically, memory-based algorithms, in particular nearest neighbor algorithms, cannot maintain acceptable performance as the number of users, items, or ratings scales up to large values. In order to determine the users who have the most similar preferences to the active user, the recommendation engine must examine every single user represented in the ratings. As the number of users increases, there is a corresponding increase in computation time. Also, as the number of items rated per user increases, there is a corresponding increase in computation time.

To overcome this theoretical limit on the scalability of the memory-based algorithms, we have implemented several methods that select a subset of ratings to be used for computation. Ratings in the subset that are selected are loaded into memory and used for computing predictions. Ratings that are not selected are completely hidden from the CF prediction algorithm. By keeping the size of the subset of ratings exposed to the CF prediction algorithm constant, we can maintain a high performance rate regardless of the size of the underlying complete dataset.

Ratings are sampled at the granularity of users. In order words, each user is either sampled and all of their ratings exist in memory, or they are not sampled, and none of their ratings are kept in memory. If a user whose ratings have not been sampled into memory requests recommendations, their ratings are temporarily loaded into memory. However, this only needs to occur once per session, regardless of the number of predictions or recommendations requested.

One could hypothesize that by only considering a small sample of the total dataset, we risk significantly reducing the accuracy of the prediction generated by the CF algorithm. However, we have learned through empirical experiments that if we carefully chose the user that we sample, then the incremental value (in terms of predictive accuracy) of additional data quickly approaches zero for available datasets. This is illustrated by the learning curve presented in Figure 8. These empirical findings lead us to believe that the quality of our recommendations will remain high in spite of the reduced size of data.

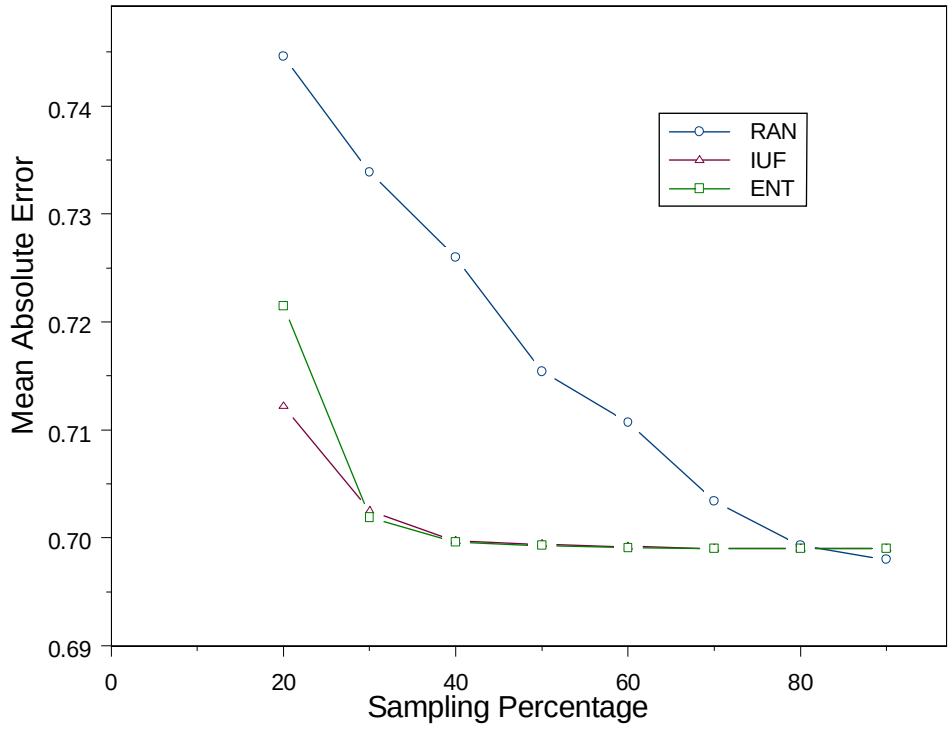

Figure 8: Learning curves using three different methods for selecting the users in a sample. RAN selects users at random. IUF stands for Inverse User Frequency and ENT stands for Entropy, two methods for estimating the "predictive value" that a user might provide if sampled. In the latter two cases, the user with the maximum predictive value is selected at each step.

The sampling approach allows good scaling as the number of users increases, but has limitations as the number of items scales to large numbers. For example, if we want to be able to recommend every single item with some confidence, then you need to sample a reasonable number of users that have rated every single item. Thus as the number of items increases, the number of users you need to sample increases, and eventually you reach a point where you have too much data to maintain high recommendation throughput. However, this problem can be addressed in most situations by limiting the items that can be recommended. For example, a retailer might limit recommendations to items that have been released recently, or even items that are more likely to be in stock.

7 Conclusion

From many different angles, collaborative filtering appears to have the potential to dramatically improve the effectiveness of information search and filtering. CF can provide recommendations based on deep understanding of content, while avoiding the very hard problem of deep machine understanding of content. In spite of this great potential, CF-based recommendation engines have not become pervasive even with almost ten years of research.

In this paper, we have presented some of what we have learned with respect to the synthesis of collaborative filtering research results. We have identified we believe are central challenges and

we believe that many of solutions to these challenges lie in the creation of reference implementations of both algorithms and experimental procedures whose maintenance is shared by the community. In an attempt to jumpstart the creation of such shared reference implementations, we have developed and released CoFE – a software framework to support collaborative filtering research. The source code for CoFE has been made publicly available and freely redistributable. We hope that CoFE will either be embraced by the CF research community for shared development of reference implementations or at least will provoke further discussion and development.

Finally, we have used the opportunity of developing a CF algorithm framework to develop a high performance implementation of a CF recommender engine, which has been released with the rest of the CoFE code. This will allow researchers and practitioners to experiment with using recommender engines within applications at no cost.

Acknowledgements

We would like to thank Dan Cosley for his extensive comments on an earlier version of this paper. We also thank Ken Goldberg for providing the Jester dataset and Compaq for providing the EachMovie dataset. This research was funded in part by National Science Foundation (NSF) award #0133994, with help from the NSF Research Experiences for Undergraduates program.

References

1. GroupLens Research Web Site. http://www.grouplens.org/ . 2004.

2. Oregon State University: Jon Herlocker (web site). http://eecs.oregonstate.edu/~herlock . 2004.

3. Aggarwal, C. C., Wolf, J. L., Wu, K.-L., Yu, P. S., 1999. Horting Hatches an Egg: A New Graph-Theoretic Approach to Collaborative Filtering. *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

4. Breese, J. S., Heckerman, D., Kadie, C., 1998. Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*. Morgan Kaufmann, San Francisco. (pp. 43-52).

5. Chee, S. H. S., Han, J., Wang, K., 2001. RecTree: An Efficient Collaborative Filtering Method. *Lecture Notes in Computer Science: Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001*. Springer Verlag, Heidleberg.

6. Chickering, D. M., 2002. The WinMine Toolkit. Microsoft Research MSR-TR-2002-103.

7. Goldberg, K., Roeder, T., Guptra, D., Perkins, C., 2001. Eigentaste: A Constant-Time Collaborative Filtering Algorithm. *Information Retrieval*, 4 (2), 133-151.

8. Heckerman, D., Chickering, D. M., Meek, C., Rounthwaite, R., Kadie, C., 2000. Dependency Networks for Inference, Collaborative Filtering, and Data Visualization. *Journal of Machine Learning Research,* 1 49-75.

9. Herlocker, J. L., Konstan, J. A., Borchers, A., Riedl, J., 1999. An algorithmic framework for performing collaborative filtering. *Proceedings of the 22nd International Conference on Research and Development in Information Retrieval (SIGIR '99)*. ACM Press, New York. (pp. 230-237).

10. Herlocker, J. L., Konstan, J. A., Terveen, L., Riedl, J., 2004. Evaluating Collaborative Filtering Recommender Systems. *Transactions on Information Systems,* 22 (1), 5-53.

11. Hofmann, T., 2001. Learning What People (Don't) Want. *Lecture Notes in Computer Science: 12th European Conference on Machine Learning*. Springer-Verlag, Heidelberg. (pp. 214-225).

12. Hofmann, T., 2003. Collaborative filtering via gaussian probabilistic latent semantic analysis. *Annual ACM Conference on Research and Development in Information Retrieval*. ACM Press, New York, NY. (pp. 259-266).

13. Karypis, G., 2001. **Evaluation of Item-Based Top-*N* Recommendation Algorithms**. *Proceedings of the tenth international conference on Information and knowledge management*. ACM Press, New York, NY. (pp. 247-254).

14. Lemire, D., 2003. Scale and Translation Invariant Collaborative Filtering Systems. *Information Retrieval,* To appear.

15. Linden, G., Smith, B., York, J., 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. (pp. 76-80).

16. McJones, P., DeTreville, J., 1997. Each to Each Programmers Reference Manual. Digital SRC Technical Note 1997-023.

17. Miyahara, K., Pazzani, M. J., 2000. **Collaborative Filtering with the Simple Bayesian Classifier**. *Pacific Rim International Conference on Artificial Intelligence (PRICAI)*. (pp. 679-689).

18. Pennock, D. M., Horvitz, E., Lawrence, S., Giles, C. L., 2000. Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-Based Approach. *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2000)*. Morgan Kaufmann, San Francisco. (pp. 473-480).

19. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J., 1994. GroupLens: An open architecture for collaborative filtering of netnews. *Proceedings of the 1994 Conference on Computer Supported Collaborative Work*. ACM Press, New York. (pp. 175-186).

20. Resnick, P., Varian, H. R., 1997. Recommender Systems. *Communications of the ACM,* 40 (3), 56-58.

21. Sarwar, B. M., Karypis, G., Konstan, J. A., Riedl, J., 2000. Analysis of Recommendation Algorithms for E-Commerce. *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC'00)*. ACM Press, New York. (pp. 285-295).

22. Sarwar, B. M., Karypis, G., Konstan, J. A., Riedl, J., 2000. Application of Dimensionality Reduction in Recommender System--A Case Study. *ACM WebKDD 2000 Web Mining for E-Commerce Workshop*.

23. Sarwar, B. M., Karypis, G., Konstan, J. A., Riedl, J., 2001. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th International World Wide Web Conference (WWW10)*.

24. Schein, A. I., Popescul, A., Ungar, L. H., Pennock, D. M., 2002. Methods and Metrics for Cold-Start Collaborative Filtering. *Proceedings of the 25th Annual international ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, New York, NY.

25. Shardanand, U., Maes, P., 1995. Social Information Filtering: Algorithms for Automating "Word of Mouth". *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*. ACM, New York. (pp. 210-217).

26. Terveen, L., Hill, W., Amento, B., McDonald, D., Creter, J., 1997. PHOAKS: A System for Sharing Recommendations. *Communications of the ACM*, 40 (3), 59-62.

27. Ungar, L. H., Foster, D. P., 1998. Clustering Methods for Collaborative Filtering. *Proceedings of the 1998 Workshop on Recommender Systems*, 114-129.