ACKNOWLEDGMENTS

I would like thank Dr. Jon Herlocker for his guidance and help in my research. His ideas and input have aided me greatly in the exploration of ideas. I would like to thank my committee members, Dr. Margaret Burnett and Dr. Prasad Tadepalli, for their feedback and support.

I would also like to the members of the CoFE group, whose input and development of the CoFE system have greatly aided the research in this area. In particular, I would like to thank Min Yang for her early work on distribution based algorithms.

I would like to thank my family support through the never ending process I call school. Finally, I would like to thank Suzanne, Joe, and Kevin for making college tolerable at the worst of times and highly entertaining at the best.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

A Collaborative Filtering Algorithm and Evaluation Metric that Accurately Model the User Experience

1. INTRODUCTION

Expansion of the internet and other sources of digital media have provided people with access to a wealth of information. This great resource has brought with it the problem of sorting through enormous amounts of information to find the pieces relevant to the user; often referred to as "information overload". There have been a variety of solutions proposed to aid users with this burden. Prior to the digital era, people often relied on the opinions of a network of friends with similar taste to filter information and decide what was relevant to them. Collaborative filtering expanded and improved upon this idea. A collaborative filtering system stores the preferences and opinions of the thousands of users of the system. These opinions are recorded as ratings by users for items. When an active user would like a recommendation, the system finds users with similar taste and uses their opinions to generate a recommendation. This methodology has proved to be very successful in a variety of domains, especially domains where multi-value ratings data are available.

While there has been almost a decade of research in collaborative filtering (referred to as CF), we believe that the user experience provided by current CF solutions could be improved upon substantially. In particular, we believe that some of the most popular current predictive algorithms create a poor and ineffective user experience.

The CF algorithm is the central computation of a CF system. The algorithm computes which items a user will like and possibly how much he/she will like them. CF algorithms are most often used to produce the "best bets" for a user, recommending a single best item or a ranked list of top items for a user. We refer to this as a top-N recommendation; web-based recommender systems are commonly asked to predict the top-N items where N is usually between 1 and 20.

In our practical experience deploying CF systems, we have observed that nearest-neighbor algorithms, which have repeatedly demonstrated the best overall prediction accuracy for multi-value rating data, are plagued with obscure or highly inaccurate recommendations at the top positions [1,3]. In other words, when using a nearest-neighbor algorithm to generate the top 50 movies that you are most likely to see, we expect most of the first 10-20 recommendations to be far from the user's interests.

The poor performance of nearest neighbor algorithms operating in top-N mode when N is small can be attributed to several factors, but is primarily due to a lack of sufficient rating information on the items recommended.

There are two interacting reasons why such blatant flaws have escaped scientific notice for such a long time. The first reason is that Mean Absolute Error (MAE), the most commonly used metric, evaluates algorithms on a per prediction basis, biasing performance results towards algorithms which predict all items well, not necessarily recommend top items well. The second is that most of the published results on CF algorithms come from analysis of offline datasets and not experiences with real users who would quickly detect these problems.

In this article, we will demonstrate flaws of two of the most popular algorithms We will propose the use of Precision/Recall evaluation techniques in addition to MAE to measure performance in order to help better estimate user experience in the future. Finally we will propose a new algorithm that demonstrates most of the good qualities of nearest-neighbor algorithms while offering far superior performance in top-N Precision/Recall metrics.

2. Flaws with popular CF algorithms

Nearest neighbor (NN) algorithms are the most frequently cited and possibly the most widely implemented CF algorithms. They consistently are rated the top performing algorithms in a variety of publications; however, we propose that they contain serious flaws. While using NN algorithms to generate movie recommendations with the EachMovie [5] or MovieLens [2] datasets, we found that many of the top movies recommended were terrible, highly questionable, or unverifiable. This leads us to believe that these algorithms perform poorly where it matters most in recommendations. We also believed that limitations of the MAE metric were responsible for concealing these flaws. We designed an experiment to test these hypotheses.

2.1 Assumptions

In designing our experiment we made several assumptions about the usage of CF solutions and the user experience.

- The CF algorithm is being used to generate the top-N recommendations of what each user should view, buy, etc.
- The user is unlikely to view any recommendations beyond the top-20 best bets.
- Very obscure items are “bad” recommendations for the majority of users. Example: obscure movies released in foreign countries that cannot be found on video or DVD in the US are bad recommendations for a person living in the US.
- A user’s confidence in the CF recommendations is dependent on the user seeing recommendations for items the user already knows are good. Example: given 10 recommended movies, a new user is more likely to believe the recommendations if they recognize 4 of the movies to be good than if the user does not recognize any of them. The 4 known movies act as proof of quality to the users.

We believe these assumptions to be valid for many of the domains where CF recommendations are used.

2.2 Experiment Design

To test our hypothesis we chose two of the most commonly used and best performing nearest neighbor algorithms: User Nearest Neighbor and Item Nearest Neighbor.

2.2.1 User Nearest Neighbor

The User Nearest Neighbor algorithm, based on the Pearson r Correlation, coefficient was used in some of the earliest collaborative filtering systems [6,9], yet it remains a popular baseline algorithm today, since it is easy to implement and demonstrates high accuracy when measured with mean absolute error. We refer to this algorithm as the *User-User* algorithm, because at its core is the computation of similarity between each pair of users. In this algorithm, Pearson’s correlation coefficient is used to define the similarity of two users based on their ratings for common items:

$$sim(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \overline{r_u})(r_{v,i} - \overline{r_v})}{\sigma_u \sigma_v} \quad \text{Eq. 1}$$

σ_u and σ_v represent the standard deviations of the ratings of users *u* and *v*, respectively. I_u is the set of items rated by user u while $r_{u,i}$ is the rating of user *u* on item *i*. Both the rating averages ($\overline{r_u}$, $\overline{r_v}$) and standard deviations are taken over the common items rated by both users. In order to achieve the best possible implementation, we have used the modification described in Herlocker et al. [3], which weights similarities by the number of item ratings in common between *u* and *v* when less than some threshold parameter □:

$$sim'(u,v) = \frac{\min(|I_u \cap I_v|, \gamma)}{\gamma} sim(u,v) \qquad \text{Eq. 2}$$

This adjustment avoids overestimating the similarity of users who happen to have rated a few items identically, but may not have similar overall preferences.

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*. U_i is the set of all users 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:

$$p_{u,i} = \overline{r_u} + \frac{\sum_{v \in V} sim'(u,v)(r_{v,i} - \overline{r_v})}{\sum_{v \in V} |sim'(u,v)|} \qquad \text{Eq. 3}$$

2.2.2 Item Nearest Neighbor (ITEM)

The nearest-neighbor algorithms introduced above finds users who have rated the active item and have interests similar to the active user. Using the same data, an alternate approach is to find items rated by the active user that are similar to the item being predicted (the active item). Sarwar et al. [8] 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. [8], we only implemented adjusted cosine similarity, the algorithm Sarwar et al. [8] found to be most accurate; here we 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 neighbor's mean rating. Specifically,

$$sim(i,j)=\frac{\sum_{u\in U_i\cap U_j}(r_{u,i}-\overline{r_u})(r_{u,j}-\overline{r_u})}{\sqrt{\sum_{v\in U_i}(r_{v,i}-\overline{r_v})^2}\sqrt{\sum_{w\in U_j}(r_{w,j}-\overline{r_w})^2}} \quad \text{Eq. 4}$$

Note that unlike User-User, 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:

$$sim'(i,j)=\frac{\min(\gamma,|U_i\cap U_j|)}{\gamma}sim(i,j) \quad \text{Eq. 5}$$

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.

$$p_{u,i}=\overline{r_i}+\frac{\sum_{j\in J}sim'(i,j)(r_{u,j}-\overline{r_j})}{\sum_{j\in J}sim'(i,j)} \qquad \text{Eq. 6}$$

2.2.3 Data sets

Each algorithm was tested using the MovieLens data set. MovieLens consists of 948 users who rated movies they had seen on a scale of 1 to 5, 5 being excellent and 1 being terrible. The dataset includes 100,000 ratings over 1,682 movies with each user having rated at least 20 movies. Based on the rating guidelines presented to the users of MovieLens, we identified ratings of 4 and 5 to signify "good" movies. These are the movies that would make good recommendations.

The 100,000 ratings were divided into 4 equal slices by giving each rating a random number from 1 to 4. Each result reported is the average of four runs, each on a mutually exclusive training set; each algorithm was trained on 25% of the data and tested on the remaining 75%.

2.2.4 Metrics

Traditionally, mean absolute error (MAE) has been used to evaluate the performance of CF algorithms. MAE works well for measuring how accurately the algorithm predicts the rating of a randomly selected item. However, it has been documented in Breese[1] and Herlocker[4] that MAE may not be applicable in evaluating rankings, which we claim to be vital in evaluating user experience. To better understand how to evaluate user experience, we analyzed how MAE failed to capture the experience. We found the following negative characteristics:

- During offline analysis, if no rating is available for a recommended item, then that recommendation has no effect on MAE.
- An error of size *e* has the same impact on MAE regardless of where that error places the item in a top-N ranking.
- Highly accurate predictions on many mediocre or bad items (perhaps rated 3 out of 5) can drown out poor performance on highly ranked items.
- MAE cannot evaluate algorithms that produce ranked recommendations, but do not produce predicted rating values.

It was our hypothesis that the problems listed in the first three bullets above were causing the MAE metric to conceal flaws in the algorithms that hurt user experience.

In search of a better metric, we turned to Precision, the percentage of "good" recommendations at a given cutoff in the ranked recommendation list. However, Precision had been used in previous evaluations of CF algorithms, (e.g. [7]) and the authors had not reported poor top-N performance for algorithms that we had observed to have poor top-N performance.

To understand this mismatch, we qualitatively evaluated the quality of the top-N recommended lists produced by the User-User and Item-Item algorithms. We found that these nearest neighbor algorithms were frequently recommending very obscure items in the top ranks. In the case of movies, the recommendations would often be for movies which no group member had heard of and there was very little retrievable information on. Because these items were obscure, rarely did we have ratings for them, and therefore they were ignored by the Precision metric. In essence, in computing top-20 Precision as it had been done before, we were essentially computing the Precision of a ranked list consisting of the top twenty *rated* items, which may or may not have any relation to the recommended items that user would actually see. The problem was critical; frequently we would find that less than two of the recommended 20 movies had been rated. Furthermore, when discarding non-rated items we might have to reach into the 200th ranked item in order to find 20 items that the user had rated.

We hypothesize that these flaws have not been noticed because their evaluation procedure ignored items for which no ratings are available. To evaluate the quality of the user experience more accurately, we choose to measure Precision and treat non-rated items as non-relevant.

Counting non-rated items as non-relevant will produce an underestimate of the true Precision, yet we believe the Precision that we measure in this way will produce a better indication of the actual user experience given our assumptions in Section 2.1. We believe this is valid for multiple reasons:

- The more items we know to be positively rated in the top 20, the more confidence we have the remaining items will be positive.
- From the user perspective. If user is recommended 20 items, but only recognizes 1, they may have less confidence in the recommender as a whole.
- People pre-filter the movies they see, choosing not to view movies that they know they are not interested. Thus, unrated items have a higher likelihood of having a true rating that is negative than a true rating that is positive.
- The dataset utilizes a discrete ratings scale of 1 to 5, and on average, users supplied a good number of the top rating (5). If there are predictable items for which we know the ratings are 5, then a good algorithm should recommend those items close to the top of the ranking. Given this, we would expect to see known liked items near the top of recommendation lists.
- The original MovieLens data that we are using was collected on a system that utilized a top-N algorithm similar to the User-User algorithm we are evaluating. Thus if our User-User algorithm recommends a movie highly for a user, then it is reasonably likely that the user was displayed the same recommendation when the data was collected. However, since they did not enter a rating, it is likely that they chose not to see the movie. MovieLens had no mechanism to say that a movie “looks awful”.

Precision with non-rated items counted as irrelevant was computed at 1, 5, 10, 15, and 20 recommendations, the most important range for users according to our assumptions.

An alternative that we could have used was proposed by Breese et al [1]. They proposed a metric based on the idea that a recommendation’s importance decreases the further it is from the top of a recommendation list. Their metric computes the “utility” of each recommended item, where the utility decreases exponentially as you move down the ranked recommendation list. Unrated items were treated as misses for this metric.

2.3 Results

The results of the experiment are summarized in Table 1 and Figure 1.

Table 1. The "modified Precision" (Precision with non-rated items counted as not relevant) with top-N recommendation lists of different sizes. Notice how exceptionally low the Precision values are.

	User-to-User	Item-Item
Top 1	2.4%	1.2%
Top 5	1.8%	1.5%
Top 10	2.0%	2.2%
Top 15	2.1%	3.2%
Top 20	2.3%	3.5%

Figure 1. The "modified Precision" (Precision with non-rated items counted as not relevant) at increasing levels of Recall. Notice the exceptionally low values of Precision – never greater than 9% Precision at their highest point.

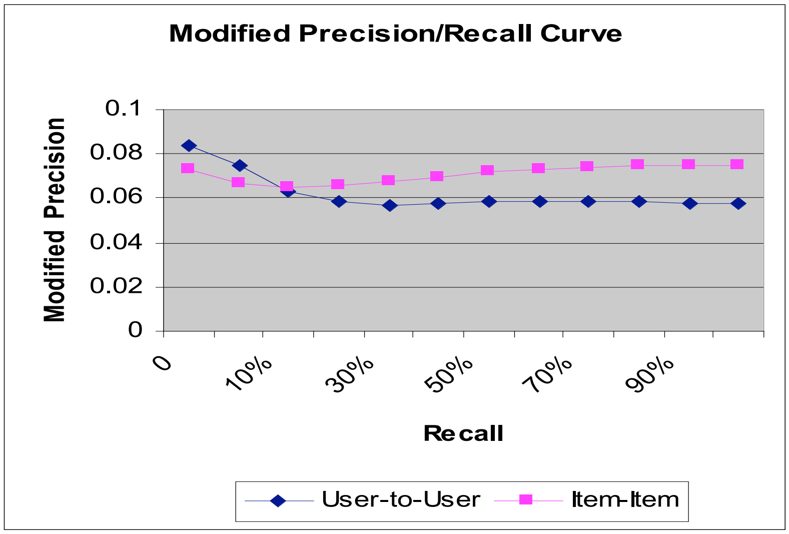


Table 1 shows that neither User-User nor Item-Item ever achieve greater than 3.5% Precision when we are considering unrated items to be bad (i.e., not of value to the user). This means that we are only confident that 3.5% of the recommendations presented to users in a top-20 list will be good using what is widely regarded as one of the most accurate prediction algorithms!

Now, to be fair, 3.5% is an overly pessimistic estimate, as we would expect some of the items for which we do not have to data to be "good". However, consider that the average user in the MovieLens dataset rated more than 50 movies as good. We would hope that a good number of those movies known to be good would appear in the top-20 recommendations. Rather, the data in Table 1 shows us that, on average, given a list of 10 top recommended movies, we can only be confident in the user recognizing a single movie as "good" in 1 in every 5 top 10 lists recommended generated using Item-Item.

Even if we assume that modified Precision underestimated the number of "good" movies by factor of 5, we still would only expect the user to recognize one movie in each list of recommendations as being good.

To explore this idea further, we examined just how far down in the ranked recommendation list would we have to go to reach the movies that we know to be good. This data is shown in Figure 1.

The relatively flat Precision/Recall curve shown in Figure 1 illustrates that the movies that we know to be good are distributed across the ranked recommendation lists created by the Item-Item and User-User prediction algorithms. This is counter to what we would expect in a good algorithm, where the good items would be clustered at the top of the ranked list.

The evidence of poor quality of the User-User and Item-Item algorithm demonstrated in this article contrasts with previously published results that have shown these two predictive algorithms to be the best performers. The reason for the differences in algorithm evaluations is the evaluation metric. In the past, MAE has been used; while we have used a modified form of Precision that we believe more accurately models the user experience.

Is it possible we can design a nearest-neighbor algorithm that overcomes this flaw? The remainder of the paper describes an algorithm we have designed to accomplish this goal.

3. The *Belief Distribution* Nearest-Neighbor algorithm

Nearest-neighbor (NN) algorithms have a variety of desirable properties that have made them popular. Our design goal was to create a new predictive algorithm that retained these properties while achieving better performance on top-N recommendations and thus an acceptable user experience. The primary advantages of NN algorithms are:

- Ability to predict a rating value for a single item.
- The best average accuracy in multi-value data (in terms of predicting the rating value).
- Good performance with large datasets (accomplished with sampling).
- Simple to understand, explain, implement, and maintain.

We designed a new algorithm using the User-User algorithm as a template, hoping to retain the benefits of the old, yet account for top-N quality and the resulting user experience.

After analyzing where the User-User algorithms had failed, we hypothesized that the majority of questionable or obscure recommendations were caused by two main sources of error. The first source of error was the active user having too few neighbors who had rated an item. An example of this with the User-User algorithm is if you have an obscure movie that is seen by small number of users, each who rate it a perfect score, that obscure movie will attain the highest possible predicted rating value for *all* other users. This will cause the obscure movie to be located high in the active user's recommendation list, despite having little confirming evidence that it is good. In fact, once a reasonable number of confirming ratings accumulate for an item, it is almost impossible for an item to be predicted as a perfect maximum rating. This occurs because as more people rate an item, the probability increases that one does not rate it a perfect score, thus making it impossible for that item to receive a perfect maximum score.

The second source of error was neighbors with very low correlation to the active user who rated the movie. These neighbors by definition have very little predictive power for the active user; the User-User algorithm accounts for this by multiplying their weight times there predictive value, effectively giving more influence to neighbors with high correlation. This method does not account for cases where all the neighbors who have rated an item have low correlation with the active user. For example, imagine if 10 minimally correlated neighbors rated item A perfect while a different highly correlated 10 neighbors rate an item B with 9 perfects and 1 above average. User-user, will recommend the first item higher, despite having those raters of A having substantially less similarity with the active user. While heuristics can be used to address some of these problems, we believe that they can be addressed more flexibly and consistently through adaptation of the predictive algorithm.

The User-User algorithm fails because the predictions it generates fail to take into account the quality and quantity of the information used to generate each prediction. We chose to create a new algorithm that enabled a system to predict not only rating values, but also to explicitly represent the uncertainty in each predicted rating.

3.1 Description of Algorithm

Past experiments have shown that users, when asked to rate the same item in two different sessions, will often rate an item differently. This change can be attributed to mood, change in taste, or a variety of other reasons. There is also the possibility that the user entered the wrong rating accidentally. Thus we consider the user's rating as noisy evidence of the user's true rating. To capture this fact, we represent our current belief of each user's rating as a belief distribution across all possible discrete (integer) rating values. If we observe a user rating of 4 on a 1 to 5 scale, we might choose to represent our belief of the true ratings as [.02, .06, .11, .70, .11] with .02 being the our belief that the user's actual rating is 1 and .7 being our belief that the user's actual rating is 4.

More formally, let R be an n by m user-item matrix containing historical item rating information of n users on m items. In this matrix, $r_{u,\ i}$ is a positive integer rating in the range [1-$r_{\max}$] of the u^{th} user for the i^{th} item. Let $\max_{\text{value}}$ represent the maximum rating an item may receive. We assume that $\max_{\text{value}}$ is small (less than 20, usually less than 10). Let the active user correspond to the user for whom we are trying to compute the top-N recommendations. The *Belief Distribution Algorithm* computes the top-N recommended items for that user as follows.

First we identify the N most similar users in the database. This is done by computing the Pearson correlation (Equation 1) between the active user and all other users in R and ranking them highest to lowest according to that correlation. For each user u, let $w(a,u)$ be the Pearson-Correlation between user a and user u.

After computing the correlations, the next step is to map each rating to a discrete belief distribution, representing our belief of user u's rating on item i. However, rather than using a vector indicating our belief in the possible rating values, we create a *belief difference distribution*. In nearest neighbor algorithms, it has been shown that normalizing a neighbor's ratings with respect to their average rating provides a substantial increase in algorithm accuracy [3]. This was done by subtracting the user's mean rating from each user's rating and then using those transformed values to predict how the active user would rate the item compared to their average rating. To incorporate this normalization into the Belief Distribution Algorithm, we represent each rating with a belief difference distribution $d(u,\ i)$. If we were to take an integer rating in the range of 1 to $r_{\max}$ and subtract an average rating in the same range, our result would be an integer in the range from $-(r_{\max-1})$ to $(r_{\max-1})$ Thus our difference distribution is a vector of size $2r_{\max-1}$, representing our belief that the user rated an item certain distances from their average rating.

Example: With a rating range from 1 to 5, if a user's average is 3 and he/she rated item *i* 5, we might choose the difference distribution to be [0.01; 0.01; 0.01; 0.02; 0.06; 0.80; 0.06; 0.02] with the cells corresponding to the possible differences from -4 to 4; the .80 indicates that we have an 80% belief that the user's true rating for this item is two points higher than their average rating.

We subtract the user's average rating from the rating, round to the nearest integer, and then map that rounded integer to a predefined difference distribution. The difference distribution that each rating is mapped to can be configured depending on our confidence in that particular rating (an inferred rating might have less confidence than an explicit rating), and our expected spread in the variability of that rating. In our data, all ratings were collected in the same manner, thus if two users' ratings have the same value of $Round(r_{u,i} - average(r_u))$, they are mapped onto the same difference distribution.

For each item *i* that is unrated by the active user and has been rated by at least one of the user's neighbors, we estimate the difference distribution *d*'(*a,i*) in a manner that is motivated by the User-User NN algorithm introduced in Section 2.2. Our approach is shown in Equation 7. We sum over the difference distributions of all the neighbors, in essence weighting by the similarity with each neighbor. The weighting is achieved by combining each neighbor's belief distribution with the uniform belief distribution. The intuition is that the less similar a neighbor is to the active user, the less belief we have that the neighbor's observed rating is the correct rating for the active user. By adding a fraction of the uniform rating in proportion to the *w*(*a*,*u*), we effectively increase the uncertainty of that belief distribution, converging the distribution closer to uniform.

$$d(a,i) = \frac{s + \sum_{u=1..N}((w(a,u) * d(u,i)) + ((1 - w(a,u)) * s)}{N+1} \qquad \textbf{Eq. 7}$$

The uniform belief distribution *s* at the beginning of the Equation 7 can be thought of a null vote with a weight of 1. It provides a threshold that the subsequent weights must overcome in order to predict a high confidence rating.

Finally, the predicted difference distribution *d*'(*a*,*i*) is transformed back into a vector of size $r_{\max}$ through the following algorithm:

- The active user's average rating is rounded to the closest integer
- We then remap the indices of the predicted belief vector by added the rounded average rating to each index. For example, on a 1-5 scale, the difference distribution represents the relative ratings -4 through 4. If the active user's average rating rounds to 2, then we transform the rating indices into -2, -1, 0, 1, 2, 3, 4, 5, and 6.
- Since we cannot have ratings less than 1, we sum the belief values for all indexes less than or equal to 1, and this becomes our end belief that the rating is 1.
- Since we cannot have ratings greater than $r_{\max}$, then we sum all belief values for ratings greater than or equal to $r_{\max}$ and this becomes our end belief that the predicted rating is $r_{\max}$.

Finally, we are left with a predicted belief distribution $d'(a, i)$ representing our discrete belief that the active user provided each of the ratings between 1 and $r_{\max}$. An example of $d'(a,i)$ for a rating range of 1 to 5 is [.07, .11, .15, .4, .27] with .07 being the probability of a user rating the item 1 and .4 the probability of the user rating the item 4.

We may then use the belief distribution to make decisions, or if we need to produce the "most probable" rating (similar to NN algorithms), we can compute the expected value of the distribution.

3.2 Improved Properties of New Algorithm

We claim that our new Belief Distribution Algorithm maintains the positive characteristics of class NN algorithms introduced at the beginning of Section 3, while at the same time providing these improvements:

1. Produces belief distributions that communicate the algorithm's confidence in a prediction, instead of just producing a predicted rating value.
2. Detects and communicates low confidence situations caused by small numbers of neighbors who have rated an item.
3. Detects and communicates low confidence situations caused by only having neighbors that are minimally similar to the active user.
4. Produces top-N recommendations lists that are substantially more valuable to the user.
5. Enables users to tune their recommendation lists to their desired risk/benefit tradeoff.

6. Supports ratings observations at different levels of confidence. For example: an explicitly entered rating vs. a rating inferred from observation of user behavior (sometimes called an implicit rating)

Properties 1-3 are introduced in Section 3.1. We will demonstrate Property 4 in Section 3.3. In support of Property 5, consider the possible different ways that ranked recommendation lists could be computed in a five-point rating scale, given the output of the Belief Distribution Algorithm. We could rank recommendations by the sum of belief that ratings are 4 or 5, or we could rank by the sum of belief that the ratings are 5, or even the sum of belief that the ratings are 3, 4, 5. In the active's user's perspective, those three rankings translate respectively to "show me items that you are confident are good", "show me items that are most likely to be excellent", and "just show me items that won't be bad!" Furthermore, users could just set thresholds for such belief levels, under which no recommendation is made.

These ranking approaches work because the sum of belief across all ratings values in a distribution is 1. A low confidence predicted item may have a most-probable rating equal to the r_{max}, but the belief in r_{max} will only be slightly higher than the belief in other rating values. If we rank only by the belief in r_{max}, we capture this lack of confidence.

Property 6 exists because we can tune the initial difference distributions. For observed ratings of high confidence, we create a more peaked distribution. For those observed ratings of low confidence, we can use a much more flat initial distribution.

3.3 Empirical Results of New Algorithm

We tested our new Belief Distribution Algorithm in the same fashion that we tested the classic nearest neighbor algorithms in Section 2. The results are summarized in Figure 2 and Figure 3.

Figure 2. Modified Precision (Precision considering non-rated items as not relevant) for recommendation lists of varying sizes for two popular NN algorithms and our new Belief Distribution Algorithm.

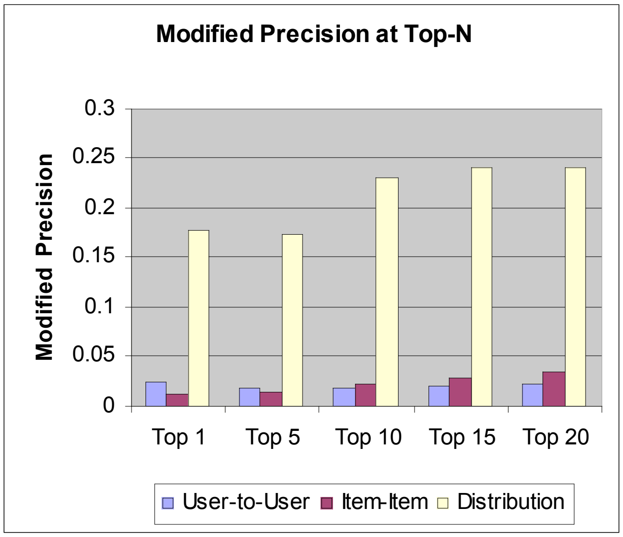
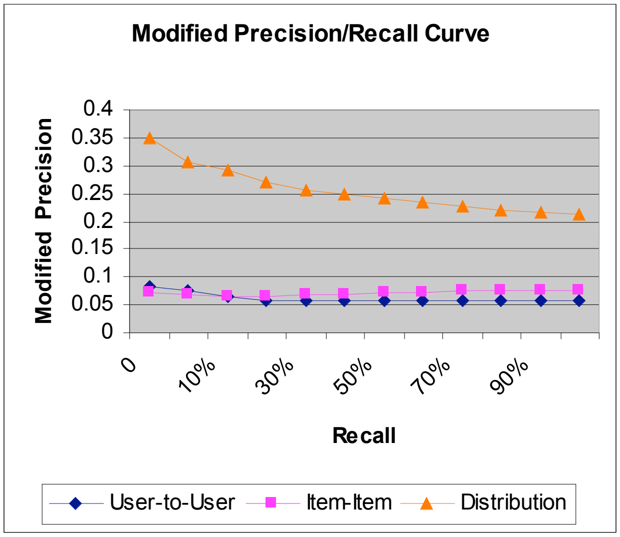


Figure 3. Modified Precision (Precision considering non-rated items as not relevant) at varying levels of Recall.



Figures 2 and 3 demonstrate the substantial improvement in Precision of top-N recommendations that our Belief Distribution Algorithm provides over the User-User and Item-Item NN algorithms, two algorithms that have consistently demonstrated the best accuracy in the scientific literature. The size of the difference is astounding; our algorithm has a decisive advantage through the entire Precision curve, with a 350% increase over the User-User algorithm at 30% Recall. These data provide substantial evidence that our Belief Distribution Algorithm will provide a better user experience.

Table 2. Comparison of new Belief Distribution Algorithm to classic CF NN algorithms using mean absolute error. To convert a belief distribution into a point value, we computed the expected value of the predicted belief distribution.

	User-to-User	**Item-Item**	**Distribution**
MAE	0.7954	0.7805	0.8556

Table 2 illustrates that our new Belief Distribution Algorithm on average cannot predict "most probable" point ratings as well as the popular NN algorithms. However, this only has bearing on recommendation systems that display one-dimensional predicted ratings to the user. Furthermore, we believe that the difference in MAE performance can be recovered with some additional tuning of the Belief Distribution Algorithm.

The far superior performance of our Belief Distribution Algorithm without a catastrophic loss in MAE gives us further confidence in our analysis from section 2.3: that Item-Item and User-User NN algorithm results are excessively low, indicating a poor user experience.

3.4 Qualitative Results of New Algorithm

Additionally, we have anecdotal evidence on the recommendations provided by the various algorithms. We found results which were in line with those reported in figures 2 and 3. Belief Distribution Algorithm provided significantly fewer obscure or horrible ratings than User-User and Item-Item. This decrease, however, was accompanied with more blockbuster movies being recommended, illustrating the trade-off between serendipity and confidence in recommendations. In practice, Belief Distribution will need to be tuned to the proper serendipity/confidence ratio for the target audience.

4. Conclusion and Future Work

Collaborative filtering algorithms' performances have been evaluated using a variety of metrics. These metrics, such as Mean Absolute Error and Precision, have often ignored recommendations for which they do not have data. Ignoring these recommendations has provided numbers which do not accurately represent the user experience. Qualitatively we have seen that User-User and Item-Item algorithms are often plagued with obscure recommendations. This flaw was illustrated quantitatively by the extremely low modified Precision scores. To address this problem and provide a better user experience we proposed a new Belief Distribution Algorithm. Our Belief Distribution Algorithm retained comparable predictive capabilities to previous nearest-neighbor algorithms while achieving far superior results for recommendations.

In the future, we plan to run a user study in which a complete set of ratings is collected, enabling us to evaluate just how accurately modified precision measures the user experience and look for significant gaps where it does not.

5. Bibliography

1. 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).

2. Dahlen, B. J., Konstan, J. A., Herlocker, J. L., Good, N., Borchers, A., Riedl, J., 1998. Jump-starting movielens: User benefits of starting a collaborative filtering system with "dead data". University of Minnesota TR 98-017.

3. Herlocker, J. L., Konstan, J. A., Riedl, J., 2002. An Empirical Analysis of Design Choices in Neighborhood-based Collaborative Filtering Algorithms. *Information Retrieval,* 5 287-310.

4. Herlocker, J. L., Konstan, J. A., Terveen, L. G., Riedl, J., 2004. Evaluating collaborative filtering recommendation systems. *ACM Transactions on Information Systems*{*TOIS*}. Vol. 22, Issue 1, p. 5-53, Jan. 2004.

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

6. 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).

7. 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).

8. 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).*

9. 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