MotorMouth: A Generic Engine for Large-Scale, Real-Time Automated Collaborative Filtering

by

Max Edward Metral

B.S., Computer Science
Massachusetts Institute of Technology
June 1994

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
In Partial Fulfillment for the degree of
MASTER OF SCIENCE IN MEDIA ARTS AND SCIENCES
at the
Massachusetts Institute of Technology
June 1995

Signature of Author Signature redacted
Program in Media Arts and Sciences
15 May 1995

Certified By Signature redacted
Pattie Maes
Assistant Professor in Media Arts and Sciences
Program in Media Arts and Sciences

Accepted By Signature redacted
Stephen A. Benton
Chairperson
Departmental Committee on Graduate Students
Program in Media Arts and Sciences

MotorMouth: A Generic Engine for Large-Scale, Real-Time Automated Collaborative Filtering

by

Max Edward Metral

The following people served as readers for this thesis:

Signature redacted

Reader _____

Mitchell Kapor
Adjunct Professor
Program in Media Arts and Sciences

Signature redacted

Reader _____

Walter Bender
Associate Director for Information Technology
MIT Media Laboratoy

Signature redacted

Reader _____

Rosalind Picard
NEC Development Professor of Computers & Communications
Assistant Professor, Program in Media Arts and Sciences

Table of Contents

List of Figures and Tables

Acknowledgements

If you look at the theses that come out of our group, almost all of them list our advisor, Pattie Maes, first. There's a reason for this: she's absolutely instrumental in making us all successful. She knows just the right questions to ask to push us in the right direction, she knows how to motivate us when demos swamp our schedules (keep those lunches!), and she gives us all the room we need to be individually creative and to work in our own style. I couldn't ask for anything more, and I can truly say that my life has changed 100% since the day I walked into her office and said "I think agents are cool."

Likewise the other members of our group: Bruce "The Plane" Blumberg, Christie "Big Hard Drive" Davidson, Lenny "Invert" Foner, Michael "Wavelet" Johnson, Akira "305 Annex" Kotani, Henry "Watch What I Do" Lieberman, David "Where Is He" Maulsby, Brad "de Jesus" Rhodes, Upendra "The Disc" Shardanand, and Alan "Laugh a Minute" Wexelblat have made this crazy three years fun. And to my office mate, summer-job mate, and former roommate Yezdi Lashkari: I never had a brother, but from what people tell me, this is what it's like. Thank god I never had a brother. ;-) I look forward to working with you, Nick, Andy, Wax, Upe, and Pattie for a long time to come. Here's to Agents!

I could never have made it without the support of my parents, obviously. I remember applying to MIT and thinking, "There's no way I'll go there, but it will sure be great to have turned them down." Thanks Mom and Dad for giving me the opportunity, and hopefully you know how much its meant to me and how much you mean to me. And stay young, because the older you get, the older I get.

Thanks to my readers for providing key insights throughout this process, and thanks for putting up with my slacking off and not getting a draft to you until last week... A special thanks to Roz who saved my thesis. I walked in her office with no clue about how clustering was going to get done, and 10 minutes later I knew it cold. I wish I would have gotten a chance to take a class from you, because you're one of the best *teachers* I've met at MIT.

A special thanks to the people on the other side of my life: the club kids. It's amazing to have a totally separate life that you can escape to when work gets to be too much. Thank you Bruno, Evans, Fran, Teddy, Cila, Jimmy, Terri, Jacob, Priscilla, Jose, and all the Loft children for supporting me and screaming for the music. Hopefully you will all read this thesis someday; and E, I hope we work together in dj'ing *and* computers someday.

Whew, if I had more pages, I could go on forever. Forgive me if I forgot someone.

Introduction

Section Summary. **The concept of Automated Collaborative Filtering (ACF) is introduced, and the need for such a system is identified. ACF is a process which attempts to automate "word of mouth" by leveraging user ratings for items to make recommendations for other users.**

In the past year, research has begun on agents which make recommendations for music, videos, and other traditionally subjective domains using automated collaborative filtering techniques. Automated collaborative filtering (ACF), strictly defined, is a technique which uses a corpus of user recommendations to suggest items to individual users. Furthermore, by our definition, SIF is independent of the items that it contains and recommends. We have developed a generic automated collaborative filtering engine based loosely on Ringo[1] which addresses the real world application of the technologies that have been developed to date and improves upon their performance. The system has been extensively tested and examined in the context of music recommendation, but the engine has also been used to recommend Web documents[2]. The system has been designed with speed and size in mind, and provides a real time, scalable architecture for automated collaborative filtering. Lastly, we have developed methods to analyze the usage of the system and gathered data to examine the way in which users interact with the system.

The Problem

There are more goods and information available to users than ever before (Table 1, "It's a big, big universe," on page 14). For users and content producers alike, the marketplace is cluttered, disorganized, and confusing. Users need help with the selection and distillation of items; producers of goods and information need better means of reaching interested users.

Automated Collaborative Filtering addresses both users and producers needs.

In certain domains such as music and movies, users have learned to accept the information overload and navigate it as best as possible. In almost all cases, some form of referrals are used to select items from the range of possibilities. In music we have Billboard, CMJ, radio, and other review sources. In movies, we have Siskel and Ebert, the New York Times, and a host of other sources. Furthermore, users often ask personal friends for recommendations. Automate collaborative filtering automates and personalizes this review process. There are two phrases which may make its methods clear:

- Automate collaborative filtering automates "word of mouth" on a global scale.

  Recommendations can be obtained from virtual friends across the world, who you've never met.
- It's just like asking your friends, except we tell you who your friends are.

  SIF attempts to match people with similar interests (friends) on the users behalf.

TABLE 1. *It's a big, big universe*

Domain	Number of Items	Recommendation Source
Music Titles	140,000	Billboard, CMJ, Radio
Videos	15,000	Siskel and Ebert, NY Times
Restaurants (US)	400,000	Zagat's, Local Newspapers
Books	15,700,905	Best-seller lists

Overview of This Document

The next chapter introduces Automated Collaborative Filtering and describes research to date. Chapter 3 presents HOMR, an automated collaborative filtering system for music recommendation available via the World Wide Web. Chapter 4 provides results for many different algorithms and discusses high-level analysis of overall data (clus-

tering). Finally, we discuss the HOMR community and attempt to draw conclusions about ACF technology and its future.

Automated Collaborative Filtering

Section Summary. **Automated Collaborative Filtering is defined. Previous work in the field is presented and analyzed.**

Definitions

Automated collaborative filtering attempts to automate the "word of mouth" process for a given user by leveraging information provided by other "similar" users. Automated collaborative filtering is especially applicable in domains which have traditionally aesthetic qualities, such as music, books and movies.

Previous Work

Automated Collaborative Filtering, as we define it, is a fairly new field. Work begun only several years ago, and was made possible by the wide proliferation of networks such as the Internet. Many of the underlying algorithms, however, have been around for quite a long time. The algorithms used attempt to measure the subjective "similarity" of users based on user ratings. This problem is a fairly straightforward statistical or pattern matching problem, and algorithms from pattern recognition or information retrieval can be examined for ideas.

For many years, direct marketers and others have been attempting to achieve the type of "molecular" demographic capabilities that HOMR provides. Many use complicated statistical analyses to try and find patterns in survey data. Unfortunately, many have not been asking the right questions. While demographers attempt to ask questions such as "What age group will drink Snapple?", ACF can do these tasks in a new way, such as "What 5,000 people are most likely to like Snapple based on their past preferences?"

Essentially, we reduce the audience to "an audience of one" which empowers users as well as content producers. A second problem with the directed marketing approach is that direct marketers do not provide their "recommendation" capabilities to their end-users. A very powerful feature of this ACF system is the win-win situation it creates for users and producers.

On the other side of the coin, researchers have been focusing on the user side of automated collaborative technology. Previous developments include:

Ringo

Ringo[1] was a system for music recommendation offered via email. It was developed at the MIT Media Lab by Upendra Shardanand and based upon ideas from Carl Feynman. The work in this thesis is heavily based on the Ringo system, and is essentially an extension and improvement upon it. Ringo gathered approximately 4,000 users before it was taken down in December and replaced with HOMR, the system developed in this thesis. Ringo was an excellent proof of concept and performed quite well; nonetheless there was quite a bit of room for improvement to the basic ACF algorithm used.

GroupLens

GroupLens[3] was an early example of automated collaborative filtering technology. The system allows users to give ratings to Usenet news articles. Custom news browsers can then interpret the data, compute correlation coefficients (Using the Pearson r formula), and provide recommendations for users.

There are two interesting comparisons of GroupLens and HOMR/Ringo. First, GroupLens is a distributed approach. Custom clients are used to interpret special news articles containing ratings. The advantage, of course, is that no central server must be maintained (other than the Usenet news server). The disadvantage is that any user wishing to use the system must download a custom client, which may not exist for all platforms, or may be difficult to compile. Secondly, users can view other users' ratings. Since ratings are merely special news articles, a user can browse those articles manually; the advantage, of course, being accessibility by those without the client and by those interested in specific user's feedback. The disadvan-

tage is the lack of privacy provided to users; if a user rates alt.sex.* highly, all other users can see that fact.

Movie Select

Movie Select is a software package for the Macintosh. Movie Select claims to recommend movies to people based on what a pre-surveyed set of users enjoyed. There are two major shortcomings of this system: its information is static, and doesn't learn from user patterns; and there is no sense of community. Across multiple Movie Select sessions, no ratings are stored. Therefore, there is no notion of "building a profile" of the user or any sort of improvement over time. Since the product is not network based, and since users do not contribute content, there is no sense of community building, bonding, or loyalty. The product's automated collaborative filtering techniques are not disclosed.

videos@bellcore.com

The videos system from Bellcore is very similar to the original Ringo system, right down to the fact that it's an email-only system. Users rate movies and receive recommendations for new movies. Until recently, no information was available about the workings of the videos system.

A paper will be published at CHI '95[13], which gives some details about the general concept. While not specific, it mentions multiple regression analysis as the similarity metric. It does not mention many error statistics, so it is very hard to compare performance of the two systems in detail. Furthermore, they have only 291 users and 55,000 ratings, less than one-tenth of HOMR's size.

The Bellcore paper mentions one very interesting measure of error. They surveyed a subset of their users and asked them to re-rate items they had previously rated. The Pearson r correlation between both sets of ratings was .83, putting an upper bound of .91 on the possible accuracy of the system.[1]

1. Statistical theory states that the best that can be done in this situation is the square root of the observed test-retest correlation, which is .91 in this case.

The I-Station

The I-Station is a commercial system featured in several record store chains, including HMV and Strawberries. While the I-Station does not employ automated collaborative filtering techniques, it does collect rating information and provides non-personalized reviews. The I-Station provides a hint at the application on automated collaborative techniques to aid users in selecting products. For example, the user can walk around the record store, choose a CD, and bring it to the I-Station kiosk. By simply passing the bar code over the scanner, the user can hear samples, get concert information, etc. By augmenting this system with ACF, the user could get a prediction of their rating, get more directed reviews from nearest neighbors, and find other artists similar to the one given.

There are several other systems like the I-Station which are "so close" to the HOMR concept, such as Cardiff's Movie Database. These systems have content, have rating data on a per user basis, but merely lack the automated collaborative technology to reduce their demographically-segmented recommendation to truly *personalized* recommendations.

HOMR: An Interactive Music Advisor

Section Summary. **The HOMR (Helpful On-line Music Recommendations) system is introduced and compared to its predecessor, Ringo[1]. Many improvements and feature additions were made and are described in detail here. By adding these features, we have made HOMR a more usable and useful system, which will in turn improve the data collected.**

The original Ringo system was offered via email. Users sent ratings and other requests to Ringo and waited several hours for a reply. The Ringo system came on-line in July, 1994 with 575 artists and 10 users (the members of our research group). When the system came to a screeching halt in early December, there were 4,000 users and 12,000 items in the database. The system took well over an hour to produce a set of recommendations and consumed over 400 Megabytes of local storage. While the original system was a fantastic success and proof of concept, it had several problems:

- Size of stored data
- Speed of prediction and recalculation
- No true interactivity
- No multimedia content
- No "people in the interface", poor sense of community

While the interface and its features do not directly relate to the algorithms used for predictions, they are an important part of the overall system. If our system is easy to use and enjoyable, users will enter more ratings, and interact with the system more often. In turn, as we learn more about the user, our algorithms can improve their predictions and recommendations.

In designing HOMR, we addressed each of these problems in depth, and the solutions are described fully in this chapter. In addition, we improved on the performance of the basic algorithm, and detailed results can be found in "Algorithms and Results" on page 37. Briefly, speed and size were vastly improved by rewriting the code in C++ and making realizations about data

access and usage; interactivity was accomplished by providing a real-time interface to predictions and other information via the World Wide Web; along with the Web came the ability to add sound samples for artists and albums. Finally, much effort was devoted to developing a sense of community among all users and among users with common interests. While HOMR aims to automate word of mouth, there is no reason it shouldn't also facilitate manual word of mouth (a.k.a. good old conversations with real people).

The Engine

Speed Improvements

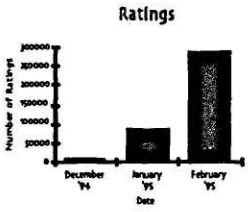
The original Ringo[1] system was written in PERL[1] and ran on a DEC Alpha. Since it was a non-interactive (email) interface, it processed input once every hour. Unfortunately, as the number of users and artists grew, the system became unmanageable. The system stored an $O(n^2)$ matrix of user-user correlation coefficients used to make predictions. With 4,000 users, this had grown to a monstrous 300 megabyte file. With 10,000 users, as we have today, it would have taken approximately 2 gigabytes of storage just for this matrix. Furthermore, the amount of email coming in began to take more than an hour to process. Therefore, since new input was processed every hour, a seemingly never-ending queue was beginning to build.

HOMR improves upon Ringo in several ways. First of all, HOMR was written in C++ using a fast freeware database package[14] as the underlying storage system. PERL had been great for testing and prototyping, but it could no longer handle the job. Second, there was no need to store the entire n^2 correlation matrix, since only a subset of users were used to calculate a given user's prediction. After the first implementation of these schemes, the entire data set (neighbors matrix, item storage, user storage) took less than 20 megabytes. As of this writing, the data, including backup and recovery information, occupies 100 megabytes.

1. Practical Extraction and Report Language, an interpreted language

A detailed analysis of the usage characteristics of Ringo was performed to come up with the appropriate database model for the system. For example, it was clear that the most popular activity was entering ratings, and that these ratings were entered in batches. Therefore, rather than storing each rating as a record, a single user's profile was stored as one large data chunk. Many optimizations such as these sped up the system by a factor of 10. Furthermore, we separated the task of generating recommendations from the task of recalculating the nearest neighbor set. At the cost of fully up-to-the-minute neighbors, we shed incredible amounts of computation time.

These programmatic improvements were adequate for quite some time, and to some degree could scale indefinitely.



The most time intensive task, by far, was the recalculation of neighbors. With 8,000 users, it took approximately 2 minutes to recalculate the neighbor set. Other functions such as generating recommendations and entering a batch of ratings took approximately 3 seconds. The reason, of course, that neighbor recalculation took so long was that the system must compare a user against all other users in the database. This comparison entails retrieving all profile records from the database. Traditionally, full sequential access is expensive. In our case, it was even more expensive since the database package[14] used a recently used cache strategy rather than pre-fetch. Even with a different, smarter database package, further improvements would be necessary. There were two possible approaches to solving our problem: incremental recalculation, or a distributed approach.

Incremental Recalculation. Instead of recalculating the neighbors from the whole set, we could make sure that all neighbor information was updated whenever new ratings data was entered. In the current engine implementation, there are "hooks" which allow algorithms to maintain such incremental information. We did not implement such an incremental scheme for several reasons:

- When using incremental recalculation, you pay the computational price when setting ratings (since you must update all potential neighbors for each new rating). Since setting ratings was the most popular activity, we did not feel the payoff would be adequate for our relatively small user base.
- Incremental schemes increase the complexity immensely, since we must manage possibly asymmetric neighbor relationships across the entire database whenever a new rating is entered. In other words, if a new rating is entered for user A, all neighbor relationships both for neighbors of A and for users whose neighbor sets contain A must be updated.

A Distributed Approach. A second possibility was a distributed approach which would allow us to add more machines or processors as the number of users grow. A simple strategy seemed promising: divide the user base among the machines. All data would be accessible to each server, and recommendations and rating information could be entered into the global database by any server (i.e. requests would be rotated among all servers or handed to the server that was least busy). When neighbors were recalculated, however, each machine would take a subset of the database (for example, user 1,000 to user 2,000), calculate the closest n neighbors from that subset, and report its results back to the "coordinating server" (another rotating responsibility). The coordinating server would then choose the n "nearest nearest neighbors" and store those for the user in question. Each server could also store all the ratings for its subset of users in memory, further speeding up the calculation. This approach still seems viable, and while there are many possible improvements, it would have been an effective first cut. There were two main reasons we did not take this approach:

- We didn't wish to dedicate many machines to the problem, and also wished to further explore optimal single-machine performance.
- Database consistency problems have been the basis of many research projects and many successful commercial ventures, and we had no desire to delve into those problems at this time.

Major Improvements

As the number of users approached 10,000, the server began to slow down, and the single machine approach was clearly becoming inadequate. Shortly before we began to implement a distributed solution,

we began thinking about the problem in a different light. It was useful to state the process explicitly. "To recalculate the neighbors, we iterate through all users. For each rating that our user has in common with another user, we compare them and update the similarity for that pair of users." The key is that only those items which both users have in common figure in the distance measurement. We have an upper bound on these artists, it is the set of items that our user has rated. Instead of iterating over all users, then, we should iterate over all users who have rated at least one of the artists that our user has rated. Unless this set is the set of all artists, the resulting calculation will involve strictly less comparisons than the "each-user" approach.

To implement our scheme, instead of simply storing rating profiles for ***users***, we also store them for items. Now, when we recalculate neighbors, we retrieve the profiles of all the ***items*** our user has rated, and iterate through each rating in this set to find the distance to all users who have at least one of these ratings in common with our user. Mathematically, our order of growth has not changed, since in the worst case we still have O(|Users|*|Items|) comparisons; however, the constants have changed, and our new method is never more computationally intensive. In practice, we've reduced the order of our calculation to

- O(Ave. Number of Ratings per Item * Ave. Number of Ratings Per User)

as opposed to:

- O(Number of Users*Ave. Number of Ratings Per User).

Structural Changes

The generic engine is easily customizable to any domain. The engine itself has no knowledge of its contents, it merely stores ratings for and makes recommendations for "items." Furthermore, different algorithms can be plugged in by writing small bits of code and slotting them into the engine. This modular design allowed great freedom in exploring new and untested algorithms without large code rewrites.

The engine also provides storage for meta-information about items and users, and a host of other informational services. The engine

also defines the concept of "containment" which allows a set of objects to be contained by a parent object. In HOMR, this relationship is manifested in the artist-album concept. In other domains, different relationships may hold. For example, in a server brought up to deal specifically with house music, singles (12" records) were contained by multiple parents: producers, artists, record labels, and compilations. Users could receive recommendations for and enter ratings for any of the parents and any of the children. In addition to "multiple containment," the containment hierarchy could be more than one level deep. In HOMR, we could have artists containing albums containing actual songs. In a research system, this was more complex than we wanted to deal with, and the informational ramifications of entering song titles were simply not worth the added capabilities. While the possibilities for use of the containment features are many, they have not been fully tested, and there may be algorithmic issues when designing particular containment schemes.

The Interfaces

Email

The email interface to the system remains largely unchanged from Ringo. The scripts which parsed email simply make calls to HOMR's engine instead of making calculations themselves. This is desirable for many reasons; for example, former email users would never have noticed a change. To summarize the description of Ringo's email interface in [1]:

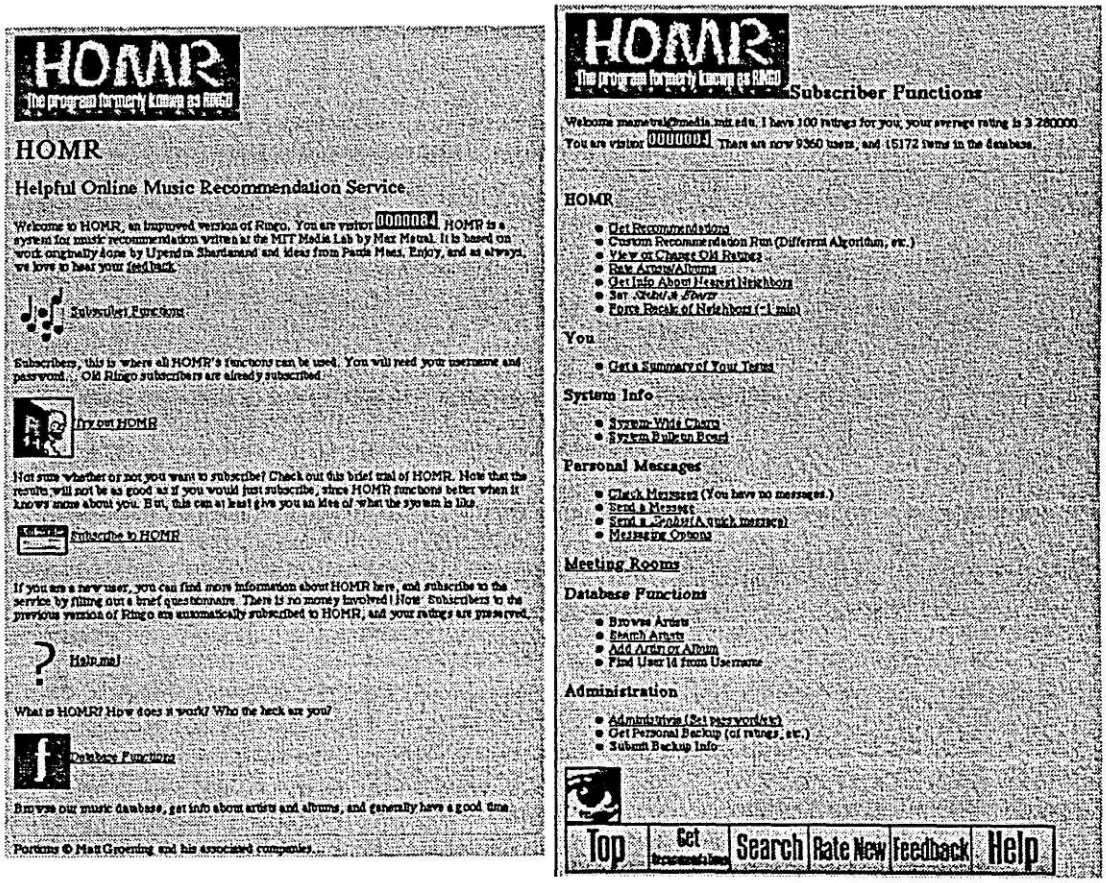
- Users rate items by submitting email with any number of { "Artist Name", "Rating" } pairs.
- Users can ask for a set of artists to rate, which is generated quasi-randomly. The new interface also allows users to get a list of "most likely" artists to rate (See "Most Rated By Neighbors" on page 31).
- Users can obtain suggestions/recommendations.
- Users can view the Top 30 and Bottom 30 artists.

Unfortunately, users can no longer submit reviews via the email interface, and can no longer obtain dossiers on artists. There is no particular reason for the loss of these capabilities, other than that we did not have time to re-implement them using the new engine.

World Wide Web

HOMR's World Wide Web interface was put on-line officially on December 15, 1995. To date (April 30, 1995), the server has processed over 400,000 requests. For more information about server statistics, see "Discussion/System Statistics" on page 67.

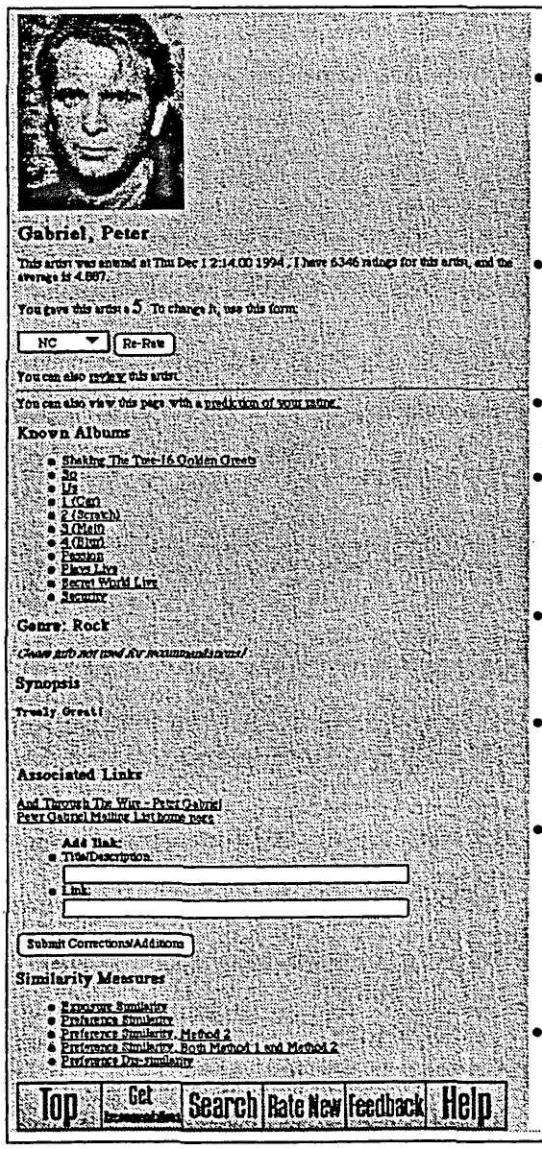
FIGURE 1. HOMR Home Page and Subscriber Page



The beauty of the Web is the ability to view large amounts of information in a point-and-click fashion. The HOMR system contains many

artists, many albums, and much other information. Through the use of scripting and forms, HOMR creates an environment in which users can browse detailed information about artists and even contribute their own information. For example, the page for Peter Gabriel shown in Figure 2 contains a user-supplied image as well as genre information, a brief synopsis, album information, and even some related hyperlinks to other web pages dealing with Peter Gabriel.

FIGURE 2. Typical Artist Page



- The image presented here was entered by a user. It doesn't reside on our server, it's merely linked into the document.
- The user is presented with information about the overall rating information for this artist, as well as their rating for him and a menu to change that rating.
- The user can enter a personal review.
- A list of known albums is presented (which users can add to via the "New Artist/Album" interface.
- Genre information can be entered by users for posterity. It is NOT used for recommendations.
- Users can enter brief synopses about each artist or album.
- A list of known links related to this artist is presented. These can be samples, info pages, or anything. Users can add arbitrary numbers of links here, and errors are corrected by the system administrator.
- Finally, a user can get "similarity metrics" for this artist to find related artists.

Of course, user contributed information is not always reliable. For example, Beethoven was classified as "Industrial" for quite some time by a mischievous user. Since we do not want to make it easy to destroy others' contributions, users must submit errors and corrections by sending "feedback" (via the buttonbar shown at the bottom of Figure 2) and the system administrator will correct the problems.

Additional Features

Improving the Survey Stage

A significant difficulty with automated collaborative filtering is the initial survey stage. Since the system must build up rating data for each particular user in order to recommend items for them, the system must solicit ratings from the user. How can the system determine which items to choose? How can the user most easily and effectively express which items they would like to rate? It is clear that there are many ways to approach this problem. Rather than deciding on a single approach, we provide users many ways to enter new ratings. Users can ask for new items to rate in the following ways:

- Initial Survey: Cluster Based
- Manual Entry / Search
- By Genre
- By Date of Entry into the system (i.e. "Newest Artists")
- Most Often Rated Artists
- Most Often Rated Artists By Nearest Neighbors
- Items Similar to a Given Artist

Initial Survey

When users subscribe to the system, they can request an initial survey based on "clusters" of artists. For details on artist clustering, see "Clustering" on page 60; the basic idea is that we attempt to learn which "sets" of artists are most similar to each other. Based on that information, we provide several often rated artists from each of those clusters, in an attempt to roughly place the user in the "user-rating" space.

Manual Entry/Search

The simplest way (conceptually) for a user to input ratings would be to allow free form entries and ratings. Unfortunately, there are several practical problems with this approach. First, users often spell or arrange artist's names differently. Secondly, some users cannot type quickly and view manual entry as a true chore. We have attempted to solve both problems by using a form of Soundex matching which allows users to enter partial names or names that "sound like" the desired artist. There are several other existing solutions which could be adopted. For example, the I-Station allows users to type the first letters of the artists names (e.g. "Sonic Youth" could be referenced as "So Yo").

In the interface, there are two items that allow manual rating entry. A user may search for a single artist, and then rate that artist on the resulting page (the standard artist page as shown in Figure 2 on page 28). A user may also enter a set of artists and have HOMR search for all of them. HOMR then attempts an exact match for each of them. If an exact match cannot be made, HOMR presents a choice box of options from which the user can select and rate the appropriate artist.

Genre

Since users can enter genre information about each artist and album, we would like to make use of this information in helping users navigate the database. Users can select any genre and receive a list of 50 (or any number) artists to rate (which they haven't rated before).

Date

A significant problem in a commercial system would be gathering a "critical mass" of ratings for a new or otherwise obscure artist. In this prototype, active users can get a list of new items to rate, in an attempt to kick-start the process. A similar method would be "billboard notification" of new artist that *may* be of interest to a user. For example, if a new band only has a few ratings, but those ratings are by a user's neighbors and are high, it may be appropriate to put a message on the top of the users recommendation screen saying "Have you checked out Band X?" which would allow them to look at general information and perhaps provide a rating. Unfortunately this was not implemented due to time constraints.

Most Rated

Users can get a list of the most popular items which they haven't yet rated. For quite some time, this method was the only way to get a list of artists to rate. Consequently, popularity was a feedback system and almost all users have rated the most popular artists.

Most Rated By Neighbors

The first of the "intelligent" selection schemes, "Most Rated by Neighbors" is exactly equivalent to a recommendation run, but irrespective of *what* rating a neighbor gave to an artist. In other words, it answered the question "What items have my nearest neighbors rated most often?" rather than "What items have my nearest neighbors given the highest ratings?"

Similar Items

Users can ask for a list of items which are similar to a given item. This similarity is measured in terms of "exposure." In other words, "If a user rated artist A, what other artists are they most likely to have rated?" For more details, see "Exposure Similarity" on page 33.

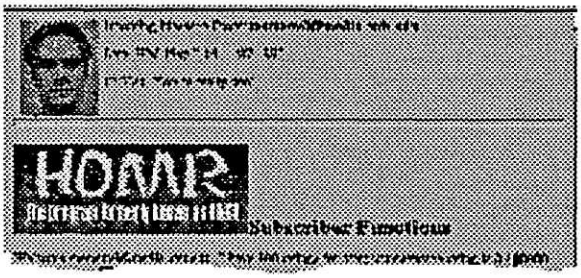
Bells and Whistles

Several other feature were incorporated into the World Wide Web interface which did not necessarily directly impact the algorithms, but provided a better interface and improved the sense of community.

Messaging

Users can send each other messages. Since users must login to use the system, and since we have anonymous user id's, messaging was fairly straightforward, and allowed a host of possibilities. There are two types of messages: standard and zephyr. Standard messages are placed in a "mailbox" which the user must check. The user is alerted via the top page (Figure 1 on page 27) when new mail is in their box. Zephyr messages are intended for rapid, casual conversation. Users send a message and it is displayed at the top of the next page the user receives from our server (Figure 3). After the user leaves that page, the message is deleted. This allows users to guide each other through the system, share a common experience, or just plain chat. Unfortunately, we weren't able to implement a means to find out who's "on the system" at any given point, so zephyrs were not all that popular.

FIGURE 3. Sample "Zephyr" Message



The Dating Service

As a result of messaging, it was a fairly simple extension to allow "neighbor messaging." Users can get a list of their nearest neighbor's anonymous id's, and send any or all of them a message. This allows users with common interests to engage in un-automated word-of-mouth, an ancient but effective means of communication.

Unfortunately, messages were not delivered to users true email addresses, but to their mailboxes on the system (to avoid being too intrusive). Therefore, many users either did not log on to the system often or did not check their messages often. The messaging feature was not used as often as one might expect, with only 1,000 messages being sent in a four month span (See Figure 23, "Web Server Usage by Function," on page 68).

Taste Summary

Using genre information provided by users, HOMR can attempt to make generalizations about your musical interests. At this point, HOMR merely calculates an average rating per genre and reports those that you rate highly or those you seem to dislike. Detailed summaries of per genre information are also provided.

Bulletin Boards

A global bulletin board for users and system administrators to discuss all issues is provided.

System Charts

System-wide data in the form of charts is available for:

- Most Rated Items (Most "Popular")
- Most Liked and Disliked Items (Highest and Lowest Average Ratings)

Similarity Metrics

By examining overall system data, we can draw conclusions about what artists are similar to each other. On each individual artist page (See Figure 2 on page 28), users can ask, "Which items are similar to this one?" HOMR measures similarity in several ways:

Exposure Similarity

Exposure similarity answers the question, "If I've rated this artist before, what other artists am I most likely to have rated?" This metric is without regard for the actual ***ratings*** given to the artists, it only checks whether or not there is a rating. The exposure metric can be very useful for guessing which artists a user will be able to rate, given that they have rated certain other artists.

Mathematically, exposure similarity between artist A and B is measured as follows:

$$E_{ab} = \frac{\sum_{n \in U} c_{na} c_{nb}}{\sum_{n \in U} c_{na} + \sum_{n \in U} c_{nb}} \qquad \text{(EQ 1)}$$

where:

- H_{ij} is user i's rating for artist j.
- $c_{na} = \begin{cases} 1 & H_{na} \neq 0 \\ 0 & H_{na} = 0 \end{cases}$
- U is the set of all users.

The value is thresholded to avoid suggesting items that have been rated by over 50% of the database, since those items have been exposed to such a large percentage of the population. Strictly, these items are similar to everything else, but users would quickly tire of seeing "The Beatles" on every list. If users wish to see these "globally similar" items, they can merely view the "Most Rated" system-wide chart.

Preference Similarity, Method 1

Preference similarity gets at a more understandable concept, namely which items do users consider aesthetically similar? It answers the

mathematical question, for all users who have rated a and b, how often do they rate them on the same side of the average? Initially, this was expressed as a percentage. Unfortunately, without weighting, a pair of items which were both rated 7 by a user would be given the same amount of credit as a pair of artists which were both given a 5 (for example).

Instead, we adopt a method which takes into account the number of standard deviations above or below the average for each rating. The preference similarity, using method 1, is then defined as:

- T_{ab} = Set of all users who have rated artists a and b
- $int(x)$ = Integer component of x
- $\overline{U_x}$ = Average rating for user x
- σ_x = Standard Dev of ratings for user x
- $m_{x(ab)} = \begin{cases} 1 & \text{If user x rated a and b on the same side of their average} \\ -1 & \text{If user x rated a and b on different sides of their average} \end{cases}$
- $\mu_{xm} = int\left(\frac{\left|H_{mx} - \overline{U_x}\right|}{\sigma_x}\right) + 1$

  a.k.a. the number of standard deviations away from the average (+1, since we still want to count items within 1 deviation).
- $\Omega_{xmn} = \left(\frac{\mu_{xm} + \mu_{xn}}{2}\right)\left(\frac{1}{1 + \mu_{xm} - \mu_{xn}}\right)$

  a.k.a. the average deviations scaled by the difference in the deviations.

$$P_{ab} = \frac{\sum_{n \in T_{ab}} m_{n(ab)} \Omega_{nab}}{\left|T_{ab}\right|} \qquad \text{(EQ 2)}$$

Preference Similarity, Method 2

Method 2 employs a similar technique, but without regard for user averages. In other words, we attempt to minimize the average distance between ratings for artist A and B for each user. Method 2 is defined as:

- T_{ab} = Set of all users who have rated artists a and b

- $d_{n(ab)} = |H_{na} - H_{nb}|$

$$Q_{ab} = \frac{\sum_{n \in T_{ab}} d_{n(ab)}}{|T_{ab}|} \quad \text{(EQ 3)}$$

Preference Similarity, Combined

The combined metric runs both method 1 and 2 and calculates the average position (rank) of a given artist in both generated similarity lists. The items with the highest average position in the combined list are reported.

Preference Dis-similarity

Preference dis-similarity merely turns preference similarity (method 1) on its head (D=-P). The results of this metric are hard to interpret, as the concept of musical dis-similarity is not a familiar one.

Clustering

Users can browse clusters of artists generated by HOMR. (See "Clustering" on page 60.) While viewing a cluster, users can ask for a recommendation of items *within* that cluster. HOMR then recalculates a user's neighbors based on ratings in common within that cluster, and generates a prediction for items in that cluster.

Users have varied tastes; for example, a user may like rock and new age music. General ACF attempts to match users with similar tastes. Cluster-based ACF attempts to match users with similar tastes in a particular subset of artists, a.ka., a cluster. Results of cluster-based ACF are presented in "Clustering" on page 60.

Algorithms and Results

Section Summary. **Comprehensive results for many prediction algorithms are presented in this chapter. Automated testing procedures were developed which generated all data automatically. Performance indices were selected which show the performance of HOMR relative to Ringo and other automated collaborative filtering systems (namely Bellcore's video system).**

In addition to improving the interface, we would also like to improve the accuracy of the system's recommendations. To do so, we must select metrics by which to measure its performance, develop automated test procedures to gather these metrics, and explore different algorithms. Nine different algorithms are examined here, and detailed results presented for each. In our particular domain, namely music recommendations, certain algorithms perform best; in different domains, other algorithms may be optimal. The development of standard metrics and testing procedures will aid in the selection of optimal algorithms in many domains.

Data Characteristics

The HOMR data is a set of user profiles. A user profile is a list of artists and ratings for that user for that artist. So, overall, the data is a matrix H of dimensions |Users| x |Artists| where H_{ij} is user j's rating for artist i. Automated collaborative filtering attempts to predict values for a given user for artists that they have not rated ($H_{ij} = 0$). To make these predictions, the rest of the data in the matrix can be used. Individual ratings for artists are based on the following scale, as used in Ringo[1]:

7: BOOM! One of my FAVORITE few! Can't live without it.
6: Solid. They are up there.
5: Good Stuff.
4: Doesn't turn me on, doesn't bother me.

3: Eh. Not really my thing.
2: Barely tolerable.
1: Pass the earplugs.

From a completely top-level view, given:

- H_{ij}, the existing user-artist rating space, and
- G_{ij}, the completely filled user-artist rating space (where the empty elements of H_{ij} would be filled with what the user ***would have*** rated artist j).

Of course, we do not know G_{ij}. We must then attempt to find an algorithm that approximates it as closely as possible. Our approximation may change over time, and is likely different for each user.

The Basic Idea

In practice, all our algorithms use a similar procedure:

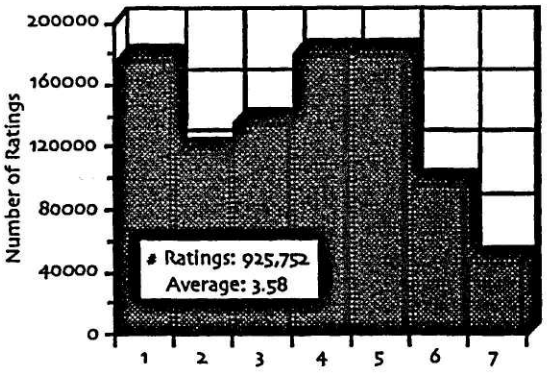
- To make a prediction for user A for artist j,
1. Calculate the set of users most "similar" to user A. Call this set N.
2. Calculate the prediction for artist j from N by calculating a weighted sum of user's ratings for j in N.

Our algorithm will succeed or fail on two main factors: definition of similarity between users, and the specifics of the weighted sum. Furthermore, we may improve the algorithm by tweaking the size of N, either by using a threshold on similarity or on sheer number of neighbors. We have tried many similarity metrics and several summation techniques, which are outlined later in this chapter.

The HOMR data set was frozen on April 1, 1995 for analysis purposes. All tests were performed on the same data set. At that point, there were 8,506 users, and 925,752 ratings. The distribution of ratings is shown in Figure 4. Note that there are almost four times as many ratings of 1 as ratings of 7. In general, users are much more able to identify things they dislike than things they like. This phenomenon may be due to several causes, among them:

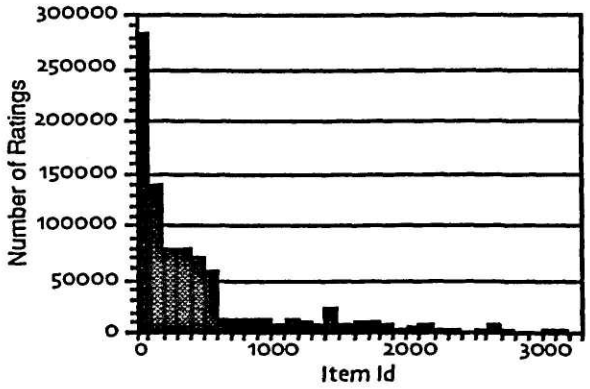
- People only like a small set of artists, and hate a much larger set.
- The Internet culture is moderately elitist, and this reflects itself in dislike of "pop" music. Since these users know pop music, but dislike it, we amass many low ratings from this factor.

FIGURE 4. Distribution of Ratings



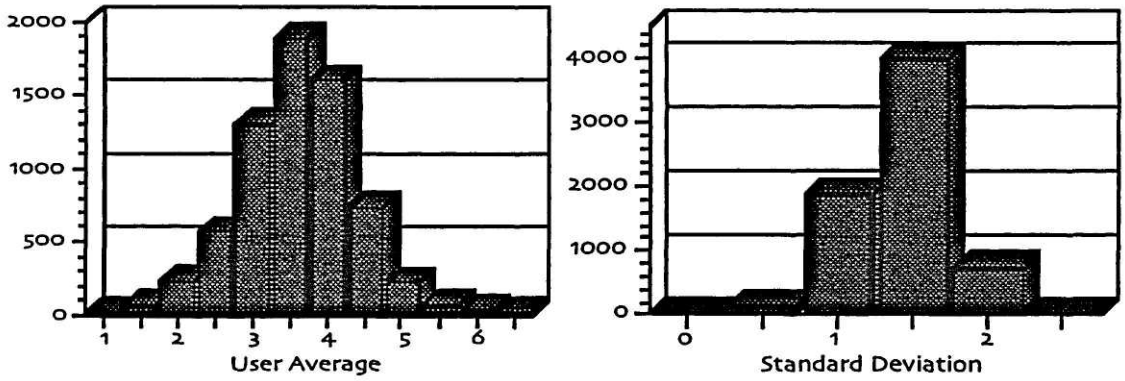
It is also very interesting to look at the distribution of ratings versus the "id" of items. Each item in the system was assigned an id that approximately corresponds to their date of entry in the system. Figure 5 shows that almost one third of the overall ratings are concentrated in ids 0-100. The first hundred artists are the core of the initial artists entered into the Ringo system, and include artists such as the Beatles, Rolling Stones, etc. Of course, these artists may have most of the ratings since they've been around the longest, but the huge disparity suggests more complex factors, such as popularity.

FIGURE 5. Number of Ratings v. Item Id



User averages, shown in Figure 6a, are distributed normally with a slight bias towards the low end. Standard deviation (Figure 6b) of user ratings peak at 1.75, with almost 4,000 users having a standard deviation between 1.5 and 2.0.

FIGURE 6. User Profile Averages



Conversely, we can examine ratings for artists (Figure 7 on page 41). Top line statistics are:

- Average of 253.3 ratings per item, with a standard deviation of 799.
- Median: 18 ratings per item

We would expect to see similar distributions when compared to user averages, which we do. There is a slight spike at around 6.5, which is mainly the "garage bands" entered by users, rated highly by a few people, and never seen again. When we restrict the sampling to only those items with over 20 ratings, our spike quickly disappears. The standard deviation in ratings is very similar to that of the users, although slightly more spread out. The greater spread proves that users tastes truly do vary; there are few artists loved by everyone or hated by everyone.

Figure 8 shows the frequency count of the number of ratings for items. From it we see that over 1500 items have less than 10 ratings, and over 1000 items have more than 100 ratings. Few items lie between these two rating spikes, showing that things are usually well-known or unknown, but not often in between.

FIGURE 7. Item Profile Averages

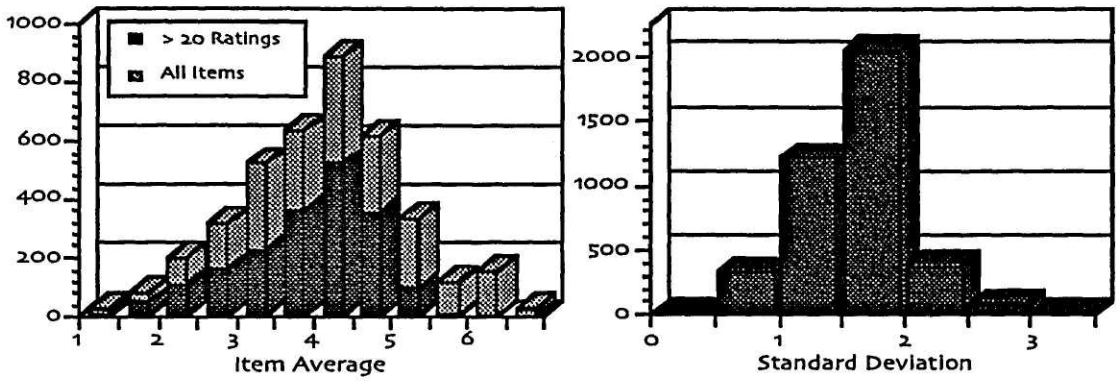
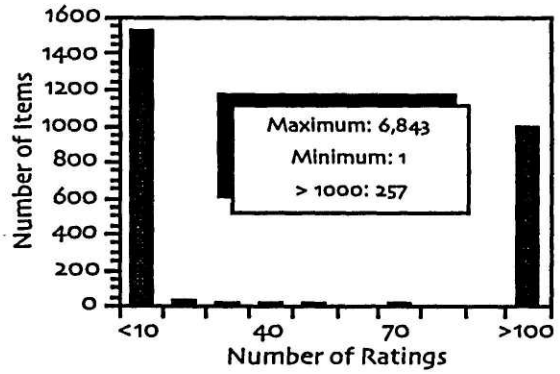


FIGURE 8. Frequency Count of Number of Ratings



Evaluation Criteria

There are numerous tests which can be applied to judge the fitness of a given prediction algorithm in various circumstances. The initial Ringo[1] tests identified several metrics by which automated collaborative filtering systems can be measured. This thesis advances these metrics and analyzes many second order effects that were not analyzed in Ringo. Furthermore, we compare the results of the larger data set in HOMR to the Ringo results to find out if the algorithm has improved over time.

Previous Results

Many of the algorithms mentioned here were originally tested in the Ringo system. When possible, results are compared for both the

original Ringo system and HOMR. There are however a few differences in the generation of the results:

- The Ringo tests were based on 1,000 users with 1,876 artists, and approximately 130,000 ratings.
- The Ringo tests dropped 20% of the entire system's ratings, and then attempted to predict that subset using the remaining 80%.
- The HOMR tests drop 20% of ***each*** user's ratings and attempt to predict that subset using the remaining, fully intact, user base. For users with few ratings, this test is quite difficult, more so than the Ringo test.

By using this new metric, our tests focus on the user experience rather than the system experience. In other words, it is much more relevant to ask "How well does the system perform for user X?" rather than "How well did the system do when it had only 80% of its total ratings?"

For a summary of other differences, see Table 2. Also, when possible, correlations between predicted ratings and actual ratings are given in the form of Pearson r correlations. The Bellcore[13] system mentions maximum correlations of .62 using its algorithms. The correlation in the Bellcore system is based on a 90%/10% split rather than 80%/20%. Our algorithms performed slightly better with a 90%/10% split than with an 80%/20% (as would be expected). For purposes of comparison with the Ringo system, 80%/20% results are presented.

TABLE 2. *Differences Between Ringo and HOMR Data Sets*

	Users	Artists	Albums	Ratings	Mean Rating	Mean # Ratings
Ringo	1,000	1,876	Unknown	132,452	3.70	106
HOMR	8,506	3,902	11,000	925,752	3.57	112

Performance data for both Ringo and HOMR was gathered for the following features:

- Average Error

  We would like to minimize the error between the predicted value for a given item and the known value.

- Extreme Average Error

  While it is important in general to reduce the average error, it is even more important to reduce error in the cases where the user would have given an artist an "extreme" rating. In other words, if a user would have rated *Madonna* with a 7 and HOMR predicts a 5, this is a much more costly error than if the user would have given her a 5 and Ringo predicted 3.

- Standard Deviation

  The standard deviation of the error tells us how wildly HOMR behaves. We can decide what behavior is acceptable for this feature, as simple minimization may not be desirable. For example, if Algorithm X has an average error of .3 but a standard deviation of 1, and Algorithm Y has a standard deviation of .1 but an average error of 1, we may choose Algorithm Y.

- Predictive Power

  To test the system, we remove 20% of the data and attempt to predict those ratings from remaining data. In some cases, there may not be enough information to make a prediction for an artist in the target set. We would like to maximize the "predictive power" of the algorithm, minimizing the number of artists for which we can make no prediction.

- Optimal threshold

  For some of the algorithms in Ringo, a threshold was applied to all nearest neighbors. That is, all nearest neighbors within distance L from the user were consulted to make a prediction. We would like to understand how the choice of threshold affects the aforementioned metrics.

The HOMR tests additionally gathered the following data:

- Optimal Number of Neighbors

  In addition to the distance threshold from Ringo, HOMR also uses a numerical threshold on the maximum number of neighbors considered to make a prediction. We would like to address two issues in regards to this numerical threshold. First, does it improve performance relative to non-numerically thresholded algorithms? Second, what is the relationship of this threshold to the performance of the system?

- Optimal Number of Ratings

What is the relationship between the number of ratings a user provides and the average error? Furthermore, for a given user, does the error improve or worsen as ratings are added? We would expect the performance to improve as the system gathers more data about a user, but we could also imagine a cut-off, after which HOMR can find no new items to recommend.

- Minimizing Impact of Errors

  If we make an error, we would like the error to be on the correct side of a users average. We would rather exaggerate a user's rating than simply be incorrect.

- Pearson r correlation

  Taking a cue from the Bellcore[13] team, we've measured the correlation between the target rating set and the predicted ratings. The Bellcore tests showed that a maximum correlation of .91 can be reliably achieved in practice, since the correlation between a user's ratings over time is on average .83. (See "videos@bellcore.com" on page 19.) Note that maximizing Pearson correlation does not directly correspond to algorithm performance. For example, an algorithm with .1 average error and a low Pearson correlation would always be preferable over an algorithm with high average error and high correlation. From a user perspective, average error is much more noticeable than correlation.

Summation Techniques

As mentioned in "The Basic Idea" on page 38, part of the recommendation process involves calculating a weighted sum of nearest neighbors ratings. There are two primary techniques we have tested for calculating recommendations:

- Weighted average of all neighbors ratings, and
- Weighted average of distance from each user's average.

Weight

We must first arrive at a weighting scheme for nearest neighbors. Clearly, we would like to have users with higher similarity (lower distance values) weighted more heavily. Currently, all our algorithms employ a ***threshold*** L above which neighbors are not considered for ratings. Effectively, this provides an upper bound on similarity, which allows us to calculate the weight for user x in relation to user y as:

$$\omega_x = \frac{L - D_{xy}}{L} \quad \text{(EQ 4)}$$

Where D_{xy} is the "distance", or dis-similarity, between two users calculated by the ACF algorithm.

Method 1: Weighted Average

The weighted average of ratings for artist j for user x is then:

$$\hat{G}_{xj} = \frac{\sum_{n \in N_{xj}} \omega_n H_{nj}}{\sum_{n \in N_{xj}} \omega_n} \quad \text{(EQ 5)}$$

Given:

- N_{xj}: The subset of the neighbors of x who have rated artist j

Method 2: Weighted Deviation

Our second method of calculation postulates that different users conceive the 1-7 rating space differently, and attempts to scale each users space differently (i.e. a 5 for user A may correspond to a 6 for user B):

$$\hat{G}_{xj} = \frac{\sum_{n \in N_{xj}} \omega_n (H_{nj} - \bar{U}_n)}{\sum_{n \in N_{xj}} \omega_n} + \bar{U}_x \quad \text{(EQ 6)}$$

Remembering:

- U_n: The set of all ratings by user n.

Both methods were tested with each of the algorithms mentioned below. In almost all cases, straight averages performed better. In one case however, extreme rating error was minimized with method 2. In general, it is a simple test to decide which method performs better. Rather than attempting to draw a general conclusion for all algorithms which would be tenuous at best, one must simply try both methods and determine which performs best for the given algorithm and parameter settings. However, computationally, method 1 is always less expensive, and thus may be preferable for speed reasons.

The Base Algorithm: Averages

We mentioned earlier that users today use "unpersonalized" recommendation as one way to navigate information spaces. For example, Billboard presents a global average of dj's opinions about music artists. We argue that personalized recommendations provided by an automated collaborative filtering system such as HOMR generate "better" recommendations, where better is defined by the host of metrics outlined above.

Straight averages then provide a basis for comparison for our more sophisticated algorithms.

Plain English

To calculate the predicted rating for user i for artist j, simply average all the ratings for artist j in the whole database.

The Algorithm

Given:

- H_{ij}: The set of all user ratings
- H_j: The set of all ratings for item j

$$\hat{G}_{ij} = \overline{H}_j \qquad \text{(EQ 7)}$$

The Results

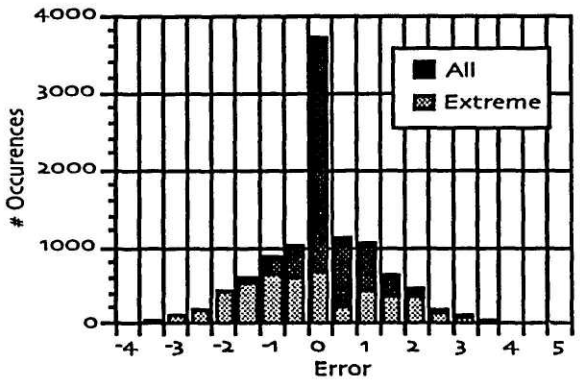
The algorithm performed much better than expected, and perhaps is a testament to the fact that many users have "average" taste, or that our user group is fairly homogeneous.

Metric	HOMR	Ringo
Average Error	1.25 (18%)	1.3 (19%)
Extreme Ratings Average Error	1.74 (25%)	1.8 (26%)
Standard Deviation of Error	.86	1.6
Pearson Correlation	.54	n/a
Predictive Power	100%	90%
Correct Side of Average	65%	n/a

We can also examine the distribution of errors (Figure 8). The error when considering all items is relatively good, with a normal distribution around 0. However, in extreme cases, the distribution breaks down, with large spikes on the negative side (when the predicted rat-

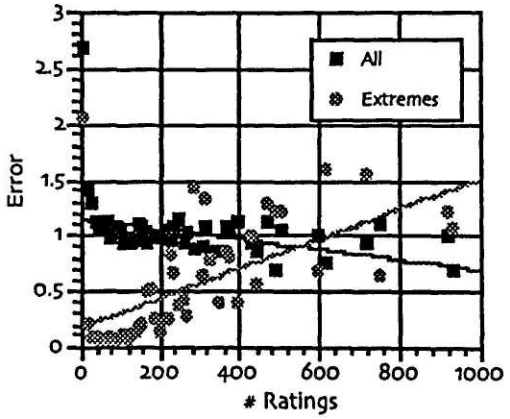
ing is higher than the actual rating). This spike is explained by the fact that item averages tend towards 4, and there are many low ratings in the system.

FIGURE 9. Base Averages: Error Distribution



Finally, we also examine average error vs. number of ratings in Figure 10. When examining all ratings, we see a desired down trend in the error as users enter more ratings. When examining only extreme ratings, we see an upward trend as the number of ratings increase. It is believed that this trend is due to two factors. First, since these ratings are "extreme", there is more room for error, since averages tend towards 4. Second, since there are more ratings, there are correspondingly more extreme ratings, yielding more chances for these more serious errors.

FIGURE 10. Base Averages: Error vs. # Ratings



Mean Squared Distance

Plain English

For each pair of users, calculate their similarity based on the average squared distance between their ratings. Calculate a prediction based on a weighted sum of the ratings of the nearest N neighbors below a given threshold (L).

The Algorithm

We follow the plan in "The Basic Idea" on page 38, and define the "dis-similarity" between two users as D_{xy}:

$$D_{xy} = \frac{\sum_{i \in I} c_{ix} c_{iy} (H_{ix} - H_{iy})^2}{\sum_{i \in I} c_{ix} c_{iy}} \quad \text{(EQ 8)}$$

Given:

- H_{ij} = Rating of item i by user j (1-7)
- I: The set of all items in the database
- $c_{na} = \begin{cases} 1 & H_{na} \neq 0 \\ 0 & H_{na} = 0 \end{cases}$

We can then set the following parameters:

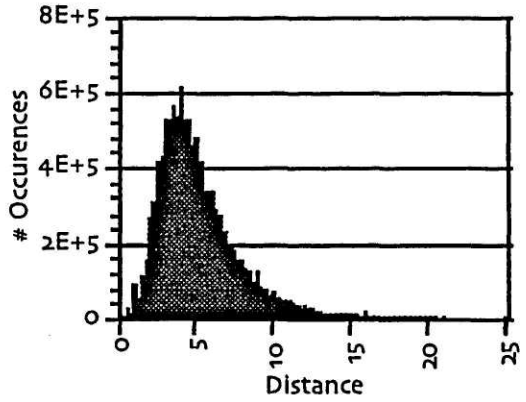
- L: A threshold on D such that if D_{xy} is below L, x will be in y's neighbor set, (and vice versa).
- max |N|: A maximum number of users that can be in a users neighbor set. Obviously, max |N| and L interact to determine the neighbor set. If there are 500 users whose distance to a given user is under L, and max |N| is 100, only the top 100 users from the set of 500 will be chosen. In theory, we can remove this threshold entirely by setting it to be the number of users in the system. In practice, however, this would be quite a painful solution in terms of storage, since we must keep the neighbor set in storage.
- C: A minimum number of ratings which two users must have in common to be considered for inclusion in N.
- P: A percentage of ratings users must have in common (which interacts with C). Note that the percentage is taken relative to the user whose neighbor set is being constructed. For example, if calculating neighbors for user A, who has 10 ratings, any user with

(P*10) ratings in common with user A would be considered for inclusion. Inclusion would not necessarily be symmetric, as this other user may have many more ratings. This allows "newbies" to receive recommendations quickly without hurting the recommendations given to users with many ratings.

The Results

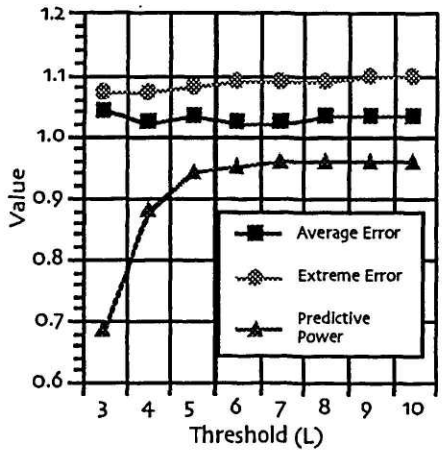
The distribution of distance values between users is shown below:

FIGURE 11. Distribution of Interuser Distances (MSD)



The distribution shows that a threshold between 0 and 15 would be possible, and more practically between 3 and 10. Further testing (Figure 12) shows that a threshold of 3 works best, with a standard setting of other parameters. Clearly these parameters interact with each other, and performance of one must be linked to the performance of others. In the time frame of this thesis however, we wish to examine each parameter independently. A genetic algorithm could be used to optimize the overall parameter set.

FIGURE 12. **Effect of Varying L with MSD**



Varying P and C

No appreciable difference was seen when varying P (percentage threshold) between 80% and 100%. Performance quickly dropped off however, when P was below 80%. Likewise, varying C (common threshold) slightly didn't affect error performance greatly, but impacted predictive power. When C dropped below 10, error quickly approaches the Base Average algorithm.

As in Base Averages, we can examine the distribution of the error, and the error vs. the number of ratings a user has provided. We see the same upward trend in the extreme errors as we did with the Base Averages algorithm.

FIGURE 13. **Error Distribution: MSD**

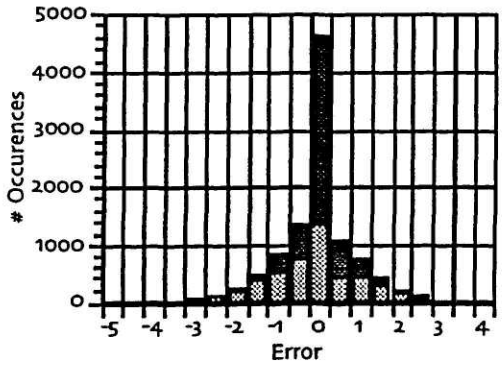
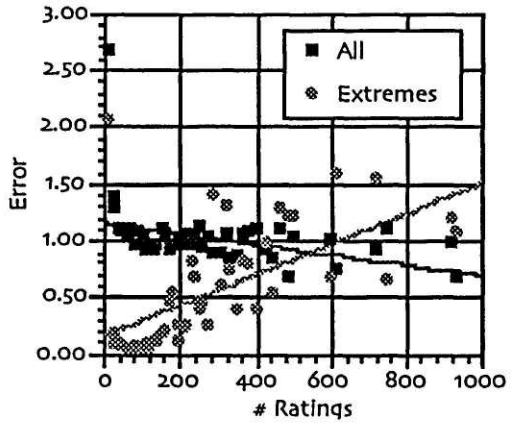
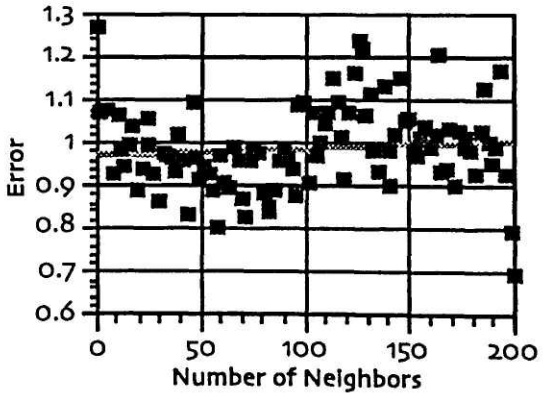


FIGURE 14. Error vs. Number of Ratings (MSD)



Finally, we can also examine the error versus the number of neighbors used to make the decision. Recall that from a given users neighbor set, all users that have rated a given artist are used to calculate the predicted rating for that artist. If we have chosen the optimal neighbor set size, we would think that the more users used to make this prediction, the better it becomes. In the extreme case, if we consult *all* users in the database, we arrive at the Base Average case. If we consult only several neighbors, single errors will have a large impact on predictions. We see in Figure 15 that there is little relationship between the number of neighbors and error for the MSD algorithm. There is a slight upward trend, but no obvious correlation.

FIGURE 15. Error vs. Number of Neighbors (MSD)



Metric	HOMR	Ringo
Average Error	.96 (14%)	1.0 (14%)
Extreme Ratings Average Error	1.23 (18%)	1.2 (17%)
Standard Deviation of Error	.77	1.3
Pearson Correlation	.65	n/a
Predictive Power	95%	70%
Correct Side of Average	65%	n/a

Overlap MSD

Motivations

General MSD has shown much promise. Unfortunately, it gives no preference or credit to users who have many ratings in common. For example, if two users have 20 ratings in common with an MSD of 1, they will be considered more similar than two users who have 50 ratings in common with an MSD of 1.001. In essence, we would like to give more credit to users who have more ratings in common, since we can be more "confident" in our measurement of similarity.

Plain English

The algorithm proceeds exactly the same as standard MSD except that we scale the distance by the percentage of ratings the users have in common.

The Algorithm

For simplification, let's define the set C_{xy} which is the set of all items that both user x and user y have rated. Mathematically:

$$C_{xy} = \{\forall I | (c_{ix}c_{iy} = 1)\} \quad \text{(EQ 9)}$$

Define the "similarity" between two users as D_{xy}:

$$D_{xy} = \frac{\sum_{i \in C_{xy}} (H_{ix} - H_{iy})^2}{|C_{xy}|} \times \frac{|C_{xy}|}{\sum_{i \in I} c_{ix} + \sum_{i \in I} c_{iy} - |C_{xy}|} \quad \text{(EQ 10)}$$

Which of course simplifies by cancellation to become:

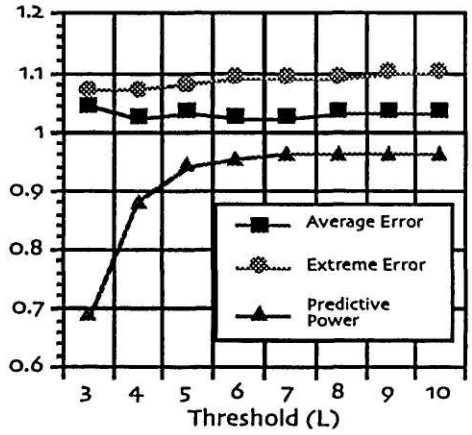
$$D_{xy} = \frac{\sum_{i \in C_{xy}} (H_{ix} - H_{iy})^2}{\sum_{i \in I} c_{ix} + \sum_{i \in I} c_{iy} - |C_{xy}|} \quad \text{(EQ 11)}$$

We can then change the same parameters available for MSD.

Results

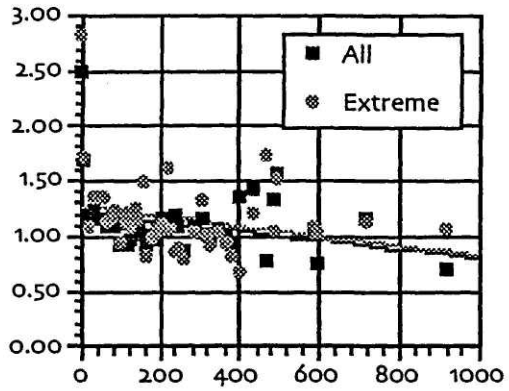
Figure 16 shows that varying L does not greatly affect performance of error parameters, but can improve predictive power drastically. It would seem that 4 would be a good choice for this parameter.

FIGURE 16. Effect of Threshold (L) on OSD Performance



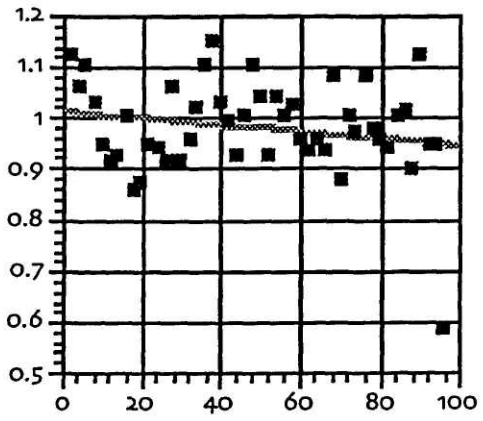
Revisiting Error vs. Number of Ratings (Figure 17), we see the desired down trend in both all ratings and extreme ratings. In Mean Squared Distance, no further credit was given to users with more ratings in common. As users gather more ratings, they are more likely to have more ratings in common with other users (obviously). Therefore, our neighbor set *does* get more "confident" as the number of ratings goes up. The improved performance proves the theory that users with more items in common should be preferred.

FIGURE 17. Error vs. Number of Ratings (OSD)



Examining error vs. number of neighbors (Figure 18) also shows a down trend as the neighbors go up. The distribution of error (shown previously in Figure 13) is a normal distribution much as in MSD, so it is not shown again.

FIGURE 18. Error vs. Number of Neighbors (OSD)



Overall results show that Overlap MSD performs significantly better than MSD, especially in the case of extreme ratings. We sacrifice a bit in predictive power which can be regained at a small cost in error by raising the threshold L (Figure 16).

Metric	HOMR
Average Error	1.02 (15%)
Extreme Ratings Average Error	1.07 (15%)
Standard Deviation of Error	.8
Pearson Correlation	.62
Predictive Power	88%
Correct Side of Average	76%

Powered MSD

Motivations

Powered MSD addresses the same issues as Overlap MSD, but in a slightly different mathematical way.

Plain English

Follow the same procedure as MSD, except instead of dividing by the number of ratings in common, divide by the number of ratings in common raised to a power greater than 1 (call this power k).

The Algorithm

$$D_{xy} = \frac{\sum_{i \in C_{xy}} (H_{ix} - H_{iy})^2}{|C_{xy}|^k} \qquad \text{(EQ 12)}$$

Given $k > 1$. In the case k=1, obviously, we have the normal MSD distance metric.

Results

Powered Squared Distance (PSD), does not provide better results than OSD, which attempts to get at the same concept. Therefore, limited tests were conducted, and only top line statistics are presented here. Optimal k seemed to be 1.5, with several values between 1.1 and 2 tested.

Metric	HOMR
Average Error	.96 (14%)
Extreme Ratings Average Error	1.24 (18%)
Standard Deviation of Error	.77
Pearson Correlation	.62

Metric	HOMR
Predictive Power	90%
Correct Side of Average	75%

Constrained Pearson

Plain English

Calculate the Pearson correlation between two users, assuming each user's average is 4. This additional constraint (assumed average) provided significantly improved results over standard Pearson correlations. Derivation of this formula and evidence of its performance can be found in [1].

The Algorithm

Define the distance between two users as:

$$D_{xy} = \frac{\sum_{i \in C_{xy}} (H_{ix} - 4)\,(H_{iy} - 4)}{\sum_{i \in U_x} (H_{ix} - 4)^2 \sum_{i \in U_y} (H_{iy} - 4)^2} \qquad \text{(EQ 13)}$$

Remembering that:

- U_n: The set of all ratings by user n.

Results

Initial testing showed very poor results. However, we must remember that the above formula expresses a *correlation* between two users rather than a distance. Therefore, we need not generate weights for each user by the weight equation on page 45, we can use the D value directly. More reasonable results were then achieved. The distribution of interuser similarities (correlations in this case) is shown in Figure 19. In the graph, lower value represents lower similarity. We see a spike at 1, which means a pair of users has equivalent ratings for all items in common. If we only examine users who have at least 10 ratings in common (C=10), we remove that spike entirely. The other line on the graph represents C=25, and is also without a spike near 1.

FIGURE 19. Interuser Correlations (Pearson)

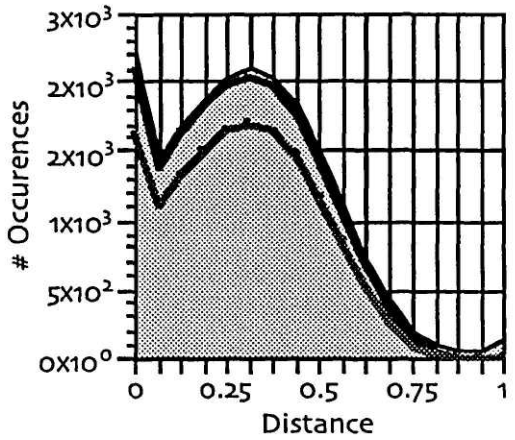


FIGURE 20. Error vs. Number of Ratings (Pearson)

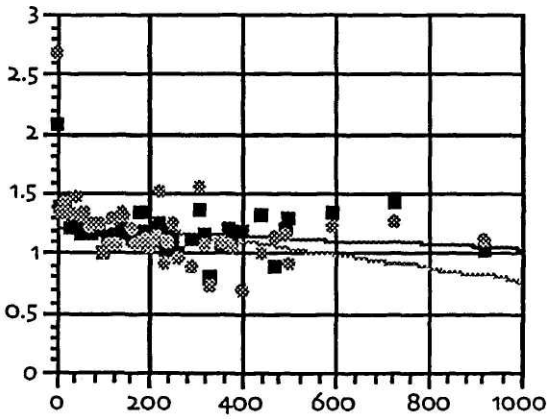
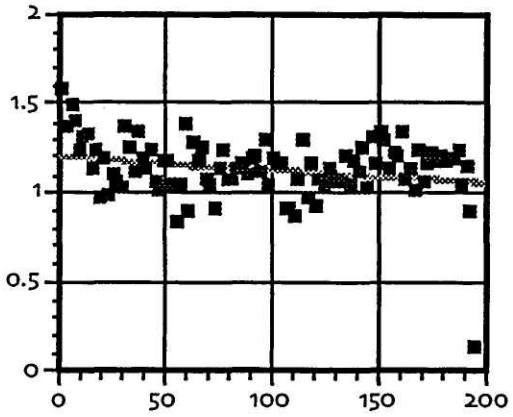


FIGURE 21. Error vs. Number of Neighbors (Pearson)



Metric	HOMR	Ringo
Average Error	1.15 (16%)	1.1 (16%)
Extreme Ratings Average Error	1.15 (16%)	1.3 (19%)
Standard Deviation of Error	.97	1.3
Pearson Correlation	.56	n/a
Predictive Power	98%	97%
Correct Side of Average	70%	n/a

Metral-Hack

Motivations

We would like to capture not only the average difference in users ratings, but also the standard deviation of this difference.

Plain English

Multiply the standard MSD calculation by the standard deviation in the error.

The Algorithm

$$D_{xy} = \sigma_{xy} \times \frac{\sum_{i \in C_{xy}} (H_{ix} - H_{iy})^2}{|C_{xy}|} \qquad \text{(EQ 14)}$$

Given:

$$\sigma_{xy} = \sqrt{\sum_{i \in C_{xy}} \frac{\left((H_{ix} - H_{iy}) - \frac{\sum_{i \in C_{xy}} (H_{ix} - H_{iy})}{|C_{xy}|} \right)^2}{|C_{xy}| - 1}} \quad \text{(EQ 15)}$$

Once again, we can modify the same parameters as MSD.

Results

Initial results using this algorithm were not promising. While standard deviation is an important component of similarity measurement, it seems that more subtle measurements such as Pearson are more effective in factoring in deviation of error.

Metric	HOMR
Average Error	1.22 (17%)
Extreme Ratings Average Error	1.55 (22%)
Standard Deviation of Error	.97
Pearson Correlation	.49
Predictive Power	84%
Correct Side of Average	67%

Popularity-Scaled MSD

Motivations

In all of the above algorithms, we've been focusing on user profiles, without regard to the actual items contained in those profiles. In other words, until now it has been "worth" the same amount if we've both rated "The Beatles" highly or if we've rated "The Garage Band Down The Street" (i.e. an unknown band) highly. Perhaps this equality is artificial. If very few users have rated a particular artist, yet two users have this rating in common, their opinions may tell us much more about their correlation. Since our number of ratings per item are so varied (standard deviation of 799), using the number of standard deviations away from the average (number of ratings) will not help. Our problem is that the distribution of ratings per item is not a normal distribution. Instead we choose a percentage of the maximum number of ratings. We chose 1% initially, which corresponds to

60 ratings. Our initial choice was fairly random, but tests of other values showed it to be optimal.

Plain English

If two users have a rating in common for an item which has less than 1% of the maximum number of ratings for an item, multiply the difference in their ratings by a factor q (q>1).

The Algorithm

Define:

- T_j: Set of all ratings for item j
- S: 1% of the maximum number of ratings for any item (max T_j)
- $Q_x = \begin{cases} q & |T_x| < S \\ 1 & \text{Otherwise} \end{cases}$

$$D_{xy} = \frac{\sum_{i \in I} Q_i c_{ix} c_{iy} (H_{ix} - H_{iy})^2}{\sum_{i \in I} c_{ix} c_{iy}} \qquad \text{(EQ 16)}$$

Results

Popularity scaled mean squared distance performs almost exactly as well as standard MSD, with a slightly higher standard deviation and a lower predictive power. However, it does predict on the right side of

Metric	HOMR
Average Error	.97 (14%)
Extreme Ratings Average Error	1.22 (17%)
Standard Deviation of Error	.85
Pearson Correlation	.59
Predictive Power	85%
Correct Side of Average	74%

the average more often (74% rather than 65%).

Clustering

Many users have expressed the fact that they have "multi-genre" interests. In other words, some users like country and pop, some like pop and rock. With general ACF as implemented in HOMR, two such

users would likely not correlate overall. However, if we were to merely analyze similarity in terms of pop, we may be able to make a better prediction for another pop artist. This is our motivation for attempting cluster-based predictions using the above algorithms. There are two phases to testing the effect of clustering on the prediction algorithm:

1. Selecting clusters
2. Generating recommendations based on selected clusters

Selecting Clusters

There are many ways to choose clusters of artists. Traditionally, artists are clustered by their "industry-given" genre. All folk artists are placed together in the stores, etc. HOMR collects such genre information from users, but it is sparse and not always correct. However, users can ask for a recommendation in a given genre. Test results are presented below.

Genre-Based Clusters

Initially, clusters were based on user-supplied genre categorizations. Top line statistics show worse performance than generalized MSD in most features. However, genre-clusters perform slightly better in extreme cases.

Metric	Cluster	Non-Cluster
Average Error	1.08 (15%)	.96 (14%)
Extreme Ratings Average Error	1.1 (16%)	1.23 (18%)
Standard Deviation of Error	.95	.77
Pearson Correlation	.49	.65
Predictive Power	73%	95%
Correct Side of Average	71%	65%

Generated Clusters

It is much more interesting to glean the clusters from the data itself. We have almost 1 million ratings to analyze, and 10,000 different

tastes. We should be able to find reasonable clusters in this data. In practice, it turns out we have too much data, and many packages were used to try and generate clusters from the raw data. In the end, a data reduction technique known in pattern recognition as eigenanalysis was used to reduce the space before clustering was run. "Eigenfaces"[5] is a specialized application of eigenanalysis that has been used for face recognition. The technique attempts to reduce the dimensionality of a space by finding a set of basis vectors which minimizes the squared-error distance between the original data and the dimensionally-reduced data. The general process to cluster artists is as follows:

1. Select a set of *S* "test artists" randomly from the set of all artists. The size of this set S should be around 100, or can be calculated based on what your computer can handle. This limit is based mainly on available memory, since you must store a large matrix. If custom packages were used, incremental approaches could be employed to raise this limit. In this thesis, off the shelf packages were used.
2. Generate the matrix A which contains all user (size=*b*) ratings for the artists in *S*. The dimensions of this matrix are then *b* x *s*. b is the entire user base, and is therefore approximately 10,000.
3. Convert each rating point to a "distance from the average" for that artist; i.e. if artist A has an average rating of 4, convert all 7's for that artist to 3's, and 3's to -1.
4. We wish to find a matrix E such that (AE) gives us a reduced dimension approximation to A. Each column of E is an element in a set of orthonormal vectors e_k such that:

$$\lambda_k = \frac{1}{|S|}\sum_{n=1}^{|S|}\langle e_k s_n\rangle^2 \qquad \text{(EQ 17)}$$

   is a maximum.
5. The vectors e_k and scalars λ_k are the eigenvectors and eigenvalues of the covariance matrix C, where by definition $C = \frac{1}{s-1}AA^T$. The eigenvectors and eigenvalues of this matrix satisfy:

$$Ce_k = \lambda_k e_k \qquad \text{(EQ 18)}$$

6. Unfortunately, C is b x b. Most computers cannot yet handle matrix problems of this size in the time it takes to complete a master's thesis; moreover, C is not of full rank, so the problem is ill-conditioned. Fortunately, we can solve a smaller, better-conditioned problem and get e_k.

Take a deep breath, and let's solve the simpler problem.

1. Form the matrix $\hat{C} = A^T A$, which is of dimension s x s.
2. Solve the eigenvector problem for $\hat{C}$, finding eigenvectors v_i and eigenvalues μ_i.

$$A^T A v_i = \mu_i v_i \qquad \text{(EQ 19)}$$

3. Left-multiply both sides of the previous equation by A:

$$AA^T (A v_i) = \mu_i (A v_i) \qquad \text{(EQ 20)}$$

From which we see the eigenvectors of the original problem (C), are $e_i = A v_i$.

4. To calculate each item's position in the new space, we take its ratings vector A_n (which consists of all ratings for item n) and multiply by our transformation matrix E.

$$\alpha_i = A_i^T E \qquad \text{(EQ 21)}$$

We can then cluster on this new space in reasonable amounts of time. The clustering algorithm chosen was k-means clustering[16] which is quite simple:

1. Randomly assign all items to a cluster (the number of desired clusters was predetermined).
2. Calculate the centroid of each cluster.
3. For each item, find out which cluster centroid it is closest to, and move it to that cutesier. Recalculate the centroids whenever you move an item.
4. Measure the intracluster distance.
5. Repeat steps 3 and 4 until the intracluster distance is under a given value, or until you've completed a certain number of iterations.

The initial run through this algorithm generated 50 clusters. While this number was not heavily justified, other sizes were tested and 50 produced the smallest intracluster distance with a reasonable number of items per cluster. Further testing would be conducted if more time was available. There were an average of 80.4 elements per cluster, and the median was 53, the minimum and maximum were 19 and 475, respectively. The standard deviation, unfortunately, was 90.2.

Results

For each user, we separated the ratings into clusters, and dropped 20% of each clusters ratings if there were more than 5 ratings in that cluster. If there were less than 5 ratings in a given cluster, we didn't attempt to predict ratings for the cluster. We then recalculated the user's nearest neighbors based only on items in a particular cluster. Based on that neighbor set, we attempt to predict the missing values for that cluster. We repeat this for each cluster and each user, and arrive at the same performance statistics we measured for other algorithms. We chose to calculate cluster neighbors based on mean squared distance, since it was computationally simplest. What we wish to learn is whether or not cluster predictions improve upon the basic algorithm.

Using the 50 cluster set described previously, the same tests were applied. The generated clusters performed better, in the extreme rating case, than both the genre-based clusters and standard MSD. With further refinement of cluster selection, this technique is quite promising.

Metric	Cluster	Non-Cluster
Average Error	.97 (14%)	.96 (14%)
Extreme Ratings Average Error	1.04 (14%)	1.23 (18%)
Standard Deviation of Error	.84	.77
Pearson Correlation	.61	.65
Predictive Power	86%	95%
Correct Side of Average	76%	65%

Key Result Summary

Table 3 on page 66 shows results for the best parameter settings of each of the tested algorithms. The best scores in each metric are highlighted. Depending on the desired application, different algorithms are optimal based on which features are most important in that domain. In the music domain, the average and extreme error are quite important. Therefore, the best algorithm would likely be Overlap Mean Squared Distance, or the dynamically generated cluster-based approach previously described.

TABLE 3. ***Summary of Results for all tested algorithms***

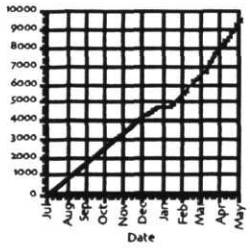
	Pearson Correlation	Correct Side	Average Error	Extreme Error	Predictive Power	Standard Deviation
Base	0.54	65%	1.25	1.74	100%	0.86
MSD	0.65	65%	0.96	1.23	95%	0.77
OSD	0.62	76%	1.02	1.07	88%	0.80
PSD	0.62	75%	0.96	1.24	90%	0.77
Pearson	0.56	70%	1.15	1.15	98%	0.97
MHack	0.49	67%	1.22	1.55	84%	0.97
Pop MSD	0.59	74%	0.97	1.22	85%	0.85
ClusterMSD	0.61	76%	0.97	1.04	86%	0.84
GenreMSD	0.49	71%	1.08	1.10	73%	0.95

Best 2 scores in each category highlighted. MSD = Mean Squared Distance. OSD = Overlap MSD. PSD = Powered MSD. PopMSD = Popularity Scaled MSD.

Discussion/System Statistics

Section Summary. **The HOMR World Wide Web server provided perhaps the first truly interactive social information filtering system. System usage statistics provide us with pointers to desired features and under-used features. Furthermore, this section analyzes a very important issue with social information filtering: vulnerability to attack. How can the data and recommendations be comprised, and if they can, how can this be avoided?**

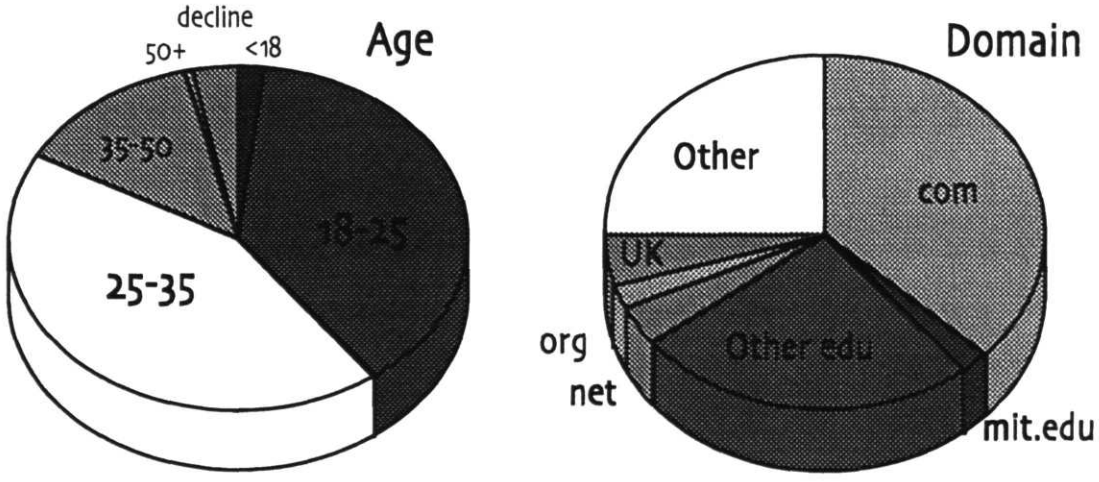
System Statistics



The World Wide Web server came on line December 11, 1995. Since then, over 5,000 users have joined. The margin figure shows the rate of growth of the system, with the initial Ringo system growth extrapolated from checkpointed statistics. Over 400,000 documents have been accessed from the server. Users must log in to the system to access recommendations and other personal functions; the server has had 29,000 such logins.

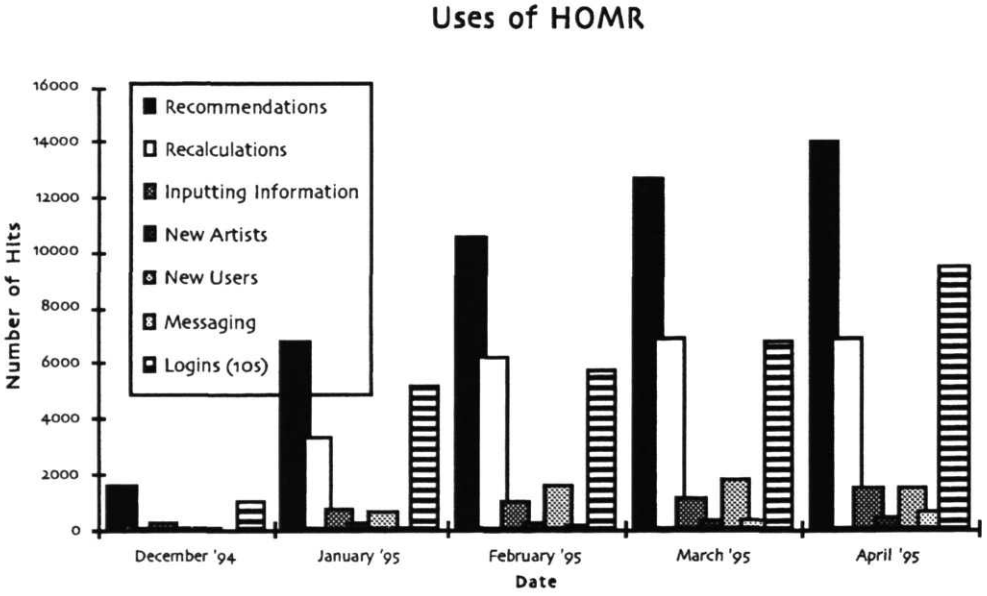
We collected demographic information on users who subscribed to the Web version, on an optional basis. The results are shown below.

FIGURE 22. HOMR User Demographics



Statistics on server usage were also gathered:

FIGURE 23. Web Server Usage by Function



Many users submitted unsolicited comments about HOMR, and most of them were good. Several "flames" arrived, but in almost all cases were quickly followed by, "It's so much better now that I entered some more ratings," or, "I didn't understand the concept, but now it makes complete sense." Some other comments are presented here:

User Comments

"HOMR IS GREAT FUN! IT'S ALREADY POINTED ME TO SEVERAL ABLUMS I WOULDN'T HAVE KNOWN ABOUT OTHERWISE."

DURING THE RINGO/HOMR TRANSITION: "GOOD LUCK AND PLEASE GET THIS BABY WORKING SOON SO I CAN GET MY CHRISTMAS LIST TOGETHER IN TIME."

"SO FAR YOUR RECCOMENDATIONS HAVE BEEN PRETTY WELL ON! GREAT JOB."

"WHAT A GREAT IDEA!...I'M ALWAYS SEARCHING FOR NEW STUFF TO LISTEN TO, AND IF THE ALGORITHMS EVEN BEGIN TO WORK AT ALL, IT'LL BE A GREAT HELP."

"I AM IMPRESSED. I HOPE YOU GET AN THE BEST SITE OF 1995 AWARD FOR THIS. CONGRATULATIONS. "

"VERY COOL SERVICE. YOUR 'RECOMMENDATIONS' WERE DEAD-ON. SCARY!"

"THE PREFERENCE SIMILARITY MEASURE IS AWESOME! THANKS! "

Vulnerability To Spoofing

A significant concern in a system such as HOMR is the integrity of the recommendations. In a not-for-profit system such as HOMR, we avoid many of these issues since we have nothing to gain by hacking the recommendations. However, we still may be vulnerable to outside forces attempting to skew the system. For example, if a record company wanted to push their new album, they could enter several hundred users who all loved their album. We would like to insure that such an attack would:

1. Be as difficult to accomplish as possible,
2. Have as little effect on "real" users as possible,
3. Be easily detectable.

Security

In many ways, all user-survey systems are vulnerable to such attacks. For example, "The Box," a request-only video channel, allows viewers to call in requests. The most popular requests then get aired. It's a well known and often utilized practice for record companies to flood the phone lines with employees calling and requesting a certain new release. Currently, *The Box* has not stopped this practice, and may likely enjoy the extra revenue.

Luckily, in HOMR, it's a bit more difficult than a simple phone call. The current system requires an application to be filled out, which requests the users email address. Mail is sent to the user with a "confirmation password" which must be used to gain access to the system. From there, a fake user would have to enter a set of ratings, and only when others logged on would they be "exposed to the risk" of getting improper recommendations. Successfully executing such

a campaign would require both large amounts of time and "know how."

Vulnerability

Assuming someone was able to enter many (say 1,000) phony users into the database, how would this affect the average person's recommendations? Overall system performance?

The HOMR system provides "System Wide Charts" which display the most liked (and hated) items in the database. This portion of the system is quite vulnerable to attack. Much like any chart-based system (Billboard, CMJ), HOMR is only as good as the validity of the reporting.

However, each users recommendations are not based on system wide data, and therefore are much less vulnerable to attack. In order to make a difference in a users predictions, the fake users would have to find their way into that users nearest neighbor set. In fact, more than one phony user would likely need to be in the neighbor set, since recommendations are based on weighted sums.

A more intelligent attack could be directed at particular "categories" of users. For example, if one wanted to sell the new Beethoven album, one could start with the fact that Mozart is similar to Beethoven. Next, the criminal could find out what other artists are similar to Mozart; similarly, they could find out what artists are dis-similar from Mozart. Then by creating fake users who rated similar items highly and dis-similar items poorly, the perpetrator would have a much greater chance of infiltrating the neighbor set of desired consumers. In a way, this violation is slightly less serious, since the users affected would have already had a disposition for classical music (in this case). However, we would still like to be able to detect these types of attacks.

Detection

There are several ways to detect attacks. First, one must examine where new accounts are being created. For example, if many accounts are created from foo.com, we may be suspicious that they are fake. On the algorithmic side, we can examine user similarity with a sliding window. In other words, if the average distance

between the last 50 users that were created is n, and suddenly our last 50 users are within n/2 (for example) of each other, we may be under attack. By then examining the ratings of those users, we can find the most highly rated artists by those users and attempt to discover the particulars of the attack.

Conclusions

Future Work

Many areas of future research are outlined in [1], but two stand out as most promising in the near term: genetic algorithms for optimization and integration with content based filtering.

Genetic Algorithms

Our problem is quite well suited to a genetic algorithm. We have a large search space of possible algorithms and parameter settings, and a fairly clear fitness function. We need to optimize fitness globally (for all users) and locally (for each user). It may be that both optimizations are quite similar, in which case a global GA could be run. If not, improving performance for each user while managing storage and complexity will be an interesting area for research.

Content-Based Filtering

In a domain such as music, feature-based approaches are not yet readily available. However, in a document such as World Wide Web documents[2] or news articles there are many features available from the document itself. What benefit, if any, can automated collaborative filtering provide to traditional content-based approaches, and conversely, how can content-based filtering aid ACF. Many of these questions are being examined in our research group, and will be published in [2].

Conclusions

Automated collaborative filtering successfully automates word of mouth on a global scale. This thesis has shown that ACF is a promising and competent method of information filtering, especially in subjective domains such as music. While the optimal algorithm and parameter settings will vary with the domain in question, we believe that automated collaborative filtering will perform comparably in many domains, including books, movies, restaurants, and even Web documents.

Automated collaborative filtering will change the way people navigate large product spaces in the future. In many ways, we are merely aiding an age old process and moving it into the electronic world. Hopefully, ACF will also empower consumers to build communities of people with common interests in any of several domains or subdomains. ACF provides a Consumer Reports of sorts, but personalized to each user. Indeed, ACF can provide personalized special interest magazines of all sorts.

In the coming years, we will likely see the spread of ACF into many existing on-line services, such as Cardiff's movie database, the I-Station, and others. While the technology is solid, the "sale" of the technology will also be critical. With any agent-like system, issues such as privacy, integrity, and trust will always manifest themselves. Users will not feel comfortable providing personal data if they believe it will be sold or used for other purposes. Users will also not trust recommendations from a system loaded with related advertisements. Lastly, each service provider will need to help the system and user build a trusting relationship. In other words, the system shouldn't recommend things with high confidence when it knows very little about a user, rather its confidence should build over time as the user enters more information.

Ringo was a first step in the right direction, and HOMR has continued along that path. We hope to see many more systems sprout with similar concepts and perhaps even better algorithms. We've successfully automated word of mouth, now we may be able to improve it.

Bibliography

[1] Shardanand, Upendra. Social Information Filtering for Music Recommendation. Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1994.

[2] Lashkari, Yezdi. Feature-Guided Social Information Filtering. Master's Thesis Proposal, Department of Media Arts and Sciences, Massachusetts Institute of Technology, October 1994.

[3] Resnick, et al. GroupLens: An open architecture for collaborative filtering of netnews. Sloan Working Paper, 1994.

[4] Sheth, Beerud. A Learning Approach to Personalized Information Filtering. Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, February 1994.

[5] Turk, Matthew; Pentland, A. Eigenfaces for Recognition.. Journal of Cognitive Neuroscience, 3(1): pp. 71-86. 1991.

[6] Stanfill, Craig; Waltz, David. Memory Based Reasoning. Communications of the ACM 29(12):1213-1228. 1986.

[7] Paramount Interactive. Movie Select. Software Application. 1993

[8] Entertainment Decisions, Inc., Evanston, IL. Claire V. Business portfolio.

[9] Hitchings, Hamilton. Personal Communcation. 1994

[10] Deerwester, Scott et. al. Indexing by Latent Semantic Indexing. Journal of the American Society for Information Science 41(6):391-407. 1990

[11] Feynman, Carl. Nearest Neighbor and Maximum Likelihood Methods for Social Information Filtering. Massachusetts Institute of Technology, internal memo, Fall 1993.

[12] Salton, Gerard. Automatic Information Retrieval. Computer, September 1980. pp. 41-56. 1980.

[13] Hill, Will; Stead, Larry; Rosenstein, Mark; and Furnas, George. Recommending and Evaluating Choices in a Virtual Community of Use. Proceedings of Computer and Human Interaction, May 1995. Denver, Co.

[14] Lentin, Kevin; et. al. "DiamondBase Documentation" October, 1994.

[15] Shardanand, Upendra; Maes, Pattie. Social Information Filtering: Algorithms for Automating "Word of Mouth." Proceedings of Computer and Human Interaction, May 1995. Denver, Co.

[16] Therrien, Charles. Decision Estimation and Classification. Wiley & Sons. New York, 1989.