Learning Automated Product Recommendations Without Observable Features: An Initial Investigation

Mary S. Lee and Andrew W. Moore

CMU–RI–TR–95–17

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

April 1995

Contents

List of Figures

List of Tables

Learning Automated Product Recommendations Without Observable Features: An Initial Investigation

Mary S. Lee
6413 Howe Street
Pittsburgh
PA 15206

mslee@cs.cmu.edu

Andrew W. Moore
Carnegie Mellon University
School of Computer Science &
Robotics Institute
Pittsburgh, PA 15213
awm@cs.cmu.edu

ABSTRACT

It is appealing to imagine software packages that provide personally tailored product recommendations to a consumer. One way to predict the rating of a particular product by a particular consumer is through inference from a database of previous ratings by many consumers of many products. Such a database consists of triplets of the form:

(product-identifier, consumer-identifier, rating)

Generally such databases will be sparse, but nevertheless we may hope to derive considerable predictive information from them. A number of groups have begun developing distributed systems to collect and predict consumer preferences. Some have put significant effort into implementation issues to do with user interfaces, and the gathering and communicating of data via Internet and Usenet. Rather than launching into the development of a distributed systhem to address a particular consumer preference domain, our goal is to first understand the computational and statistical nature of the general problem. In this paper we develop two new algorithms for this purpose and also relate them to a nearest-neighbor based algorithm of [Resnick *et al.*, 1994]. We then examine their predictive perfomance and quality of recommendations on a number of synthetic and real-world databases. The real-world results suggest that a significant improvement can be obtained over simply recommending the most popular product in some but not all domains. At the end of the paper we discuss computational expense on large databases, the use of explicit features, and our ideas for improved inference algorithms.

1 Introduction

The ability to predict consumer preferences would be of great benefit in a wide variety of domains. Such a tool could be used to help consumers select which video to rent, which book to read, which restaurant to eat at, or which Usenet articles to sample. Recommendations could be tailored to the individual concerned, based on their own past preferences and the wider database of other consumers' preferences. In this paper we describe and compare three memory-based learning algorithms to make such predictions. We present results both for synthetic data and real data sets, including a database with a total of over eighty thousand movie ratings.

Our algorithms are based on the hypothesis that there are a set of hidden features that determine people's ratings of various products. Given a database of known consumer ratings, we can try to infer the underlying features directly or indirectly. Assume there are N_p products, each identified by an integer $i \in \{1, \ldots, N_p\}$; N_c consumers, each identified by an integer $j \in \{1, \ldots, N_c\}$; and a partial set of known ratings, $R_{i,j}$, where $R_{i,j}$ is the rating of product i by consumer j. Such a database has an unconventional form from the perspective of traditional regression and machine learning approaches, which usually map a set of features onto outputs.

The paper discusses three algorithms for learning in such domains. The first algorithm, linfeat, uses the direct approach to prediction. It makes the assumption that consumer preferences are determined by a linear relationship:-

$$R_{i,j} = \sum_{q=1}^{n} P_{i,q} C_{j,q} + \text{ i.i.d. noise} \tag{1}$$

where n is the number of hidden features, $P_{i,q}$ is the qth feature of the ith product, and $C_{j,q}$ is the qth feature of the jth consumer. Thus a consumer's rating of a product is a weighted sum of that product's features, with the weights being the model of that consumer's underlying preferences. As a matrix equation, $R = PC^T + \text{noise}$. The linfeat algorithm finds the best fit for the two feature matrices, C and P. These can then be used to predict any unknown ratings.

Our second and third algorithms use an *indirect* approach and do not assume a linear model. The second algorithm is very similar to the methods described in [Resnick *et al.*, 1994, Shardanand and Maes, 1995]. These indirect approaches look for individuals whose past ratings have been in close agreement, and then assume that these individuals will agree on future ratings. Given a large enough pool of people, it seems reasonable to expect that for any individual you will be able to find other people whose opinions are similar. One theoretical drawback is that once a group of like-minded individuals have all sampled every product that any of them liked, it is unclear which products to try next. One theoretical advantage of these methods is that they make very few assumptions about the underlying model.

Next we describe our general approach, followed by descriptions of the algorithms we have currently tested, before progressing to empirical results. We then suggest how this initial work could be extended, and discuss related work.

2 Our Approach

A number of groups have begun developing distributed systems to collect and predict consumer preferences, including [Shardanand and Maes, 1995] for the music domain, [Resnick *et al.*, 1994] and [Lang, 1994] for the Usenet domain, and [Hill, 1994] for the video domain. Some of these groups have put significant effort into implementation issues to do with user interfaces, and the gathering and communicating of data via Internet and Usenet.

The ability to analyze and predict consumer preferences will be increasingly valuable in coming years. Potential applications include video-on-demand, TV remote controllers with a trainable "channel-I'll-want-to-watch" button, marketing, book recommendations, and restaurant selection. Rather than launching into the development of a distributed system to address a particular consumer preference domain, our goal is to first understand the computational and statistical nature of the general problem. We are working to answer questions such as:

- How accurate can we hope to be for general predictions on real world data? Our initial results–and those reported in [Shardanand and Maes, 1995]–offer perhaps a twenty percent improvement over the naive algorithm that merely looks at the average popularity of products. We hope and expect that further improvement is possible. If not, it is important to be fully aware of the limitations of the data.

- Can we give more reliable predictions for some products? Even if learning algorithms are unable to produce accurate general predictions, they may be able to give highly reliable predictions for a subset of products. It would be especially advantageous if the algorithms could automatically provide statistical bounds on the accuracy of individual predictions.

- To what extent does the use of explicit features change performance? Incorporating a small number of explicit features, such as the genre of a movie, might be sufficient to increase performance notably.

- How many ratings do we need from consumers? Which ratings help most? It should be harder to predict the preferences of a consumer who has only rated a few products. We would like to know how performance changes with the number of ratings from a user. We would also like to know which products' ratings would most improve our model, so that we can recommend the best experiments.

To begin answering these and other questions, we decided to test the statistical performance of a variety of algorithms on pre-existing databases.

3 The Algorithms

3.1 Common assumptions

In each of the algorithms we begin with the hypothesis that consumers' preferences depend on the values of certain product features. These features can be such things as whether a music album is jazz or country, or whether the main conflict in a movie is resolved by violence.

The underlying features are never given to us explicitly (see Section 5 for possible extensions to this work where some features are given explicitly). Instead the algorithms use a database that contains simple numerical ratings: consumer j tried product i and gave it a rating of $R_{i,j}$. The database is incomplete: not every consumer has already rated every product. Our goal is to learn a model of people's preferences. If we achieve this, we can predict the missing ratings. These predictions could then be used to give individually tailored advice to people—recommending products they are predicted to like, warning them of products they are predicted to hate.

3.2 The linfeat algorithm

The linfeat algorithm makes the assumption that consumers' preferences are determined by a linear model, where a consumer's rating of a product is a weighted linear sum of that product's features. As given above:-

$$R_{i,j} = \sum_{q=1}^{n} P_{i,q} C_{j,q} + \text{ i.i.d. noise} \tag{2}$$

where $R_{i,j}$ is the rating of product i by consumer j, P is the product feature matrix, C is the consumer feature matrix, and n is the number of features.

Given the number of features to use, n, and a partial set of known ratings, $R_{i,j}$, the linfeat algorithm finds the best fit for the feature matrices, C and P. This involves finding a minimum of the sum-squared-error on the training set

$$\sum_{R_{i,j} \in \text{Training set}} \left(R_{i,j} - \sum_{q=1}^{n} P_{i,q} C_{j,q} \right)^2 \tag{3}$$

and as such gives a maximum likelihood estimator of the hidden features subject to the i.i.d. gaussian noise assumption.

The algorithm proceeds by iteratively refining its estimates of C and P using singular value decomposition. We now explain one step in this procedure. Suppose we are trying to find a new estimate for P given our current estimate of C.

Consider product k. Assume it has been rated by a subset consisting of m consumers. Denote the identifiers of those consumers who have rated product k as $\{j_1^k, j_2^k \ldots j_m^k\}$. The ratings they gave are, respectively, $\{R_{k,j_1^k}, R_{k,j_2^k}, \ldots R_{k,j_m^k}\}$. We wish to solve for the product features $P_{k,1}, P_{k,2} \ldots P_{k,n}$, where n is the number of features, using the equations

$$\begin{aligned}
R_{k,j_1^k} &= P_{k,1}C_{j_1^k,1} + P_{k,2}C_{j_1^k,2} + \ldots + P_{k,n}C_{j_1^k,n} \\
R_{k,j_2^k} &= P_{k,1}C_{j_2^k,1} + P_{k,2}C_{j_2^k,2} + \ldots + P_{k,n}C_{j_2^k,n} \\
&\vdots \\
R_{k,j_m^k} &= P_{k,1}C_{j_m^k,1} + P_{k,2}C_{j_m^k,2} + \ldots + P_{k,n}C_{j_m^k,n}
\end{aligned}$$

This is a system of m linear equations in n unknowns. Typically, we expect the number of consumers m who have rated a product to be far greater than the number of features n. In general there will be no exact solutions to these equations, but we can use singular value decomposition to yield the least squares solution for $P_{k,1}, P_{k,2} \ldots P_{k,n}$.

We do this for each product $k \in \{1, 2, \ldots, N_p\}$ in the database in turn, and amalgamate the results to yield the product feature matrix, P.

A similar procedure is used to derive a new estimate of C given the current estimate of P. The algorithm as a whole is summarized next. For further details see appendix A.

1 Initialize the feature matrix, C. The algorithm should not be sensitive to the initial values (we set them pseudo-randomly in the range $[0, 1]$ with the first column values explicitly set to 1).

2 Set quit $:=$ FALSE.

3 Loop until quit $=$ TRUE:

3.1 Given the known ratings, R, and the current value for C, use singular value decomposition to find the best fit for the matrix P.

3.2 Given the known ratings, R, and the current value for P, use singular value decomposition to find the best fit for the matrix C.

3.3 If and only if C and P have converged, set quit := TRUE. Convergence could be determined in several ways. Currently we stop when the RMS discrepancy between the known ratings and the values predicted for those ratings by C and P changes by less than a small value, epsilon, between one iteration and the next.

4 Return the final estimates of C and P, and use them to make any requested predictions.

The algorithm converges very quickly because of the huge jumps provided in feature-space by the alternate steps of maximizing the likelihood of P then C.

3.3 The nearest neighbor based algorithm

In the nearest neighbor based algorithm we assume that there are groups of individuals with similar tastes. These tastes may be complex functions of many product features, but we do not attempt to solve for the features directly. Instead when we wish to predict an individual's rating of a product, we look for other people whose past ratings are in good agreement with this individual's ratings. We call these people the individual's nearest neighbors. Our prediction is then just a weighted sum of the means-adjusted nearest neighbor ratings for the product concerned. This algorithm is very similar to the methods described in [Resnick *et al.*, 1994, Shardanand and Maes, 1995].

We would expect prediction accuracy to improve significantly as the number of consumers in the database increases (and thus the probability of finding people with similar tastes likewise increases).

We now summarize the algorithm to predict consumer j's rating of product k.

1 Let $\{j_1^k, j_2^k \ldots j_m^k\}$ be the set of other consumers who have (a) rated product k **and** (b) rated at least **MIN-COMMON** products that consumer j

has rated.

2 For each $j' \in \{j_1^k, j_2^k \dots j_m^k\}$:

2.1 Let $\mathbf{common}(j, j')$ be the set of products that have been rated both by j and by j'.

2.2 Compute μ_j and σ_j^2 respectively as the sample mean and variance of the ratings of consumer j over all the products in $\mathbf{common}(j, j')$. Define $\mu_{j'}$ and $\sigma_{j'}^2$ similarly.

2.3 Compute the Pearson correlation coefficient between the common ratings of consumers j and j':

$$\rho_{j,j'} = \sum_{i \in \mathbf{common}(j,j')} \frac{(R_{i,j,} - \mu_j)(R_{i,j',} - \mu_{j'})}{\sqrt{\sigma_j^2 \sigma_{j'}^2}} \tag{4}$$

4 Provided $\{j_1^k, j_2^k \dots j_m^k\}$ has at least one member (i.e. $m > 0$) , return the prediction:

$$\text{prediction} = \frac{\sum_{j' \in \{j_1^k, j_2^k \cdots j_m^k\}} (\mathbf{weight}(\rho_{j,j'})(\mu_{j,j'} + (R_{k,j',} - \mu_{j',j})))}{\sum_{j' \in \{j_1^k, j_2^k \cdots j_m^k\}} \mathbf{weight}(\rho_{j,j'})} \tag{5}$$

where

$$\mathbf{weight}(x) = \begin{cases} \exp(K^2 \log(x)) & \text{if } x > 0 \\ 0.0 & \text{if } x \leq 0 \end{cases} \tag{6}$$

and K^2 is a positive number, and $\mu_{j,j'}$ is the mean value of j's ratings over those products that both j and j' have rated.

5 If $\{j_1^k, j_2^k \dots j_m^k\}$ has no members at all, return the average of all j's ratings.

See appendix B for further details.

Note that while the nearest neighbor algorithm may perform adequately on many real datasets, it contains a theoretical weakness. We judge the closeness of two people by the correlation between their ratings, and then make predictions biased toward the ratings of a consumer's nearest neighbors. Yet in theory two consumers' ratings may be perfectly correlated even if their ratings are wildly different. For instance, one consumer might always give a rating precisely fifty times greater than another individual, yielding a correlation coefficient of 1.0 but producing a very unhelpful contribution toward the prediction. Our next algorithm addresses this problem.

3.4 The regression-based algorithm

This algorithm is closely related to the nearest neighbor based method. It arises from the observation that the nearest neighbor method can sometimes make poor predictions even when individuals' ratings are highly correlated.

Given two individuals, the regression-based algorithm solves for the best-fit linear relationship between the two consumers' ratings. We can then directly predict the value of one consumer's ratings from the other consumer's ratings, and use this direct estimate in our weighted sum.

As before, define **common**(j, j') as the set of products rated by both j and j'. We will model the relation between the ratings of j and j' by a line passing through the centroid of their common ratings at angle $\theta_{j,j'}$ to the horizontal, as in Figure 1.

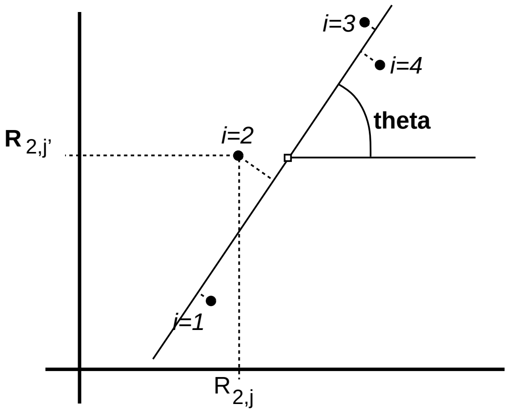

Figure 1: A simple example in which users j and j' share four ratings. The diagram shows the ratings on a 2-d plane. A line through the centroid (the square) is placed at an angle $\theta_{j,j'}$ to minimize the sum of squared perpendicular errors.

The least squares fit (in a perpendicular errors sense) of the line has, as its direction, the principal component of the ratings data. It can be verified that

$$\tan(2\theta_{j,j'}) = 2\frac{\displaystyle\sum_{i\in\mathbf{common}(j,j')} (R_{i,j}-\mu_{j,j'})\,(R_{i,j'}-\mu_{j',j})}{\displaystyle\sum_{i\in\mathbf{common}(j,j')} \left((R_{i,j}-\mu_{j,j'})^2-(R_{i,j'}-\mu_{j',j})^2\right)} \tag{7}$$

where $\mu_{j,j'}$ is the sample mean of j's ratings of the products in $\mathbf{common}(j,j')$, and $\mu_{j',j}$ is the sample mean of j''s ratings of the products in $\mathbf{common}(j,j')$.

To predict $R_{k,j}$, the rating of product k by consumer j, interpolation from our linear model gives:

$$\text{prediction using consumer } j' = \mu_{j,j'} + \frac{R_{k,j'}-\mu_{j',j}}{\tan\theta_{j,j'}} \tag{8}$$

Thus, to predict j's rating of product k, the regression-based algorithm is identical to the nearest neighbor algorithm, except for step 4 which now has[1]:

$$\text{prediction} = \frac{\displaystyle\sum_{j'\in\{j_1^k,j_2^k\cdots j_m^k\}} \mathbf{weight}(|\,\rho_{j,j'}\,|)\left(\mu_j+\frac{R_{k,j'}-\mu_{j',j}}{\tan(\theta_{j,j'})}\right)}{\displaystyle\sum_{j'\in\{j_1^k,j_2^k\cdots j_m^k\}} \mathbf{weight}(|\,\rho_{j,j'}\,|)} \tag{9}$$

To prevent ridiculous predictions, we clip our predictions to lie within the minimum and maximum marks in the known ratings. In the pathological case where $\tan(\theta_{j,j'}) = 0.0$ we return the average of all j's ratings as our prediction.

Note that the regression-based algorithm can make equally good use of strong negative correlations as it can of strong positive correlations.

4 The Experiments

4.1 Test methodology and baseline comparisons

We tested the three algorithms on a variety of datasets, ranging from synthetic datasets to a movie database with over eighty thousand movie ratings. To assess

[1] Instead of weighting according to correlation, we could now weight according to sum-squared errors, or the width of the confidence interval on the estimate of θ. Future work will investigate this.

performance on a given dataset, we randomly split the data into a training set and a test set[2]. Each algorithm in turn is presented with the ratings in the training set (consumer j gave mark m to product i). Having seen the training data, the algorithm then predicts the values of each of the ratings that has been withheld in the test set.

Each algorithm was run for a few values of its main parameter, and the results shown here are for the best setting found. For linfeat we varied the number of features that it attempted to find. For the nearest neighbor and regression-based methods we varied the value of K^2, the parameter that determines how much the weights should be biased toward highly correlated individuals. See Section 5 for a discussion of how cross-validation could be used to tune these parameters autonomously.

In order to provide a baseline for comparison, we also ran the experiments using two very crude prediction methods. The first, globalav, predicts the same value for every single rating, that value being the mean of all the ratings in the training set. The second method, prodav, predicts the same value for every rating of a particular product, that value being the mean of all the ratings in the training set for that particular product.

Clearly we would hope that our learning algorithms would outperform globalav and prodav. In addition, the difference between the errors for globalav and prodav gives us a rough feel for how much improvement we can hope to make. We would expect real world data for such things as movies to contain some products that are widely regarded as better than others. If this is so, then prodav should have lower errors than globalav. If not, or if prodav's errors are only slightly lower than globalav's, then our intuition about real world data may be incorrect—perhaps consumers' ratings are too inconsistent to be helpful in predicting their future likes and dislikes.

[2]When doing the random split, we first ensure that each consumer and product has a few ratings in the training set (randomly selected from all their ratings). This prevents us having to predict ratings for a product that no one in the training set has sampled, or having to predict the opinions of a consumer who has no ratings in the training set.

Method	Mean absolute error	Method - Prodav (95% confidence interval)		Method - Regression (95% confidence interval)	
Linfeat	0.26	[-0.191, -0.185]	$\surd$	[-0.029, -0.026]	$\surd$
Nearest Neigh.	0.29	[-0.164, -0.159]	$\surd$	[-0.0017, 0.0005]	NS
Regression	0.29	[-0.164, -0.158]	$\surd$	(same)	
Prodav	0.45	(same)		[0.158, 0.164]	X
Globalav	0.62	[0.166, 0.174]	X	[0.327, 0.335]	X

Table 1: Synthetic 1: 3213 ratings; training set density = 75%; 50 splits

4.2 Linear model synthetic datasets.

First we present the results for synthetic data generated in accordance with a linear model:

$$R_{i,j} = \sum_{q=1}^{n} P_{i,q} C_{j,q} + E_{ij} \tag{10}$$

where $R_{i,j}$ is the rating of product i by consumer j, n is the true number of features, E_{ij} is a gaussian error term.

This is the same model that the linfeat algorithm assumes, and so we expect linfeat to perform well. Table 1 shows the results for a dataset of 51 products and 63 consumers, generated using three features and gaussian noise with $\sigma = 0.3$; the true ratings range from -2.0 to +2.0. As expected linfeat performs best, with a mean absolute prediction error of 0.26 (this is the mean averaged over the predictions for every withheld rating in each of fifty random splittings of the source data into a test set and a training set). The nearest neighbor and regression methods perform nearly as well, both giving a mean absolute prediction error of 0.29. Prodav performs markedly worse, and globalav lags behind all the others.

Note that the results also list 95% confidence intervals on the difference in performance between each method and prodav, and between each method and the regression algorithm. For clarity, we have added a tick if the method is better at the 95% confidence level, a cross if it is inferior, and NS if the performance difference is insignificant. Thus on this dataset we were unable to reliably distinguish which of nearest neighbor and regression performs best.

Method	Mean absolute error	Method - Prodav (95% confidence interval)		Method - Regression (95% confidence interval)	
Linfeat	0.17	[-0.309, -0.304]	$\surd$	[-0.036, -0.034]	$\surd$
Nearest Neigh.	0.20	[-0.271, -0.267]	$\surd$	[0.0027, 0.0034]	X
Regression	0.20	[-0.274, -0.270]	$\surd$	(same)	
Prodav	0.47	(same)		[0.270, 0.274]	X
Globalav	0.66	[0.185, 0.190]	X	[0.457, 0.461]	X

Table 2: Synthetic 2: 15000 ratings; training set density = 75%; 25 splits

But we can see that linfeat, nearest neighbor, and regression are all significantly better than prodav.

Table 2 shows the results for a dataset of 150 products and 100 consumers, generated using four features and with gaussian noise of $\sigma = 0.2$; the true ratings range from -3.2 to +2.7. As before linfeat performs best, closely followed by nearest neighbor and regression.

So far we have assessed performance by the mean absolute errors over the test sets. Other measures might be more appropriate in some domains. For instance in a commercial application we might only need to reliably recommend a few products that someone will enjoy. This may be easier than making arbitrary predictions. Suppose that for each consumer we want to predict the test-set product they will most enjoy. We can compare the products that our methods predict with the true preferred product. We define the *regret* to be the difference between the ratings of the recommended product and the true optimal test-set product.

For each algorithm, we predicted the products that each consumer would most enjoy. Table 3 shows the resulting mean regrets, together with confidence intervals on the differing regrets between methods. The results are averaged over all the consumers in each of twenty-five test-sets for the same synthetic data as in Table 2. Once again linfeat does best, with a mean regret of just 0.07, and again nearest neighbor and regression perform nearly as well. But this time the disparity between these three methods and prodav and globalav has been magnified. Prodav's mean regret is 0.72, over ten times that of linfeat,

Method	Mean regret	Reg/Method - Reg/Prodav (95% confidence interval)		Reg/Method - Reg/Regression (95% confidence interval)	
Linfeat	0.07	[-0.67, -0.62]	$\surd$	[-0.03, -0.02]	$\surd$
Nearest Neigh.	0.09	[-0.65, -0.60]	$\surd$	[-0.003, 0.002]	NS
Regression	0.09	[-0.65, -0.60]	$\surd$	(same)	
Prodav	0.72	(same)		[0.60, 0.65]	X
Globalav	1.60	[0.84, 0.94]	X	[1.47, 1.55]	X

Table 3: Synthetic 2: 15000 ratings; training set density = 75%; 25 splits

and globalav's mean regret is 1.60. (Globalav's predictions are the same for every product, so when asked to make a recommendation it just picks a random product.) The results for linfeat, nearest neighbor, and regression demonstrate that a good model of the data can lead to excellent recommendations.

The experiments on synthetic data show how the algorithms can perform in theory. If real data followed this type of linear model, linfeat would be ideal. In addition to producing the most accurate predictions, it was also the fastest algorithm. In the following two sections we show how the algorithms performed on real data, firstly on sets of up to four thousand Star Trek episode ratings, and then on databases of up to eighty thousand movie ratings.

4.3 Star Trek Episode Poll Data

We now present results based on people's ratings of Star Trek: The Next Generation and Star Trek: Deep Space Nine television episodes. The source data is collected by Joe Reiss, who solicits votes from readers of the Star Trek Usenet groups. We looked at the performance on two subsets of the raw data, the first a relatively small subset selected to have a high density of ratings. This subset contains 1055 votes from thirty-six people on thirty-three episodes, which is 88.8% of the maximum possible number of ratings for this size group. The ratings were normalized to lie in the range $[0, 10]$. Table 4 shows the results.

On this subset of the data, the regression algorithm has the lowest mean error of 0.95, followed by nearest neighbor and linfeat. Prodav has a mean error of 1.10 (about 15% higher than regression). As expected, globalav does worst

Method	Mean absolute error	Method - Prodav (95% confidence interval)		Method - Regression (95% confidence interval)	
Linfeat	1.03	[-0.07, -0.05]	√	[0.07, 0.10]	X
Nearest Neigh.	1.00	[-0.11, -0.09]	√	[0.04, 0.06]	X
Regression	0.95	[-0.16, -0.14]	√	(same)	
Prodav	1.10	(same)		[0.14, 0.16]	X
Globalav	1.38	[0.27, 0.29]	X	[0.41, 0.44]	X

Table 4: Star Trek 1: 1055 ratings; training set density = 67%; 50 splits

Method	Mean absolute error	Method - Prodav (95% confidence interval)		Method - Regression (95% confidence interval)	
Linfeat	1.10	[-0.12, -0.10]	√	[0.01, 0.03]	X
Nearest Neigh.	1.09	[-0.13, -0.11]	√	[0.002, 0.016]	X
Regression	1.09	[-0.14, -0.12]	√	(same)	
Prodav	1.21	(same)		[0.12, 0.14]	X
Globalav	1.47	[0.25, 0.27]	X	[0.37, 0.40]	X

Table 5: Star Trek 2: 4109 ratings; training set density = 23%; 25 splits

with a mean error of 1.38.

Table 5 shows the results for a larger group: 4109 ratings from 141 people on 94 episodes (31% of the maximum possible number of ratings). This time the mean errors for linfeat, nearest neighbor, and regression are all very similar at 1.09 to 1.10. Prodav's error is about 10% greater at 1.21, and globalav has a mean error of 1.47.

These results show that linfeat, nearest neighbor, and regression all outperform the simple prodav method on at least some types of real world data. Nonetheless the improvement over prodav is lower than we might have initially expected. Moreover the performance difference between prodav and globalav is also lower than we might have anticipated. Globalav predicts a constant value for every single rating, and yet its mean error is only 21% to 25% higher than prodav's, which predicts the average of the known ratings for the episode concerned.

Each week people send in their votes on the latest Star Trek episode to add

Method	Mean regret	Reg/Method - Reg/Prodav (95% confidence interval)		Reg/Method - Reg/Regression (95% confidence interval)	
Linfeat	0.59	[-0.02, 0.04]	NS	[-0.07, 0.02]	NS
Nearest Neigh.	0.58	[-0.03, 0.02]	NS	[-0.07, -0.02]	√
Regression	0.62	[0.005, 0.071]	X	(same)	
Prodav	0.58	(same)		[-0.071, -0.005]	√
Globalav	1.67	[1.02, 1.17]	X	[0.97, 1.14]	X

Table 6: Star Trek 2: 4109 ratings; training set density = 23%; 25 splits

to the database. Perhaps consumers' votes are inherently noisy, reflecting their own mood as much as an episode's quality (though note that the algorithms performed well on synthetic datasets in the presence of appreciable noise). Or perhaps some consumers' average marks drift over time as they become more or less generous, without this trend matching any underlying change in the episodes. Or perhaps there is information in the data that these algorithms fail to exploit.

As with the synthetic datasets, we also looked at the regrets for the Star Trek data (the regret is the difference between the rating of the test-set episode that a method predicts a consumer will most enjoy and the true optimum). Table 6 shows the regrets for the larger Star Trek subset. While globalav now does much worse than the other methods, the other four algorithms all perform very similarly. The mean regret for these four algorithms is substantially lower than their average prediction error: they are better at recommending a single episode that an individual will like than they are at predicting that individual's general tastes. It is interesting that prodav's regret is as low as any other method's. This implies that there is a strong popular consensus on the best episodes. In order to recommend which episode someone will most enjoy, we only need to recommend the episode with the highest average rating.

In another experiment, not described here, we preprocessed the Star Trek ratings, replacing each consumer's raw ratings by their normalized ranked values. The results were similar to those shown here.

There are many potential application domains for consumer preference pre-

Method	Mean absolute error	Method - Prodav (95% confidence interval)		Method - Regression (95% confidence interval)	
Linfeat	1.08	[-0.199, -0.191]	√	[0.017, 0.023]	X
Nearest Neigh.	1.06	[-0.218, -0.213]	√	[-0.002, 0.002]	NS
Regression	1.06	[-0.218, -0.213]	√	(same)	
Prodav	1.27	(same)		[0.213, 0.218]	X
Globalav	1.50	[0.231, 0.234]	X	[0.446, 0.451]	X

Table 7: Movies 1: 56743 ratings; training set density = 42%; 25 splits

diction, ranging from a computer that suggests which video to rent, to an intelligent newsreader that recommends which Usenet articles to browse. In many of these domains we would expect a larger pool of ratings. In the next section we look at results based on sets of up to eighty thousand movie ratings.

4.4 Movie Ratings Data

In this section we show the results of applying our algorithms on a database of movie ratings compiled by Col Needham from an ongoing poll of Usenet readers. Almost ten thousand people have sent in their votes, and the full database contains more than fourteen thousand movies and over three hundred thousand ratings. Each rating is an integer between one and ten.

We first tested our algorithms on a relatively dense subset of this source data: a group of 262 people and 390 movies selected so that no consumer had rated fewer than a hundred of the movies, and no movie had fewer than a hundred votes. Table 7 shows the results for the mean absolute prediction errors over the whole test set. Table 8 shows the results for the mean regret when recommending the movie someone will most enjoy. We see that linfeat, nearest neighbor and regression all perform very similarly, except that linfeat's regrets are about eight percent higher than the other two methods'. All three methods do significantly better than prodav, whose predictive errors and regrets are both about twenty percent higher than for either nearest neighbor or regression. Globalav, as expected, comes a poor fifth.

As with the Star Trek data, it is perhaps surprising that globalav's predic-

Method	Mean regret	Reg/Method - Reg/Prodav (95% confidence interval)		Reg/Method - Reg/Regression (95% confidence interval)	
Linfeat	1.23	[-0.19, -0.10]	√	[0.05, 0.13]	X
Nearest Neigh.	1.15	[-0.26, -0.20]	√	[-0.01, 0.03]	NS
Regression	1.14	[-0.26, -0.21]	√	(same)	
Prodav	1.38	(same)		[0.21, 0.26]	X
Globalav	3.06	[1.63, 1.73]	X	[1.86, 1.97]	X

Table 8: Movies 1: 56743 ratings; training set density = 42%; 25 splits

tion error isn't still greater. Despite predicting the same mark every single time, globalav's mean prediction error is only 18% higher than prodav's. Either the data is noisier than we might have hoped, or there is comparatively little consensus on the quality of films. Given that prodav only improves over globalav by eighteen percent, it is encouraging that the other three algorithms achieve a further twenty percent improvement on prodav.

Figures 2 and 3 give the distribution of the absolute prediction errors for regression and prodav on this dataset. The histograms clearly show that prodav has a higher percentage of significant errors.

Figures 4 to 5 give the distributions of the regrets on this dataset. Prodav has slightly fewer low regrets of value 0 and 1, and slightly more large regrets of value 6 or more.

Tables 9 and 10 show the results on a second subset of the movie database. This time the subset was selected so that every consumer in the group had rated at least two hundred of the movies, but movies could have as few as thirty-four ratings. The total number of ratings in the set was 83305. The results are similar to those just discussed, except that this time linfeat's regret is further behind that of nearest neighbor and regression.

We have now seen that our methods behave reasonably well on real world datasets selected to have a generous number of ratings for each product and each consumer. Tables 11 and 12 show some results of what happens when we relax these constraints. They are based on a subset of the movie database where some movies have just nine ratings, and where some people have only

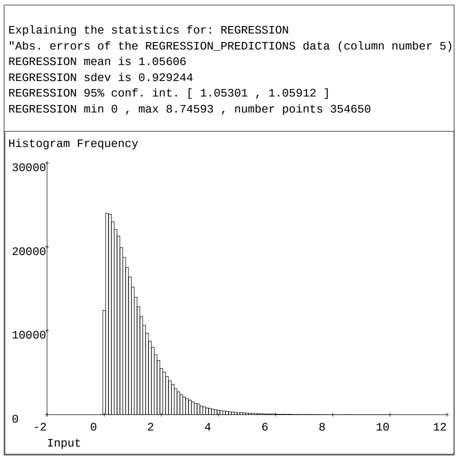

Figure 2: Absolute errors using regression

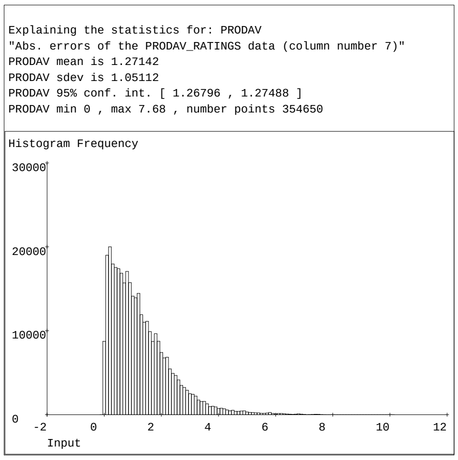

Figure 3: Absolute errors using prodav

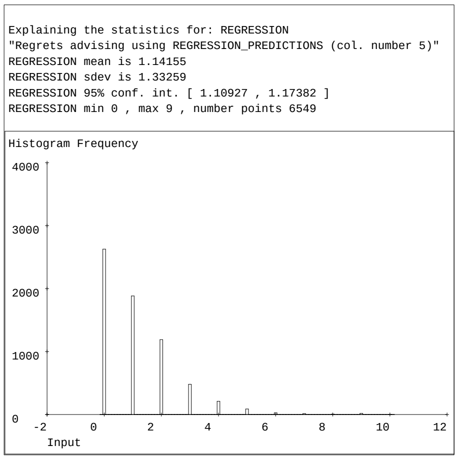

Figure 4: Regrets using regression

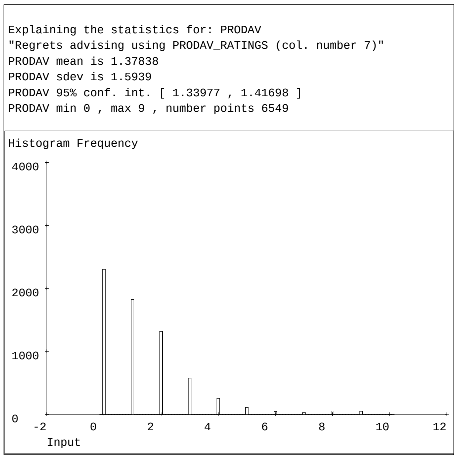

Figure 5: Regrets using prodav

Method	Mean absolute error	Method - Prodav (95% confidence interval)		Method - Regression (95% confidence interval)	
Linfeat	1.09	[-0.188, -0.179]	$\surd$	[0.025, 0.033]	X
Nearest Neigh.	1.07	[-0.211, -0.207]	$\surd$	[0.002, 0.004]	X
Regression	1.07	[-0.214, -0.210]	$\surd$	(same)	
Prodav	1.28	(same)		[0.210, 0.214]	X
Globalav	1.52	[0.243, 0.248]	X	[0.454, 0.460]	X

Table 9: Movies 2: 83305 ratings; training set density = 32%; 20 splits

Method	Mean regret	Reg/Method - Reg/Prodav (95% confidence interval)		Reg/Method - Reg/Regression (95% confidence interval)	
Linfeat	1.35	[-0.10, -0.01]	$\surd$	[0.12, 0.21]	X
Nearest Neigh.	1.22	[-0.22, -0.14]	$\surd$	[0.01, 0.07]	X
Regression	1.18	[-0.26, -0.18]	$\surd$	(same)	
Prodav	1.40	(same)		[0.18, 0.26]	X
Globalav	3.21	[1.74, 1.86]	X	[1.96, 2.09]	X

Table 10: Movies 2: 83305 ratings; training set density = 32%; 20 splits

given eleven ratings. Linfeat, nearest neighbor, and regression still have lower mean prediction errors than prodav, but the margin has narrowed.

Moreover these results show that prodav now has the lowest regrets of the five methods. This last result, however, is somewhat misleading. All the results on regrets were derived from the same runs used to find the mean prediction errors; hence they used the same parameter settings. In practice this is usually a close approximation to the minimal regrets. But with this particular dataset,

Method	Mean absolute error	Method - Prodav (95% confidence interval)		Method - Regression (95% confidence interval)	
Linfeat	1.31	[-0.05, -0.04]	$\surd$	[0.05, 0.06]	X
Nearest Neigh.	1.23	[-0.123, -0.116]	$\surd$	[-0.020, -0.016]	$\surd$
Regression	1.25	[-0.11, -0.10]	$\surd$	(same)	
Prodav	1.35	(same)		[0.10, 0.11]	X
Globalav	1.53	[0.18, 0.19]	X	[0.286, 0.294]	X

Table 11: Movies 3: 20290 ratings; training set density = 9%; 30 splits

Method	Mean regret	Reg/Method - Reg/Prodav (95% confidence interval)		Reg/Method - Reg/Regression (95% confidence interval)	
Linfeat	1.47	[0.14, 0.20]	X	[-0.04, 0.06]	NS
Nearest Neigh.	1.36	[0.02, 0.10]	X	[-0.14, -0.06]	$\surd$
Regression	1.46	[0.11, 0.20]	X	(same)	
Prodav	1.30	(same)		[-0.20, -0.11]	$\surd$
Globalav	2.47	[1.12, 1.22]	X	[0.95, 1.07]	X

Table 12: Movies 3: 20290 ratings; training set density = 9%; 30 splits

linfeat's regrets are substantially lower when it uses one feature than when it uses two (the number that minimized prediction error). Using one feature, linfeat's behavior mimics prodav and it obtains the same regret.

So far we we have compared the accuracy of the various methods. In the next section we consider their computational expense.

4.5 Computational Expense

In this section we briefly compare the computational expense of the different methods.

We ran our algorithms on a Sparc 5. Runtimes varied from under a second up to about twenty minutes for the slower algorithms on large datasets. For instance on the movie data used in Table 7 , which contained some fifty-seven thousand ratings, linfeat took 42 seconds to process the training set and generate predictions for every element in the test set. By comparison, nearest neighbor took seventeen minutes, and regression took twenty-one minutes.

Linfeat is much faster than the nearest neighbor and regression methods. In a practical application it might well be worth trading off its slightly poorer predictions for its greatly increased speed.

5 Possible Extensions

There are various ways to extend this preliminary work. The following are a few of the more interesting possibilities.

5.1 Explicit features

In some domains, it would be relatively simple to give certain explicit features in advance. For instance a video store may already have the genre of the videos stored on computer. The algorithms could then use both the known features and the known ratings to build a model of consumer preferences.

In the case of the linfeat algorithm it is straightforward to see how explicit numerical features would be used: we simply add the appropriate number of columns to the product feature matrix, containing the explicit values of the known features for each product. Non-numerical features, such as the genre of a movie, could probably also be incorporated with a little work.

In the case of the other two algorithms, it is less clear how to proceed. If we have a single explicit feature and it has a small number of possible values (such as the genre of a movie), then we could partition the database so that each genre is considered separately. To predict a consumer's rating of a comedy film, we search for other people who have had similar opinions of comedies, and use their ratings to make our prediction. This may well produce more accurate predictions, as people could have very similar tastes in comedies yet wildly differing opinions on westerns.

Now suppose that we are given n explicit features each of which can take m values. There are n^m sets of possible values for these features. If n and m are of any significant size, partitioning the database into each possible category will produce tiny subsets. Perhaps a sophisticated algorithm could be devised that would search for a small number of optimal partitions of the data based on the explicit features. Alternatively, we could adjust the correlation measure according to the explicit features. Thus when we are trying to make a prediction for a given product, k, we pay most attention to the ratings of those products whose explicit features are closest to k's. It is more significant that consumers' opinions are closely correlated over products similar to k than over very different products.

5.2 Discussion

Here we briefly discuss other ways to extend this work.

- **Other weight functions and improved regression.**

  The nearest neighbor and regression methods both use a weighting function to determine how much to bias ratings toward highly correlated individuals. In our experiments we used the weight function given in section 3.3.

  The performance of the regression method is currently very similar to that of nearest neighbor. In the case of regression, however, we have a statistical tool readily available that may improve performance. A byproduct of a least squares prediction is a measure of the confidence in that prediction. We could use this measure as a statistically sound basis for weighting other people's ratings.

  We would also like to test other weight functions for both regression and nearest neighbor, such as using the k-nearest neighbors.

- **Cross-validation as a higher level control.**

  With our current implementation, for each new dataset one first has to tune the parameters for each algorithm. Using cross-validation ([Stone, 1974], [Wahba and Wold, 1975], [Moore and Lee, 1994]) this process could be automated, and also extended to select such things as the best type of weight function. Ideally cross-validation would be presented with a dataset, and would go away and autonomously test the various algorithms, weight functions, parameter settings, and then return with the optimal tuned algorithm for that dataset. By using its own test set, cross-validation can optimize either for general prediction accuracy or for the expected regret.

- **Faster implementations.**

  Both the nearest neighbor and regression methods are much slower than linfeat, with runtimes of twenty minutes or more to generate complete test-set predictions on large datasets. We would like to speed up our implementations of these algorithms as much as possible. Possible directions here are branch and bound strategies, and the use of non-euclidean kd-tree structures, [Bentley, 1980].

- **Non-numerical data.**

  So far we have only applied our algorithms to numerical ratings and numerical features. It would be a simple extension to allow letter grades (A, B, C) or boolean ratings (liked/disliked). Similarly when extending our algorithms to allow explicit features as discussed in section 5.1, we would like to be able to handle features with a fixed number of categories (western, thriller, science fiction) as well as numerical features.

- **Proposing experiments.**

  In a practical application of these algorithms, people may be willing to risk an occasional experiment in order to improve the average quality of the recommendations. We would like to be able to select the experiment that will best improve our model of the consumer's preferences. This links into the disciplines of experimental design, [Box and Draper, 1987], and active learning, [Cohn *et al.*, 1995].

6 Related Work

In [Shardanand and Maes, 1995] they describe a system called Ringo for making personalized recommendations of music albums and artists. The system has been running on Internet since July 1994, gathering ratings from users via email, and returning recommendations and predictions. They have tested four variants of nearest neighbor techniques. Although their results are drawn from a different domain, they appear to be very similar to those reported here, with mean prediction errors about twenty percent lower than the naive algorithm that uses the average popularity of products.

In the Usenet domain, [Resnick *et al.*, 1994] discuss the use of collaborative filters and describe a prototype system called GroupLens. They have given considerable thought to the architectural issues involved in integrating such filters into Usenet readers and communicating the article ratings from one Usenet site to another. They describe one nearest-neighbor prediction scheme, but to our knowledge have not yet published results on appreciable test sets.

Lang is also investigating collaborative filtering of Usenet articles, and is developing a system called NewsWeeder, [Lang, 1994]. Unlike the above two projects, NewsWeeder looks at the content of the products concerned, processing the text in the Usenet articles. Lang began by trying to predict which newsgroup an article came from, and achieved up to 73% accuracy with a training set of articles drawn from twenty newsgroups. Lang is currently working to extend these results to predicting people's ratings of Usenet articles.

Will Hill at Bellcore is working on a video recommendation service, [Hill, 1994], and has an Internet site that receives ratings from people. But to our knowledge he has yet to publish results.

7 Conclusions

Practical tools to model consumer preferences would be of great benefit in a wide range of domains. With more and more electronic data available on which videos people rent, which Usenet articles they read, even which groceries they buy, it is appealing to use the information to aid people in future choices.

Our goal is to gain a sound understanding of the computational and statistical nature of this general problem. We wish to answer questions such as: "How accurate can we hope to be for predictions on real world data?" "Can we provide statistical confidence intervals on the reliability of our predictions?" "To what extent does the use of explicit features aid performance?"

Note that it is entirely possible that user response would be enthusiastic even for simple schemes that only consider the average ratings of products. Such schemes already offer considerable improvement over random selection, especially when recommending a few products that a user will enjoy. Unless a complex method can perform significantly better still, there may be little demand for it.

In this report we gave some preliminary results on algorithms to analyze past ratings and predict future ratings. We showed that these algorithms can produce predictions that are about 20% more accurate than naive methods on large, real world databases. In future work we hope to devise increasingly accurate, increasingly autonomous tools to model consumer preferences.

8 Appendix A: Linfeat Implementation Details

We briefly mention three refinements to the basic algorithm presented in section 3.2. Firstly, from looking at Equation 2 we see that C and P are underdetermined: if we were to multiply every element of C by gamma and divide every element of P by gamma, their product would have the same value. In an effort to keep the numerical values of a reasonable magnitude, we arbitrarily set the values in the first column of C to be 1.0. Secondly, we keep a record of any products or consumers that led to singularities during the most recent singular value decomposition. If there are any, then they are withheld from the next iterative step. This is intended to prevent deductions based on the most dubious values in the feature matrices. Thirdly, we clip our predictions to lie within the minimum and maximum marks in the known ratings; this is intended to prevent ridiculous predictions.

On the large movie subsets, linfeat used five features to minimize the mean prediction errors. On the Star Trek data and the small movie subset, linfeat used two features to minimize the mean prediction errors. These values were optimized by means of cross-validation.

9 Appendix B: Nearest Neighbor Implementation Details

The weighting function determines how much the ratings are biased toward the consumers whose ratings are best correlated with j's. Many other weighting functions could be used. The particular weighting function given in section 3.3 allows us to smoothly vary the precedence given to j's nearest neighbors. For $K^2 >> 1$, the weighting is only significant for very close neighbors. In the limit as K^2 tends to zero, the prediction for product k is the average of all the ratings for k by positively correlated neighbors who have rated at least **MIN-COMMON** of the same products as j. On the Star Trek and movie ratings databases, the nearest neighbor mean prediction errors were minimized for values of K^2 between 0.5 and 4.0.

If we had used the identity function as the weighting function, $\mathbf{weight}(x) = x$, then this method would be essentially equivalent to the method mentioned in [Resnick *et al.*, 1994] (see section 6).

The restriction to consumers who have rated at least **MIN-COMMON** of the same products as j is a crude measure. It is intended to prevent us from basing predictions on individuals who have only seen a very few products in common with j, yet happen to have given them marks close to j's. In order to be confident that two individuals have similar tastes, we need a reasonable number of pairs of ratings to compare. In our experiments we set **MIN-COMMON** to seven.

Ideally we would dispense with the **MIN-COMMON** ratings restriction. Instead of using $\rho_{j,j'}$ as the parameter to the weighting function in Equation 5, we would use the *probability* that j and j''s tastes are closely correlated.

Since we usually need to make many predictions, we precompute quantities such as the correlation coefficients and store them.

Acknowledgements

We are grateful to Joe Reiss for permission to use the Star Trek episode ratings, and to Col Needham for permission to use the database of movie ratings.

References

[Bentley, 1980] J. L. Bentley. Multidimensional Divide and Conquer. *Communications of the ACM*, 23(4):214—229, 1980.

[Box and Draper, 1987] G. E. P. Box and N. R. Draper. *Empirical Model-Building and Response Surfaces.* Wiley, 1987.

[Cohn *et al.*, 1995] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7.* MIT Press, 1995.

[Hill, 1994] W. Hill. Personal Communication. , 1994.

[Lang, 1994] Ken Lang. Personal Communication. , 1994.

[Moore and Lee, 1994] A. W. Moore and M. S. Lee. Efficient Algorithms for Minimizing Cross Validation Error. In W. W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning.* Morgan Kaufmann, 1994.

[Resnick *et al.*, 1994] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of Computer Supported Cooperative Work*, 1994.

[Shardanand and Maes, 1995] Upendra Shardanand and Pattie Maes. Social Information Filtering: Algorithms for Automating "Word of Mouth". To appear in Proceedings of the International Conference on Computer-Human Interfaces, 1995.

[Stone, 1974] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, B36, 1974.

[Wahba and Wold, 1975] G. Wahba and S. Wold. A completely automatic French Curve: Fitting Spline functions by Cross-validation. *Communications in Statistics*, 4(1), 1975.