Collaborative Filtering with Privacy via Factor Analysis

John Canny
Computer Science Division
University of California
Berkeley, CA 94720
jfc@cs.berkeley.edu

ABSTRACT

Collaborative filtering is valuable in e-commerce, and for direct recommendations for music, movies, news etc. But today's systems use centralized databases and have several disadvantages, including privacy risks. As we move toward ubiquitous computing, there is a great potential for individuals to share all kinds of information about places and things to do, see and buy, but the privacy risks are severe. In this paper we introduce a peer-to-peer protocol for collaborative filtering which protects the privacy of individual data. A second contribution of this paper is a new collaborative filtering algorithm based on factor analysis which appears to be the most accurate method for CF to date. The new algorithm has other advantages in speed and storage over previous algorithms. It is based on a careful probabilistic model of user choice, and on a probabilistically sound approach to dealing with missing data. Our experiments on several test datasets show that the algorithm is more accurate than previously reported methods, and the improvements increase with the sparseness of the dataset. Finally, factor analysis with privacy is applicable to other kinds of statistical analyses of survey or questionaire data scientists (e.g. web surveys or questionaires).

Keywords

Collaborative filtering, recommender systems, personalization, privacy, CSCW, surveys, sparseness, missing data, context-aware computing

1. INTRODUCTION

Our work here builds on the recent paper [2] that introduced collaborative filtering with privacy. That paper describes a protocol for encrypting data and aggregating it and publishing the result. The first contribution of the present paper is to introduce the privacy-preserving CF scheme to the SIGIR community, and give a simplified description of the protocol. The second contribution is a new collaborative filtering scheme preserving privacy which also appears to be the most accurate CF scheme developed so far. The third contribution is a set of experiments using the new method, characterizing its accuracy, speed and storage requirements.

The genesis of this work is the disturbing privacy afronts that have followed from widespread computerization and ecommerce. The fragility of the digital marketplace has put many e–companies in bankruptcy. The courts have supported the rights of liquidators to sell off databases of personal information as an asset. From Amazon.com's policy "In the unlikely event that Amazon.com, Inc., or substantially all of its assets are acquired, customer information will of course be one of the transferred assets." This policy is typical of many other companies and, in spite of Amazon's language, the papers are filled with news of such "unlikely events" which lead to bankrupcies and transfer of customer information.

We are moving towards ubiquitous and pervasive computing environments. Computers on our desks will be supplemented by an array of sensors and small devices in offices and on or in our clothing. It follows that it will be easy and inexpensive to capture much daily activity, and we believe there is great value to being able to share it. Users would be able to get recommendations about restaurants, performances, stores and a variety of other real-world services from logs of other users' locations (time spent in those places) and purchases (via credit card logs and e-wallets). Inferences could be made on implicit data (normal user logs) or explicit data (user's ratings of the services they use). But clearly privacy would be at great risk and must be protected.

Finally, as collaborative filtering techniques become more widespread, they constitute an important part of the process of diffusion of innovations through society [11]. Diffusion in real societies relies on both *homophilous diffusion* where recommendations come from others like the inquirer, and *heterophilous* diffusion, where the inquirer explicitly seeks recommendations from individuals *not* like them (typically from experts or earlier adopters). Today's collaborative filtering systems support homophilous diffusion well, but offer no plausible version of heterophily. One goal of our work is to support recommendations *from designated communities*, such as students, gardeners, surfers, antique collectors, elder citizens etc. etc. We wanted to design a system that would allow non-members of those communities to gain recommendations from them, just as in natural heterophilous diffusion. This pushes two of our design criteria: (i) the pro-

tocol should protect the privacy of individuals in the community while allowing individuals both inside *and* outside the community to gain recommendations from it (ii) a peer-to-peer design is very desirable so that it is as easy as possible for communities to construct electronic CF groups, even for communities with limited resources (e.g. ability to afford, configure and run a server).

We show here that it is possible (and we claim practical) to do useful collaborative computation with personal data without exposing the data, so that it can remain private. This technology opens the door to many novel applications of CF in ubiquitous and everyday settings. In the next section, we describe the probabilistic model of user preferences, and how this model used to generate new recommendations. Then we briefly explain how the model can be computed from encrypted data. Finally, we describe some experiments with the factor analysis model on CF datasets, including a common benchmark dataset. Our experiments support the claim that this is the most accurate collaborative filtering method to date, and that it has other advantages in speed and storage requirements.

2. PROBABILISTIC MODEL

In the most general setting, we want to construct probabilistic models of user behavior/preferences from observations of it. In this paper, we want to extrapolate ratings from these observations for collaborative filtering. We chose a low-dimension linear model of user preferences after consideration of the most successful previous CF algorithms. The following design issues are important:

1. Among the most accurate CF methods to date are neighbor methods using Pearson correlation [1], [7], Singular Value Decomposition [12] and "Personality Diagnosis" [10]. SVD has an explicit linear model, Pearson correlation is known to be equivalent to linear fit. PD is harder to characterize, but if it is indeed "personality" diagnosis, it is known that linear models are good models for human personality traits [8]. So we opted to go with a linear model.
2. CF data is extremely sparse. The EachMovie dataset described later, which contains only 3% of the possible ratings, is considered a "dense" dataset. Other datasets may have 0.1% or even 0.01% of the possible ratings. A good CF method should deal with missing data in a principled way (not by filling missing values with defaults).
3. Signal and noise variances are comparable in typical CF applications (this is an empirical observation we have made). So an explicit probabilistic model should be used, but not e.g. the simple least-squares fitting of SVD which effectively assumes that signal variance is infinite.

For these reasons we chose a linear factor analysis model. Factor analysis [4] is a probabilistic formulation of linear fit, which generalizes SVD and linear regression. Let $Y = (Y_1, \ldots, Y_n)$ be a random variable representing user preferences for n items. Then Y_i is the preference for item i. An observation (lower case) y_j would be a particular set of ratings by user j, so e.g. y_{ij} would be user j's rating of movie number i, say $y_{ij} = 5$ on a scale of 0 to 10.

Let $X = (X_1, \ldots, X_k)$ be a random variable representing a user's k canonical preferences, which is a kind of user profile. For instance, X_1 could be an individual's affinity for blockbuster movies. Other X_i's would affinities for other movie types. We do not need to know the meanings of the dimensions at any stage of the algorithm, and the model is automatically generated. The linear model for user preferences looks like this:

$$Y = \Lambda X + N \tag{1}$$

where Λ is a $n \times k$ matrix. $N = (N_1, \ldots, N_n)$ is a random variable representing "noise" in user choices. The linear model we are looking for comprises Λ and the variances $\mathrm{VAR}(N_i)$ of the noise variables. X is automatically scaled so that it has unit variance, and therefore we dont need to find $\mathrm{VAR}(X)$. For simplicity, we assume all the $\mathrm{VAR}(N_i)$'s are the same, $\mathrm{VAR}(N_i) = \psi$. We also assume that X and N have gaussian probability distributions. If we could observe X and Y at the same time we would have a classical linear regression problem. We can't observe X, and instead we have a factor analysis problem.

There are several algorithms available for factor analysis. We chose to use an EM approach (Expectation Maximization) [3] for two reasons: firstly because it has a particularly simple recursive definition which can be combined with our privacy method, and secondly because it can be adapted to sparse data. When we solve the EM equations, we start by computing an approximation to Λ using linear regression. We do this because the regression equations are also compatible with our privacy scheme, and each regression iteration is faster than an EM iteration.

2.1 Linear Regression

We noted earlier that linear regression can be used to generate a low dimensional linear model if we new the values of X. A slightly different version of linear regression is also possible by ignoring the distribution of X, and minimizing the mean-square noise. This version produces the same, low-dimensional subspace as SVD and can will generate the same recommendations as [12]. Because it ignores both the prior on X and sparseness, this method is always less accurate than factor analysis. Nevertheless it is useful both as a very simple and reasonably accurate method for generating recommendations, and also as a quick way to initialize the factor analysis model.

The input to the regression scheme is an $n \times m$ matrix y, containing the ratings of m users on the n items. Here y_{ij} is the rating of user j on item i. Each column of the matrix y consists of one user's ratings. Each user's observations can be thought of as an observation of the random variable Y. Corresponding to the matrix y is a $k \times m$ matrix x. Each column of x is an estimate of the value of the random variable X (user's preferences in the k dimension preference space) for one user. The linear regression iterations give good starting values for Λ for EM iteration. In the interests of completeness, we give the regression iteration formula here. Let $\Lambda^{(p)}$ be the value of Λ after the p^{th} iteration, and Λ with no superscript is the value before the p^{th} iteration.

Then the iteration is:

$$\begin{aligned} x &= (\Lambda^T\Lambda)^{-1}\Lambda^T y \\ \Lambda^{(p)} &= yx^T(xx^T)^{-1} \end{aligned} \quad (2)$$

where Λ^T is the transpose of Λ. Both the matrices to be inverted are $k \times k$ matrices. Here k is the dimension of the preference space, and it is small and independent of the number of users or number of items. We typically used k values of between 4 and 20.

In typical cases, the matrix y is very sparse, because most users rate only a small fraction of all items. The regression recurrence assumes a dense matrix y. If the data is sparse, the usual remedy is to replace missing elements with the average of some non-missing elements, either the per-user (column) average, or the per-item (row) average, or the average over all ratings. We used the third method for simplicity, and because the regression model is only an initializer for EM, so great accuracy isnt necessary. Note that defaults are only used in the regression phase. The factor analysis iteration works directly with the sparse data, so no defaults are needed.

To initialize Λ *before* the linear regression begins, we fill it with random values with gaussian distribution. Then we scale each of its columns to have a magnitude which matches the expected RMS noise for that CF domain (RMS noise is $\sqrt{< N^2 >}$ where $<>$ is expected value). The RMS noise for typical collaborative filtering domains is about 20% of the rating range. The variable ψ should be initialized to the variance of the noise (the square of the RMS noise value).

2.2 EM Factor Analysis

We use an Expectation Maximization recurrence to solve the factor analysis model. The recurrence we use is well-known [4] although the way we deal with sparseness (missing values) is new. We believe that careful treatment of sparseness is essential for good performance of CF algorithms, given the (usually) very high fraction of missing data. Most previous approaches have used defaults or inferred the missing values. This is simply incorrect from a statistical point of view. The available user ratings induce a probability distribution for each missing rating, which cannot be represented by any single value. For a detailed discussion of these issues we refer the reader to [5].

We start first with the dense situation, assuming that every user has rated every item. Once again y is the $n \times m$ matrix of ratings of n items by m users, and x models the users' coordinates in preference space. The recurrence is

$$\begin{aligned} M &= (\psi I + \Lambda^T\Lambda)^{-1} \\ x &= M\Lambda^T y \\ \Lambda^{(p)} &= yx^T(xx^T + m\psi M)^{-1} \\ \psi^{(p)} &= (1/nm)\mathrm{trace}(yy^T - \Lambda^{(p)}xy^T) \end{aligned}$$

This recurrence is based on a maximum likelihood model, and because of that it can be adapted to any subset of the evidence y. First of all, we rewrite the recurrence in terms of individual user's data. Let y_j be user j's vector of n ratings, and x_j be user j's estimated coordinates in preference space.

Our approach to dealing with sparseness is based [5]. The idea is to compute symbolic expected values for all the missing quantities *including quantities derived from missing ratings*, and to substitute them in the formulas so they are complete. Note that this is not equivalent to substituting only the expectations of missing ratings [5]. One must effectively work with the *distributions* of missing ratings, not just their expectations. We will not give the derivation here, and it turns out that substituting for quantities derived from missing ratings causes them to cancel later in the analysis. One can in fact derive the same set of equations by treating missing values as if they were never there. We will give an informal exposition here, and rely on [5] for proofs of the validity of this formulation.

2.2.1 Sparse Formula

Because user j hasnt rated every item, we need to introduce an $n \times n$ "trimming" matrix I_j. The matrix I_j is a diagonal matrix with 1 in positions (i, i) where i is an item that user j *has* rated, and I_j is zero everywhere else. The matrix I_j allows us to restrict the formulas for user j to the items they have rated. We have

$$\begin{aligned} \Lambda_j &= I_j\Lambda \\ M_j &= (\psi I + \Lambda_j^T\Lambda_j)^{-1} \\ x_j &= M_j\Lambda_j^T y_j \end{aligned} \quad (3)$$

and notice that all of these calculations can be done by user j, assuming they know Λ, which is public information.

The next line comes from the maximization step of EM [4]

$$I_j\Lambda^{(p)}(x_jx_j^T + \psi M_j) = y_jx_j^T$$

which is a tricky equation to solve when summed over all users because Λ, the matrix we would like to solve for, is surrounded by non-constant factors. We will skip the details, but we can rewite the above as:

$$I_j \otimes (x_jx_j^T + \psi M_j)\mathrm{L}(\Lambda^{(p)}) = \mathrm{L}(y_jx_j^T)$$

where $\otimes$ is the kronecker product, and $\mathrm{L}(A)$ for a matrix A is the vector obtained by "stacking" columns of A one on top of another. These operations are available in many linear algebra packages (e.g. $\otimes$ is "kron" in Matlab and $\mathrm{L}(A)$ is $A(:)$). Once we have this form for the equations, we can add them up for all users and solve for $\Lambda^{(p)}$. The result is

$$\mathrm{L}(\Lambda^{(p)}) = \left(\sum_{j=1}^{m}\frac{1}{n_j}I_j \otimes (x_jx_j^T + \psi M_j)\right)^{-1}\sum_{j=1}^{m}\frac{1}{n_j}\mathrm{L}(y_jx_j^T) \quad (4)$$

where n_j is the number of items that user j actually rated. The factors $(1/n_j)$ give appropriate weights to each user, and without them users who had voted for many items would overly influence the model.

Finally, we need to update ψ, which is slightly changed from before to

$$\psi^{(p)} = \frac{1}{m}\sum_{j=1}^{m}\frac{1}{n_j}(y_j^Ty_j - \mathrm{trace}(\Lambda^{(p)}x_jy_j^T)) \quad (5)$$

Equations (3), (4) and (5) are the complete EM iteration for sparse factor analysis, which we will call EM-FA. In order

to distribute the iteration among users, we decompose the sparse formula in the following way. Each user j computes

$$\begin{aligned} A_j &= \tfrac{1}{n_j} I_j \otimes (x_j x_j^T + \psi M_j) \\ B_j &= \tfrac{1}{n_j} y_j x_j^T \\ C_j &= \tfrac{1}{n_j} y_j^T y_j \end{aligned} \tag{6}$$

where $\otimes$ is the Kronecker product (e.g. kron() in Matlab). The user sends the results in encrypted form (see next section) to a totaler(s). The totaler(s) computes

$$\begin{aligned} \mathrm{L}(\Lambda^{(p)}) &= \left(\textstyle\sum_{j=1}^m A_j\right)^{-1} \sum_{j=1}^m \mathrm{L}(B_j) \\ \psi^{(p)} &= \tfrac{1}{m} \textstyle\sum_{j=1}^m (C_j - \mathrm{trace}(\Lambda^{(p)} B_j)) \end{aligned} \tag{7}$$

where $L(A)$ is the matrix A written as a vector with its columns stacked one on top of another ($A(:)$ in Matlab). The totaler(s) decrypts the results and sends to all the clients. So the iterative procedure goes in rounds, with back-and-forth communication between clients and totalers using equations (3), (6) and (7). We call these the distributed EM-FA equations.

As suggested in the last section, our procedure is to initialize Λ and ψ, run a few (say 10) iterations of linear regression (which also easily distributes among clients) to move Λ nearer the maximum likelihood value, and then to run iterations of EM-FA until convergence is obtained. In practice, the EM-FA procedure converges so reliably that we use a fixed number of iterations (15-25 typically) of EM-FA. From then on, we continue to run iterations periodically, which will keep the model fitted to the data as it changes.

2.3 Obtaining Recommendations

The result of the iteration above is a model Λ, ψ of the original dataset y. It can be used to predict a user's ratings from any subset of that user's ratings. Let y_j be user j's ratings. User j should download Λ, ψ and then locally compute x_j using equation (3). From x_j, we compute Λx_j which is a vector of ratings for all items. The ratings in this vector for missing items are the predictions for user j.

2.4 Removing Means

The factor analysis model assumes that the random variable Y has mean zero. In practice this isnt true. We can make it approximately true by subtracting an estimator of the mean for each user rating. If y is a training set, then the means should be subtracted before the model Λ, ψ is built from y. Then when recommendations are computed, we take the j^{th} user's ratings y_j, subtract the means for all items and make the predictions for missing ratings using equation (3). The means should be added back to the missing ratings before they are used. There are two basic estimates for the mean of a given rating: the per-user mean, or the per-item mean. Either of these can be used. For datasets where the number of users is much larger than the number of items (as for Eachmovie or Jester), the per-item average will usually be more accurate. For other datasets, either estimator may work well, and it is best to experiment to see which is better. We implemented both methods.

3. PRESERVING PRIVACY

Having shown that we can reduce factor analysis to an iteration based on vector addition of per-user data A_j, B_j and C_j, we next sketch how to do vector addition with privacy. Putting both procedures together gives us factor analysis with privacy. The scheme we use for vector addition is the same as [2]. We will not repeat the derivation or mathematics here. We will give a high-level sketch of the method which we think should be useful for understanding the method. Among other things, it uses a social notion of trust that could well be the source of some interesting studies. We assume that a fraction α of users are honest. The value α must be at least 0.5 and preferably > 0.8. The goals of the protocol are that: The server should gain no information about an individual user's data y_j, and both user's data and totaler's calculations should be proved correct (no cheating). Our protocol achieves those goals [2]

The method uses a property of several common encryption schemes (RSA, Diffie-Hellman, ECC) called homomorphism. If M_1 and M_2 are messages, and $E(.)$ is an encryption function, it turns out that

$$E(M_1)E(M_2) = E(M_1 + M_2)$$

where multiplication is ring multiplication for RSA, or elementwise for DH or ECC. By induction, multiplying encryptions of several messages gives us the encryption of their sum. This seems to get us halfway there - we can add up encrypted items by just multiplying them. But how to decrypt the total?

The decryption scheme is somewhat more involved. It relies on key-sharing. The key needed to decrypt the total is not owned by anyone. It does not exist on any single machine. But it is "shared" among all the users. Like a jigsaw puzzle, if enough users put their shares together, we would see the whole key. There is some redundancy for practical reasons - we would not want to require all the users to contribute their shares in order to get back the key, or we could probably never get it back. Because the item that has been shared among the users is a decryption key, they can use it to create a share of the decryption of the total. To clarify this, everyone has a copy of the encrypted total $E(T)$. Each person can decrypt $E(T)$ with their share of the key, and the result turns out to be a share of the decryption of T. By putting these shares together, the users can compute T.

To summarize the outcome of the protocol: Each user starts with their own preference data, and knowledge of who their peers are in their community. By running the protocol, users exchange various encrypted messages. At the end of the protocol, every user has an *unencrypted* copy of the linear model Λ, ϕ of the community's preferences. They can then use this to extrapolate their own ratings using equations as described in section 2.3. At no stage does unencypted information about a user's preferences leave their own machine. Users outside the community can request a copy of the model Λ, ϕ from any community member, and derive recommendations for themselves as described in section 2.3.

There are some extra details to make sure that users and totalers (who compute the totals) are not cheating. We use a technique called zero-knowledge proofs (ZKPs) to require each user to prove that their hidden vote is valid (within

the allowable range of ratings, so they cant excessively influence the total). And we use a sampling technique to check totalers' totals for accuracy. As well as making errors, totalers can compromise users' privacy by deliberately leaving out (or multiply adding) their data. The interested reader is refered to [2]. The protocols are simple and efficient, and although the numbers required are quite large (160 to 1024 bits) the bandwidth and computation demands are reasonable.

This protocol is designed to be robust with a fraction (say up to 0.2) of cheating totalers and users. It is also robust against a reasonable number of sites being offline (these parameters can be adjusted but 50require a server, although it does leverage some non-trivial peer services which are available now (in systems like Groove or Oceanstore [9]) and which are likely to be built into infrastructure in the near future. We would like any group of users to be able to create and maintain a collaborating affinity group that shares data internally and allows other groups to use it. The privacy model may seem unorthodox but we believe it is very natural in distributed settings. This claim merits a long discussion which we cannot give here. Obviously there are many interesting sociological and policy questions raised by the model and we hope to see some of these explored in the future.

4. RELATED WORK

Two recent survey papers on collaborative filtering [1], [7], compared a number of algorithms for accuracy on widely-available test data. Generally speaking, they found that neighbor weighting using Pearson correlation gave best accuracy among the algorithms considered at that time. Slight modifications to the Pearson method, like the significance weighting scheme from [7], can improve its performance by one or two percent. We implemented several of these extensions, but significance weighting was the only extension that reliably improved accuracy. We used it in our comparison experiments.

Since the surveys, there have been a few papers which gave comparable or better results than Pearson correlation on some datasets. The first uses SVD [12], which gives a linear least-squares fit to a dataset. SVD was used in our first paper [2] on CF with privacy. There are differences in the method of generating recommendations from the SVD however, and our scheme from [2] is based on a maximum likelihood model, and was more accurate than the scheme from [12] in experiments. The present work (sparse factor analysis) differs on both [2] and [12] by using the same probabilistic formulation to generate recommendations from a model, and to construct the model itself. It is always more accurate than either of the SVD schemes.

Another recent paper uses a probabilistic method called "personality diagnosis" (PD) [10], and gives better accuracy than Pearson. As we will see in the next section, PD is more accurate than Pearson, but less accurate than sparse factor analysis on available data.

5. EXPERIMENTS

To provide better comparability with earlier results, we reimplemented the best performing method (Pearson correlation with significance weighting [7]) from prior papers, and repeated published experiments on available datasets to check our implementation. Then we compared Peason and our scheme on some new datasets to give a broader comparison. All the code for our algorithms, along with our implementations of Pearson as well as the SVD algorithm from [12], are available in MATLAB from the project website `www.anonymous.website`.

5.1 Evaluation Metric

Several evaluation metrics for collaborative filtering have appeared in the literature. The most common is the MAE or Mean Absolute Error between predicted and actual ratings for a set of users. We used MAE exclusively in our experiments for several reasons. First of all, it is the most commonly used metric and allows us to compare our results with the largest set of previous works. Secondly, it correlates well with other metrics. In a recent paper by the Grouplens group [12] they noted that statistical metrics such as MAE, RMSE (Root Mean Squared Error) and Correlation "track each other closely". They also noted that MAE and the decision metric ROC "provide the same ordering of different experimental schemes". Third, the differences in MAE between our scheme and others are quite large compared to earlier comparison papers. It seems unlikely that this disparity would be reversed under a different metric. Instead we concentrated on testing on a diversity of datasets to see how strong the disparity was across them.

5.2 Eachmovie Dataset

The EachMovie dataset was created by Compaq Equipment Corporation from a collaborative filtering site for movies that they ran. Eachmovie comprises ratings of 1628 movies by 72916 users. The ratings are on a scale of 0-5. The dataset has a density of approximately 3%, meaning that 97% of possible ratings are missing. Eachmovie was one of the datasets used in the recent survey by Breese [1]. Breese et al. tested several methods on the Eachmovie dataset, and used two different metrics, one of which was MAE. They found that that best performing algorithm on the EachMovie dataset under both metrics was a neighborhood scheme based on Pearson correlation.

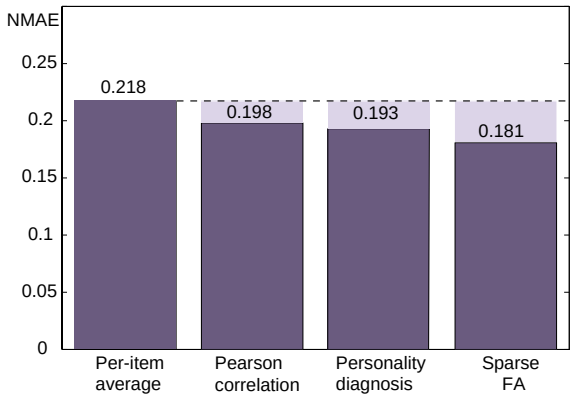

Fig 1. Results of sparse factor analysis, Pearson correlation and Personality diagnosis against a baseline predictor for the EachMovie dataset. Lower bars indicate better performance. Lightly shaded regions show improvement over baseline.

Breese's experiments were done on a sample of 5000 users. Since our method involves a training phase, we created two disjoint sets Y_T, Y_P of 5000 users. We computed the model Λ, ψ on the first training set Y_T, and tested the model on the set Y_P. We reimplemented Pearson correlation, ran it on the Y_P set, and obtained MAE ratings that matched Breese's within statistical error. We compared our predictions with Breese's "Allbut1" case, which used all but one of a user's ratings to make a prediction, and compared the prediction with the remaining value. We felt this was the most realistic prediction case.

Because we tested against several datasets, we used a normalized value of MAE, or NMAE, as suggested by Goldberg [6]. The results appear in figure 1. The baseline predictor was are respectively the per-item average (average of a row of y). Per-item average is not personalized. Each movie's rating is the same for all users. These predictors provide a useful reality check for more sophisticated schemes. A prediction scheme should be significantly more accurate than these baseline predictors, or it is not adding value. Also, many schemes, Breese's and ours included, make predictions *relative* to one of the baseline predictors. In Breese's case, their model is computed as an offset from the user's mean. In our case, predictions are computed relative to per-item means. We tested per-user mean baselines for factor analysis as well, and they were 2% less accurate.

Note that Pearson correlation, the most accurate reported scheme on Eachmovie from Breese's survey, achieves about a 9% improvement in MAE over non-personalized recommendations based on per-item average. Personality diagnosis achieves an 11% improvement over baseline. By contrast, sparse factor analysis achieves an 18% improvement over baseline, and a 9% improvement over Pearson correlation. It was a 10% improvement over simple SVD prediction. The predictions from sparse factor analysis improve if there is more training data. This will often be important because as we will see later, sparse FA is orders of magnitude faster at generating recommendations than Pearson correlation or PD on typical datasets. Therefore SFA can be often used on larger datasets than is practical with those methods. We tried training with 50,000 users, and the NMAE dropped a further 2.5% to 0.175. The standard deviations in all estimates are less than 0.25 %.

5.3 Jester Dataset

The Jester dataset comes from Ken Goldberg's joke recommendation website, Jester [6]. In Jester, users rate a core set of jokes, and then receive recommendations about others that they should like. The database has 100 jokes, and records of 17988 users. Some users end up reading and rating all the jokes, so Jester is much more dense than the other datasets we considered. Roughly 50% of all possible ratings are present. Jester has a rating scale from -10 to 10. Ratings are implemented with a slider, so Jester's scale is continuous. Perhaps because of the density, and/or because the continuous scale introduces less quantization error in ratings, Jester exhibits lower NMAE values than the other datasets we tested.

We split the data into training and test sets with approximately 9000 users in each. We computed a factor analysis model with $k = 14$ factors as before. The results are shown in figure 2. Notice that this time, the per-user average is a better baseline predictor than per-item average. This time Pearson correlation is a 10% improvement over the better of the two baseline predictors, while factor analysis is a 5% improvement over Pearson. Given that this data is not very sparse, we expected a lower improvement over Pearson. The best reported NMAE from among the methods described in [6] was 0.187, which is somewhat higher than Pearson on this dataset.

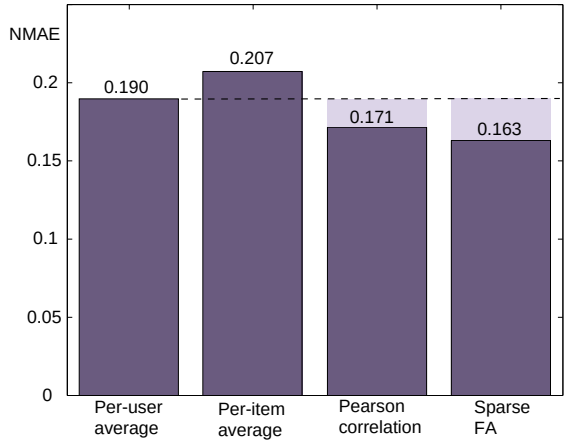

Fig 2. Results of sparse factor analysis and Pearson correlation on the Jester dataset. Lower bars indicate better performance, and the lightly shaded regions are improvement over baseline.

5.4 Clickthru Dataset

The last experiment used data from a large internet service provider. This dataset has anonymized web clickthru information from 15,000 users for 6 months. The basic dataset was a log file listing user ID number, URL accessed and the date and time. To limit the number of sites that might be considered, we truncated the URLs to 20 characters after the domain name. We also eliminated sites that had not been visited by at least 3 distinct users. The result was 210832 web sites. Our factor analysis model is most effective on datasets that have ratings of the items on some scale. So instead of a binary matrix indicating whether a user visited a site or not, we built an integer valued matrix y, where y_{ij} was the number of times that user j visited site i. The number of visits of a user to a site is an implicit rating of that site by the user. The implicit rating was clipped at 10, so the range of ratings for a site varied from 1 to 10 (0 represents no visit rather than a low rating). The dataset has 2200000 non-null entries, and is the most sparse dataset we used. It contains 0.07% of the possible ratings.

The results are shown in figure 3. Both Pearson correlation and factor analysis methods used per-user mean as their baseline predictor, which is shown as a dotted line on the figure. For this example, factor analysis gives more than double the decrease in NMAE compared to Pearson. Note that the implicit rating (by frequency of visit) is not a real user rating and gives smaller error improvements than the earlier datasets, which is why the plot has been scaled on the Y-axis. However, the implicit rating *does* behave like a rating, in the sense that there is a significant error reduction

by using the predictive model. Sparse FA is able to give nearly a 10% improvement over baseline prediction on this dataset, which is the same improvement as Pearson over baseline on the two previous datasets, which were real user ratings.

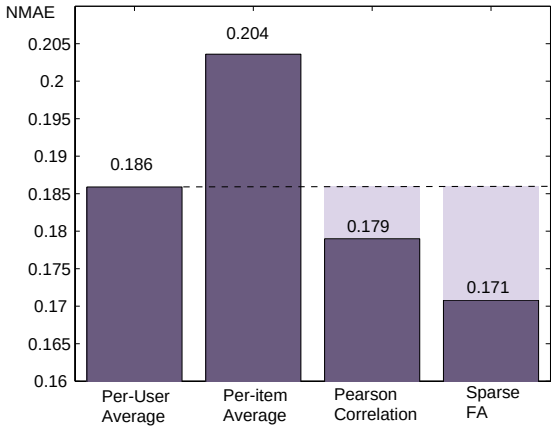

Fig 3. Results of sparse factor analysis and Pearson correlation on the web clickthru dataset. Lower bars indicate better performance, lightly shaded region is gain over baseline. Note that the Y-axis has been expanded.

5.5 Performance: Speed

The code for EM and Pearson correlation was written in Matlab. The complexity of one iteration of the sparse FA Matlab algorithm is $O(nmk^2)$, so it scales linearly with both n the number of items and m the number of users. For the Eachmovie dataset with 5000 users, the factor analysis training time was 11 minutes (20 iterations of the EM recurrence) on a PII-400 MHz machine with 256MB ram. Training with 50,000 users took a little under two hours. The time to predict all missing ratings for a user is $O(nk^2)$ and is independent of the number of users. Predictions for Eachmovie took 7 milliseconds (to generate approximately 1600 ratings for one user). For comparison, Breese reported a computing time to generate ratings for one user using Pearson correlation of about 300ms on a PII-266 MHz machine. Our Matlab implementation of Pearson correlation had similar performance to Breese's at 300ms per rec. With the same effort at optimization, our Matlab implementation of sparse factor analysis was about 50 times faster than our implementation of Pearson for generating ratings on a 5k dataset. It was 500 times faster on a 50k dataset. Of course, our method involves an additional training phase, while Pearson does not. Including model generation, the overall times to generate recommendations for everyone are comparable for SFA and Pearson with 5k users. But with 50k users, SFA is an order of magnitude faster, including model generation.

For the Jester dataset with 100 items, 9000 users and $k = 14$, time to construct the factor analysis model was 8 minutes. Generating all recommendations for one user took 7 milliseconds on the same hardware as the previous experiment. For the Clickthru dataset with 210832 items and 15000 users, we computed a factor analysis model with $k = 4$. The training time was 5 hours. Generating all recommendations for one user took 60 milliseconds.

We did not have access to the code for personality diagnosis. But as a hybrid memory/model-based approach, its speed at generating recommendations should not be better than Pearson correlation. The discussion in [10] suggests it is somewhat slower.

5.6 Performance: Space

For the Eachmovie dataset with 5000 users and k=14, The model Λ was a dense array with 23072 elements compared with the original training array which had 234934 non-nulls. In other words, the model was a 10-fold compression of the original data. For Jester, which had a high density of available ratings, the model was a 300-fold compression.

The curve below shows how cross-validation NMAE varies with model size k and number of users m. To the left of the curve, it is clear that high k leads to large errors, implying that the model is over-fitting. As the number of users (and amount of non-null data) increases, the curves cross over, and with 50000 users, the model error decreases with k. However, it was difficult to detect a significant difference between $k = 14$ and $k = 20$, and we did not try larger values of k.

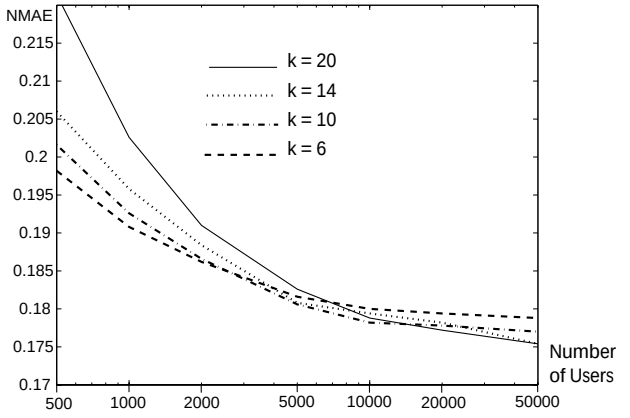

Fig 4. NMAE as a function of model dimension k and number of users m, for the Eachmovie dataset.

We can abstract the information above to the following heuristic to avoid over-fitting for typical collaborative filtering data:

Maximum Model Size The model dimension k should be chosen so that the total size of the model Λ is at most one tenth of the number of non-null elements in the original data. This criterion is equivalent to

$$10 \times k < \rho \times m$$

where m is the number of users as before, and ρ is the density (fraction of non-nulls) in the dataset.

A second test for overfitting is provided by our Matlab code mentioned earlier, which estimates ψ at both model creation and at prediction time. If these estimates differ significantly (say by more than 20%), the data has been overfit and k should be reduced.

The model size criterion implies that the factor analysis model is a substantial compression of the original dataset. Ten is the minimum compression ratio, twenty to thirty are desirable, and larger numbers will still give very accurate prediction. e.g. with 50000 users and $k = 14$, the model still gives very good accuracy from figure 4. In this case, the model is 100 times smaller than the original data.

6. DISCUSSION

Sparse factor analysis is an extremely promising approach to collaborative filtering. We ran it on three datasets of varying sparseness, including a well-studied reference dataset. In all cases, its accuracy improved significantly over other methods. The improvements increased with the sparseness of the dataset, as expected because sparse FA correctly handles sparseness. For memory-based methods such as Pearson correlation or personality diagnosis (PD), sparse FA is much faster per recommendation (50 times typical). It also and provides typical compression of the dataset of 10-100 times over memory-based methods. It is closest in speed and space requirements to singular value decomposition, but has a large accuracy advantage (about 10% on EachMovie data) over SVD. Sparse FA was implemented using a simple EM recurrence, so that it adapts easily to user datasets that are continually being updated. One or two iterations are enough to update the model after an update to the user data. It is the first CF algorithm to have an incremental implementation. Sparse FA supports computation on encrypted data, thereby protecting user privacy. Finally, the method is fully described by the recurrence equations (3), (4) and (5) and is easy to implement. It infers the statistical quantities it needs, and the only parameter to be set is the model dimension k, for which we gave some principles. The lack of other adjustable parameters or heuristics makes its performance easy to replicate. In our experience, its convergence is fast and reliable.

6.1 Limitations

The main limitation of this model is its applicability to binary recommendation problems, such as purchase records. If the only data on a user is what they have bought, then this model is not helpful. The reason is that with binary data, there is no range of ratings. Every item "rated" by a user has rating 1, meaning they bought that item. If the user did not buy an item, it does not mean they did not like it, but that we have no information on whether they did. But we cannot generate recommendations from a sparse dataset with identical values everywhere (all the predictions would be 1s for every other item). The best we can do with this dataset is to fill in missing values with 0 as a default value, indicating the user has apparently less interest in those items than in the ones she bought. This makes missing data look like low rating, and it is likely to do rather poorly. Also the data is not sparse, because we have a 1 or 0 for everything. A better solution would be to extract some tacit rating info. For instance, if you own the site providing the commerce, you could record when a user views an item as well as when they purchase it. If they view and do not purchase, record a 0 for that item and user. If they do purchase, record a 1. In that way, you still acquire a (typically very) sparse matrix, and the factor analysis model would likely do better than other models.

7. REFERENCES

[1] Breese, Heckermen, and Kadie. Empirical analysis of predictive algorithms for collaborative filtering. Technical report, Microsoft Research, October 1998.

[2] J. Canny. Collaborative filtering with privacy. In *IEEE Security and Privacy Conference (to appear)*, 2002.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[4] B. Frey. Turbo factor analysis. *Adv. Neural Information Processing*, 1999. (submitted).

[5] Z. Ghahramani and M. I. Jordan. Learning from incomplete data. Technical Report AIM-1509, MIT AI Lab, 1994.

[6] K. Goldberg, D. Gupta, M. Digiovanni, and H. Narita. Jester 2.0 : Evaluation of a new linear time collaborative filtering algorithm. In *22nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, August 1999. Poster Session and Demonstration.

[7] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. ACM SIGIR*, 1999.

[8] O. P. John. The "big five" factor taxonomy: Dimensions of personality in the natural language and in questionnaires. In L. A. Pervin, editor, *Handbook of personality: Theory and research*. Guilford, New York, 1990.

[9] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.

[10] D. Pennock and E. Horvitz. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In *IJCAI Workshop on Machine Learning for Information Filtering, International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden, August 1999.

[11] E. M. Rogers. *Diffusion of Innovations, Fourth Edition*. The Free Press, 1995.

[12] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system – a case study. In *ACM WebKDD 2000 Web Mining for E-Commerce Workshop*, 2000. Full length paper.