Collaborative Filtering for Implicit Feedback Datasets

Yifan Hu
AT&T Labs – Research
Florham Park, NJ 07932

Yehuda Koren*
Yahoo! Research
Haifa 31905, Israel

Chris Volinsky
AT&T Labs – Research
Florham Park, NJ 07932

Abstract

A common task of recommender systems is to improve customer experience through personalized recommendations based on prior implicit feedback*. These systems passively track different sorts of user behavior, such as purchase history, watching habits and browsing activity, in order to model user preferences. Unlike the much more extensively researched* explicit feedback*, we do not have any direct input from the users regarding their preferences. In particular, we lack substantial evidence on which products consumer dislike. In this work we identify unique properties of implicit feedback datasets. We propose treating the data as indication of positive and negative preference associated with vastly varying confidence levels. This leads to a factor model which is especially tailored for implicit feedback recommenders. We also suggest a scalable optimization procedure, which scales linearly with the data size. The algorithm is used successfully within a recommender system for television shows. It compares favorably with well tuned implementations of other known methods. In addition, we offer a novel way to give explanations to recommendations given by this factor model.*

1 Introduction

As e-commerce is growing in popularity, an important challenge is helping customers sort through a large variety of offered products to easily find the ones they will enjoy the most. One of the tools that address this challenge is recommender systems, which are attracting a lot of attention recently [1, 4, 12]. These systems provide users with personalized recommendations for products or services, which hopefully suit their unique taste and needs. The technology behind those systems is based on profiling users and products, and finding how to relate them.

Broadly speaking, recommender systems are based on two different strategies (or combinations thereof). The *content based approach* creates a profile for each user or product to characterize its nature. As an example, a movie profile could include attributes regarding its genre, the participating actors, its box office popularity, etc. User profiles might include demographic information or answers to a suitable questionnaire. The resulting profiles allow programs to associate users with matching products. However, content based strategies require gathering external information that might not be available or easy to collect.

An alternative strategy, our focus in this work, relies only on past user behavior without requiring the creation of explicit profiles. This approach is known as *Collaborative Filtering* (CF), a term coined by the developers of the first recommender system - Tapestry [8]. CF analyzes relationships between users and interdependencies among products, in order to identify new user-item associations. For example, some CF systems identify pairs of items that tend to be rated similarly or like-minded users with similar history of rating or purchasing to deduce unknown relationships between users and items. The only required information is the past behavior of users, which might be their previous transactions or the way they rate products. A major appeal of CF is that it is domain free, yet it can address aspects of the data that are often elusive and very difficult to profile using content based techniques. While generally being more accurate than content based techniques, CF suffers from the *cold start* problem, due to its inability to address products new to the system, for which content based approaches would be adequate.

Recommender systems rely on different types of input. Most convenient is the high quality *explicit feedback*, which includes explicit input by users regarding their interest in products. For example, Netflix collects star ratings for movies and TiVo users indicate their preferences for TV shows by hitting thumbs-up/down buttons. However, explicit feedback is not always available. Thus, recommenders can infer user preferences from the more abundant *implicit feedback*, which indirectly reflect opinion through observing user behavior [14]. Types of implicit feedback include purchase history, browsing history, search patterns, or even mouse movements. For example, a user that pur-

*Work done while author was at AT&T Labs – Research

chased many books by the same author probably likes that author.

The vast majority of the literature in the field is focused on processing explicit feedback; probably thanks to the convenience of using this kind of pure information. However, in many practical situations recommender systems need to be centered on implicit feedback. This may reflect reluctance of users to rate products, or limitations of the system that is unable to collect explicit feedback. In an implicit model, once the user gives approval to collect usage data, no additional explicit feedback (e.g. ratings) is required on the user's part.

This work conducts an exploration into algorithms specifically suitable for processing implicit feedback. It reflects some of the major lessons and developments that were achieved while we built a TV shows recommender engine. Our setup prevents us from actively gathering explicit feedback from users, so the system was solely based on implicit feedback – analyzing watching habits of anonymized users.

It is crucial to identify the unique characteristics of implicit feedback, which prevent the direct use of algorithms that were designed with explicit feedback in mind. In the following we list the prime characteristics:

1. No negative feedback. By observing the users behavior, we can infer which items they probably like and thus chose to consume. However, it is hard to reliably infer which items a user did not like. For example, a user that did not watch a certain show might have done so because she dislikes the show or just because she did not know about the show or was not available to watch it. This fundamental asymmetry does not exist in explicit feedback where users tell us both what they like and what they dislike. It has several implications. For example, explicit recommenders tend to focus on the gathered information – those user-item pairs that we know their ratings – which provide a balanced picture on the user preference. Thus, the remaining user-item relationships, which typically constitute the vast majority of the data, are treated as "missing data" and are omitted from the analysis. This is impossible with implicit feedback, as concentrating only on the gathered feedback will leave us with the positive feedback, greatly misrepresenting the full user profile. Hence, it is crucial to address also the missing data, which is where most negative feedback is expected to be found.

2. Implicit feedback is inherently noisy. While we passively track the users behavior, we can only guess their preferences and true motives. For example, we may view purchase behavior for an individual, but this does not necessarily indicate a positive view of the product. The item may have been purchased as a gift, or perhaps the user was disappointed with the product. We may view that a television is on a particular channel at a particular time, but the viewer might be asleep.

3. The numerical value of explicit feedback indicates *preference*, whereas the numerical value of implicit feedback indicates *confidence*. Systems based on explicit feedback let the user express their level of preference, e.g. a star rating between 1 ("totally dislike") and 5 ("really like"). On the other hand, numerical values of implicit feedback describe the frequency of actions, e.g., how much time the user watched a certain show, how frequently a user is buying a certain item, etc. A larger value is not indicating a higher preference. For example, the most loved show may be a movie that the user will watch only once, while there is a series that the user quite likes and thus is watching every week. However, the numerical value of the feedback is definitely useful, as it tells us about the confidence that we have in a certain observation. A one time event might be caused by various reasons that have nothing to do with user preferences. However, a recurring event is more likely to reflect the user opinion.

4. Evaluation of implicit-feedback recommender requires appropriate measures. In the traditional setting where a user is specifying a numeric score, there are clear metrics such as mean squared error to measure success in prediction. However with implicit models we have to take into account availability of the item, competition for the item with other items, and repeat feedback. For example, if we gather data on television viewing, it is unclear how to evaluate a show that has been watched more than once, or how to compare two shows that are on at the same time, and hence cannot both be watched by the user.

2 Preliminaries

We reserve special indexing letters for distinguishing users from items: for users u, v, and for items i, j. The input data associate users and items through r_{ui} values, which we henceforth call *observations*. For explicit feedback datasets, those values would be ratings that indicate the preference by user u of item i, where high values mean stronger preference. For implicit feedback datasets, those values would indicate observations for user actions. For example, r_{ui} can indicate the number of times u purchased item i or the time u spent on webpage i. In our TV recommender case, r_{ui} indicates how many times u fully watched show i. For example, $r_{ui} = 0.7$ indicates that u watched 70% of the show, while for a user that watched the show twice we will set $r_{ui} = 2$.

Explicit ratings are typically unknown for the vast majority of user-item pairs, hence applicable algorithms work

with the relatively few known ratings while ignoring the missing ones. However, with implicit feedback it would be natural to assign values to all r_{ui} variables. If no action was observed r_{ui} is set to zero, thus meaning in our examples zero watching time, or zero purchases on record.

3 Previous work

3.1 Neighborhood models

The most common approach to CF is based on neighborhood models. Its original form, which was shared by virtually all earlier CF systems, is user-oriented; see [9] for a good analysis. Such user-oriented methods estimate unknown ratings based on recorded ratings of like minded users. Later, an analogous item-oriented approach [13, 19] became popular. In those methods, a rating is estimated using known ratings made by the same user on similar items. Better scalability and improved accuracy make the item-oriented approach more favorable in many cases [2, 19, 20]. In addition, item-oriented methods are more amenable to explaining the reasoning behind predictions. This is because users are familiar with items previously preferred by them, but usually do not know those allegedly like minded users.

Central to most item-oriented approaches is a similarity measure between items, where s_{ij} denotes the similarity of i and j. Frequently, it is based on the Pearson correlation coefficient. Our goal is to predict r_{ui} – the unobserved value by user u for item i. Using the similarity measure, we identify the k items rated by u, which are most similar to i. This set of k neighbors is denoted by $\mathrm{S}^k(i;u)$. The predicted value of r_{ui} is taken as a weighted average of the ratings for neighboring items:

$$\hat{r}_{ui} = \frac{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij} r_{uj}}{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}} \quad (1)$$

Some enhancements of this scheme are well practiced for explicit feedback, such as correcting for biases caused by varying mean ratings of different users and items. Those modifications are less relevant to implicit feedback datasets, where instead of having ratings which are all on the same scale, we use frequencies in which items are consumed by the same user. Frequencies for disparate users might have very different scale depending on the application, and it is less clear how to calculate similarities. A good discussion on how to use an item-oriented approach with implicit feedback is given by Deshpande and Karypis [6].

All item-oriented models share a disadvantage in regards to implicit feedback - they do not provide the flexibility to make a distinction between user preferences and the confidence we might have in those preferences.

3.2 Latent factor models

Latent factor models comprise an alternative approach to Collaborative Filtering with the more holistic goal to uncover latent features that explain observed ratings; examples include pLSA [11], neural networks [16], and Latent Dirichlet Allocation [5]. We will focus on models that are induced by Singular Value Decomposition (SVD) of the user-item observations matrix. Recently, SVD models have gained popularity, thanks to their attractive accuracy and scalability; see, e.g., [3, 7, 15, 17, 20]. A typical model associates each user u with a user-factors vector $x_u \in \mathbb{R}^f$, and each item i with an item-factors vector $y_i \in \mathbb{R}^f$. The prediction is done by taking an inner product, i.e., $\hat{r}_{ui} = x_u^T y_i$. The more involved part is parameter estimation. Many of the recent works, applied to explicit feedback datasets, suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model, such as:

$$\min_{x_\star, y_\star} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2) \quad (2)$$

Here, λ is used for regularizing the model. Parameters are often learnt by stochastic gradient descent; see, e.g., [7, 15, 20]. The results, as reported on the largest available dataset – the Netflix dataset [4] – tend to be consistently superior to those achieved by neighborhood models. In this work we borrow this approach to implicit feedback datasets, which requires modifications both in the model formulation and in the optimization technique.

4 Our model

In this section we describe our model for implicit feedback. First, we need to formalize the notion of confidence which the r_{ui} variables measure. To this end, let us introduce a set of binary variables p_{ui}, which indicates the preference of user u to item i. The p_{ui} values are derived by binarizing the r_{ui} values:

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

In other words, if a user u consumed item i ($r_{ui} > 0$), then we have an indication that u likes i ($p_{ui} = 1$). On the other hand, if u never consumed i, we believe no preference ($p_{ui} = 0$). However, our beliefs are associated with greatly varying confidence levels. First, by the nature of the data zero values of p_{ui} are associated with low confidence, as not taking any positive action on an item can stem from many other reasons beyond not liking it. For example, the user might be unaware of the existence of the item, or unable to consume it due to its price or limited availability. In

addition, consuming an item can also be the result of factors different from preferring it. For example, a user may watch a TV show just because she is staying on the channel of the previously watched show. Or a consumer may buy an item as gift for someone else, despite not liking the item for himself. Thus, we will have different confidence levels also among items that are indicated to be preferred by the user. In general, as r_{ui} grows, we have a stronger indication that the user indeed likes the item. Consequently, we introduce a set of variables, c_{ui}, which measure our confidence in observing p_{ui}. A plausible choice for c_{ui} would be:

$$c_{ui} = 1 + \alpha r_{ui}$$

This way, we have some minimal confidence in p_{ui} for every user-item pair, but as we observe more evidence for positive preference, our confidence in $p_{ui} = 1$ increases accordingly. The rate of increase is controlled by the constant α. In our experiments, setting $\alpha = 40$ was found to produce good results.

Our goal is to find a vector $x_u \in \mathbb{R}^f$ for each user u, and a vector $y_i \in \mathbb{R}^f$ for each item i that will factor user preferences. In other words, preferences are assumed to be the inner products: $p_{ui} = x_u^T y_i$. These vectors will be known as the user-factors and the item-factors, respectively. Essentially, the vectors strive to map users and items into a common latent factor space where they can be directly compared. This is similar to matrix factorization techniques which are popular for explicit feedback data, with two important distinctions: (1) We need to account for the varying confidence levels, (2) Optimization should account for all possible u, i pairs, rather than only those corresponding to observed data. Accordingly, factors are computed by minimizing the following cost function:

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \tag{3}$$

The $\lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$ term is necessary for regularizing the model such that it will not overfit the training data. Exact value of the parameter λ is data-dependent and determined by cross validation.

Notice that the cost function contains $m \cdot n$ terms, where m is the number of users and n is the number of items. For typical datasets $m \cdot n$ can easily reach a few billion. This huge number of terms prevents most direct optimization techniques such as stochastic gradient descent, which was widely used for explicit feedback datasets. Thus, we suggest an alternative efficient optimization process, as follows.

Observe that when either the user-factors or the item-factors are fixed, the cost function becomes quadratic so its global minimum can be readily computed. This leads to an alternating-least-squares optimization process, where we alternate between re-computing user-factors and item-factors, and each step is guaranteed to lower the value of the cost function. Alternating least squares was used for explicit feedback datasets [2], where unknown values were treated as missing, leading to a sparse objective function. The implicit feedback setup requires a different strategy to overcome the dense cost function and to integrate the confidence levels. We address these by exploiting the structure of the variables so that this process can be implemented to be highly scalable.

The first step is recomputing all user factors. Let us assume that all item-factors are gathered within an $n \times f$ matrix Y. Before looping through all users, we compute the $f \times f$ matrix $Y^T Y$ in time $O(f^2 n)$. For each user u, let us define the diagonal $n \times n$ matrix C^u where $C^u_{ii} = c_{ui}$, and also the vector $p(u) \in \mathbb{R}^n$ that contains all the preferences by u (the p_{ui} values). By differentiation we find an analytic expression for x_u that minimizes the cost function (3):

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u) \tag{4}$$

A computational bottleneck here is computing $Y^T C^u Y$, whose naive calculation will require time $O(f^2 n)$ (for each of the m users). A significant speedup is achieved by using the fact that $Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y$. Now, $Y^T Y$ is independent of u and was already precomputed. As for $Y^T (C^u - I) Y$, notice that $C^u - I$ has only n_u non-zero elements, where n_u is the number of items for which $r_{ui} > 0$ and typically $n_u \ll n$. Similarly, $C^u p(u)$ contains just n_u non-zero elements. Consequently, recomputation of x_u is performed in time $O(f^2 n_u + f^3)$. Here, we assumed $O(f^3)$ time for the matrix inversion $(Y^T C^u Y + \lambda I)^{-1}$, even though more efficient algorithms exist, but probably are less relevant for the typically small values of f. This step is performed over each of the m users, so the total running time is $O(f^2 \mathcal{N} + f^3 m)$, where $\mathcal{N}$ is the overall number of non-zero observations, that is $\mathcal{N} = \sum_u n_u$. Importantly, running time is linear in the size of the input. Typical values of f lie between 20 and 200; see experimental study in Sec. 6.

A recomputation of the user-factors is followed by a recomputation of all item-factors in a parallel fashion. We arrange all user-factors within an $m \times f$ matrix X. First we compute the $f \times f$ matrix $X^T X$ in time $O(f^2 m)$. For each item i, we define the diagonal $m \times m$ matrix C^i where $C^i_{uu} = c_{ui}$, and also the vector $p(i) \in \mathbb{R}^m$ that contains all the preferences for i. Then we solve:

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i) \tag{5}$$

Using the same technique as with the user-factors, running time of this step would be $O(f^2 \mathcal{N} + f^3 n)$. We employ a few sweeps of paired recomputation of user- and item-factors, till they stabilize. A typical number of sweeps is 10.

The whole process scales linearly with the size of the data. After computing the user- and item-factors, we recommend to user u the K available items with the largest value of $\hat{p}_{ui} = x_u^T y_i$, where $\hat{p}_{ui}$ symbolizes the predicted preference of user u for item i.

Now that the basic description of our technique is completed we would like to further discuss it, as some of our decisions can be modified. For example, one can derive p_{ui} differently from r_{ui}, by setting a minimum threshold on r_{ui} for the corresponding p_{ui} to be non-zero. Similarly, there are many ways to transform r_{ui} into a confidence level c_{ui}. One alternative method that also worked well to us is setting

$$c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon). \quad (6)$$

Regardless of the exact variant of the scheme, it is important to realize its main properties, which address the unique characteristics of implicit feedback:

1. Transferring the raw observations (r_{ui}) into two separate magnitudes with distinct interpretations: preferences (p_{ui}) and confidence levels (c_{ui}). This better reflect the nature of the data and is essential to improving prediction accuracy, as shown in the experimental study (Sec. 6).

2. An algorithm that addresses all possible ($n \cdot m$) user-item combinations in a linear running time, by exploiting the algebraic structure of the variables.

5 Explaining recommendations

It is well accepted [10] that a good recommendation should be accompanied with an explanation, which is a short description to why a specific product was recommended to the user. This helps in improving the users' trust in the system and their ability to put recommendations in the right perspective. In addition, it is an invaluable means for debugging the system and tracking down the source of unexpected behavior. Providing explanations with neighborhood-based (or, "memory-based") techniques is straightforward, as recommendations are directly inferred from past users' behavior. However, for latent factor models explanations become trickier, as all past user actions are abstracted via the user factors thereby blocking a direct relation between past user actions and the output recommendations. Interestingly, our alternating least squares model enables a novel way to compute explanations. The key is replacing the user-factors by using Eq. (4) : $x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$. Thus, the predicted preference of user u for item i, $\hat{p}_{ui} = y_i^T x_u$, becomes: $y_i^T (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$. This expression can be simplified by introducing some new notation. Let us denote the $f \times f$ matrix $(Y^T C^u Y + \lambda I)^{-1}$ as W^u, which should be considered as a weighting matrix associated with user u. Accordingly, the weighted similarity between items i and j from u's viewpoint is denoted by $s_{ij}^u = y_i^T W^u y_j$. Using this new notation the predicted preference of u for item i is rewritten as:

$$\hat{p}_{ui} = \sum_{j:r_{uj}>0} s_{ij}^u c_{uj} \quad (7)$$

This reduces our latent factor model into a linear model that predicts preferences as a linear function of past actions ($r_{uj} > 0$), weighted by item-item similarity. Each past action receives a separate term in forming the predicted $\hat{p}_{ui}$, and thus we can isolate its unique contribution. The actions associated with the highest contribution are identified as the major explanation behind the recommendation. In addition, we can further break the contribution of each individual past action into two separate sources: the significance of the relation to user u – c_{uj}, and the similarity to target item i – s_{ij}^u.

This shares much resemblance with item-oriented neighborhood models, which enables the desired ability to explain computed predictions. If we further adopt this viewpoint, we can consider our model as a powerful preprocessor for a neighborhood based method, where item similarities are learnt through a principled optimization process. In addition, similarities between items become dependent on the specific user in question, reflecting the fact that different users do not completely agree on which items are similar.

Giving explanation through (7) involves solving a linear system $(Y^T C^u Y + \lambda I)^{-1} y_j$, followed by a matrix vector product, and can be done in time $O(f^2 n_u + f^3)$, assuming that $Y^T Y$ is precomputed.

6 Experimental study

Data description Our analysis is based on data from a digital television service. We were able to collect data on about 300,000 set top boxes. All data was collected in accordance with appropriate end user agreements and privacy policies. The analysis was done with data that was aggregated and/or fully anonymized. No personally identifiable information was collected in connection with this research.

We collected all channel tune events for these users, indicating the channel the set-top box was tuned into, and a time stamp. There are approximately 17,000 unique programs which aired during a four week period. The training data contains r_{ui} values, for each user u and program i, which represent how many times user u watched program i (related is the number of *minutes* that a given show was watched - for all of our analysis we focus on show length based units). Notice that r_{ui} is a real value, as users may watch parts of shows. After aggregating multiple watches

of the same program, the number of non-zero r_{ui} values is about 32 million.

In addition, we use a similarly constructed test set, which is based on all channel tune events during the single week following a 4-week training period. Our system is trained using the recent 4 weeks of data in order to generate predictions of what users will watch in the ensuing week. The training period of 4 weeks is chosen based on an experimental study which showed that a shorter period tends to deteriorate the prediction results, while a longer period does not add much value (since television schedules change seasonally, long training periods do not necessarily have an advantage, even though we found that our core model is robust enough to avoid being contaminated by the seasonality). The observations in the test set are denoted by r^t_{ui} (distinguished with a superscript ^{t}).

One characteristic of television watching is the tendency to repetitively watch the same programs every week. It is much more valuable to a user to be recommended programs that she has not watched recently, or that she is not aware of. Thus, in our default setting, for each user we remove the "easy" predictions from the test set corresponding to the shows that had been watched by that user during the training period. To make the test set even more accurate, we toggle to zero all entries with $r^t_{ui} < 0.5$, as watching less than half of a program is not a strong indication that a user likes the program. This leaves us with about 2 million non-zero r^t_{ui} values in the test set.

The tendency to watch the same programs repeatedly also makes r_{ui} vary significantly over a large range. While there are a lot of viewing events close to 0 (channel flipping), 1, 2 or 3 (watching a film or a couple of episodes of a series), there are also some viewing events that accumulate to hundreds (have the DVR recording the same program for many hours per day over a period of 4 weeks). Therefore we employ the log scaling scheme (6) with $\epsilon = 10^{-8}$.

One other important adjustment is needed. We observe many cases where a single channel is watched for many hours. It is likely that the initial show that was tuned into is of interest to the viewer, while the subsequent shows are of decreasing interest. The television might simply have been left on while the viewer does something else (or sleeps!). We call this a momentum effect, and programs watched due to momentum are less expected to reflect real preference. To overcome this effect, we down-weight the second and subsequent shows after a channel tuning event. More specifically, for the t-th show after a channel tune, we assign it a weighting $\frac{e^{-(at-b)}}{1+e^{-(at-b)}}$. Experimentally we found $a = 2$ and $b = 6$ to work well and is intuitive: the third show after the channel tune gets its r_{ui} value halved, by the fifth show without a channel change, r_{ui} is reduced by 99%.

Evaluation methodology We evaluate a scenario where we generate for each user an ordered list of the shows, sorted from the one predicted to be most preferred till the least preferred one. Then, we present a prefix of the list to the user as the recommended shows. It is important to realize that we do not have a reliable feedback regarding which programs are unloved, as not watching a program can stem from multiple different reasons. In addition, we are currently unable to track user reactions to our recommendations. Thus, precision based metrics are not very appropriate, as they require knowing which programs are undesired to a user. However, watching a program is an indication of liking it, making recall-oriented measures applicable.

We denote by $rank_{ui}$ the percentile-ranking of program i within the ordered list of all programs prepared for user u. This way, $rank_{ui} = 0\%$ would mean that program i is predicted to be the most desirable for user u, thus preceding all other programs in the list. On the other hand, $rank_{ui} = 100\%$ indicates that program i is predicted to be the least preferred for user u, thus placed at the end of the list. (We opted for using percentile-ranks rather than absolute ranks in order to make our discussion general and independent of the number of programs.) Our basic quality measure is the expected percentile ranking of a watching unit in the test period, which is:

$$\overline{rank} = \frac{\sum_{u,i} r^t_{ui} rank_{ui}}{\sum_{u,i} r^t_{ui}} \tag{8}$$

Lower values of $\overline{rank}$ are more desirable, as they indicate ranking actually watched shows closer to the top of the recommendation lists. Notice that for random predictions, the expected value of $rank_{ui}$ is 50% (placing i in the middle of the sorted list). Thus, $\overline{rank} \geqslant 50\%$ indicates an algorithm no better than random.

Evaluation results We implemented our model with different number of factors (f), ranging from 10 to 200. In addition, we implemented two other competing models. The first model is sorting all shows based on their popularity, so that the top recommended shows are the most popular ones. This naive measure is surprisingly powerful, as crowds tend to heavily concentrate on few of the many thousands available shows. We take this as a baseline value.

The second model is neighborhood based (item-item), along the lines described in Sec. 3.1. We explored many variants of this scheme, and found the following two decisions to yield best results: (1) Take all items as "neighbors", not only a small subset of most similar items. (2) Use cosine similarity for measuring item-item similarity. Formally, for an item i let us arrange within $r_i \in \mathbb{R}^m$ the r_{ui} values associated with all m users. Then, $s_{ij} = \frac{r_i^T r_j}{\|r_i\|\|r_j\|}$. The predicted preference of user u for show i is: $\sum_j s_{ij} r_{uj}$. As a

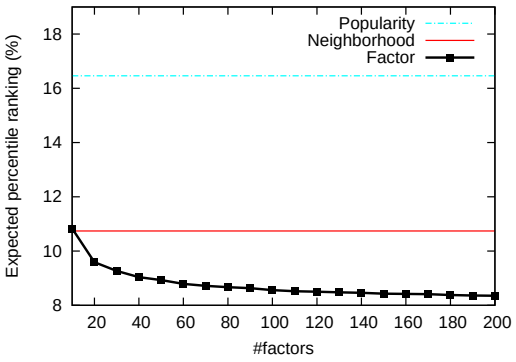

Figure 1. Comparing factor model with popularity ranking and neighborhood model.

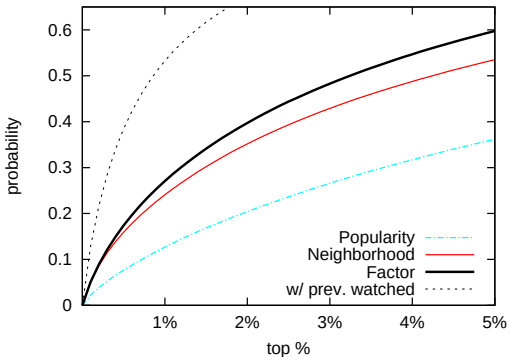

Figure 2. Cumulative distribution function of the probability that a show watched in the test set falls within top x% of recommended shows.

side remark, we would like to mention that we recommend very different settings for neighborhood based techniques when applied to explicit feedback data.

Figure 1 shows the measured values of $\overline{rank}$ with different number of factors, and also the results by the popularity ranking (cyan, horizontal line), and the neighborhood model (red, horizontal line). We can see that based only on popularity, we can achieve $\overline{rank} = 16.46\%$, which is much lower than the $\overline{rank} = 50\%$ that would be achieved by a random predictor. However, a popularity based predictor is clearly non-personalized and treats all users equally. The neighborhood based method offers a significant improvement ($\overline{rank} = 10.74\%$) achieved by personalizing recommendations. Even better results are obtained by our factor model, which offers a more principled approach to the problem. Results keep improving as number of factors increases, till reaching $\overline{rank} = 8.35\%$ for 200 factors. Thus, we recommend working with the highest number of factors feasible within computational limitations.

We further dig into the quality of recommendations, by studying the cumulative distribution function of $rank_{ui}$. Here, we concentrate on the model with 100 factors, and compare results to the popularity-based and the neighborhood-based techniques, as shown in Fig. 2. We asked the following: what is the distribution of percentiles for the shows that were actually watched in the test set? If our model does well, all of the watched shows will have low percentiles. From the figure, we see that a watched show is in the top 1% of the recommendations from our model about 27% of the time. These results compare favorably to the neighborhood based approach, and are much better than the baseline popularity-based model.

Here we would like to comment that results get much better had we left all previously watched programs in the test set (without removing all user-program events that already occurred in the training period). Predicting rewatching a program is much easier than predicting a first time view of a program. This is shown by the black dotted line in the figure, which evaluates our algorithm when previously watched shows are not taken out of the test set. Although suggesting a previously watched show might not be very exciting, it does come useful. For example, our system informs users on which programs are running today that might interest them. Here, users are not looking to be surprised, but for being reminded not to miss a favorite show. The high predictive accuracy of retrieving previously watched shows comes handy for this task.

We would also like to evaluate our decision to transform the raw observations (the r_{ui} values), into distinct preference-confidence pairs (p_{ui}, c_{ui}). Other possible models were studied as well. First, we consider a model which works directly on the given observations. Thus, our model (3) is replaced with a factor model that strives to minimize:

$$\min_{x_\star, y_\star} \sum_{u,i} (r_{ui} - x_u^T y_i)^2 + \lambda_1 \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \tag{9}$$

Notice that this is a regularized version of the dense SVD algorithm, which is an established approach to collaborative filtering [18]. Results without regularization ($\lambda_1 = 0$) were very poor and could not improve upon the popularity based model. Better results are achieved when the model is regularized – here, we used $\lambda_1 = 500$ for regularizing the model, which proved to deliver best recommendations. While results consistently outperform those of the popularity model, they were substantially poorer even than the neighborhood model. For example, for 50 factors we got $\overline{rank} = 13.63\%$, while 100 factors yield $\overline{rank} = 13.40\%$. This relatively low quality is not surprising as earlier we argued that taking r_{ui} as raw preferences is not sensible.

Therefore, we also tried another model, which factorizes the derived binary preference values, resulting in:

$$\min_{x_\star, y_\star} \sum_{u,i} (p_{ui} - x_u^T y_i)^2 + \lambda_2 \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \tag{10}$$

The model was regularized with $\lambda_2 = 150$. Results are indeed better than those of model (9), leading to $\overline{rank} = 10.72\%$ with 50 factors and $\overline{rank} = 10.49\%$ with 100 factors. This is slightly better than the results achieved with the neighborhood model. However, this is still materially inferior to our full model, which results in $\overline{rank} = 8.93\% - 8.56\%$ for 50 – 100 factors. This shows the importance of augmenting (10) with confidence levels as in (3).

We now analyze the performance of the full model (with 100 factors) on different types of shows and users. Different shows receive significantly varying watching time in the training data. Some shows are popular and watched a lot, while others are barely watched. We split the positive observations in the test set into 15 equal bins, based on increasing show popularity, We measured the performance of our model in each bin, ranging from bin 1 (least popular shows) to bin 15 (most popular shows). As Fig. 3 (blue line) shows, there is a big gap in the accuracy of our model, as it becomes much easier to predict popular programs, while it is increasingly difficult to predict watching a non popular show. To some extent, the model prefers to stay with the safe recommendations of familiar shows, on which it gathered enough data and can analyze well. Interestingly, this effect is not carried over to partitioning users according to their watching time. Now, we split users into bins based on their overall watching time; see Fig. 3 (red line). Except for the first bin, which represents users with almost no watching history, the model performance is quite similar for all other user groups. This was somewhat unexpected, as our experience with explicit feedback datasets was that as we gather more information on users, prediction quality significantly increases. The possible explanation to why the model could not do much better for heavy watchers is that those largely represent heterogeneous accounts, where many different people are watching the same TV.

Finally, we demonstrate the utility of our recommendation explanations. Explanations for recommendations are common for neighbor methods since the system can always return the nearest neighbors of the recommended item. However, there is no previous work discussing how to do explanations for matrix decomposition methods, which in our experience outperform the neighbor based methods. Table 1 shows three recommended shows for one user in our study. Following the methods in Section 5, we show the top five watched shows which explain the recommended show (shown in bold). These explanations make sense: the reality show *So You Think You Can Dance* is explained by other reality shows, while *Spider-Man* is explained by other comic-related shows and *Life in the E.R.* is explained by medical documentaries. These common-sense explanations help the user understand why certain shows are recommended, and are similar to explanations returned by neighbor methods. We also report the total percent of the recommendation accounted for by the top 5. In this case, the top five shows only explain between 35 and 40% of the recommendation, indicating that many other watched shows give input to the recommendations.

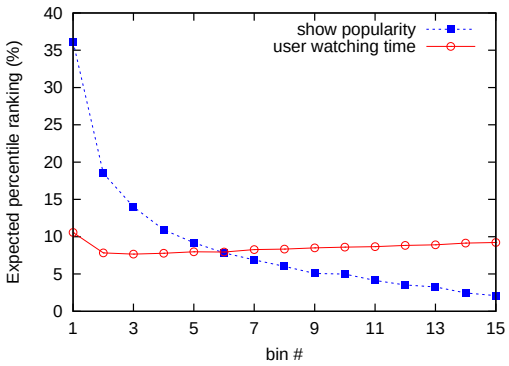

Figure 3. Analyzing the performance of the factor model by segregating users/shows based on different criteria.

7 Discussion

In this work we studied collaborative filtering on datasets with implicit feedback, which is a very common situation. One of our main findings is that implicit user observations should be transformed into two paired magnitudes: preferences and confidence levels. In other words, for each user-item pair, we derive from the input data an estimate to whether the user would like or dislike the item ("preference") and couple this estimate with a confidence level. This preference-confidence partition has no parallel in the widely studied explicit-feedback datasets, yet serves a key role in analyzing implicit feedback.

We provide a latent factor algorithm that directly addresses the preference-confidence paradigm. Unlike explicit datasets, here the model should take all user-item preferences as an input, including those which are not related to any input observation (thus hinting to a zero preference). This is crucial, as the given observations are inherently biased towards a positive preference, and thus do not reflect well the user profile. However, taking all user-item values as an input to the model raises serious scalability issues – the number of all those pairs tends to significantly exceed the input size since a typical user would provide feedback

So You Think You Can Dance	Spider-Man	Life In The E.R.
Hell's Kitchen	Batman: The Series	Adoption Stories
Access Hollywood	Superman: The Series	Deliver Me
Judge Judy	Pinky and The Brain	Baby Diaries
Moment of Truth	Power Rangers	I Lost It!
Don't Forget the Lyrics	The Legend of Tarzan	Bringing Home Baby
Total Rec = 36%	Total Rec = 40%	Total Rec = 35%

Table 1. *Three recommendations with explanations for a single user in our study. Each recommended show is recommended due to a unique set of already-watched shows by this user.*

only on a small fraction of the available items. We address this by exploiting the algebraic structure of the model, leading to an algorithm that scales linearly with the input size while addressing the full scope of user-item pairs without resorting to any sub-sampling.

An interesting feature of the algorithm is that it allows explaining the recommendations to the end user, which is a rarity among latent factor models. This is achieved by showing a surprising and hopefully insightful link into the well known item-oriented neighborhood approach.

The algorithm was implemented and tested as a part of a large scale TV recommender system. Our design methodology strives to find a right balance between the unique properties of implicit feedback datasets and computational scalability. We are currently exploring modifications with a potential to improve accuracy at the expense of increasing computational complexity. As an example, in our model we decided to treat all user-item pairs associated with a zero preference with the same uniform confidence level. Since the vast majority of pairs is associated with a zero preference, this decision saved a lot of computational effort. However, a more careful analysis would split those zero values into different confidence levels, perhaps based on availability of the item. In our television recommender example, the fact that a user did not watch a program might mean that the user was not aware of the show (it is on an 'unusual' channel or time of day), or that there is another favorite show on concurrently, or that the user is simply not interested. Each of these correspond to different scenarios, and each might warrant a distinctive confidence level in the "no preference" assumption. This leads us to another possible extension of the model – adding a dynamic time variable addressing the tendency of a user to watch TV on certain times. Likewise, we would like to model that certain program genres are more popular in different times of the day. This is part of an ongoing research, where the main challenge seems to be how to introduce an added flexibility into the model while maintaining its good computational scalability.

Finally, we note that the standard training and test setup is designed to evaluate how well a model can predict future user behavior. However, this is not the purpose of a recommender system, which strives to point users to items that they might not have otherwise purchased or consumed. It is difficult to see how to evaluate that objective without using in depth user study and surveying. In our example, we believe that by evaluating our methods by removing the "easy" cases of re-watched shows, we somehow get closer to the ideal of trying to capture user discovery of new shows.

References

[1] G. Adomavicius and A. Tuzhilin, "Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions", *IEEE Transactions on Knowledge and Data Engineering* **17** (2005), 634–749.

[2] R. Bell and Y. Koren, "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights", *IEEE International Conference on Data Mining (ICDM'07)*, pp. 43–52, 2007.

[3] R. Bell, Y. Koren and C. Volinsky, "Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems", *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

[4] J. Bennet and S. Lanning, "The Netflix Prize", *KDD Cup and Workshop*, 2007. `www.netflixprize.com`.

[5] D. Blei, A. Ng, and M. Jordan, "Latent Dirichlet Allocation", *Journal of Machine Learning Research* **3** (2003), 993–1022.

[6] M. Deshpande, G. Karypis, "Item-based top-N recommendation algorithms", *ACM Trans. Inf. Syst.* **22** (2004) 143-177.

[7] S. Funk, "Netflix Update: Try This At Home", http://sifter.org/˜simon/journal/20061211.html, 2006.

[8] D. Goldberg, D. Nichols, B. M. Oki and D. Terry, "Using Collaborative Filtering to Weave an Information Tapestry", *Communications of the ACM* **35** (1992), 61–70.

[9] J. L. Herlocker, J. A. Konstan, A. Borchers and John Riedl, "An Algorithmic Framework for Performing Collaborative Filtering", *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.

[10] J. L. Herlocker, J. A. Konstan, and J. Riedl. "Explaining collaborative filtering recommendations", In Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, ACM Press, pp. 241-250, 2000.

[11] T. Hofmann, "Latent Semantic Models for Collaborative Filtering", *ACM Transactions on Information Systems* **22** (2004), 89–115.

[12] Z. Huang, D. Zeng and H. Chen, "A Comparison of Collaborative-Filtering Recommendation Algorithms for E-commerce", *IEEE Intelligent Systems* **22** (2007), 68–78.

[13] G. Linden, B. Smith and J. York, "Amazon.com Recommendations: Item-to-item Collaborative Filtering", *IEEE Internet Computing* **7** (2003), 76–80.

[14] D.W. Oard and J. Kim, "Implicit Feedback for Recommender Systems", *Proc. 5th DELOS Workshop on Filtering and Collaborative Filtering*, pp. 31–36, 1998.

[15] A. Paterek, "Improving Regularized Singular Value Decomposition for Collaborative Filtering", *Proc. KDD Cup and Workshop*, 2007.

[16] R. Salakhutdinov, A. Mnih and G. Hinton, "Restricted Boltzmann Machines for Collaborative Filtering", *Proc. 24th Annual International Conference on Machine Learning*, pp. 791–798, 2007.

[17] R. Salakhutdinov and A. Mnih, "Probabilistic Matrix Factorization", *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pp. 1257–1264, 2008.

[18] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Application of Dimensionality Reduction in Recommender System – A Case Study", *WEBKDD'2000*.

[19] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, "Item-based Collaborative Filtering Recommendation Algorithms", *Proc. 10th International Conference on the World Wide Web*, pp. 285-295, 2001.

[20] G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk, "Major Components of the Gravity Recommendation System", *SIGKDD Explorations* **9** (2007), 80–84.