Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model

Yehuda Koren
AT&T Labs – Research
180 Park Ave, Florham Park, NJ 07932
yehuda@research.att.com

ABSTRACT

Recommender systems provide users with personalized suggestions for products or services. These systems often rely on Collaborating Filtering (CF), where past transactions are analyzed in order to establish connections between users and products. The two more successful approaches to CF are latent factor models, which directly profile both users and products, and neighborhood models, which analyze similarities between products or users. In this work we introduce some innovations to both approaches. The factor and neighborhood models can now be smoothly merged, thereby building a more accurate combined model. Further accuracy improvements are achieved by extending the models to exploit both explicit and implicit feedback by the users. The methods are tested on the Netflix data. Results are better than those previously published on that dataset. In addition, we suggest a new evaluation metric, which highlights the differences among methods, based on their performance at a top-K recommendation task.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms

Keywords

collaborative filtering, recommender systems

1. INTRODUCTION

Modern consumers are inundated with choices. Electronic retailers and content providers offer a huge selection of products, with unprecedented opportunities to meet a variety of special needs and tastes. Matching consumers with most appropriate products is not trivial, yet it is a key in enhancing user satisfaction and loyalty. This emphasizes the prominence of *recommender systems*, which provide personalized recommendations for products that suit a user's taste [1]. Internet leaders like Amazon, Google, Netflix, TiVo and Yahoo are increasingly adopting such recommenders.

Recommender systems are often based on *Collaborative Filtering* (CF) [10], which relies only on past user behavior—e.g., their previous transactions or product ratings—and does not require the creation of explicit profiles. Notably, CF techniques require no domain knowledge and avoid the need for extensive data collection. In addition, relying directly on user behavior allows uncovering complex and unexpected patterns that would be difficult or impossible to profile using known data attributes. As a consequence, CF attracted much of attention in the past decade, resulting in significant progress and being adopted by some successful commercial systems, including Amazon [15], TiVo and Netflix.

In order to establish recommendations, CF systems need to compare fundamentally different objects: items against users. There are two primary approaches to facilitate such a comparison, which constitute the two main disciplines of CF: *the neighborhood approach* and *latent factor models*.

Neighborhood methods are centered on computing the relationships between items or, alternatively, between users. An item-oriented approach evaluates the preference of a user to an item based on ratings of similar items by the same user. In a sense, these methods transform users to the item space by viewing them as baskets of rated items. This way, we no longer need to compare users to items, but rather directly relate items to items.

Latent factor models, such as Singular Value Decomposition (SVD), comprise an alternative approach by transforming both items and users to the same latent factor space, thus making them directly comparable. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback. For example, when the products are movies, factors might measure obvious dimensions such as comedy vs. drama, amount of action, or orientation to children; less well defined dimensions such as depth of character development or "quirkiness"; or completely uninterpretable dimensions.

The CF field has enjoyed a surge of interest since October 2006, when the Netflix Prize competition [5] commenced. Netflix released a dataset containing 100 million movie ratings and challenged the research community to develop algorithms that could beat the accuracy of its recommendation system, Cinematch. A lesson that we learnt through this competition is that the neighborhood and latent factor approaches address quite different levels of structure in the data, so none of them is optimal on its own [3].

教训一：neighborhood and latent factor approaches 的缺点

Neighborhood models are most effective at detecting very localized relationships. They rely on a few significant neighborhood-relations, often ignoring the vast majority of ratings by a user. Consequently, these methods are unable to capture the totality of weak signals encompassed in all of a user's ratings. Latent factor models are generally effective at estimating overall structure that relates simultaneously to most or all items. However, these models are poor at detecting strong associations among a small set of closely related items, precisely where neighborhood models do best.

In this work we suggest a combined model that improves prediction accuracy by capitalizing on the advantages of both neighborhood and latent factor approaches. To our best knowledge, this is the first time that a single model has integrated the two approaches. In fact, some past works (e.g., [2, 4]) recognized the utility of combining those approaches. However, they suggested post-processing the factorization results, rather than a unified model where neighborhood and factor information are considered symmetrically.

教训二： Another lesson learnt from the Netflix Prize competition is the importance of integrating different forms of user input into the models [3]. 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 [16]. Types of implicit feedback include purchase history, browsing history, search patterns, or even mouse movements. For example, a user that purchased many books by the same author probably likes that author. Our main focus is on cases where explicit feedback is available. Nonetheless, we recognize the importance of implicit feedback, which can illuminate users that did not provide enough explicit feedback. Hence, our models integrate explicit and implicit feedback.

The structure of the rest of the paper is as follows. We start with preliminaries and related work in Sec. 2. Then, we describe a new, more accurate neighborhood model in Sec. 3. The new model is based on an optimization framework that allows smooth integration with latent factor models, and also inclusion of implicit user feedback. Section 4 revisits SVD-based latent factor models while introducing useful extensions. These extensions include a factor model that allows explaining the reasoning behind recommendations. Such explainability is important for practical systems [11, 23] and known to be problematic with latent factor models. The methods introduced in Sec. 3-4 are linked together in Sec. 5, through a model that integrates neighborhood and factor models within a single framework. Relevant experimental results are brought within each section. In addition, we suggest a new methodology to evaluate effectiveness of the models, as described in Sec. 6, with encouraging results.

2. PRELIMINARIES

We reserve special indexing letters for distinguishing users from items: for users u, v, and for items i, j. A rating r_{ui} indicates the preference by user u of item i, where high values mean stronger preference. For example, values can be integers ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. We distinguish predicted ratings from known ones, by using the notation $\hat{r}_{ui}$ for the predicted value of r_{ui}. The (u, i) pairs for which r_{ui} is known are stored in the set $\mathcal{K} = \{(u, i) \mid r_{ui} \text{ is known}\}$. Usually the vast majority of ratings are unknown. For example, in the Netflix data 99% of the possible ratings are missing. In order to combat overfitting the sparse rating data, models are regularized so estimates are shrunk towards baseline defaults. Regularization is controlled by constants which are denoted as: $\lambda_1, \lambda_2, \ldots$ Exact values of these constants are determined by cross validation. As they grow, regularization becomes heavier.

2.1 Baseline estimates

Typical CF data exhibit large user and item effects – i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. It is customary to adjust the data by accounting for these effects, which we encapsulate within the *baseline estimates*. Denote by μ the overall average rating. A baseline estimate for an unknown rating r_{ui} is denoted by b_{ui} and accounts for the user and item effects:

$$b_{ui} = \mu + b_u + b_i \tag{1}$$

The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average. For example, suppose that we want a baseline estimate for the rating of the movie Titanic by user Joe. Now, say that the average rating over all movies, μ, is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline estimate for Titanic's rating by Joe would be 3.9 stars by calculating $3.7 - 0.3 + 0.5$. In order to estimate b_u and b_i one can solve the least squares problem:

$$\min_{b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$$

这里bu，bi是要求出来的参数

Here, the first term $\sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu + b_u + b_i)^2$ strives to find b_u's and b_i's that fit the given ratings. The regularizing term – $\lambda_1(\sum_u b_u^2 + \sum_i b_i^2)$ – avoids overfitting by penalizing the magnitudes of the parameters.

2.2 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 [12] 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 [15, 21] 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, 21, 22]. 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 do not know those allegedly like minded users. Thus, our focus is on item-oriented approaches, but parallel techniques can be developed in a user-oriented fashion, by switching the roles of users and items.

Central to most item-oriented approaches is a similarity measure between items. Frequently, it is based on the Pearson correlation coefficient, ρ_{ij}, which measures the tendency of users to rate items i and j similarly. Since many ratings are unknown, it is expected that some items share only a handful of common raters. Computation of the correlation coefficient is based only on the common user support. Accordingly, similarities based on a greater user support are more reliable. An appropriate similarity measure, denoted by s_{ij}, would be a shrunk correlation coefficient:

$$s_{ij} \overset{\text{def}}{=} \frac{n_{ij}}{n_{ij} + \lambda_2} \rho_{ij} \tag{2}$$

The variable n_{ij} denotes the number of users that rated both i and j. A typical value for λ_2 is 100. Notice that the literature suggests additional alternatives for a similarity measure [21, 22].

Our goal is to predict r_{ui} – the unobserved rating 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 of neighboring items, while adjusting for user and item effects through the baseline estimates:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j\in \mathrm{S}^k(i;u)} s_{ij}(r_{uj} - b_{uj})}{\sum_{j\in \mathrm{S}^k(i;u)} s_{ij}} \tag{3}$$

Neighborhood-based methods of this form became very popular because they are intuitive and relatively simple to implement. However, in a recent work [2], we raised a few concerns about such neighborhood schemes. Most notably, these methods are not justified by a formal model. We also questioned the suitability of a similarity measure that isolates the relations between two items, without analyzing the interactions within the full set of neighbors. In addition, the fact that interpolation weights in (3) sum to one forces the method to fully rely on the neighbors even in cases where neighborhood information is absent (i.e., user u did not rate items similar to i), and it would be preferable to rely on baseline estimates. 优点

This led us to propose a more accurate neighborhood model, which overcomes these difficulties. Given a set of neighbors $\mathrm{S}^k(i;u)$ we need to compute *interpolation weights* $\{\theta_{ij}^u | j \in \mathrm{S}^k(i;u)\}$ that enable the best prediction rule of the form:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{S}^k(i;u)} \theta_{ij}^u (r_{uj} - b_{uj}) \tag{4}$$

Derivation of the interpolation weights can be done efficiently by estimating all inner products between item ratings; for a full description refer to [2].

2.3 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 [13], neural networks [18], and Latent Dirichlet Allocation [7]. We will focus on models that are induced by Singular Value Decomposition (SVD) on the user-item ratings matrix. Recently, SVD models have gained popularity, thanks to their attractive accuracy and scalability. A typical model associates each user u with a user-factors vector $p_u \in \mathbb{R}^f$, and each item i with an item-factors vector $q_i \in \mathbb{R}^f$. The prediction is done by taking an inner product, i.e., $\hat{r}_{ui} = b_{ui} + p_u^T q_i$. The more involved part is parameter estimation.

In information retrieval it is well established to harness SVD for identifying latent semantic factors [8]. However, applying SVD in the CF domain raises difficulties due to the high portion of missing ratings. Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting. Earlier works [14, 20] relied on imputation to fill in missing ratings and make the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. In addition, the data may be considerably distorted due to inaccurate imputation. Hence, more recent works [4, 6, 9, 17, 18, 22] suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model, such as:

$$\min_{p_*,q_*,b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda_3(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2) \tag{5}$$

Zhi-Lin Zhao's svd implement the algorithm

A simple gradient descent technique was applied successfully to solving (5).

Paterek [17] suggested the related NSVD model, which avoids explicitly parameterizing each user, but rather models users based on the items that they rated. This way, each item i is associated with two factor vectors q_i and x_i. The representation of a user u is through the sum: $\left(\sum_{j\in\mathrm{R}(u)} x_j\right) / \sqrt{|\mathrm{R}(u)|}$, so r_{ui} is predicted as: $b_{ui} + q_i^T \left(\sum_{j\in\mathrm{R}(u)} x_j\right) / \sqrt{|\mathrm{R}(u)|}$. Here, $\mathrm{R}(u)$ is the set of items rated by user u. Later in this work, we adapt Paterek's idea with some extensions.

2.4 The Netflix data

We evaluated our algorithms on the Netflix data of more than 100 million movie ratings performed by anonymous Netflix customers [5]. We are not aware of any publicly available CF dataset that is close to the scope and quality of this dataset. To maintain compatibility with results published by others, we adopted some standards that were set by Netflix, as follows. First, quality of the results is usually measured by their root mean squared error (RMSE): $\sqrt{\sum_{(u,i)\in TestSet} (r_{ui} - \hat{r}_{ui})^2 / |TestSet|}$. In addition, we report results on a test set provided by Netflix (also known as the Quiz set), which contains over 1.4 million recent ratings. Netflix compiled another 1.4 million recent ratings into a validation set, known as the Probe set, which we employ in Section 6. The two sets contain many more ratings by users that do not rate much and are harder to predict. In a way, they represent real requirements from a CF system, which needs to predict new ratings from older ones, and to equally address all users, not only the heavy raters.

The Netflix data is part of the ongoing Netflix Prize competition, where the benchmark is Netflix's proprietary system, Cinematch, which achieved a RMSE of 0.9514 on the test set. The grand prize will be awarded to a team that manages to drive this RMSE below 0.8563 (10% improvement). Results reported in this work lower the RMSE on the test set to levels around 0.887, which is better than previously published results on this dataset.

2.5 Implicit feedback

As stated earlier, an important goal of this work is devising models that allow integration of explicit and implicit user feedback. For a dataset such as the Netflix data, the most natural choice for implicit feedback would be movie rental history, which tells us about user preferences without requiring them to explicitly provide their ratings. However, such data is not available to us. Nonetheless, a less obvious kind of implicit data does exist within the Netflix dataset. The dataset does not only tell us the rating values, but also *which* movies users rate, regardless of *how* they rated these movies. In other words, a user implicitly tells us about her preferences by choosing to voice her opinion and vote a (high or low) rating. This reduces the ratings matrix into a binary matrix, where "1" stands for "rated", and "0" for "not rated". Admittedly, this binary data is not as vast and independent as other sources of implicit feedback could be. Nonetheless, we have found that incorporating this kind of implicit data – which inherently exist in every rating based recommender system – significantly improves prediction accuracy. Some prior techniques, such as Conditional RBMs [18], also capitalized on the same binary view of the data.

implicit feedback matrix is the use-item matrix 转化为binary matrix

The models that we suggest are not limited to a certain kind of implicit data. To keep generality, each user u is associated with two sets of items, one is denoted by $\mathrm{R}(u)$, and contains all the items for which ratings by u are available. The other one, denoted by $\mathrm{N}(u)$, contains all items for which u provided an implicit preference.

3. A NEIGHBORHOOD MODEL

In this section we introduce a new neighborhood model, which allows an efficient global optimization scheme. The model offers improved accuracy and is able to integrate implicit user feedback. We will gradually construct the various components of the model, through an ongoing refinement of our formulations.

Previous models were centered around *user-specific* interpolation weights – θ_{ij}^u in (4) or $s_{ij} / \sum_{j\in\mathrm{S}^k(i;u)} s_{ij}$ in (3) – relating item i to the items in a user-specific neighborhood $\mathrm{S}^k(i;u)$. In order to facilitate global optimization, we would like to abandon such user-specific weights in favor of global weights independent of a specific user. The weight from j to i is denoted by w_{ij} and will be learnt from the data through optimization. An initial sketch of the model describes each rating r_{ui} by the equation:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} \tag{6}$$

For now, (6) does not look very different from (4), besides our claim that the w_{ij}'s are not user specific. Another difference, which will be discussed shortly, is that here we sum over all items rated by u, unlike (4) that sums over members of $\mathrm{S}^k(i;u)$.

Let us consider the interpretation of the weights. Usually the weights in a neighborhood model represent interpolation coefficients relating unknown ratings to existing ones. Here, it is useful to adopt a different viewpoint, where weights represent offsets to baseline estimates. Now, the residuals, $r_{uj} - b_{uj}$, are viewed as the coefficients multiplying those offsets. For two related items i and j, we expect w_{ij} to be high. Thus, whenever a user u rated j higher than expected ($r_{uj} - b_{uj}$ is high), we would like to increase our estimate for u's rating of i by adding $(r_{uj} - b_{uj})w_{ij}$ to the baseline estimate. Likewise, our estimate will not deviate much from the baseline by an item j that u rated just as expected ($r_{uj} - b_{uj}$ is around zero), or by an item j that is not known to be predictive on i (w_{ij} is close to zero). This viewpoint suggests several enhancements to (6). First, we can use implicit feedback, which provide an alternative way to learn user preferences. To this end, we add another set of weights, and rewrite (6) as:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + \sum_{j \in \mathrm{N}(u)} c_{ij} \tag{7}$$

Much like the w_{ij}'s, the c_{ij}'s are offsets added to baseline estimates. For two items i and j, an implicit preference by u to j lead us to modify our estimate of r_{ui} by c_{ij}, which is expected to be high if j is predictive on i.[1]

Viewing the weights as global offsets, rather than as user-specific interpolation coefficients, emphasizes the influence of missing ratings. In other words, a user's opinion is formed not only by what he rated, but also by what he did not rate. For example, suppose that a movie ratings dataset shows that users that rate "Lord of the Rings 3" high also gave high ratings to "Lord of the Rings 1–2". This will establish high weights from "Lord of the Rings 1–2" to "Lord of the Rings 3". Now, if a user did not rate "Lord of the Rings 1–2" at all, his predicted rating for "Lord of the Rings 3" will be penalized, as some necessary weights cannot be added to the sum.

For prior models ((3),(4)) that interpolated $r_{ui} - b_{ui}$ from $\{r_{uj} - b_{uj} | j \in \mathrm{S}^k(i;u)\}$, it was necessary to maintain compatibility between the b_{ui} values and the b_{uj} values. However, here we do not use interpolation, so we can decouple the definitions of b_{ui} and b_{uj}. Accordingly, a more general prediction rule would be: $\hat{r}_{ui} = \tilde{b}_{ui} + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + \sum_{j \in \mathrm{N}(u)} c_{ij}$. Here, $\tilde{b}_{ui}$ represents predictions of r_{ui} by other methods such as a latent factor model. We elaborate more on this in Section 5. For now, we suggest the following rule that was found to work well:

$$\hat{r}_{ui} = \mu + b_u + b_i + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + \sum_{j \in \mathrm{N}(u)} c_{ij} \tag{8}$$

Importantly, the b_{uj}'s remain constants, which are derived as explained in Sec. 2.1. However, the b_u's and b_i's become parameters, which are optimized much like the w_{ij}'s and c_{ij}'s.

[1]In many cases it would be reasonable to attach significance weights to implicit feedback. This requires a modification to our formula which, for simplicity, will not be considered here.

A characteristic of the current scheme is that it encourages greater deviations from baseline estimates for users that provided many ratings (high $|\mathrm{R}(u)|$) or plenty of implicit feedback (high $|\mathrm{N}(u)|$). In general, this is a good practice for recommender systems. We would like to take more risk with well modeled users that provided much input. For such users we are willing to predict quirkier and less common recommendations. On the other hand, we are less certain about the modeling of users that provided only a little input, in which case we would like to stay with safe estimates close to the baseline values. However, our experience shows that the current model somewhat overemphasizes the dichotomy between heavy raters and those that rarely rate. Better results were obtained when we moderated this behavior, replacing the prediction rule with:

$$\begin{aligned}\hat{r}_{ui} =& \mu + b_u + b_i + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} \\ &+ |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} c_{ij}\end{aligned} \tag{9}$$

Complexity of the model can be reduced by pruning parameters corresponding to unlikely item-item relations. Let us denote by $\mathrm{S}^k(i)$ the set of k items most similar i, as determined by the similarity measure s_{ij}. Additionally, we use $\mathrm{R}^k(i;u) \overset{\mathrm{def}}{=} \mathrm{R}(u) \cap \mathrm{S}^k(i)$ and $\mathrm{N}^k(i;u) \overset{\mathrm{def}}{=} \mathrm{N}(u) \cap \mathrm{S}^k(i)$.[2] Now, when predicting r_{ui} according to (9), it is expected that the most influential weights will be associated with items similar to i. Hence, we replace (9) with:

$$\begin{aligned}\hat{r}_{ui} =& \mu + b_u + b_i + |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} \\ &+ |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}\end{aligned} \tag{10}$$

When $k = \infty$, rule (10) coincides with (9). However, for other values of k it offers the potential to significantly reduce the number of variables involved.

This is our final prediction rule, which allows fast online prediction. More computational work is needed at a pre-processing stage where parameters are estimated. A major design goal of the new neighborhood model was facilitating an efficient global optimization procedure, which prior neighborhood models lacked. Thus, model parameters are learnt by solving the regularized least squares problem associated with (10):

$$\begin{aligned}\min_{b_*, w_*, c_*} \sum_{(u,i) \in \mathcal{K}} \Bigg(& r_{ui} - \mu - b_u - b_i - |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij} \\ & - |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} \Bigg)^2 \\ & + \lambda_4 \Bigg(b_u^2 + b_i^2 + \sum_{j \in \mathrm{R}^k(i;u)} w_{ij}^2 + \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}^2 \Bigg)\end{aligned} \tag{11}$$

An optimal solution of this convex problem can be obtained by

[2]Notational clarification: With other neighborhood models it was beneficial to use $\mathrm{S}^k(i;u)$, which denotes the k items most similar to i among those rated by u. Hence, if u rated at least k items, we will always have $|\mathrm{S}^k(i;u)| = k$, regardless of how similar those items are to i. However, $|\mathrm{R}^k(i;u)|$ is typically smaller than k, as some of those items most similar to i were not rated by u.

least square solvers, which are part of standard linear algebra packages. However, we have found that the following simple gradient descent solver works much faster. Let us denote the prediction error, $r_{ui} - \hat{r}_{ui}$, by e_{ui}. We loop through all known ratings in $\mathcal{K}$. For a given training case r_{ui}, we modify the parameters by moving in the opposite direction of the gradient, yielding:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_i)$
- $\forall j \in \mathrm{R}^k(i; u)$:
  $w_{ij} \leftarrow w_{ij} + \gamma \cdot \left(|\mathrm{R}^k(i; u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_4 \cdot w_{ij} \right)$
- $\forall j \in \mathrm{N}^k(i; u)$:
  $c_{ij} \leftarrow c_{ij} + \gamma \cdot \left(|\mathrm{N}^k(i; u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_4 \cdot c_{ij} \right)$

The meta-parameters γ (step size) and λ_4 are determined by cross-validation. We used $\gamma = 0.005$ and $\lambda_4 = 0.002$ for the Netflix data. A typical number of iterations throughout the training data is 15. Another important parameter is k, which controls the neighborhood size. Our experience shows that increasing k always benefits the accuracy of the results on the test set. Hence, the choice of k should reflect a tradeoff between prediction accuracy and computational cost.

Experimental results on the Netflix data with the new neighborhood model are presented in Fig. 1. We studied the model under different values of parameter k. The pink curve shows that accuracy monotonically improves with rising k values, as root mean squared error (RMSE) falls from 0.9139 for $k = 250$ to 0.9002 for $k = \infty$. (Notice that since the Netflix data contains 17,770 movies, $k = \infty$ is equivalent to k =17,770, where all item-item relations are explored.) We repeated the experiments without using the implicit feedback, that is, dropping the c_{ij} parameters from our model. The results depicted by the yellow curve show a significant decline in estimation accuracy, which widens as k grows. This demonstrates the value of incorporating implicit feedback into the model.

For comparison we provide the results of two previous neighborhood models. First is a correlation-based neighborhood model (following (3)), which is the most popular CF method in the literature. We denote this model as CorNgbr. Second is a newer model [2] that follows (4), which will be denoted as WgtNgbr. For both these two models, we tried to pick optimal parameters and neighborhood sizes, which were 20 for CorNgbr, and 50 for WgtNgbr. The results are depicted by the green and cyan lines. Notice that the k value (the x-axis) is irrelevant to these models, as their different notion of neighborhood makes neighborhood sizes incompatible. It is clear that the popular CorNgbr method is noticeably less accurate than the other neighborhood models, though its 0.9406 RMSE is still better than the published Netflix's Cinematch RMSE of 0.9514. On the opposite side, our new model delivers more accurate results even when compared with WgtNgbr, as long as the value of k is at least 500.

Finally, let us consider running time. Previous neighborhood models require very light pre-processing, though, WgtNgbr [2] requires solving a small system of equations for each provided prediction. The new model does involve pre-processing where parameters are estimated. However, online prediction is immediate by following rule (10). Pre-processing time grows with the value of k. Typical running times per iteration on the Netflix data, as measured on a single processor 3.4GHz Pentium 4 PC, are shown in Fig. 2.

4. LATENT FACTOR MODELS REVISITED

As mentioned in Sec. 2.3, a popular approach to latent factor models is induced by an SVD-like lower rank decomposition of the

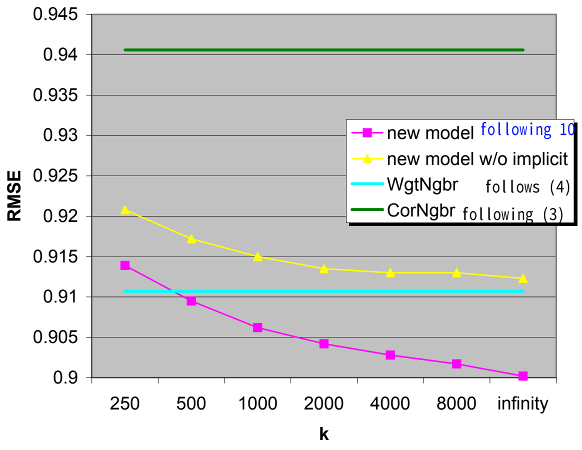


Figure 1: Comparison of neighborhood-based models. We measure the accuracy of the new model with and without implicit feedback. Accuracy is measured by RMSE on the Netflix test set, so lower values indicate better performance. RMSE is shown as a function of varying values of k, which dictates the neighborhood size. For reference, we present the accuracy of two prior models as two horizontal lines: the green line represents a popular method using Pearson correlations, and the cyan line represents a more recent neighborhood model.

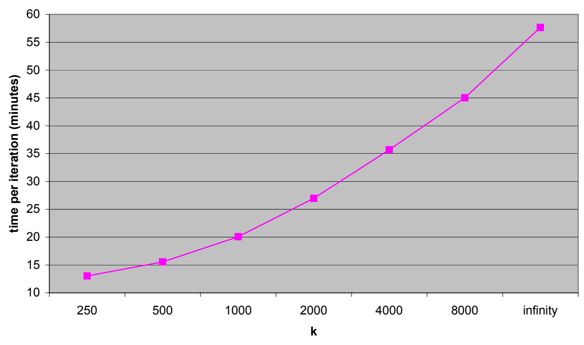


Figure 2: Running times (minutes) per iteration of the neighborhood model, as a function of the parameter k.

ratings matrix. Each user u is associated with a user-factors vector $p_u \in \mathbb{R}^f$, and each item i with an item-factors vector $q_i \in \mathbb{R}^f$. Prediction is done by the rule:

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i \tag{12}$$

Parameters are estimated by minimizing the associated squared error function (5). Funk [9] popularized gradient descent optimization, which was successfully practiced by many others [17, 18, 22]. Henceforth, we will dub this basic model "SVD". We would like to extend the model by considering also implicit information. Following Paterek [17] and our work in the previous section, we suggest the following prediction rule:

$$\hat{r}_{ui} = b_{ui} + q_i^T \Bigg(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \Bigg) \tag{13}$$

Here, each item i is associated with three factor vectors $q_i, x_i, y_i \in \mathbb{R}^f$. On the other hand, instead of providing an explicit parameterization for users, we represent users through the items that they prefer. Thus, the previous user factor p_i was replaced by the sum $|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{R}(u)}(r_{uj} - b_{uj})x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{N}(u)} y_j$. This new model, which will be henceforth named "Asymmetric-SVD", offers several benefits:

1. *Fewer parameters.* Typically the number of users is much larger than the number of products. Thus, exchanging user-parameters with item-parameters lowers the complexity of the model.

2. *New users.* Since Asymmetric-SVD does not parameterize users, we can handle new users as soon as they provide feedback to the system, without needing to re-train the model and estimate new parameters. Similarly, we can immediately exploit new ratings for updating user preferences. Notice that for new items we do have to learn new parameters. Interestingly, this asymmetry between users and items meshes well with common practices: systems need to provide immediate recommendations to new users who expect quality service. On the other hand, it is reasonable to require a waiting period before recommending items new to the system. As a side remark, it is worth mentioning that item-oriented neighborhood models exhibit the same desired asymmetry.

3. *Explainability.* Users expect a system to give a reason for its predictions, rather than facing "black box" recommendations. This not only enriches the user experience, but also encourages users to interact with the system, fix wrong impressions and improve long-term accuracy. In fact, the importance of explaining automated recommendations is widely recognized [11, 23]. Latent factor models such as SVD face real difficulties to explain predictions. After all, a key to these models is abstracting users via an intermediate layer of user factors. This intermediate layer separates the computed predictions from past user actions and complicates explanations. However, the new Asymmetric-SVD model does not employ any level of abstraction on the users side. Hence, predictions are a direct function of past users' feedback. Such a framework allows identifying which of the past user actions are most influential on the computed prediction, thereby explaining predictions by most relevant actions. Once again, we would like to mention that item-oriented neighborhood models enjoy the same benefit.

4. *Efficient integration of implicit feedback.* Prediction accuracy is improved by considering also implicit feedback, which provides an additional indication of user preferences. Obviously, implicit feedback becomes increasingly important for users that provide much more implicit feedback than explicit one. Accordingly, in rule (13) the implicit perspective becomes more dominant as $|\mathrm{N}(u)|$ increases and we have much implicit feedback. On the other hand, the explicit perspective becomes more significant when $|\mathrm{R}(u)|$ is growing and we have many explicit observations. Typically, a single explicit input would be more valuable than a single implicit input. The right conversion ratio, which represents how many implicit inputs are as significant as a single explicit input, is automatically learnt from the data by setting the relative values of the x_j and y_j parameters.

As usual, we learn the values of involved parameters by minimizing the regularized squared error function associated with (13):

Asymmetric-SVD

$$\min_{q_*,x_*,y_*,b_*} \sum_{(u,i)\in\mathcal{K}} \Bigg(r_{ui} - \mu - b_u - b_i - q_i^T \Bigg(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{R}(u)} (r_{uj} - b_{uj})x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{N}(u)} y_j \Bigg) \Bigg)^2 + \lambda_5 \Bigg(b_u^2 + b_i^2 + \|q_i\|^2 + \sum_{j\in\mathrm{R}(u)} \|x_j\|^2 + \sum_{j\in\mathrm{N}(u)} \|y_j\|^2 \Bigg) \tag{14}$$

We employ a simple gradient descent scheme to solve the system. On the Netflix data we used 30 iterations, with step size of 0.002 and $\lambda_5 = 0.04$.

An important question is whether we need to give up some predictive accuracy in order to enjoy those aforementioned benefits of Asymmetric-SVD. We evaluated this on the Netflix data. As shown in Table 1, prediction quality of Asymmetric-SVD is actually slightly better than SVD. The improvement is likely thanks to accounting for implicit feedback. This means that one can enjoy the benefits that Asymmetric-SVD offers, without sacrificing prediction accuracy. As mentioned earlier, we do not really have much independent implicit feedback for the Netflix dataset. Thus, we expect that for real life systems with access to better types of implicit feedback (such as rental/purchase history), the new Asymmetric-SVD model would lead to even further improvements. Nevertheless, this still has to be demonstrated experimentally.

In fact, as far as integration of implicit feedback is concerned, we could get more accurate results by a more direct modification of (12), leading to the following model:

Svd++

$$\hat{r}_{ui} = b_{ui} + q_i^T \Bigg(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{N}(u)} y_j \Bigg) \tag{15}$$

Now, a user u is modeled as $p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{N}(u)} y_j$. We use a free user-factors vector, p_u, much like in (12), which is learnt from the given explicit ratings. This vector is complemented by the sum $|\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{N}(u)} y_j$, which represents the perspective of implicit feedback. We dub this model "SVD++". Similar models were discussed recently [3, 19]. Model parameters are learnt by minimizing the associated squared error function through gradient descent. SVD++ does not offer the previously mentioned benefits of having less parameters, conveniently handling new users and readily explainable results. This is because we do abstract each user with a factors vector. However, as Table 1 indicates, SVD++ is clearly advantageous in terms of prediction accuracy. Actually, to our best knowledge, its results are more accurate than all previously published methods on the Netflix data. Nonetheless, in the next section we will describe an integrated model, which offers further accuracy gains.

5. AN INTEGRATED MODEL

The new neighborhood model of Sec. 3 is based on a formal model, whose parameters are learnt by solving a least squares problem. An advantage of this approach is allowing easy integration with other methods that are based on similarly structured global cost functions. As explained in Sec. 1, latent factor models and neighborhood models nicely complement each other. Accordingly, in this section we will integrate the neighborhood model with our most accurate factor model – SVD++. A combined model will sum

Model	50 factors	100 factors	200 factors
SVD	0.9046	0.9025	0.9009
Asymmetric-SVD	0.9037	0.9013	0.9000
SVD++	0.8952	0.8924	0.8911

Table 1: Comparison of SVD-based models: prediction accuracy is measured by RMSE on the Netflix test set for varying number of factors (f). Asymmetric-SVD offers practical advantages over the known SVD model, while slightly improving accuracy. Best accuracy is achieved by SVD++, which directly incorporates implicit feedback into the SVD model.

the predictions of (10) and (15), thereby allowing neighborhood and factor models to enrich each other, as follows:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j\in \mathrm{N}(u)} y_j \right) + |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j\in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj})w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j\in \mathrm{N}^k(i;u)} c_{ij} \tag{16}$$

In a sense, rule (16) provides a 3-tier model for recommendations. The first tier, $\mu + b_u + b_i$, describes general properties of the item and the user, without accounting for any involved interactions. For example, this tier could argue that "The Sixth Sense" movie is known to be good, and that the rating scale of our user, Joe, tends to be just on average. The next tier, $q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j\in \mathrm{N}(u)} y_j \right)$, provides the interaction between the user profile and the item profile. In our example, it may find that "The Sixth Sense" and Joe are rated high on the Psychological Thrillers scale. The final "neighborhood tier" contributes fine grained adjustments that are hard to profile, such as the fact that Joe rated low the related movie "Signs".

Model parameters are determined by minimizing the associated regularized squared error function through gradient descent. Recall that $e_{ui} \stackrel{\mathrm{def}}{=} r_{ui} - \hat{r}_{ui}$. We loop over all known ratings in $\mathcal{K}$. For a given training case r_{ui}, we modify the parameters by moving in the opposite direction of the gradient, yielding:

- $b_u \leftarrow b_u + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma_2 \cdot (e_{ui} \cdot (p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j\in \mathrm{N}(u)} y_j) - \lambda_7 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma_2 \cdot (e_{ui} \cdot q_i - \lambda_7 \cdot p_u)$
- $\forall j \in \mathrm{N}(u)$:
  $y_j \leftarrow y_j + \gamma_2 \cdot (e_{ui} \cdot |\mathrm{N}(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_7 \cdot y_j)$
- $\forall j \in \mathrm{R}^k(i;u)$:
  $w_{ij} \leftarrow w_{ij} + \gamma_3 \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_8 \cdot w_{ij} \right)$
- $\forall j \in \mathrm{N}^k(i;u)$:
  $c_{ij} \leftarrow c_{ij} + \gamma_3 \cdot \left(|\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_8 \cdot c_{ij} \right)$

When evaluating the method on the Netflix data, we used the following values for the meta parameters: $\gamma_1 = \gamma_2 = 0.007$, $\gamma_3 = 0.001$, $\lambda_6 = 0.005$, $\lambda_7 = \lambda_8 = 0.015$. It is beneficial to decrease step sizes (the γ's) by a factor of 0.9 after each iteration. The neighborhood size, k, was set to 300. Unlike the pure neighborhood model (10), here there is no benefit in increasing k, as adding neighbors covers more global information, which the latent factors already capture adequately. The iterative process runs for around 30 iterations till convergence. Table 2 summarizes the performance over the Netflix dataset for different number of factors. Once again, we report running times on a Pentium 4 PC for processing the 100 million ratings Netflix data. By coupling neighborhood and latent factor models together, and recovering signal from implicit feedback, accuracy of results is improved beyond other methods.

	50 factors	100 factors	200 factors
RMSE	0.8877	0.8870	0.8868
time/iteration	17min	20min	25min

Table 2: Performance of the integrated model. Prediction accuracy is improved by combining the complementing neighborhood and latent factor models. Increasing the number of factors contributes to accuracy, but also adds to running time.

Recall that unlike SVD++, both the neighborhood model and Asymmetric-SVD allow a direct explanation of their recommendations, and do not require re-training the model for handling new users. Hence, when explainability is preferred over accuracy, one can follow very similar steps to integrate Asymmetric-SVD with the neighborhood model, thereby improving accuracy of the individual models while still maintaining the ability to reason about recommendations to end users.

6. EVALUATION THROUGH A TOP-K RECOMMENDER

So far, we have followed a common practice with the Netflix dataset to evaluate prediction accuracy by the RMSE measure. Achievable RMSE values on the Netflix test data lie in a quite narrow range. A simple prediction rule, which estimates r_{ui} as the mean rating of movie i, will result in RMSE=1.053. Notice that this rule represents a sensible "best sellers list" approach, where the same recommendation applies to all users. By applying personalization, more accurate predictions are obtained. This way, Netflix Cinematch system could achieve a RMSE of 0.9514. In this paper, we suggested methods that lower the RMSE to 0.8870. In fact, by blending several solutions, we could reach a RMSE of 0.8645. Nonetheless, none of the 3,400 teams actively involved in the Netflix Prize competition could reach, as of 20 months into the competition, lower RMSE levels, despite the big incentive of winning a \$1M Grand Prize. Thus, the range of attainable RMSEs is seemingly compressed, with less than 20% gap between a naive non-personalized approach and the best known CF results. Successful improvements of recommendation quality depend on achieving the elusive goal of enhancing users' satisfaction. Thus, a crucial question is: what effect on user experience should we expect by lowering the RMSE by, say, 10%? For example, is it possible, that a solution with a slightly better RMSE will lead to completely different and better recommendations? This is central to justifying research on accuracy improvements in recommender systems. We would like to shed some light on the issue, by examining the effect of lowered RMSE on a practical situation.

A common case facing recommender systems is providing "top K recommendations". That is, the system needs to suggest the top K products to a user. For example, recommending the user a few specific movies which are supposed to be most appealing to him. We would like to investigate the effect of lowering the RMSE on the quality of top K recommendations. Somewhat surprisingly, the Netflix dataset can be used to evaluate this.

Recall that in addition to the test set, Netflix also provided a validation set for which the true ratings are published. We used all 5-star ratings from the validation set as a proxy for movies that interest users.[3] Our goal is to find the relative place of these "interesting movies" within the total order of movies sorted by predicted ratings for a specific user. To this end, for each such movie i, rated 5-stars by user u, we select 1000 additional random movies and predict the ratings by u for i and for the other 1000 movies. Finally, we order the 1001 movies based on their predicted rating, in a decreasing order. This simulates a situation where the system needs to recommend movies out of 1001 available ones. Thus, those movies with the highest predictions will be recommended to user u. Notice that the 1000 movies are random, some of which may be of interest to user u, but most of them are probably of no interest to u. Hence, the best hoped result is that i (for which we know u gave the highest rating of 5) will precede the rest 1000 random movies, thereby improving the appeal of a top-K recommender. There are 1001 different possible ranks for i, ranging from the best case where none (0%) of the random movies appears before i, to the worst case where all (100%) of the random movies appear before i in the sorted order. Overall, the validation set contains 384,573 5-star ratings. For each of them (separately) we draw 1000 random movies, predict associated ratings, and derive a ranking between 0% to 100%. Then, the distribution of the 384,573 ranks is analyzed. (Remark: since the number 1000 is arbitrary, reported results are in percentiles (0%–100%), rather than in absolute ranks (0–1000).)

判断算法优劣的标准

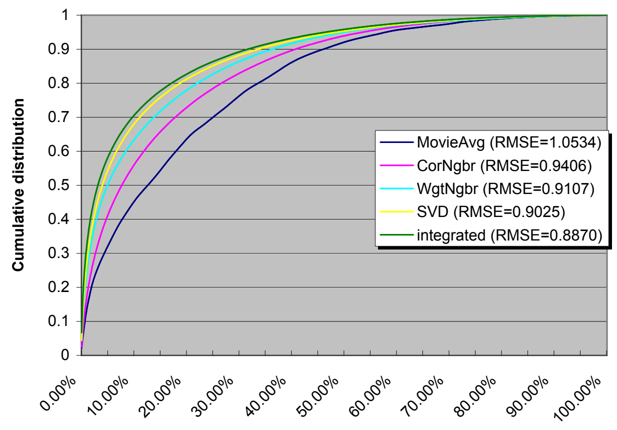
We used this methodology to evaluate five different methods. The first method is the aforementioned non-personalized prediction rule, which employs movie means to yield RMSE=1.053. Henceforth, it will be denoted as MovieAvg. The second method is a correlation-based neighborhood model, which is the most popular approach in the CF literature. As mentioned in Sec. 3, it achieves a RMSE of 0.9406 on the test set, and was named CorNgbr. The third method is the improved neighborhood approach of [2], which we named WgtNgbr and could achieve RMSE= 0.9107 on the test set. Fourth is the SVD latent factor model, with 100 factors thereby achieving RMSE=0.9025 as reported in Table 1. Finally, we consider our most accurate method, the integrated model, with 100 factors, achieving RMSE=0.8870 as shown in Table 2.

Figure 3(top) plots the cumulative distribution of the computed percentile ranks for the five methods over the 384,573 evaluated cases. Clearly, all methods would outperform a random/constant prediction rule, which would have resulted in a straight line connecting the bottom-left and top-right corners. Also, the figure exhibits an ordering of the methods by their strength. In order to achieve a better understanding, let us zoom in on the head of the x-axis, which represents top-K recommendations. After all, in order to get into the top-K recommendations, a product should be ranked before almost all others. For example, if 600 products are considered, and three of them will be suggested to the user, only those ranked 0.5% or lower are relevant. In a sense, there is no difference between placing a desired 5-star movie at the top 5%, top 20% or top 80%, as none of them is good enough to be presented to the user. Accordingly, Fig. 3(bottom), plots the cumulative ranks distribution between 0% and 2% (top 20 ranked items out of 1000).

随机排序的概率都是50%

As the figure shows, there are very significant differences among the methods. For example, the integrated method has a probability of 0.067 to place a 5-star movie before all other 1000 movies (rank=0%). This is more than three times better than the chance of the MovieAvg method to achieve the same. In addition, it is 2.8 times better than the chance of the popular CorNgbr to achieve the same. The other two methods, WgtNbr and SVD, have a probability of around 0.043 to achieve the same. The practical interpretation is that if about 0.1% of the items are selected to be suggested to the user, the integrated method has a significantly higher chance to pick a specified 5-star rated movie. Similarly, the integrated method has a probability of 0.157 to place the 5-star movie before at least 99.8% of the random movies (rank$\leqslant$0.2%). For comparison, MovieAvg and CorNgbr have much slimmer chances of achieving the same: 0.050 and 0.065, respectively. The remaining two methods, WgtNgbr and SVD, lie between with probabilities of 0.107 and 0.115, respectively. Thus, if one movie out of 500 is to be suggested, its probability of being a specific 5-stars rated one becomes noticeably higher with the integrated model.

We are encouraged, even somewhat surprised, by the results. It is evident that small improvements in RMSE translate into significant improvements in quality of the top K products. In fact, based on RMSE differences, we did not expect the integrated model to deliver such an emphasized improvement in the test. Similarly, we did not expect the very weak performance of the popular correlation based neighborhood scheme, which could not improve much upon a non-personalized scheme.



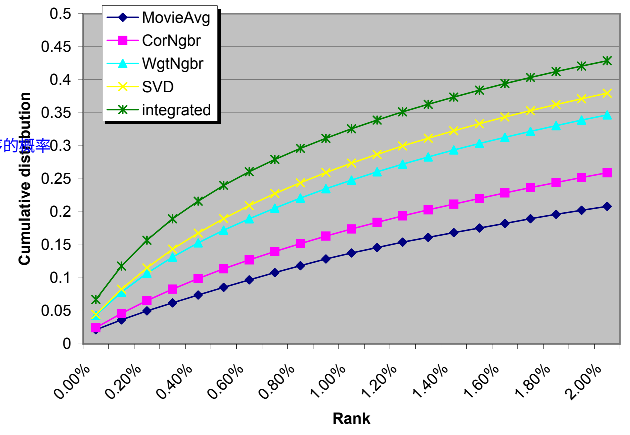


Figure 3: Comparing the performance of five methods on a top-K recommendation task, where a few products need to be suggested to a user. Values on the x-axis stand for the percentile ranking of a 5-star rated movie; lower values represent more successful recommendations. We experiment with 384,573 cases and show the cumulative distribution of the results. The lower plot concentrates on the more relevant region, pertaining to low x-axis values. The plot shows that the integrated method has the highest probability of obtaining low values on the x-axis. On the other hand, the non-personalized MovieAvg method and the popular correlation-based neighborhood method (CorNgbr) achieve the lowest probabilities.

[3]In this study, the validation set is excluded from the training data.

7. DISCUSSION

This work proposed improvements to two of the most popular approaches to Collaborative Filtering. First, we suggested a new neighborhood based model, which unlike previous neighborhood methods, is based on formally optimizing a global cost function. This leads to improved prediction accuracy, while maintaining merits of the neighborhood approach such as explainability of predictions and ability to handle new users without re-training the model. Second, we introduced extensions to SVD-based latent factor models that allow improved accuracy by integrating implicit feedback into the model. One of the models also provides advantages that are usually regarded as belonging to neighborhood models, namely, an ability to explain recommendations and to handle new users seamlessly. In addition, the new neighborhood model enables us to derive, for the first time, an integrated model that combines the neighborhood and the latent factor models. This is helpful for improving system performance, as the neighborhood and latent factor models address the data at different levels and complement each other.

Quality of a recommender system is expressed through multiple dimensions including: accuracy, diversity, ability to surprise with unexpected recommendations, explainability, appropriate top-K recommendations, and computational efficiency. Some of those criteria are relatively easy to measure, such as accuracy and efficiency that were addressed in this work. Some other aspects are more elusive and harder to quantify. We suggested a novel approach for measuring the success of a top-K recommender, which is central to most systems where a few products should be suggested to each user. It is notable that evaluating top-K recommenders significantly sharpens the differences between the methods, beyond what a traditional accuracy measure could show.

A major insight beyond this work is that improved recommendation quality depends on successfully addressing different aspects of the data. A prime example is using implicit user feedback to extend models' quality, which our methods facilitate. Evaluation of this was based on a very limited form of implicit feedback, which was available within the Netflix dataset. This was enough to show a marked improvement, but further experimentation is needed with better sources of implicit feedback, such as purchase/rental history. Other aspects of the data that may be integrated to improve prediction quality are content information like attributes of users or products, or dates associated with the ratings, which may help to explain shifts in user preferences.

Acknowledgements

The author thanks Suhrid Balakrishnan, Robert Bell and Stephen North for their most helpful remarks.

8. 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 and Y. Koren, "Lessons from the Netflix Prize Challenge", *SIGKDD Explorations* **9** (2007), 75–79.

[4] R. M. 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.

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

[6] J. Canny, "Collaborative Filtering with Privacy via Factor Analysis", *Proc. 25th ACM SIGIR Conf.on Research and Development in Information Retrieval (SIGIRŠ02)*, pp. 238–245, 2002.

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

[8] S. Deerwester, S. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, "Indexing by Latent Semantic Analysis", *Journal of the Society for Information Science* **41** (1990), 391–407.

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

[10] 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.

[11] J. L. Herlocker, J. A. Konstan and J. Riedl, , "Explaining Collaborative Filtering Recommendations", *Proc. ACM conference on Computer Supported Cooperative Work*, pp. 241–250, 2000.

[12] 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.

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

[14] D. Kim and B. Yum, "Collaborative Filtering Based on Iterative Principal Component Analysis", *Expert Systems with Applications* **28** (2005), 823–830.

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

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

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

[18] 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.

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

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

[21] 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.

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

[23] N. Tintarev and J. Masthoff, "A Survey of Explanations in Recommender Systems", *ICDE'07 Workshop on Recommender Systems and Intelligent User Interfaces*, 2007.