Factor in the Neighbors: Scalable and Accurate Collaborative Filtering

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 most common approach to CF is based on neighborhood models, which is based on similarities between products or users. In this work we introduce a new neighborhood model with an improved prediction accuracy. The model works by minimizing a global cost function. Further accuracy improvements are achieved by extending the model to exploit both explicit and implicit feedback by the users. Past models were limited by the need to compute all pairwise similarities between items or users, which grow quadratically with input size. In particular, this limitation vastly complicates adopting user similarity models, due to the typical large number of users. Our new model solves these limitations by factoring the neighborhood model, thus making both item-item and user-user implementations scale linearly with the size of the data. The methods are tested on the Netflix data, with encouraging results. In addition, we suggest a new evaluation metric, which highlights the differences among methods, based on their performance at a top-K recommendation task. Our study reveals a very significant improvement in quality of top-K recommendation.

1. INTRODUCTION

[1] 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 [18], Google [9], Netflix [6], TiVo [2] and Yahoo! [21] are increasingly adopting such recommenders.

Recommender systems are often based on *Collaborative Filtering* (CF) [12], 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 [18], TiVo [2] 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-item 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. For users, each factor measures how much the user likes movies that score high on the corresponding movie factor.

Latent factor models offer high expressive ability to describe various aspects of the data. Thus, they tend to provide more accurate results than neighborhood models; see, e.g., [4, 7, 15, 17, 22, 27]. However, most literature and commercial systems (e.g., those of Amazon [18], TiVo [2] and Netflix[2]) are based on the neighborhood models. The prevalence of neighborhood models is partly thanks to their relative simplicity and intuitiveness. However, there are more important reasons for real life systems to stick with those less accurate models. First, they naturally provide intuitive explanations of the reasoning behind recommendations, which often enhance user experience beyond what improved accuracy may achieve. Second, they can immediately provide recommendations based on just entered user feedback.

In this work we introduce a new neighborhood model with an improved accuracy on par with recent latent factor models. In the same time, the model retains the two aforementioned fundamental advantages of neighborhood models. Namely, it can explain its recommendations and handle new ratings without requiring retraining. Admittedly, the new model is not as simple as common neighborhood models, because it requires a training stage much like latent factor models. This reflects a more principled way to modeling neighborhood relations, which rewards the model not only with improved accuracy, but also with greater flexibility to address different aspects of the data and to integrate other models. In particular, we can make the model significantly more scalable than

[1] Parts of this work overlap a KDD'08 publication [17].

[2] Based on personal knowledge.

previous neighborhood models, without compromising its accuracy or other desired properties.

The CF field has enjoyed a surge of interest since October 2006, when the Netflix Prize competition [6] 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 the importance of integrating different forms of user input into the models [4]. 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 [20]. 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 other models, and also inclusion of implicit user feedback. Section 4 shows how to make the model highly scalable by introducing novel factorization techniques. 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. 5, with encouraging results.

2. PRELIMINARIES

We are given ratings about m users and n items. 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}. Usually the vast majority of ratings are unknown. For example, in the Netflix data 99% of the possible ratings are missing because a user typically rates only a small portion of the movies. The (u, i) pairs for which r_{ui} is known are stored in the set $\mathcal{K} = \{(u, i) \mid r_{ui} \text{ is known}\}$. 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:

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

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.

An easier, yet somewhat less accurate way to estimate the parameters is by decoupling the calculation of the b_i's from the calculation of the b_u's. First, for each item i we set:

$$b_i = \frac{\sum_{u:(u,i)\in\mathcal{K}} (r_{ui} - \mu)}{\lambda_2 + |\{u|(u,i) \in \mathcal{K}\}|}$$

Then, for each user u we set:

$$b_u = \frac{\sum_{i:(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_i)}{\lambda_3 + |\{i|(u,i) \in \mathcal{K}\}|}$$

Averages are shrunk towards zero by using the regularization parameters, $\lambda_2.\lambda_3$, which are determined by cross validation. Typical values on the Netflix dataset are: $\lambda_2 = 25, \lambda_3 = 10$.

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-user based; see [14] for a good analysis. Such user-user methods estimate unknown ratings based on recorded ratings of like-minded users. Later, an analogous item-item approach [18, 26] 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-item approach more favorable in many cases [3, 26, 27]. In addition, item-item 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. We focus mostly on item-item approaches, but techniques developed in this work make the user-user approach as computationally efficient; sec Sec. 4.1.

Central to most item-item 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:

$$s_{ij} \stackrel{\text{def}}{=} \frac{n_{ij}}{n_{ij} + \lambda_4} \rho_{ij} \tag{2}$$

The variable n_{ij} denotes the number of users that rated both i and j. A typical value for λ_4 is 100. Notice that the literature suggests additional alternatives for a similarity measure [26, 27].

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:

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

Neighborhood-based methods of this form became very popular because they are intuitive and relatively simple to implement. They also offer the following two useful properties:

1. *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 [13, 28]. The neighborhood framework allows identifying which of the past user actions are most influential on the computed prediction. In prediction rule (3), each past rating (r_{uj} for $j \in \mathrm{S}^k(i;u)$) receives a separate term in forming the predicted $\hat{r}_{ui}$, and thus we can isolate its unique contribution. The past ratings associated with highest contribution are identified as the major explanation behind the recommendation.
2. *New ratings.* Item-item neighborhood models can provide updated recommendations immediately after users entered new ratings. This includes handling new users as soon as they provide feedback to the system, without needing to retrain the model and estimate new parameters. Here, we assume that relationships between items (the s_{ij} values) are stable and barely change on a daily basis. Thus, prediction rule (3) can be reliably used with the most recent set of ratings given by the user. Notice that for items new to the system 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 (or new ratings by old users) who expect quality service. On the other hand, it is reasonable to require a waiting period before recommending items new to the system.

However, in a recent work [3], we raised a few concerns about traditional 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.

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* $\{\theta^u_{ij} | j \in \mathrm{S}^k(i;u)\}$ that enable the best prediction rule of the form:

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

Derivation of the interpolation weights can be done efficiently by estimating all inner products between item ratings; for a full description refer to [3].

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 [15], neural networks [23], Latent Dirichlet Allocation [8], or 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 [10]. 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 [16, 25] 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 [5, 7, 11, 17, 22, 23, 27] suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model. The reported results compare very favorably with neighborhood models.

However, Latent factor models such as SVD face real difficulties when needed 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. Similarly, reflecting new ratings requires re-learning the user factors, and cannot be done at virtually no cost as in the neighborhood models. Thus, we believe that for practical applications neighborhood models are still expected to be a common choice.

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 [6]. 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):

$$\sqrt{\sum_{(u,i) \in TestSet} (r_{ui} - \hat{r}_{ui})^2 / |TestSet|}$$

a measure that puts more emphasis on large errors compared with the alternative of mean absolute error. 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 5. 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).

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 probably 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 [23], also capitalized on the same binary view of the data. The benefit of using the binary data is closely related to the fact that ratings do not miss at random, but users are deliberate in which items they choose to rate; see Marlin et al. [19].

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^u_{ij} 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:

$$\hat{r}_{ui} = b_{ui} + \sum_{j\in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} \tag{5}$$

For now, (5) 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 (5). 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 (5) as:

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

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.[3]

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: $\hat{r}_{ui} = \tilde{b}_{ui} + \sum_{j\in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + \sum_{j\in \mathrm{N}(u)} c_{ij}$. The constant $\tilde{b}_{ui}$ can represent predictions of r_{ui} by other methods such as a latent factor model. Here, we suggest the following rule that was found to work well:

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

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.

We have found that it is beneficial to normalize sums in the model leading to the form:

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

The constant α controls the extent of normalization. A non-normalized rule ($\alpha = 0$), encourages greater deviations from baseline estimates for users that provided many ratings (high $|\mathrm{R}(u)|$) or plenty of implicit feedback (high $|\mathrm{N}(u)|$). On the other hand, a fully normalized rule, with $\alpha = 1$, will eliminate the effect of number of ratings on deviations from baseline estimates. In many cases it would be a good practice for recommender systems to have greater deviation from baselines for users that rate a lot. This way, we take more risk with well modeled users that provided much input. For such users we are willing to predict quirkier and less common recommendations. At the same time, we are less certain about the modeling of

[3] 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.

users that provided only a little input, in which case we would like to stay with safe estimates close to the baseline values. Our experience with the Netflix dataset shows that best results are achieved with partial normalization, which allows moderately higher deviations for heavy raters. Thus, we set $\alpha = 0.5$, as in the prediction rule:

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

As an optional refinement, 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 $\mathrm{R}^k(i;u) \stackrel{\mathrm{def}}{=} \mathrm{R}(u) \cap \mathrm{S}^k(i)$ and $\mathrm{N}^k(i;u) \stackrel{\mathrm{def}}{=} \mathrm{N}(u) \cap \mathrm{S}^k(i)$.[4] 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:

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

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):

$$\min_{b_*, w_*, c_*} \sum_{(u,i) \in \mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij} - |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} \right)^2 + \lambda_5 \left(b_u^2 + b_i^2 + \sum_{j \in \mathrm{R}^k(i;u)} w_{ij}^2 + \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}^2 \right) \quad (11)$$

An optimal solution of this convex problem can be obtained by 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:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_i)$
- $\forall j \in \mathrm{R}^k(i;u)$: $w_{ij} \leftarrow w_{ij} + \gamma \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_5 \cdot w_{ij} \right)$
- $\forall j \in \mathrm{N}^k(i;u)$: $c_{ij} \leftarrow c_{ij} + \gamma \cdot \left(|\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_5 \cdot c_{ij} \right)$

The meta-parameters γ (step size) and λ_5 are determined by cross-validation. We used $\gamma = 0.005$ and $\lambda_5 = 0.002$ for the Netflix data. 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. In the next section we will describe a factored version of the model, which cancels this tradeoff by allowing us to work with the most accurate $k = \infty$ while lowering running time.

A typical number of iterations throughout the training data is 15–20. As for time complexity per iteration, let us analyze the most accurate case where $k = \infty$, which is equivalent to using prediction rule (9). For each user u and item $i \in \mathrm{R}(u)$ we need to modify both $\{w_{ij} | j \in \mathrm{R}(u)\}$ and $\{c_{ij} | j \in \mathrm{N}(u)\}$. Thus the overall time complexity of the training phase is $O(\sum_u |\mathrm{R}(u)|(|\mathrm{R}(u)| + |\mathrm{N}(u)|))$.

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 [3] 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. 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. Notice that the k value (the x-axis) is irrelevant to previous models, as their different notion of neighborhood makes neighborhood sizes incompatible. Yet, we observed that while the performance of our algorithm keeps improving as more neighbors are added, this was not true with previous models. For those past models performance peaks with a relatively small number of neighbors and since then declines. This may be explained by the fact that in our model parameters are directly learnt from the data through a formal optimization procedure, what facilitates using many more parameters effectively.

Finally, let us consider running time. Previous neighborhood models require very light pre-processing, though, WgtNgbr [3] 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

[4]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.

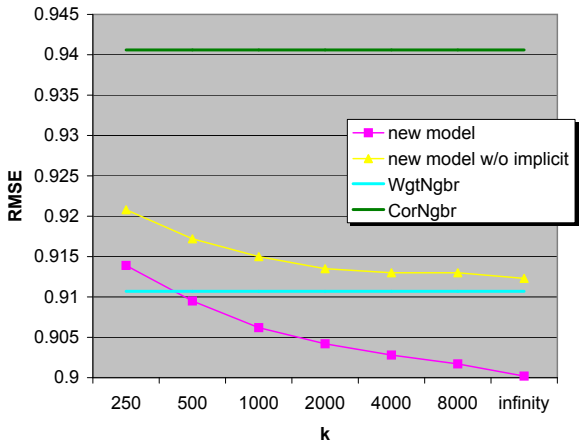

Figure 1: Comparison of neighborhood-based models. We measure the accuracy of the new model with and without implicit feedback. Accuracy is measured by RMSE on the Netflix test set, so lower values indicate better performance. RMSE is shown as a function of varying values of k, which dictates the neighborhood 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.

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.

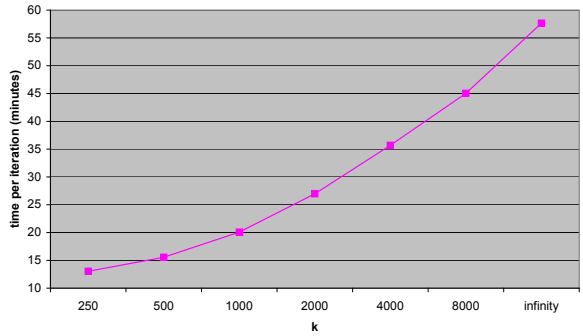

Figure 2: Running times (minutes) per iteration of the neighborhood model, as a function of the parameter k.

4. A FACTORIZED NEIGHBORHOOD MODEL

In the previous section we presented our most accurate neighborhood model, which is based on prediction rule (9) with training time complexity $O(\sum_u |\mathrm{R}(u)|(|\mathrm{R}(u)|+|\mathrm{N}(u)|))$ and space complexity $O(m+n^2)$. (Recall that m is the number of users, and n is the number of items.) We could improve time and space complexity by sparsifying the model through pruning unlikely item-item relations. Sparsification was controlled by the parameter $k \leqslant n$, which reduced running time and allowed space complexity of $O(m+nk)$. However, as k gets lower, the accuracy of the model declines as well. Now, we will show how to retain the accuracy of the full dense prediction rule (9), while significantly lowering time and space complexity.

We will factor item-item relationships by associating each item i with three vectors: $q_i, x_i, y_i \in \mathbb{R}^f$. This way, we will confine w_{ij} to be $q_i^T x_i$. Similarly, we impose $c_{ij} = q_i^T y_j$. Essentially, these vectors strive to map items into a latent factor space where they are measured against various aspects that are revealed automatically by learning the data. By substituting this into (9) we get the following prediction rule:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) q_i^T x_i + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} q_i^T y_j \quad (12)$$

Computational gains become more obvious by using the equivalent rule:

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

Notice that the bulk of the rule – $\left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$ – depends only on u while being independent of i. This leads to an efficient way to learn the model parameters. As usual, we minimize the regularized squared error function associated with (12):

$$\min_{q_*, x_*, y_*, b_*} \sum_{(u,i) \in \mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - q_i^T \left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right) \right)^2 + \lambda_6 \left(b_u^2 + b_i^2 + \|q_i\|^2 + \sum_{j \in \mathrm{R}(u)} \|x_j\|^2 + \sum_{j \in \mathrm{N}(u)} \|y_j\|^2 \right) \quad (14)$$

We employ a simple gradient descent scheme, which is described in the following pseudo code:

LearnFactorizedNeighborhoodModel(Known ratings: r_{ui}, rank: f)
% For each item i compute $q_i, x_i, y_i \in \mathbb{R}^f$
% which form a neighborhood model
Const $\#Iterations = 20, \gamma = 0.002, \lambda = 0.04$
% Gradient descnet sweeps:
for $count = 1, \ldots, \#Iterations$ **do**
  for $u = 1, \ldots, m$ **do**
    % Compute the component independent of i:
    $p_u \leftarrow |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j$
    $p_u \leftarrow p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j$
    $sum \leftarrow 0$
    for all $i \in \mathrm{R}(u)$ **do**
      $\hat{r}_{ui} \leftarrow \mu + b_u + b_i + q_i^T p_u$
      $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$
      % Accumulate information for gradient steps on x_i, y_i:
      $sum \leftarrow sum + e_{ui} \cdot q_i$
      % Perform gradient step on q_i, b_u, b_i:
      $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$
      $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u)$
      $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i)$
    for all $i \in \mathrm{R}(u)$ **do**
      % Perform gradient step on x_i:
      $x_i \leftarrow x_i + \gamma \cdot (|\mathrm{R}(u)|^{-\frac{1}{2}} \cdot (r_{ui} - b_{ui}) \cdot sum - \lambda \cdot x_i)$
    for all $i \in \mathrm{N}(u)$ **do**
      % Perform gradient step on y_i:
      $y_i \leftarrow y_i + \gamma \cdot (|\mathrm{N}(u)|^{-\frac{1}{2}} \cdot sum - \lambda \cdot y_i)$
return $\{q_i, x_i, y_i | i = 1, \ldots, n\}$

The time complexity of this model is linear with the input size - $O(f \cdot \sum_u (|\mathrm{R}(u)| + |\mathrm{N}(u)|))$, which is significantly better than the non-factorized model that required time $O(\sum_u |\mathrm{R}(u)|(|\mathrm{R}(u)| + |\mathrm{N}(u)|))$. We measured the performance of the model on the Netflix data; see Table 1. Accuracy is improved as we use more factors (increasing f). However, going beyond 200 factors could barely improve performance, while slowing running time. Interestingly, we have found that with $f \geqslant 200$ accuracy negligibly exceeds that of the best non-factorized model (with $k = \infty$). In addition, the improved time complexity translates into a big difference in wall-clock measured running time. For example, the time-per-iteration for the non-factorized model (with $k = \infty$) was close to 58 minutes. On the other hand, a factorized model with $f = 200$ could complete an iteration in 14 minutes without degrading accuracy at all.

The most important benefit of the factorized model is the reduced space complexity, which is $O(m + nf)$ – linear in the input size. Previous neighborhood models required storing all pairwise relations between items, leading to a quadratic space complexity of $O(m + n^2)$. For the Netflix dataset which contains 17,770 movies, such quadratic space can still fit within core memory. Some commercial recommenders process a much higher number of items. For example, a leading online movie rental service is currently advertising offering over 100,000 titles. Music download shops offer even more titles. Such more comprehensive systems with data on 100,000s items eventually need to resort to external storage in order to fit the entire set of pairwise relations. However, as number of items is growing towards millions, as in the Amazon item-item recommender system, which accesses stored similarity information for several million catalog items [18], designers must keep a sparsified version of the pairwise relations. To this end, only values relating an item to its top-k most similar neighbors are stored thereby reducing space complexity to $O(m + nk)$. However, a sparsification technique will inevitably degrade accuracy by missing important relations, as demonstrated in the previous section. In addition, identification of the top k most similar items in such a high dimensional space is a non-trivial task that can require considerable computational efforts. All these issues do not exist in our factorized neighborhood model, which offers a linear time and space complexity without trading off accuracy.

#factors	50	100	200	500
RMSE	0.9037	0.9013	0.9000	0.8998
time/iteration	4.5 min	8 min	14 min	34 min

Table 1: Performance of the factorized item-item neighborhood model. The models with $\geqslant 200$ factors slightly outperform the non-factorized model, while providing much shorter running time.

Discussion The factorized neighborhood model resembles principles of some latent factor models. The important distinction is that here we factorize the item-item relationships, rather than the ratings themselves. The results reported in Table 1 are comparable to those of the widely used SVD model, which generates results with RMSEs ranging in 0.9046–0.9009 for 50–200 factors [17]. However, some sophisticated extensions of the SVD model would result in even better accuracy [17]. Nonetheless, the factorized neighborhood model retains the practical advantages of traditional neighborhood models, which are difficult to achieve with latent factor models. Here, we refer to the ability to explain recommendations and to immediately reflect new ratings, as discussed in Sec. 2.2.

As a side remark, we would like to mention that the decision to use three separate sets of factors was intended to give us more flexibility. Indeed, on the Netflix data this allowed achieving most accurate results. However, another reasonable choice could be using a smaller set of vectors, e.g., by requiring: $q_i = x_i$ (implying symmetric weights: $w_{ij} = w_{ji}$).

4.1 A user-user model

A user-user neighborhood model predicts ratings by considering how like-minded users rated the same items. Such models can be implemented by switching the roles of users and items throughout our derivation of the item-item model. Here, we would like to concentrate on a user-user model, which is analogous to the item-item model of (9). The major difference is replacing the w_{ij} weights relating item pairs, with weights relating user pairs:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathrm{R}(i)} (r_{vj} - b_{vj}) w_{uv} \qquad (15)$$

The set $\mathrm{R}(i)$ contains all the users for who rated item i. Notice that here we decided to not account for implicit feedback. This is because adding such feedback was not very beneficial for the user-user model when working with the Netflix data.

User-user models can become useful in various situations. For example, some recommenders may deal with items that are rapidly replaced, thus making item-item relations very volatile. On the other hand, the user base is pretty stable enabling to establish long term relations between users. An example of such a case is a recommender system for web articles or news items which are rapidly changing by their nature; see, e.g., [9]. In such cases, systems centered around user-user relations are more appealing.

In addition, user-user approaches identify different kinds of relations that item-item approaches may fail to recognize, and thus can be useful on certain occasions. For example, suppose that we

#factors	50	100	200	500
RMSE	0.9119	0.9110	0.9101	0.9093
time/iteration	3 min	5 min	8.5 min	18 min

Table 2: Performance of the factorized user-user neighborhood model.

want to predict r_{ui}, but none of the items rated by user u is really relevant to i. In this case, an item-item approach will face obvious difficulties. However, when employing a user-user perspective, we may find a set of users similar to u, who rated i. The ratings of i by these users would allow us to improve prediction of r_{ui}.

The major disadvantage of user-user model is computational. Since typically there are many more users than items, precomuting and storing all user-user relations, or even a reasonably sparsified version thereof, is overly expensive or completely impractical. In addition to the high $O(m^2)$ space complexity, the time complexity for optimizing model (15) is also much higher than its item-item counterpart, being $O(\sum_i |\mathrm{R}(i)|^2)$ (notice that $|\mathrm{R}(i)|$ is expected to be much higher than $|\mathrm{R}(u)|$). These issues render user-user models as a less practical choice.

A factorized model All those computational differences disappear by factorizing the user-user model along the same lines as in the item-item model. Now, we associate each user u with two vector $p_u, z_u \in \mathbb{R}^f$. We assume the user-user relations to satisfy: $w_{uv} = p_u^T z_v$. Let us substitute this into (15) to get:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathrm{R}(i)} (r_{vj} - b_{vj}) p_u^T z_v \qquad (16)$$

Once again, an efficient computation is achieved by including the terms that depends on i but are independent of u in a separate sum, so the prediction rule is presented in the equivalent form:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T |\mathrm{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathrm{R}(i)} (r_{vj} - b_{vj}) z_v \qquad (17)$$

In a parallel fashion to the item-item model, all parameters are learnt in linear time $O(f \cdot \sum_i |\mathrm{R}(i)|)$. The space complexity is also linear with the input size being $O(n+mf)$. This significantly lowers the complexity of the user-user model compared to previously known results. In fact, running time measured on the Netflix data shows that now the user-user model is even faster than the item-item model; see Table 2. We should remark that unlike the item-item model, our implementation of the user-user model did not account for implicit feedback, what probably lead to its shorter running time. Accuracy of the user-user model is significantly better than that of the widely-used correlation-based item-item models, which achieves RMSE=0.9406 as reported in Sec. 3. Furthermore, accuracy is slightly better than the variant of the item-item model that also did not account for implicit feedback (yellow curve in Fig. 1). This is quite surprising given the common wisodm that item-item methods are more accurate than user-user ones. It appears that a well implemented user-user model can match speed and accuracy of an item-item model. However, our item-item model could significantly benefit by accounting for implicit feedback.

Fusing item-item and user-user models Since item-item and user-user models address different aspects of the data, overall accuracy is expected to improve by combining predictions of both models. Such an approach was previously suggested and was shown to improve accuracy; see, e.g. [5, 29]. However, past efforts were based on blending the item-item and user-user predictions during a post-processing stage, after each individual model is separately trained independently of the other model. A more principled approach will optimize the two models simultaneously, letting them know of each other while parameters are being learnt. Thus, throughout the entire training phase each model is aware of the capabilities of the other model and strives to complement it. Our approach, which states the neighborhood models as formal optimization problems, allows doing that naturally. We devise a model which sums the item-item model (12) and the user-user model (16), leading to:

$$\begin{aligned}\hat{r}_{ui} =& \mu + b_u + b_i + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) q_i^T x_i \\ &+ |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} q_i^T y_j \\ &+ |\mathrm{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathrm{R}(i)} (r_{vj} - b_{vj}) p_u^T z_v \end{aligned} \qquad (18)$$

Model parameters are learnt by gradient descent optimization of the associated squared error function. Our experiments with the Netflix data show that prediction accuracy is indeed better than that of each individual model. For example, with 100 factors the obtained RMSE is 0.8966, while with 200 factors the obtained RMSE is 0.8953.

Here we would like to comment that our approach allows integrating the neighborhood models also with completely different models in a similar way. For example, in [17] we showed an integrated model that combines the item-item model with a latent factor model (called SVD++), thereby achieving improved prediction accuracy with RMSE below 0.887. Therefore, other possiblities with potentially better accuracy should be explored before considering the integration of an item-item and user-user models.

5. 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 below 0.9. 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, 5%? 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.[5] 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).)

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 [3], 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 (see [17]). Finally, we consider our new neighborhood model with a full set of neighbors ($k = \infty$), achieving RMSE=0.9002.

Figure 3 plots the cumulative distribution of the computed percentile ranks for the five methods over the 384,573 evaluated cases. 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, we concentrate on cumulative ranks distribution between 0% and 2% (top 20 ranked items out of 1000).

As the figure shows, there are very significant differences among the methods. For example, the new neighborhood (NewNgbr) method has a probability of 0.077 to place a 5-star movie before all other 1000 movies (rank=0%). This is more than 3.5 times better than the chance of the MovieAvg method to achieve the same. In addition, it is over three times better than the chance of the popular CorNgbr to achieve the same. The other two methods, WgtNgbr 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 new neighborhood method has a significantly higher chance to pick a specified 5-star rated movie. Similarly, NewNgbr has a probability of 0.163 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-star rated one becomes noticeably higher with the new neighborhood 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 new neighborhood 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.

Notice that our evaluation answers the question: given a user and a 5-star rated movie, what is the chance that a particular method will rank this movie within the top X%? One can come up with other related questions, which we did not analyze here. For example, we can move from a single specific 5-star movie, to all movies in which the user is interested, thus answering a question like: given a user and set of all movies she would rate 5 stars, what is the chance that a method will rank at least one 5-star rated movie within the top X%? However, since the validation set provides us with a very limited view of user preferences, we do not really know the full set of movies that she likes, and hence could not evaluate performance regarding the latter question.

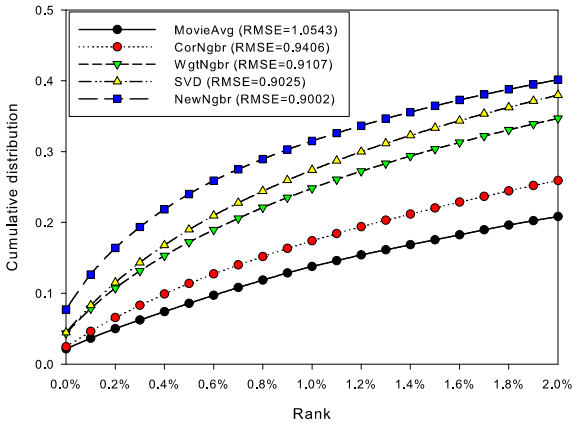

Figure 3: Comparing the performance of five methods on a top-K recommendation task, where a few products need to be suggested to a user. Values on the x-axis stand for the percentile ranking of a 5-star rated movie; lower values represent more successful recommendations. We experiment with 384,573 cases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

6. CONCLUSIONS

This work proposed a new neighborhood based model, which unlike previous neighborhood methods is based on formally optimizing a global cost function. This leads to improved prediction

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

accuracy, while maintaining merits of the neighborhood approach such as explainability of predictions and ability to handle new ratings (or new users) without re-training the model.

In addition we suggested a factorized version of the neighborhood model, which improves its computational complexity while retaining prediction accuracy. In particular, it is free from the quadratic storage requirements that limited past neighborhood models. Thus, the new method offers scalability to a very large number of items without resorting to complicated sparsification techniques that tend to lower accuracy. The improved complexity of the factorized model is especially emphasized with the user-user models, which were considered so far as computationally inefficient. Now they become as efficient as the item-item ones, thus making them a viable option. In fact, thanks to their vastly improved efficiency, we could build full user-user models that achieve accuracy competitive with item-item models.

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.

7. 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] K. Ali and W. van Stam, "TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture", *Proc. 10th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining*, pp. 394–401, 2004.

[3] 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.

[4] R. Bell and Y. Koren, "Lessons from the Netflix Prize Challenge", *SIGKDD Explorations* **9** (2007), 75–79.

[5] 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.

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

[7] J. Canny, "Collaborative Filtering with Privacy via Factor Analysis", *Proc. 25th ACM SIGIR Conf.on Research and Development in Information Retrieval (SIGIR02)*, pp. 238–245, 2002.

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

[9] A. Das, M. Datar, A. Garg and S. Rajaram, "Google News Personalization: Scalable Online Collaborative Filtering, *WWW07*, pp. 271-280, 20007.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

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

[17] Y. Koren, "Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model", *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

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

[19] Benjamin M. Marlin, Richard S. Zemel, Sam Roweis, and Malcolm Slaney, " Collaborative filtering and the Missing at Random Assumption", *Proc. 23rd Conference on Uncertainty in Artificial Intelligence*, 2007.

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

[21] S.-T. Park and D. M. Pennock, "Applying Collaborative Filtering Techniques to Movie Search for Better Ranking and Browsing, *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 550559, 2007.

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

[23] 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.

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

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

[26] 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.

[27] G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk, "Major Components of the Gravity Recommendation System",

SIGKDD Explorations **9** (2007), 80–84.

[28] N. Tintarev and J. Masthoff, “A Survey of Explanations in Recommender Systems”, *ICDE’07 Workshop on Recommender Systems and Intelligent User Interfaces*, 2007.

[29] J. Wang, A. P. de Vries and M. J. T. Reinders,“Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion”, *Proc. 29th ACM SIGIR Conference on Information Retrieval*, pp. 501–508, 2006.