Collaborative Filtering with Temporal Dynamics

Yehuda Koren
Yahoo! Research, Haifa, Israel
yehuda@yahoo-inc.com

ABSTRACT

Customer preferences for products are drifting over time. Product perception and popularity are constantly changing as new selection emerges. Similarly, customer inclinations are evolving, leading them to ever redefine their taste. Thus, modeling temporal dynamics should be a key when designing recommender systems or general customer preference models. However, this raises unique challenges. Within the eco-system intersecting multiple products and customers, many different characteristics are shifting simultaneously, while many of them influence each other and often those shifts are delicate and associated with a few data instances. This distinguishes the problem from concept drift explorations, where mostly a single concept is tracked. Classical time-window or instance-decay approaches cannot work, as they lose too much signal when discarding data instances. A more sensitive approach is required, which can make better distinctions between transient effects and long term patterns. The paradigm we offer is creating a model tracking the time changing behavior throughout the life span of the data. This allows us to exploit the relevant components of all data instances, while discarding only what is modeled as being irrelevant. Accordingly, we revamp two leading collaborative filtering recommendation approaches. Evaluation is made on a large movie rating dataset by Netflix. Results are encouraging and better than those previously reported on this dataset.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

collaborative filtering, recommender systems, concept drift

1. INTRODUCTION

Modeling time drifting data is a central problem in data mining. Often, data is changing over time, and up to date modeling should be continuously updated to reflect its present nature. The analysis of such data needs to find the right balance between discounting temporary effects that have very low impact on future behavior, while capturing longer-term trends that reflect the inherent nature of the data. This led to many works on the problem, which is also widely known as *concept drift*; see e.g. [15, 25].

Modeling temporal changes in customer preferences brings unique challenges. One kind of concept drift in this setup is the emergence of new products or services that change the focus of customers. Related to this are seasonal changes, or specific holidays, which lead to characteristic shopping patterns. All those changes influence the whole population, and are within the realm of traditional studies on concept drift. However, many of the changes in user behavior are driven by localized factors. For example, a change in the family structure can drastically change shopping patterns. Likewise, individuals gradually change their taste in movies and music. All those changes cannot be captured by methods that seek a global concept drift. Instead, for each customer we are looking at different types of concept drifts, each occurs at a distinct time frame and is driven towards a different direction.

The need to model time changes at the level of each individual significantly reduces the amount of available data for detecting such changes. Thus we should resort to more accurate techniques than those that suffice for modeling global changes. For example, it would no longer be adequate to abandon or simply underweight far in time user transactions. The signal that can be extracted from those past actions might be invaluable for understanding the customer herself or be indirectly useful to modeling other customers. Yet, we need to distill long term patterns while discounting transient noise. This requires a more sensitive methodology for addressing drifting customer preferences. It would not be adequate to concentrate on identifying and modeling just what is relevant to the present or the near future. Instead, we require an accurate modeling of each point in the past, which will allow us to distinguish between persistent signal that should be captured and noise that should be isolated from the longer term parts of the model.

Modeling user preferences is relevant to multiple applications ranging from spam filtering to market-basket analysis. Our main focus in the paper is on modeling user preferences for building a recommender system, but we believe that general lessons that we learn would apply to other applications as well. Automated recommendations is a very active research field [12]. Such systems analyze patterns of user interest in items or products to provide personalized recommendations of items that will suit a user's taste. We expect user preferences to change over time. This may stem from multiple factors, some are fundamental while others are more circumstantial. For example, in a movie recommender system, users may change their preferred genre or adopt a new viewpoint on an actor or director. In addition, they may alter the appearance of their feedback. E.g., in a system where users provide star ratings to products, a user that used to indicate a neutral preference by a

KDD'09, June 28–July 1, 2009, Paris, France.

"3 stars" input, may now indicate dissatisfaction by the same "3 stars" feedback. Similarly, it is known that user feedback is influenced by anchoring, where current ratings should be taken as relative to other ratings given at the same short period. Finally, in many instances systems cannot separate different household members accessing the same account, even though each member has a different taste and deserves a separate model. This creates a de facto multifaceted meta-user associated with the account. A way to get some distinction between different persons is by assuming that time-adjacent accesses are being done by the same member (sometimes on behalf of other members), which can be naturally captured by a temporal model that assumes a drifting nature of a customer.

All these patterns and the likes should have made temporal modeling a predominant factor in building recommender systems. Nonetheless, with very few exceptions (to be mentioned in Sec. 7), the recommenders literature does not address temporal changes in user behavior. Perhaps, because user behavior is composed of many different concept drifts, all acting in a different timeframe and different directions, thus making common methodologies for dealing with concept drift and temporal data less successful at this setup. We are showing that capturing time drifting patterns in user behavior is essential to improving accuracy of recommenders. This also gives us hope that the insights from successful time modeling for recommenders will be useful in other data mining applications.

Our test bed is a large movie rating dataset released by Netflix as the basis of a well publicized competition [4]. This dataset combines several merits for the task at hand. First, it is not a synthetic dataset, but contains user-movie ratings by real paying Netflix subscribers. In addition, its relatively large size – above 100 million date-stamped ratings – makes it a better proxy for real life large scale datasets, while putting a premium on computational efficiency. Finally, unlike some other dominant datasets, time effects are natural and are not introduced artificially. Two interesting (if not surprising) temporal effects that emerge within this dataset are shown in Fig. 1. One effect is an abrupt shift of rating scale that happened in early 2004. At that time, the mean rating value jumped from around 3.4 stars to above 3.6 stars. Another significant effect is that ratings given to movies tend to increase with the movie age. That is, older movies receive higher ratings than newer ones. In Sec. 6 we will return to these phenomena and use our temporal modeling to shed some light on their origins.

The major contribution of this work is presenting a methodology and specific techniques for modeling time drifting user preferences in the context of recommender systems. The proposed approaches are applied on the aforementioned extensively analyzed movie ratings dataset, enabling us to firmly compare our methods with those reported recently. We show that by incorporating temporal information we achieve best results reported so far, indicating the significance of uncovering temporal effects.

The rest of the paper is organized as follows. In the next section we describe basic notions and notation. Then, in Sec. 3 our principles for addressing time changing user preferences are evolved. Those principles are then materialized, in quite different ways, within two leading recommender techniques: factor modeling (Sec. 4) and item-item neighborhhod modeling (Sec. 5). In Sec. 6 we describe an exploratory study, followed by surveying related work in Sec. 7.

2. PRELIMINARIES

We are given ratings about m users (henceforth, interchangeable with "customers") and n items (henceforth, interchangeable with "products"). We reserve special indexing letters for distinguishing users from items: for users u, v, and for items i, j. We use t for time (or, date). A rating $r_{ui}(t)$ indicates the preference by user u of item i at day t, where high values mean stronger preferences.

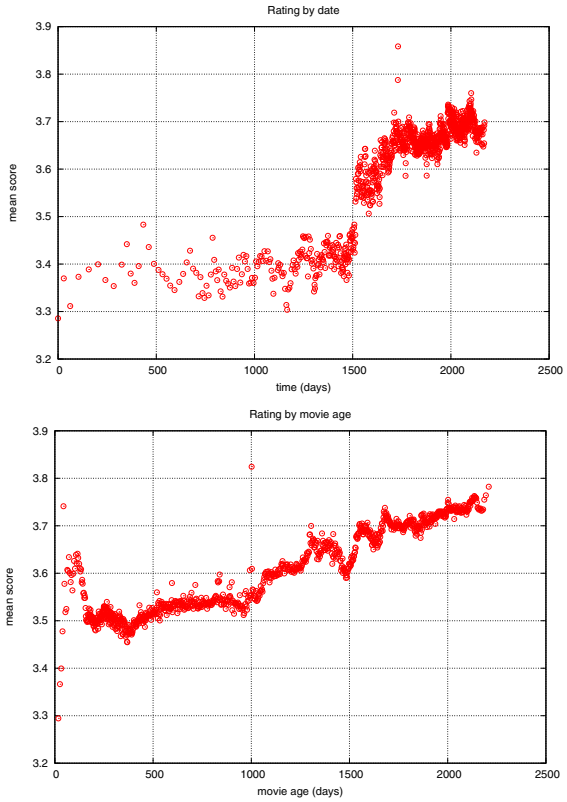

Figure 1: Two temporal effects emerging within the Netflix movie rating dataset. *Top:* the average movie rating made a sudden jump in early 2004 (1500 days since the first rating in the dataset). *Bottom:* ratings tend to increase with the movie age at the time of the rating. Here, movie age is measured by the time span since its first rating event within the dataset. In both charts each point averages 100,000 rating instances.

For example, values can be integers ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. User u rates item i at most once, otherwise we take only the freshest rating, so given u and i, the day of rating is unique. Sometimes, when the day of rating is not relevant, we will use the short notation r_{ui}. We distinguish predicted ratings from known ones, by using the notation $\hat{r}_{ui}(t)$ for the predicted value of $r_{ui}(t)$. Usually the vast majority of ratings are unknown. The (u, i, t) triples for which $r_{ui}(t)$ is known are stored in the set $\mathcal{K} = \{(u, i, t) \mid r_{ui}(t) \text{ is known}\}$.

We evaluated our algorithms on a movie rating dataset of more than 100 million date-stamped ratings performed by about half million anonymous Netflix customers on 17,770 movies between Dec 31, 1999 and Dec 31, 2005 [4]. We are not aware of any publicly available comparable dataset that is close to the scope and quality of this one. To maintain compatibility with results published by others, we adopted some common standards. We evaluated our methods on two comparable sets designed by Netflix: a hold-out set ("Probe set") and a test set ("Quiz set"), each of which contains over 1.4 million ratings. Reported results are on the test set, while experiments on the hold-out set show the same findings. In our time-modeling context, it is important to note that the test instances of each user come later in time than his/her training instances. The quality of the results is 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. Achievable RMSE values on the test set lie in a quite compressed range, as reported by many partici-

pants in the related competition. Nonetheless, there is evidence that small improvements in RMSE terms can have a significant impact on the quality of the top few presented recommendations [8].

Recommender systems are often based on *Collaborative Filtering* (CF), which relies only on past user behavior—e.g., their previous transactions or product ratings—and does not require the creation of explicit profiles. When enough ratings were gathered per item, as in the Netflix movie rating dataset, CF becomes the preferred and more accurate technique. 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 [9], 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-item approach [9, 14] 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.

Latent factor models, such as matrix factorization, 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.

3. TRACKING DRIFTING CUSTOMER PREFERENCES

One of the frequently mentioned examples of concept drift is changing customer preferences over time, e.g.: "customer preferences change as new products and services become available" [7]. This aspect of drifting customer preferences highlights a common paradigm in the literature of having global drifting concepts influencing the data as a whole. However, in many applications, including our focus application of recommender systems, we also face a more complicated form of concept drift where interconnected preferences of many users are drifting in different ways at different time points. This requires the learning algorithm to keep track of multiple changing concepts. In addition the typically low amount of data instances associated with individual customers calls for more concise and efficient learning methods, which maximize the utilization of signal in the data.

In a survey on the problem of concept drift, Tsymbal [22] argues that three approaches can be distinguished in the literature. The *instance selection* approach discards instances that are less relevant to the current state of the system. A common variant is time window approaches were only recent instances are considered. A possible disadvantage of this simple model is that it is giving the same significance to all instances within the considered time window, while completely discarding all other instances. This might be reasonable when the time shift is abrupt, but less so when time shift is gradual. Thus, a refinement is *instance weighting* were instances are weighted based on their estimated relevance. Frequently, a time decay function is used, under-weighting instances as they occur deeper into the past. The third approach is based on *ensemble learning*, which maintains a family of predictors that together produce the final outcome. Those predictors are weighted by their perceived relevance to the present time point, e.g., predictors that were more successful on recent instances get higher weights.

We performed extensive experiments with instance weighting schemes, trying different exponential time decay rates on both neighborhood and factor models. The consistent finding was that prediction quality improves as we moderate that time decay, reaching best quality when there is no decay at all. This is despite the fact that users do change their taste and rating scale over the years, as we show later. However, much of the old preferences still persist or, more importantly, help in establishing useful cross-user or cross-product patterns in the data. Thus, just underweighting past actions loses too much signal along with the lost noise, which is detrimental given the scarcity of data per user.

As for ensemble learning, having multiple models, each of which considers only a fraction of the total behavior may miss those global patterns that can be identified only when considering the full scope of user behavior. What makes them even less appealing in our case is the need to keep track of the independent drifting behaviors of many customers. This, in turn, would require building a separate ensemble for each user. Such a separation will significantly complicate our ability to integrate information across users along multiple time points, which is the cornerstone of *collaborative filtering*. For example, an interesting relation between products can be established by related actions of many users, each of them at a totally different point of time. Capturing such a collective signal requires building a single model encompassing all users and items together.

All those considerations led us to the following guidelines we adopt for modeling drifting user preferences.

- We seek models that explain user behavior along the full extent of the time period, not only the present behavior (while subject to performance limitations). This is key to being able to extract signal from each time point, while neglecting only the noise.
- Multiple changing concepts should be captured. Some are user-dependent and some are item-dependent. Similarly, some are gradual while others are sudden.
- While we need to model separate drifting "concepts" or preferences per user and/or item, it is essential to combine all those concepts within a single framework. This allows modeling interactions crossing users and items thereby identifying higher level patterns.
- In general, we do not try to extrapolate future temporal dynamics, e.g., estimating future changes in a user's preferences. This could be very helpful but is seemingly too difficult, especially given a limited amount of known data. Rather than that, our goal is to capture past temporal patterns in order to isolate persistent signal from transient noise. This, indeed, helps in predicting *future* behavior.

Now we turn to how these desirable principles materialize into concrete methods when dealing with two leading approaches to collaborative filtering - matrix factorization and neighborhood models.

4. TIME-AWARE FACTOR MODEL

4.1 The anatomy of a factor model

One of the more successful approaches to CF is based on a matrix factorization model [2, 5, 10, 13, 18]. This approach lends itself well to an adequate modeling of temporal effects. Before

we deal with those temporal effects, we would like to establish the foundations of a static factor model.

Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that ratings are modeled as inner products in that space. Accordingly, each user u is associated with a vector $p_u \in \mathbb{R}^f$ and each item i is associated with a vector $q_i \in \mathbb{R}^f$. A rating is predicted by the rule:

$$\hat{r}_{ui} = q_i^T p_u \tag{1}$$

In order to learn the vectors p_u and q_i we minimize the regularized squared error:

$$\min_{q_*, p_*} \sum_{(u,i,t) \in \mathcal{K}} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

The constant λ controls the extent of regularization, as usually determined by cross validation. Minimization is typically performed by either stochastic gradient descent or alternating least squares.

Such a pure factor model serves well in capturing the interaction between users and items. However, much of the observed rating values are due to effects associated with either users or items, independently of their interaction. A prime example is that typical CF data exhibit large user and item biases – i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others.

We will encapsulate those effects, which do not involve user-item interaction, within the *baseline predictors*. These baseline predictors tend to capture much of the observed signal, in particular much of the temporal dynamics within the data. Hence, it is vital to model them accurately, which enables better identification of the part of the signal that truly represents user-item interaction and should be subject to factorization.

A suitable way to construct a static baseline predictor is as follows. Denote by μ the overall average rating. A baseline predictor for an unknown rating r_{ui} is denoted by b_{ui} and accounts for the user and item main effects:

$$b_{ui} = \mu + b_u + b_i \tag{2}$$

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 $3.7 - 0.3 + 0.5$.

The baseline predictor should be integrated back into the factor model. To achieve this we extend rule (1) to be:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{3}$$

Here main effects are explicitly isolated, thus letting the $q_i^T p_u$ factorization deal effectively with the relevant portions of the signal; see also [8, 10].

The factor model we are using in this work is SVD++ [8], which slightly differs from (3). This model was shown to offer a superior accuracy by also accounting for the more implicit information recorded by which items were rated (regardless of their rating value). To this end a second set of item factors is added, relating item i to a factor vector $y_i \in \mathbb{R}^f$. Those new item factors are used to characterize users based on the set of items that they rated. The exact model is as follows (see [8] for further explanations):

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \left(p_u + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} y_j \right) \tag{4}$$

The set $\mathrm{R}(u)$ contains the items rated by user u.

The decomposition of a rating into distinct portions is convenient here, as it allows us to treat different temporal aspects in separation. More specifically, we identify the following effects: (1) user biases (b_u) change over time; (2) Item biases (b_i) change over time; (3) User preferences (p_u) change over time. On the other hand, we would not expect a significant temporal variation of item characteristics (q_i), as items, unlike humans, are static in their nature. We start with a detailed discussion of the temporal effects that are contained within the baseline predictors.

4.2 Time changing baseline predictors

Much of the temporal variability is included within the baseline predictors, through two major temporal effects. First is addressing the fact that an item's popularity is changing over time. For example, movies can go in and out of popularity as triggered by external events such as the appearance of an actor in a new movie. This is manifested in our models by the fact that item bias b_i will not be a constant but a function that changes over time. The second major temporal effect is related to user biases - users change their baseline ratings over time. For example, a user who tended to rate an average movie "4 stars", may now rate such a movie "3 stars", for various reasons explained earlier. Hence, in our models we would like to take the parameter b_u as a function of time. This induces the following template for a time sensitive baseline predictor:

$$b_{ui}(t) = \mu + b_u(t) + b_i(t) \tag{5}$$

The function $b_{ui}(t)$ represents the baseline estimate for u's rating of i at day t. Here, $b_u(t)$ and $b_i(t)$ are real valued functions that change over time. The exact way to build these functions should reflect a reasonable way to parameterize the involving temporal changes. We will detail our choice in the context of the movie rating dataset, which demonstrates some typical considerations. A major distinction is between temporal effects that span extended periods of time and more transient effects. In the movie rating case, we do not expect movie likeability to fluctuate on a daily basis, but rather to change over more extended periods. On the other hand, we observe that user effects can change on a daily basis, reflecting inconsistencies natural to customer behavior. This requires finer time resolution when modeling user-biases compared to a lower resolution that suffices for capturing item-related time effects.

Let us start with our choice of time-changing item biases - $b_i(t)$, which are easier to capture since we do not need finest resolution there. Thus, an adequate decision would be to split the item biases into time-based bins. During each time period corresponding to a single bin we use a distinct item bias. The decision of how to split the timeline into bins should balance the desire to achieve finer resolution (hence, smaller bins) with the need to have enough ratings per bin (hence, larger bins). For the movie rating data, there is a wide variety of bin sizes that yield about the same accuracy. In our implementation each bin corresponds to roughly ten consecutive weeks of data, leading to an overall number of 30 bins spanning all days in the dataset. A day t is associated with an integer $\mathrm{Bin}(t)$ (a number between 1 and 30 in our data), such that the movie bias is split into a stationary part and a time changing part:

$$b_i(t) = b_i + b_{i,\mathrm{Bin}(t)} \tag{6}$$

While binning the parameters works well on the items, it is more of a challenge on the users side. On the one hand, we would like a

finer resolution for users to detect very short lived temporal effects. On the other hand, we do not expect having enough ratings per user to produce reliable estimates for isolated bins. Different function forms can be considered for parameterizing temporal user behavior, with varying complexity and accuracy.

The first modeling choice is very concise, and uses a linear function to capture a possible gradual drift of user bias. Let us first introduce some new notation. For each user u, we denote the mean date of rating by t_u. Now, if u rated a movie on day t, then the associated time deviation of this rating is defined as:

$$\text{dev}_u(t) = \text{sign}(t - t_u) \cdot |t - t_u|^\beta$$

Here $|t - t_u|$ measures the time distance (e.g., number of days) between dates t and t_u. We set the value of β by cross validation; in our implementation $\beta = 0.4$. We introduce a single new parameter for each user called α_u so that we get our first definition of a time-dependent user-bias:

$$b_u^{(1)}(t) = b_u + \alpha_u \cdot \text{dev}_u(t) \tag{7}$$

This offers a simple linear model for approximating a drifting behavior, which requires learning two parameters per user: b_u and α_u. A more flexible parameterization is offered by splines. Let u be a user associated with n_u ratings. We designate k_u time points – $\{t_1^u, \ldots, t_{k_u}^u\}$ – spaced uniformly across the dates of u's ratings as kernels that control the following function:

$$b_u^{(2)}(t) = b_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}} \tag{8}$$

The parameters $b_{t_l}^u$ are associated with the control points (or, kernels), and are automatically learnt from the data. This way the user bias is formed as a time-weighted combination of those parameters. The number of control points, k_u, balances flexibility and computational efficiency. In our application we set k_u=$n_u^{0.25}$, letting it grow with the number of available ratings. The constant γ determines the smoothness of the spline; we set γ=0.3 by cross validation.

So far we have discussed smooth functions for modeling the user bias, which mesh well with *gradual concept drift*. However, in many applications there are *sudden drifts* emerging as "spikes" associated with a single day or session. For example, in the movie rating dataset we have found that multiple ratings a user gives in a single day, tend to concentrate around a single value. Such an effect does not span more than a single day. This may reflect the mood of the user that day, the impact of ratings given in a single day on each other, or changes in the actual rater in multi-person accounts. To address such short lived effects, we assign a single parameter per user and day, absorbing the day-specific variability. This parameter is denoted by $b_{u,t}$. Notice that in some applications the basic primitive time unit to work with can be shorter or longer than a day. E.g., our notion of day can be exchanged with a notion of a user session.

In the Netflix movie rating data, a user rates on 40 different days on average. Thus, working with $b_{u,t}$ requires, on average, 40 parameters to describe each user bias. It is expected that $b_{u,t}$ is inadequate as a standalone for capturing the user bias, since it misses all sorts of signals that span more than a single day. Thus, it serves as an additive component within the previously described schemes. The time-linear model (7) becomes:

$$b_u^{(3)}(t) = b_u + \alpha_u \cdot \text{dev}_u(t) + b_{u,t} \tag{9}$$

Similarly, the spline-based model becomes:

$$b_u^{(4)}(t) = b_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}} + b_{u,t} \tag{10}$$

model	static	mov	linear	spline	linear+	spline+
RMSE	.9799	.9771	.9731	.9714	.9605	.9603

Table 1: Comparing baseline predictors capturing main movie and user effects. As temporal modeling becomes more accurate, prediction accuracy improves (lowering RMSE).

A baseline predictor on its own cannot yield personalized recommendations, as it misses all interactions between users and items. In a sense, it is capturing the portion of the data that is less relevant for establishing recommendations and in doing so enables deriving accurate recommendations. Nonetheless, to better assess the relative merits of the various choices of time-dependent user-bias, we will compare their accuracy as standalone predictors. In order to learn the involved parameters we minimize the associated regularized squared error by using stochastic gradient descent. For example, in our actual implementation we adopt rule (9) for modeling the drifting user bias, thus arriving at the baseline predictor:

$$b_{ui}(t) = \mu + b_u + \alpha_u \cdot \text{dev}_u(t) + b_{u,t} + b_i + b_{i,\text{Bin}(t)} \tag{11}$$

To learn the involved parameters, $b_u, \alpha_u, b_{u,t}, b_i$ and $b_{i,\text{Bin}(t)}$, one should solve:

$$\min \sum_{(u,i,t)\in\mathcal{K}} \left(r_{ui}(t) - \mu - b_u - \alpha_u \text{dev}_u(t) - b_{u,t} - b_i - b_{i,\text{Bin}(t)}\right)^2 + \lambda(b_u^2 + \alpha_u^2 + b_{u,t}^2 + b_i^2 + b_{i,\text{Bin}(t)}^2)$$

Here, the first term strives to construct parameters that fit the given ratings. The regularization term, $\lambda(b_u^2 + \ldots)$, avoids overfitting by penalizing the magnitudes of the parameters, assuming a neutral 0 prior. Learning is done by a stochastic gradient descent algorithm running 20–30 iterations, with $\lambda = 0.01$.

Table 1 compares the ability of various suggested baseline predictors to explain signal in the data. As usual, the amount of captured signal is measured by the root mean squared error on the test set. As a reminder, test cases come later in time than the training cases for the same user. We code the predictors as follows:

- *static* no temporal effects: $b_{ui}(t) = \mu + b_u + b_i$.
- *mov* accounting only to movie-related temporal effects: $b_{ui}(t) = \mu + b_u + b_i + b_{i,\text{Bin}(t)}$.
- *linear* linear modeling of user biases: $b_{ui}(t) = \mu + b_u + \alpha_u \cdot \text{dev}_u(t) + b_i + b_{i,\text{Bin}(t)}$.
- *spline* spline modeling of user biases: $b_{ui}(t) = \mu + b_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}} + b_i + b_{i,\text{Bin}(t)}$.
- *linear+* linear modeling of user biases and single day effect: $b_{ui}(t) = \mu + b_u + \alpha_u \cdot \text{dev}_u(t) + b_{u,t} + b_i + b_{i,\text{Bin}(t)}$.
- *spline+* spline modeling of user biases and single day effect: $b_{ui}(t) = \mu + b_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-d_l|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}} + b_{u,t} + b_i + b_{i,\text{Bin}(t)}$.

The table shows that while temporal movie effects reside in the data (lowering RMSE from 0.9799 to 0.9771), the drift in user biases is much more influential. The additional flexibility of splines at modeling user effects leads to better accuracy compared to a linear model. However, sudden changes in user biases, which are captured by the per-day parameters, are most significant. Indeed, when including those changes, the difference between linear modeling ("linear+") and spline modeling ("spline+") virtually vanishes.

Beyond the temporal effects described so far, one can use the same methodology to capture more effects. A prime example is capturing periodic effects. For example, some products can be more popular in specific seasons or near certain holidays. Similarly,

different types of television or radio shows are popular throughout different segments of the day (known as "dayparting"). Periodic effects can be found also on the user side. As an example, a user may have different attitudes or buying patterns during the weekend compared to the working week. A way to model such periodic effects is to dedicate a parameter for the combinations of time periods with items or users. This way, the item bias of (6), becomes:

$$b_i(t) = b_i + b_{i,\mathrm{Bin}(t)} + b_{i,\mathrm{period}(t)}$$

E.g., if we try to capture the change of item bias with the season of the year, then $\mathrm{period}(t) \in \{\mathrm{fall}, \mathrm{winter}, \mathrm{spring}, \mathrm{summer}\}$. Similarly, recurring user effects are modeled by modifying (9) to be:

$$b_u(t) = b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{u,t} + b_{u,\mathrm{period}(t)}$$

E.g., if we try to model a day-of-week user effect, then $\mathrm{period}(t) \in \{\mathrm{Sun}, \mathrm{Mon}, \mathrm{Tue}, \mathrm{Wed}, \mathrm{Thu}, \mathrm{Fri}, \mathrm{Sat}\}$. We could not find periodic effects with a significant predictive power within the movie-rating dataset, thus our reported results do not include those.

Another temporal effect within the scope of basic predictors is related to the changing scale of user ratings. While $b_i(t)$ is a user-independent measure for the merit of item i at time t, users tend to respond to such a measure differently. For example, different users employ different rating scales, and a single user can change his rating scale over time. Accordingly, the raw value of the movie bias is not completely user-independent. To address this, we add a time-dependent scaling feature to the baseline predictors, denoted by $c_u(t)$. Thus, the baseline predictor (11) becomes:

$$b_{ui}(t) = \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{u,t} + (b_i + b_{i,\mathrm{Bin}(t)}) \cdot c_u(t) \quad (12)$$

All discussed ways to implement $b_u(t)$ would be valid for implementing $c_u(t)$ as well. We chose to dedicate a separate parameter per day, resulting in: $c_u(t) = c_u + c_{u,t}$. As usual, c_u is the stable part of $c_u(t)$, whereas $c_{u,t}$ represents day-specific variability. Adding the multiplicative factor $c_u(t)$ to the baseline predictor lowers RMSE to 0.9555. Interestingly, this basic model, which captures just main effects disregarding user-item interactions, can explain almost as much of the data variability as the commercial Netflix Cinematch recommender system, whose published RMSE on the same test set is 0.9514 [4].

4.3 Time changing factor model

In the previous subsection we discussed the way time affects baseline predictors. However, as hinted earlier, temporal dynamics go beyond this, they also affect user preferences and thereby the interaction between users and items. Users change their preferences over time. For example, a fan of the "psychological thrillers" genre may become a fan of "crime dramas" a year later. Similarly, humans change their perception on certain actors and directors. This effect is modeled by taking the user factors (the vector p_u) as a function of time. Once again, we need to model those changes at the very fine level of a daily basis, while facing the built-in scarcity of user ratings. In fact, these temporal effects are the hardest to capture, because preferences are not as pronounced as main effects (user-biases), but are split over many factors.

The same way we treat user biases we also treat each component of the user preferences $p_u(t)^T = (p_{u1}(t), \ldots, p_{uf}(t))$. In our application, we have found modeling after (9) effective leading to:

$$p_{uk}(t) = p_{uk} + \alpha_{uk} \cdot \mathrm{dev}_u(t) + p_{uk,t} \quad k = 1, \ldots, f \quad (13)$$

Here p_{uk} captures the stationary portion of the factor, $\alpha_{uk} \cdot \mathrm{dev}_u(t)$ approximates a possible portion that changes linearly over time, and $p_{uk,t}$ absorbs the very local, day-specific variability.

Model	f=10	f=20	f=50	f=100	f=200
SVD	.9140	.9074	.9046	.9025	.9009
SVD++	.9131	.9032	.8952	.8924	.8911
timeSVD++	.8971	.8891	.8824	.8805	.8799

Table 2: Comparison of three factor models: prediction accuracy is measured by RMSE (lower is better) for varying factor dimensionality (f). For all models accuracy improves with growing number of dimensions. Most significant accuracy gains are achieved by addressing the temporal dynamics in the data through the timeSVD++ model.

At this point, we can tie all pieces together and extend the SVD++ factor model by incorporating the time changing parameters. This leads to a model, which will be denoted as *timeSVD++*, where the prediction rule is as follows:

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T \left(p_u(t) + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} y_j \right) \quad (14)$$

The exact definitions of the time drifting parameters $b_i(t), b_u(t)$ and $p_u(t)$ were given in (6),(9) and (13). Learning is performed by minimizing the associated squared error function on the training set using a regularized stochastic gradient descent algorithm. The procedure is analogous to the one involving the original SVD++ algorithm [8]; details are omitted here for brevity. Time complexity per iteration is still linear with the input size, while wall clock running time is approximately doubled compared to SVD++, due to the extra overhead required for updating the temporal parameters. Importantly, convergence rate was not affected by the temporal parameterization, and the process converges in around 30 iterations.

Addressing temporal dynamics leads to significant accuracy gains within the movie rating dataset, when considering past RMSE improvements on the dataset. In Table 2 we compare results of three algorithms. First is the plain matrix factorization algorithm as per (3), denoted by SVD. Second, is the SVD++ method (4), which was considered as a significant improvement over SVD by incorporating also a kind of implicit feedback [8]. Finally is the newly proposed timeSVD++, which accounts for temporal effects as in (14). The three methods are compared over a range of factorization dimensions (f). All methods benefit from a growing number of factor dimensions, what enables them to better express complex movie-user interactions. Notice that the improvement delivered by timeSVD++ over SVD++ is consistently more significant than the improvement SVD++ achieves over SVD. In fact, we are not aware of any single algorithm in the literature that could deliver such accuracy. We attribute this to the importance of properly addressing temporal effects. What further demonstrates the importance of capturing temporal dynamics is the fact that a timeSVD++ model of dimension 10 is already more accurate than an SVD model of dimension 200. Similarly, a timeSVD++ model of dimension 20 is enough to outperform an SVD++ model of dimension 200.

5. TEMPORAL DYNAMICS AT NEIGHBORHOOD MODELS

The most common approach to CF is based on neighborhood models. While typically less accurate than their factorization counterparts, neighborhood methods enjoy popularity thanks to some of their merits, such as explaining the reasoning behind computed recommendations, and seamlessly accounting for new entered ratings.

Recently, we suggested an item-item model based on global optimization [8], which will enable us here to capture time dynamics in a principled manner. The static model, without temporal dynamics,

is centered on the following prediction rule:

$$\hat{r}_{ui} = \mu + b_i + b_u + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + c_{ij} \quad (15)$$

Here, the item-item weights w_{ij} and c_{ij} represent the adjustments we need to make to the predicted rating of item i, given a known rating of item j. It was proven greatly beneficial to use two sets of item-item weights: one (the w_{ij}s) is related to the values of the ratings, and the other disregards the rating value, considering only which items were rated (the c_{ij}s). These weights are automatically learnt from the data together with the biases b_i and b_u. The constants b_{uj} are precomputed according to (2). Recall that $\mathrm{R}(u)$ is the set of items rated by user u.

When adapting rule (15) to address temporal dynamics, two components should be considered separately. First, is the baseline predictor portion, $\mu + b_i + b_u$, which explains most of the observed signal. Second, is the part that captures the more informative signal, dealing with user-item interaction $|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + c_{ij}$. As for the baseline part, nothing changes from the factor model, and we replace it with $\mu + b_i(t) + b_u(t)$, according to (6) and (9). However, capturing temporal dynamics within the interaction part requires a different strategy.

Item-item weights (w_{ij} and c_{ij}) reflect inherent item characteristics and are not expected to drift over time. Learning process should make sure that they capture unbiased long term values, without being too affected from drifting aspects. Indeed, the time changing nature of the data can mask much of the longer term item-item relationships if not treated adequately. For instance, a user rating both items i and j high in a short time period, is a good indicator for relating them, thereby pushing higher the value of w_{ij}. On the other hand, if those two ratings are given five years apart, while the user's taste (if not her identity) could considerably change, this is less of an evidence of any relation between the items. On top of this, we would argue that those considerations are pretty much user-dependent – some users are more consistent than others and allow relating their longer term actions.

Our goal here is to distill accurate values for the item-item weights, despite the interfering temporal effects. First we need to parameterize the decaying relations between two items rated by user u. We adopt exponential decay formed by the function $e^{-\beta_u \cdot \Delta t}$, where $\beta_u > 0$ controls the user specific decay rate and should be be learnt from the data. We also experimented with other decay forms, such as a power law decay $\Delta t^{-\beta_u}$, which resulted in slightly inferior results. This leads to the prediction rule:

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{(j,t_j) \in \mathrm{R}(u)} e^{-\beta_u \cdot |t - t_j|} ((r_{uj} - b_{uj}) w_{ij} + c_{ij}) \quad (16)$$

Here, in a slight abuse of notation, we assume that the set $\mathrm{R}(u)$ contains not only the items rated by u, but also the time of those ratings. The involved parameters, $b_i(t) = b_i + b_{i,\mathrm{Bin}(t)}$, $b_u(t) = b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{u,t}$, β_u, w_{ij} and c_{ij}, are learnt by minimizing the associated regularized squared error:

$$\sum_{(u,i,t) \in \mathcal{K}} (r_{ui}(t) - \mu - b_i - b_{i,\mathrm{Bin}(t)} - b_u - \alpha_u \mathrm{dev}_u(t) - b_{u,t} - |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{(j,t_j) \in \mathrm{R}(u)} e^{-\beta_u \cdot |t - t_j|} ((r_{uj} - b_{uj}) w_{ij} + c_{ij}))^2 + \lambda (b_i^2 + b_{i,\mathrm{Bin}(t)}^2 + b_u^2 + \alpha_u^2 + b_{u,t}^2 + w_{ij}^2 + c_{ij}^2) \quad (17)$$

Minimization is performed by stochastic gradient descent. We run the process for 25 iterations, with $\lambda = 0.002$, and step size (learning rate) of 0.005. An exception is the update of the exponent β_u, where we are using a much smaller step size of 10^{-7}. Training time complexity is the same as the original algorithm, which is: $O(\sum_u |\mathrm{R}(u)|^2)$. One can tradeoff complexity with accuracy by sparsifying the set of item-item relations as explained in [8].

Like in the factor case, properly considering temporal dynamics improves the accuracy of the neighborhood model within the movie ratings dataset. The RMSE decreases from 0.9002 [8] to 0.8885. To our best knowledge, this is significantly better than previously known results by neighborhood methods. To put this in some perspective, this result is even better than those reported [1, 10, 21] by using hybrid approaches such as applying a neighborhood approach on residuals of other algorithms. A lesson is that addressing temporal dynamics in the data can have a more significant impact on accuracy than designing more complex learning algorithms.

We would like to highlight an interesting point related to the basic methodology described in Sec. 3. Let u be a user whose preferences are quickly drifting (β_u is large). Hence, old ratings by u should not be very influential on his status at the current time - t. One could be tempted to decay the weight of u's older ratings, leading to "instance weighting" through a cost function like:

$$\sum_{(u,i,t_{ui}) \in \mathcal{K}} e^{-\beta_u \cdot |t - t_{ui}|} (r_{ui} - \mu - b_i - b_{i,\mathrm{Bin}(t_{ui})} - b_u - \alpha_u \mathrm{dev}_u(t_{ui}) - b_{u,t_{ui}} - |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{(j,t_{uj}) \in \mathrm{R}(u)} ((r_{uj} - b_{uj}) w_{ij} + c_{ij}))^2 + \lambda(\cdots)$$

Such a function is focused at the *current* state of the user (at time t), while de-emphasizing past actions. We would argue against this choice, and opt for equally weighting the prediction error at all past ratings as in (17), thereby modeling *all* past user behavior. This allows us to exploit the signal at each of the past ratings, a signal that is extracted as item-item weights. Learning those weights would equally benefit from all ratings by a user. In other words, we can deduce that two items are related if users rated them similarly within a short timeframe, even if this happened long ago.

6. AN EXPLORATORY STUDY

In Fig. 1 we showed two strong temporal effects within the Netflix movie-rating data. First effect exhibits a sudden rise in the average movie rating beginning around 1500 days into the dataset, corresponding to early 2004. The second effect shows that people tend to give higher ratings as movies become older (movie age is measured by number of days since its first rating in the dataset). The patterns that our temporal models capture may help in explaining what created those two global temporal effects.

Let us start with the first effect. We can come up with several hypotheses on what caused the sudden increase of rating scores.

1. Since 2004 people are matched with movies better suited for them leading to higher entered ratings. This may result by technical improvements in Netflix recommendation technology (Cinematch) and/or GUI improvements making people more aware of movies they like. Notice that an improvement in Cinematch's effectiveness can have a significant impact on which movies members rent and subsequently rate, as Cinematch suggestions drive 60% of Netflix's rentals [20].
2. Since 2004 people are biased to give higher ratings in general. A possible cause is a hypothetical change of the labels associated with the star scores. While at the present, stars reflect subjective feelings on the movies (e.g., 5 stars="loved it", 4 stars="really liked it"), in the past they might have used to denote a more objective quality (e.g., 5 stars="superb movie"), setting a higher bar for a 5 star rating.

3. The vast majority of the users in the Netflix dataset gave their first rating no earlier than 2004. It is possible that unlike early adopters those newer users have a less refined taste and shifted the overall rating average higher.

A straightforward analysis rejects the third hypothesis – even when concentrating on earlier customers, e.g., those who have rated earlier than 2003, we can find a strong shift in rating scale since early 2004. As for the two other hypotheses, we use our models for examining them. The first hypothesis corresponds to the interaction part of the models (e.g., $q_i^T \left(p_u(t) + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} y_j \right)$ for the timeSVD++ model), which measures how well user and movie characteristics match together. On the other hand, the second hypothesis, deals with general biases that have nothing to do with the matching of users to movies. Thus, it corresponds to the baseline predictor portion of them model ($\mu + b_i(t) + b_u(t)$). In order to analyze this, we modeled the data using the timeSVD++ model (of dimensionality $f = 50$). While the full model could accurately regenerate the shifts in rating values over time, more interesting to us is to separate the model predictions into baseline and interaction components, and examine how each of them evolve over time. Results are shown in Fig. 2. We observe that since early 2004 (1500 days into the data), the score due to interaction between users and movies steadily rises, indicating that users are increasingly rating movies that are more suitable for their own taste. This supports the first hypothesis of an ongoing improvement in the way Netflix matches users to movies beginning at early 2004 and continuing since then. Apparently, this could be expected knowing the large effort that company invests in improving their recommender system [4]. At the same time, the various biases, captured by the baseline predictors, exhibit a onetime phase transition around the 1500 days time point. While shallower than the change in the interaction part, the jump is clear and supports the more surprising second hypothesis. This hints that beyond a constant improvement in matching people to movies they like, something else happened in early 2004 causing an overall shift in rating scale. Uncovering this may require extra information on the related circumstances.

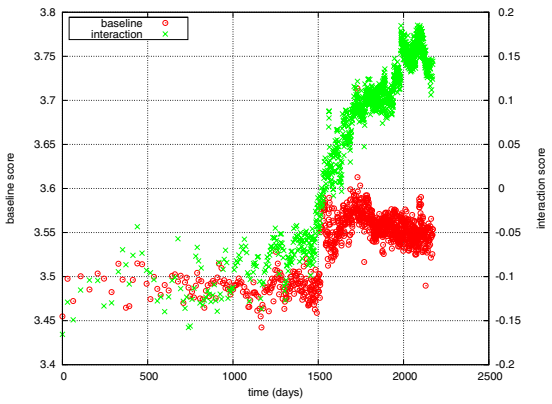

Figure 2: Tracking the change of the two components of modeled movie-ratings over time. First component ("baseline") represents rating behavior influenced by exterior considerations, while the other component ("interaction") captures the rating behavior that is explained by the match between users and movies.

Now we move to the second temporal effect. We would like to suggest two hypotheses to why ratings rise as movies become older.

1. Older movies are getting rated by users better matching them. This might indicate that people watch (and then rate) a new movie even if it is less appropriate for them, but will watch an older movie only after a more careful selection process. Such an improved match between users and movies can be captured by the fact that the interaction part of the model is rising with movies' age.
2. Older movies are just inherently better than newer ones. This would be captured by the baseline part of the model.

Once again we split the modeled behavior into two parts – the interaction part measuring the match between users and movies and the baseline part capturing other effects. In Fig. 3 we track how those two components change over time. Much like the original raw data (in Fig. 1), the interaction part shows a steady increase virtually throughout the whole movie age range at a close to linear pace. On the other hand, the baseline portion is increasing only between days 1000 and 1500, while being captured within a narrow range elsewhere. Since we measure movie age by number of days since first rating, as movies become older they are more exposed to the aforementioned early 2004 rating jump effect. In particular, all movies older than 1500 days must be fully susceptible to this effect. Thus, it is possible that the increase in baseline values between days 1000 and 1500 reflects such a side effect. To wipe out this interfering effect we concentrate only on ratings to movies aged 1500 days or older. This leaves us with about 44% of the points (~44 million rating instances) and makes the picture much clearer. While the raw ratings as well the interaction part continue to steadily increase beyond day 1500, the baseline portion does not increase after that day. We view this as an indication that the first hypothesis is closer to the truth than the second one.

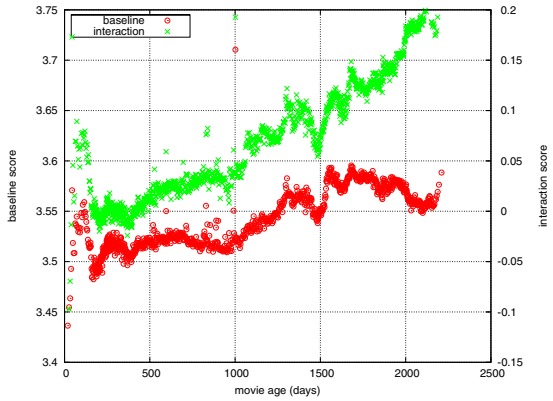

Figure 3: Tracking the change of the two components of modeled ratings against age of rated movie. We observe a consistent improvement in the match between users and movies as movie age rises (captured by the "interaction" component).

7. RELATED WORKS

In the past few years, much research was devoted to the Netflix dataset. Many works were published in the two KDD workshops dedicated to that dataset [3, 23]. Other notable works include [8, 13, 19]. Best reported results were obtained by integrating the factorization and neighborhood models. Results reported in this paper by pure factorization are more accurate, in a sense showing that addressing temporal dynamics is not less important than algorithmic sophistication created by integration of two different models.

Despite the high impact of temporal effects on user preferences, the subject attracted a quite negligible attention in the recommender literature. Notable discussions of temporal effects include Ding and Li [6], who suggested a time weighting scheme for a similarity-based collaborative filtering approach. At the prediction stage, sim-

ilarities to previously rated items are decayed as time difference increases. The decay rate is both user-dependent and item-dependent. Sugiyama et al. [17] proposed a personalized web search engine, where they let the user profile evolve over time. There, they distinguish between aspects of user behavior computed over a fixed time decay window, and ephemeral aspects captured within the current day. In a prior work, we suggested an incremental modeling of global effects [1], which include some baseline time effects. This scheme was later enhanced [11, 21].

Our work is within the topic of tracking and modeling concept drift, which has gathered much interest in the data mining community. Early works in the field (e.g. [15, 25]) used techniques like adjusted and decayed weights of past instances or using a sliding time window. Another approach popular in newer publications (e.g. [7, 16, 24]) is based on maintaining an ensemble of models capable of capturing various states of the data. As explained in Sec. 3, the problem of tracking user preferences, especially in a collaborative filtering scenario, requires different approaches.

8. CONCLUSIONS

Tracking the temporal dynamics of customer preferences to products raises unique challenges. Each user and product potentially goes through a distinct series of changes in their characteristics. Moreover, we often need to model all those changes within a single model thereby interconnecting users (or, products) to each other to identify communal patterns of behavior. A mere decay of older instances or usage of multiple separate models lose too much signal, thus degrading prediction accuracy. The solution we adopted is to model the temporal dynamics along the whole time period, allowing us to intelligently separate transient factors from lasting ones. We applied this methodology with two leading recommender techniques. In a factorization model, we modeled the way user and product characteristics change over time, in order to distill longer term trends from noisy patterns. In an item-item neighborhood model, we showed how the more fundamental relations among items can be revealed by learning how influence between two items rated by a user decays over time. In both factorization and neighborhood models, the inclusion of temporal dynamics proved very useful in improving quality of predictions, more than various algorithmic enhancements. This led to the best results published so far on a widely analyzed movie rating dataset.

9. REFERENCES

[1] 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.

[2] 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 (KDD'07)*, pp. 95–104, 2007.

[3] J. Bennett, C. Elkan, B. Liu, P. Smyth and D. Tikk (eds.). *KDD Cup and Workshop in conjunction with KDD'07*, 2007.

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

[5] J. Canny. Collaborative filtering with privacy via factor analysis. *Proc. 25th ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR'02)*, pp. 238–245, 2002.

[6] Y. Ding and X. Li. Time weight collaborative filtering. *Proc. 14th ACM international conference on Information and knowledge management (CIKM'04)*, pp. 485–492, 2004.

[7] J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. *Proc. IEEE Conf. on Data Mining (ICDM'03)*, pp. 123–130, 2003.

[8] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'08)*, pp. 426–434, 2008.

[9] G. Linden, B. Smith and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7:76–80, 2003.

[10] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. *Proc. KDD Cup and Workshop*, 2007.

[11] G. Potter. Putting the collaborator back into collaborative filtering. *KDD'08 Workshop on Large Scale Recommenders Systems and the Netflix Prize*, 2008.

[12] P. Pu, D. G. Bridge, B. Mobasher and F. Ricci (eds.). *Proc. 2008 ACM Conference on Recommender Systems*, 2008.

[13] 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.

[14] 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.

[15] J. Schlimmer and R. Granger. Beyond incremental processing: Tracking concept drift. *Proc. 5th National Conference on Artificial Intelligence*, pp. 502-507, 1986.

[16] W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. *Proc. 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, pp. 377-382, 2001.

[17] K. Sugiyama, K. Hatano and M. Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. *Proc. 13th international conference on World Wide Web (WWW'04)*, pp. 675–684, 2004.

[18] G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk. Major components of the gravity recommendation aystem. *SIGKDD Explorations*, 9:80–84, 2007.

[19] G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk. Matrix factorization and neighbor based algorithms for the Netflix Prize problem. *Proc. 2008 ACM Conference on Recommender Systems (RECSYS'08)*, pp. 267–274, 2008.

[20] C. Thompson. If you liked this, you're sure to love that. *The New York Times*, Nov 21, 2008.

[21] A. Toscher, M. Jahrer and R. Legenstein. Improved neighborhood-based algorithms for large-scale recommender systems. *KDD'08 Workshop on Large Scale Recommenders Systems and the Netflix Prize*, 2008.

[22] A. Tsymbal. The problem of concept drift: Definitions and related work. Technical Report TCD-CS-2004-15, Trinity College Dublin, 2004.

[23] A. Tuzhilin, Y. Koren, J. Bennett, C. Elkan and D. Lemire (eds.). *Workshop on large scale recommender systems and the Netflix Prize in conjunction with KDD'08*, 2008.

[24] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept drifting data streams using ensemble classifiers. *Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pp. 226-235, 2003.

[25] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.