DOI:10.1145/1721654.1721677

Collaborative Filtering with Temporal Dynamics

By Yehuda Koren

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 is essential for designing recommender systems or general customer preference models. However, this raises unique challenges. Within the ecosystem 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 many signals when discarding data instances. A more sensitive approach is required, which can make better distinctions between transient effects and long-term patterns. We show how to model the time changing behavior throughout the life span of the data. Such a model 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 underlying the Netflix Prize contest. Results are encouraging and better than those previously reported on this dataset. In particular, methods described in this paper play a significant role in the solution that won the Netflix contest.

1. INTRODUCTION

Modeling time drifting data is a central problem in data mining. Often, data is changing over time, and models 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., Schlimmer and Granger, and Widmer and Kubat.[15, 20]

Temporal changes in customer preferences bring unique modeling 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. Such 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 toward 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. These considerations require 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 are 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. The change may stem from multiple factors; some of these factors 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. For example, in a system

A previous version of this paper appeared in the *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2009), 447–456.

where users provide star ratings to products, a user that used to indicate a neutral preference by a "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 distinguish 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.

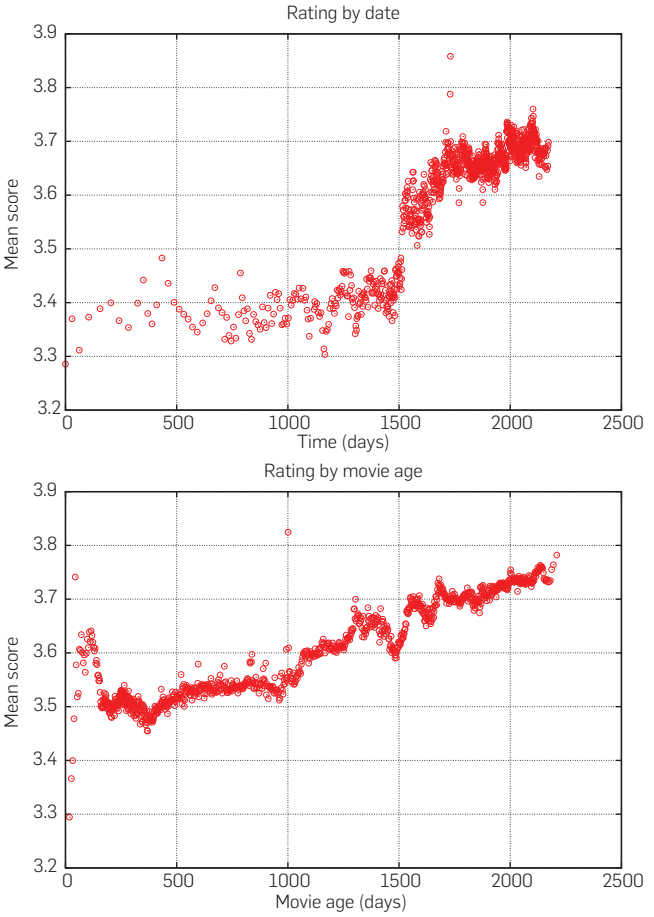
All these patterns and the likes should have made temporal modeling a predominant factor in building recommender systems. Nonetheless, with very few exceptions (e.g., Ding and Li, and Sugiyama et al.[4,16]), the recommenders' literature does not address temporal changes in user behavior. Perhaps this is because user behavior is composed of many different concept drifts, acting in different timeframes and directions, thus making common methodologies for dealing with concept drift and temporal data less successful. We show that capturing time drifting patterns in user behavior is essential for improving accuracy of recommenders. Our findings also give 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.[3] 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 Figure 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 Koren,[8] we shed some light on the origins of these effects.

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 Section 3, our principles for addressing time changing user preferences are evolved. Those principles are then incorporated, in quite different ways, into two leading recommender techniques: factor modeling (Section 4) and item–item neighborhood modeling (Section 5).

Figure 1. Two temporal effects emerging within the Netflix movie-rating dataset. *Top*: the average movie-rating made a sudden jump in early 2004 (1,500 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.



2. PRELIMINARIES

2.1. Notation

We are given ratings for m users (aka customers) and n items (aka products). We reserve special indexing letters to distinguish users from items: for users u, v, and for items i, j. A rating r_{ui} indicates the preference by user u of item i, where high values mean stronger preference. For example, values can be integers ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. We distinguish predicted ratings from known ones, by using the notation $\hat{r}_{ui}$ for the predicted value of r_{ui}.

The scalar t_{ui} denotes the time of rating r_{ui}. One can use different time units, based on what is appropriate for the

application at hand. For example, when time is measured in days, then t_{ui} counts the number of days elapsed since some early time point. 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 $\mathcal{K} = \{(u, i)|r_{ui} \text{ is known}\}$, which is known as the *training set*.

Models for the rating data are learned by fitting the previously observed ratings. However, our goal is to generalize those in a way that allows us to predict future, unknown ratings. Thus, caution should be exercised to avoid overfitting the observed data. We achieve this by using a technique called *regularization*. Regularization restricts the complexity of the models, thereby preventing them from being too specialized to the observed data. We employ L2-regularization, which penalizes the magnitude of the learned parameters. Extent of regularization is controlled by constants which are denoted as: $\lambda_1, \lambda_2, \ldots$

2.2. The Netflix data

We evaluated our algorithms on a movie-rating dataset of more than 100 million date-stamped ratings performed by about 480,000 anonymous Netflix customers on 17,770 movies between 31 December 1999 and 31 December 2005.[3] Ratings are integers ranging between 1 and 5. On average, a movie receives 5,600 ratings, while a user rates 208 movies, with substantial variation around each of these averages. 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 holdout 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 holdout 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) $\sqrt{\sum_{(u,i)\in TestSet}(r_{ui}-\hat{r}_{ui})^2/|TestSet|}$.

The Netflix data is part of the Netflix Prize contest, with the target of improving the accuracy of Netflix movie recommendations by 10%. The benchmark is Netflix's proprietary system. Cinematch, which achieved an RMSE of 0.9514 on the test set. The grand prize was awarded to a team that managed to drive this RMSE to 0.8554 after almost 3 years of extensive efforts. Achievable RMSE values on the test set lie in a quite compressed range, as evident by the difficulty to win the grand prize. Nonetheless, there is evidence that small improvements in RMSE terms can have a significant impact on the quality of the top few presented recommendations.[7] The algorithms described in this work played a central role in reaching the grand prize.

2.3. Collaborative filtering

Recommender systems are often based on *collaborative filtering* (CF), a term coined by the developers of the first recommender system—Tapestry.[5] This technique relies only on past user behavior—e.g., their previous transactions or product ratings—without requiring the creation of explicit profiles. CF analyzes relationships between users and interdependencies among products, in order to identify new user–item associations.

A major appeal of CF is that it is domain-free and avoids 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,[10] TiVo,[1] and Netflix.

The two primary areas of CF are the *neighborhood methods* and *latent factor models*. The neighborhood methods are centered on computing the relationships between items or, alternatively, between users. The item-oriented approach evaluates the preference of a user to an item based on ratings of "neighboring" items by the same user. A product's neighbors are other products that tend to be scored similarly when rated by the same user. For example, consider the movie "Saving Private Ryan." Its neighbors might include other war movies, Spielberg movies, and Tom Hanks movies, among others. To predict a particular user's rating for "Saving Private Ryan," we would look for the movie's nearest neighbors that were actually rated by that user. A dual to the item-oriented approach is user-oriented approach, which identifies like-minded users who can complement each other's missing ratings.

Latent factor models comprise an alternative approach that tries to explain the ratings by characterizing both items and users on, say, 20–200 factors inferred from the pattern of ratings. For movies, factors discovered by the decomposition 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. One of the most successful realizations of latent factor models is based on *matrix factorization*; see, e.g., Koren et al.[9]

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."[6] 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[19] 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. Equal significance 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, underweighting 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 finding 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 lose too many signals 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). Such modeling 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 combination 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. Extrapolation 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. The result, indeed, helps in predicting *future* behavior.

Now we turn to how these desirable principles are incorporated into two leading approaches to CF—matrix factorization and neighborhood methods.

4. TIME-AWARE FACTOR MODEL

4.1. The anatomy of a factor model

Matrix factorization is a well-recognized approach to CF.[9, 11, 17] 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.

In its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from patterns of item ratings. High correspondence between item and user factors leads to recommendation of an item to a user. More specifically, both users and items are mapped 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

$$\hat{r}_{ui} = q_i^T p_u = \left(\sum_{k=1}^{f} q_i[k] \cdot p_u[k] \right). \quad (1)$$

The major challenge is computing the mapping of each item and user to factor vectors $q_i, p_u \in \mathbb{R}^f$. After this mapping is accomplished, we can easily compute the ratings a user will give to any item by using Equation 1.

Such a model is closely related to singular value decomposition (SVD), which is a well-established technique for identifying latent semantic factors in the information retrieval. Applying SVD in the CF domain would require factoring the user–item rating matrix. Such a factorization raises difficulties due to the high portion of missing values, due to the sparseness in the user–item ratings matrix. 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[13] 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 (e.g., Koren, Paterek, and Takacs et al.[7, 11, 17]) suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model. In order to learn the factor vectors (p_u and q_i), we minimize the regularized squared error on the set of known ratings:

$$\min_{q^*, p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) . \quad (2)$$

Minimization is typically performed by stochastic gradient descent.

Model (1) tries to capture the interactions between users and items that produce the different rating values. However, much of the observed variation in rating values is due to effects associated with either users or items, independently of their interaction, which are known as biases. A prime example is that typical CF data exhibits large systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. After all, some products are widely received as better (or worse) than others.

Thus, it would be unwise to explain the full rating value by an interaction of the form $q_i^T p_u$. Instead, we will try to identify the portion of these values that can be explained by individual user or item effects (biases). The separation of interaction and biases will allow us to subject only the true interaction portion of the data to factor modeling.

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:

$$b_{ui} = \mu + b_u + b_i. \quad (3)$$

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

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u. \quad (4)$$

Here, the observed rating is separated to its four components: global average, item-bias, user-bias, and user–item interaction. The separation allows each component to explain only the part of signal relevant to it. Learning is done analogously to before, by minimizing the squared error function

$$\min_{q^*, p^*, b^*} \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2) . \quad (5)$$

Schemes along these lines were described in, e.g., Koren and Paterek.[7, 11]

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; and (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. The first addresses the fact that an item's popularity may change 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 treating the item bias b_i as a function of time. The second major temporal effect allows users to 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." This may reflect several factors including a natural drift in a user's rating scale, the fact that ratings are given in relevance to other ratings that were given recently and also the fact that the identity of the rater within a household can change over time. Hence, in our models we take the parameter b_u as a function of time. This induces a template for a time sensitive baseline predictor for u's rating of i at day t_{ui}:

$$b_{ui} = \mu + b_u(t_{ui}) + b_i(t_{ui}). \quad (6)$$

Here, $b_u(\cdot)$ and $b_i(\cdot)$ 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. Our choice in the context of the movie-rating dataset 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 with a lower resolution that suffices for capturing item-related time effects.

We start with our choice of time-changing item biases $b_i(t)$. We found it adequate to split the item biases into time-based bins, using a constant item bias for each time period. The decision of how to split the timeline into bins should

balance the desire to achieve finer resolution (hence, smaller bins) with the need for 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 10 consecutive weeks of data, leading to 30 bins spanning all days in the dataset. A day t is associated with an integer 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:

$$b_i(t) = b_i + b_{i,\mathrm{Bin}(t)}. \quad (7)$$

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 enough ratings per user to produce reliable estimates for isolated bins. Different functional forms can be considered for parameterizing temporal user behavior, with varying complexity and accuracy.

One simple modeling choice uses a linear function to capture a possible gradual drift of user-bias. 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

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

Here $|t - t_u|$ measures the 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

$$b_u^{(1)}(t) = b_u + \alpha_u \cdot \mathrm{dev}_u(t). \quad (8)$$

A more flexible spline-based rule is described in Koren.[8]

A smooth function for modeling the user-bias meshes 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 need not span more than a single day. The effect 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 multiperson 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_{ut}. Notice that in some applications the basic primitive time unit to work with can be shorter or longer than a day.

In the Netflix movie-rating data, a user rates on 40 different days on average. Thus, working with b_{ut} requires, on average, 40 parameters to describe each user-bias. It is expected that b_{ut} is inadequate as a stand-alone 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 (8) becomes

$$b_u^{(3)}(t) = b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{ut}. \quad (9)$$

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. Nonetheless, to better assess the relative merits of the various choices of time-dependent user-bias, we compare their accuracy as stand-alone 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

$$b_{ui} = \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t_{ui}) + b_{u,t_{ui}} + b_i + b_{i,\mathrm{Bin}(t_{ui})}. \quad (10)$$

To learn the involved parameters, b_u, α_u, b_{ut}, b_i, and $b_{i,\mathrm{Bin}(t)}$, one should solve

$$\min \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_u - \alpha_u \mathrm{dev}_u(t_{ui}) - b_{u,t_{ui}} - b_i - b_{i,\mathrm{Bin}(t_{ui})})^2 + \lambda_7 (b_u^2 + \alpha_u^2 + b_{u,t_{ui}}^2 + b_i^2 + b_{i,\mathrm{Bin}(t_{ui})}^2).$$

Here, the first term strives to construct parameters that fit the given ratings. The regularization term, $\lambda_7(b_u^2 + \ldots)$, 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 $\lambda_7 = 0.01$.

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 RMSE on the test set. As a reminder, test cases come later in time than the training cases for the same user, so predictions often involve extrapolation in terms of time. We code the predictors as follows:

- *static*, no temporal effects: $b_{ui} = \mu + b_u + b_i$,
- *mov*, accounting only for movie-related temporal effects: $b_{ui} = \mu + b_u + b_i + b_{i,\mathrm{Bin}(t_{ui})}$,
- *linear*, linear modeling of user-biases: $b_{ui} = \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t_{ui}) + b_i + b_{i,\mathrm{Bin}(t_{ui})}$, and
- *linear*$^+$, linear modeling of user-biases and single day effect: $b_{ui} = \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t_{ui}) + b_{u,t_{ui}} + b_i + b_{i,\mathrm{Bin}(t_{ui})}$.

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. In particular, sudden changes in user-biases, which are captured by the per-day parameters, are most significant.

Beyond the temporal effects described so far, one can

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

Model	Static	Mov	Linear	Linear$^+$
RMSE	0.9799	0.9771	0.9731	0.9605

use the same methodology to capture more effects. A prime example is capturing periodic effects. For example, some products may 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 (7) becomes

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

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

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

However, we have not found 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 (10) becomes

$$b_{ui} = \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t_{ui}) + b_{u,t_{ui}} + (b_i + b_{i,\mathrm{Bin}(t_{ui})}) \cdot c_u(t_{ui}). \quad (11)$$

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: $c_u(t) = c_u + c_{ut}$. As usual, c_u is the stable part of $c_u(t)$, whereas c_{ut} 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.[3]

4.3. Time changing factor model

In Section 4.2 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.

We modeled each component of the user preferences $p_u(t)^T = (p_u(t)[1], p_u(t)[2], \ldots, p_u(t)[f])$ in the same way that we treated user-biases. Within the movie-rating dataset, we have found modeling after (9) effective, leading to

$$p_u(t)[k] = p_{uk} + \alpha_{uk} \cdot \mathrm{dev}_u(t) + p_{ukt} \qquad k = 1, \ldots, f. \quad (12)$$

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_{ukt} absorbs the very local, day-specific variability.

At this point, we can tie all pieces together and extend the SVD factor model (4) by incorporating the time changing parameters. The resulting model will be denoted as *timeSVD*, where the prediction rule is as follows:

$$\hat{r}_{ui} = \mu + b_i(t_{ui}) + b_u(t_{ui}) + q_i^T p_u(t_{ui}). \quad (13)$$

The exact definitions of the time drifting parameters $b_i(t)$, $b_u(t)$, and $p_u(t)$ were given in Equations 7, 9, and 12. 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. 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.

4.4. Comparison

The factor model we are using in practice is slightly more involved than the one described so far. The model, which is known as SVD++,[7] offers an improved accuracy by also accounting for the more implicit information recorded by which items were rated (regardless of their rating value). While details of the SVD++ algorithm are beyond the scope of this article, they do not influence the introduction of temporal effects, and the model is extended to account for temporal effects following exactly the same procedure described in this section. The resulting model is known as timeSVD++, and is described in Koren.[8]

In Table 2 we compare results of three matrix factorization algorithms. First is SVD, the plain matrix factorization algorithm. Second is the SVD++ method, which improves upon SVD by incorporating a kind of implicit feedback. Third is timeSVD++, which also accounts for temporal effects. The three methods are compared over a range of factorization dimensions (f). All benefit from a growing number of factor dimensions that enables them to better express complex movie–user interactions. Addressing implicit feedback by the SVD++ model leads to accuracy gains within the movie-rating dataset. Yet, the improvement delivered by

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.

Model	*f* = 10	*f* = 20	*f* = 50	*f* = 100	*f* = 200
SVD	0.9140	0.9074	0.9046	0.9025	0.9009
SVD++	0.9131	0.9032	0.8952	0.8924	0.8911
timeSVD++	0.8971	0.8891	0.8824	0.8805	0.8799

timeSVD++ over SVD++ is consistently more significant. 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. Further evidence of 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.

4.5. Predicting future days

Our models include day-specific parameters. An aparaent question would be how these models can be used for predicting ratings in the future, on new dates for which we cannot train the day-specific parameters? The simple answer is that for those future (untrained) dates, the day-specific parameters should take their default value. In particular for Equation 11, $c_u(t_{ui})$ is set to c_u, and $b_{u,t_{ui}}$ is set to zero. Yet, one wonders, if we cannot use the day-specific parameters for predicting the future, why are they good at all? After all, prediction is interesting only when it is about the future. To further sharpen the question, we should mention the fact that the Netflix test sets include many ratings on dates for which we have no other rating by the same user and hence day-specific parameters cannot be exploited.

To answer this, notice that our temporal modeling makes no attempt to capture future changes. All it is trying to do is to capture transient temporal effects, which had a significant influence on past user feedback. When such effects are identified, they must be tuned down, so that we can model the more enduring signal. This allows our model to better capture the long-term characteristics of the data, while letting dedicated parameters absorb short-term fluctuations. For example, if a user gave many higher than usual ratings on a particular single day, our models discount those by accounting for a possible day-specific good mood, which does not reflect the longer term behavior of this user. This way, the day-specific parameters contribute to cleaning the data, which improves prediction of future dates.

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,[7] 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:

$$\hat{r}_{ui} = \mu + b_i + b_u + \left|\mathrm{R}(u)\right|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + c_{ij}. \qquad (14)$$

Here, the set R(u) contains the items rated by user u. 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 learned from the data together with the biases b_i and b_u. The constants b_{uj} are precomputed according to Equation 3. Recall that R(u) is the set of items rated by user u.

When adapting rule (14) to address temporal dynamics, two components should be considered separately. First component, $\mu + b_i + b_u$, corresponds to the the baseline predictor portion. Typically, this component explains most variability in the observed signal. Second component, $\left|R(u)\right|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - b_{uj}) w_{ij} + c_{ij}$, captures the more informative signal, which deals with user–item interaction. As for the baseline part, nothing changes from the factor model, and we replace it with $\mu + b_i(t_{ui}) + b_u(t_{ui})$, according to Equations 7 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. The learning process should 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 within 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 5 years apart, while the user's taste (if not her identity) could considerably change, this provides less 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 learned from the data. We also experimented with other decay forms, like the computationally cheaper $(1 + \beta_u \Delta t)^{-1}$, which resulted in about the same accuracy, with an improved running time.

This leads to the prediction rule

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

The involved parameters, $b_i(t_{ui}) = b_i + b_{i,\mathrm{Bin}(t_{ui})}$, $b_u(t_{ui}) = b_u + \alpha_u \cdot \mathrm{dev}_u(t_{ui}) + b_{u,t_{ui}}$, β_u, w_{ij} and c_{ij}, are learned by minimizing the associated regularized squared error

$$\sum_{(u,i) \in \kappa} \Big(r_{ui} - \mu - b_i - b_{i,\mathrm{Bin}(t_{ui})} - b_u - \alpha_u \mathrm{dev}_u(t_{ui}) - b_{u,t_{ui}} - |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} ((r_{uj} - b_{uj}) w_{ij} + c_{ij}) \Big)^2 + \lambda_{12} (b_i^2 + b_{i,\mathrm{Bin}(t_{ui})}^2 + b_u^2 + \alpha_u^2 + b_{ut}^2 + w_{ij}^2 + c_{ij}^2). \quad (16)$$

Minimization is performed by stochastic gradient descent. As 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[7] 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 by using hybrid approaches such as applying a neighborhood approach on residuals of other algorithms.[2, 11, 18] 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 Section 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

$$\sum_{(u,i) \in \kappa} e^{-\beta_u \cdot |t - t_{ui}|} \Big(r_{ui} - \mu - b_i - b_{i,\mathrm{Bin}(t_{ui})} - b_u - \alpha_u \mathrm{dev}_u(t_{ui}) - b_{u,t_{ui}} - |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} ((r_{uj} - b_{uj}) w_{ij} + c_{ij}) \Big)^2 + \lambda_{12}(\cdots).$$

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 Equation 16, thereby modeling *all* past user behavior. Therefore, equal-weighting 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 time frame, even if this happened long ago.

6. CONCLUSION

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 many signals, 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 to 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. C

References

1. Ali, K., van Stam, W. TiVo: making show recommendations using a distributed collaborative filtering architecture. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2004), 394–401.
2. Bell, R., Koren, Y. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. *IEEE International Conference on Data Mining* (*ICDM'07*) (2007), 43–52.
3. Bennet, J., Lanning, S. The Netflix Prize. *KDD Cup and Workshop*, 2007. www.netflixprize.com.
4. Ding, Y., Li, X. Time weight collaborative filtering. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management* (*CIKM'04*) (2004), 485–492.
5. Goldberg, D., Nichols, D., Oki, B.M., Terry, D. Using collaborative filtering to weave an information tapestry. *Commun. ACM 35* (1992), 61–70.
6. Kolter, J.Z., Maloof, M.A. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proceedings of the IEEE Conference on Data Mining* (*ICDM'03*) (2003), 123–130.
7. Koren, Y. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD'08*) (2008), 426–434.
8. Koren, Y. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD'09*) (2009), 447–456.
9. Koren, Y., Bell, R., Volinsky, C. Matrix factorization techniques for recommender systems. *IEEE Comput. 42* (2009), 30–37.
10. Linden, G., Smith, B., York, J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Comput. 7* (2003), 76–80.
11. Paterek, A. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of the KDD Cup and Workshop* (2007).
12. Pu, P., Bridge, D.G., Mobasher, B., Ricci, F. (eds.). In *Proceedings of the 2008 ACM Conference on Recommender Systems* (2008).
13. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J. Application of dimensionality reduction in recommender system—A case study. *WEBKDD'2000*.
14. Sarwar, B., Karypis, G., Konstan, J., Riedl, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on the World Wide Web* (2001), 285–295.
15. Schlimmer, J., Granger, R. Beyond incremental processing: Tracking concept drift. In *Proceedings of the 5th National Conference on Artificial Intelligence* (1986), 502–507.
16. Sugiyama, K., Hatano, K., Yoshikawa, M. Adaptive web search based on user profile constructed without any effort from users. In *Proceedings of the 13th International Conference on World Wide Web* (*WWW'04*) (2004), 675–684.
17. Takacs, G., Pilaszy, I., Nemeth, B., Tikk, D. Major components of the gravity recommendation aystem. *SIGKDD Explor. 9* (2007), 80–84.
18. Toscher, A., Jahrer, M., Legenstein, R. Improved neighborhood-based algorithms for large-scale recommender systems. *KDD'08 Workshop on Large Scale Recommenders Systems and the Netflix Prize* (2008).
19. Tsymbal, A. The problem of concept drift: Definitions and related work. Technical Report TCD-CS-2004-15. Trinity College Dublin, 2004.
20. Widmer, G., Kubat, M. Learning in the presence of concept drift and hidden contexts. *Mach. Learn. 23*, 69 (1996), 101.

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