INVITED PAPER

Web Scale Media Recommendation Systems

By Gideon Dror, Noam Koenigstein, and Yehuda Koren

ABSTRACT | Modern consumers are inundated with choices. A variety of products are offered to consumers, who have unprecedented opportunities to select products that meet their needs. The opportunity for selection also presents a time-consuming need to select. This has led to the development of recommender systems that direct consumers to products expected to satisfy them. One area in which such systems are particularly useful is that of media products, such as movies, books, television, and music. We study the details of media recommendation by focusing on a large scale music recommender system. To this end, we introduce a music rating data set that is likely to be the largest of its kind, in terms of both number of users, items, and total number raw ratings. The data were collected by Yahoo! Music over a decade. We formulate a detailed recommendation model, specifically designed to account for the data set properties, its temporal dynamics, and the provided taxonomy of items. The paper demonstrates a design process that we believe to be useful at many other recommendation setups. The process is based on gradual modeling of additive components of the model, each trying to reflect a unique characteristic of the data.

KEYWORDS | Collaborative filtering; matrix factorization; recommender systems; Yahoo! Music

I. INTRODUCTION

Web retailers and content providers offer a large selection of products, with unprecedented opportunities to meet a variety of special needs and tastes. Consequently, the web has generated huge collections of recorded human ratings and interactions with an assortment of products. This opened an opportunity for recommender systems that analyze patterns of user interest in products to provide personalized recommendations that suit a user's taste. As leading e-commerce companies and websites have made recommender systems prominent, the last decade has brought extensive research and rapid developments to the field [1].

Recommender systems are particularly useful for media products such as movies, music, and TV shows. Indeed, media products come in large varieties, cater to different tastes, and are usually cheap to purchase—an ideal setup for making automated recommender systems vital and successful. In addition, customers have proven willing to indicate their level of satisfaction with specific media items, so that a massive volume of data is available as to which items appeal to which customers. Proper analysis of this data can be used to recommend suitable items to particular customers.

Even though media products are a natural domain for a recommender system, creating such a successful system is challenging. The designer needs to model the very elusive human taste appropriately, and deal with the fact that humans constantly change and redefine their preferences. Furthermore, balancing between a detailed modeling of heavy users and an adequate modeling of newcomers within a single model makes the task all the more challenging. Finally, systems often need to combine various types of signals, such as different kinds of feedback originating from its users together with external third-party data describing the offered items.

Given the challenges above, we describe a recommender system built for Yahoo! Music as a case study and an example to approaching large scale recommendations. Accordingly, we compiled one of the largest publicly

Manuscript received June 15, 2011; revised December 01, 2011; accepted February 08, 2012.
G. Dror and **Y. Koren** are with Yahoo! Research Labs, Haifa 31905, Israel (e-mail: gideondr@yahoo-inc.com; yehuda@yahoo-inc.com).
N. Koenigstein was with Yahoo! Research Labs, Haifa 31905, Israel. He is now with the School of Electrical Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel (e-mail: noamk@eng.tau.ac.il).
Digital Object Identifier: 10.1109/JPROC.2012.2189529

available data sets, which contains over 250 million ratings, more than a million users and above half a million items. We analyze the data properties, which genuinely reflect the wisdom of a large crowd and its cumulative taste in music, as well as phenomena related to the operation of the service.

In order to facilitate quality personalized item recommendations, the system learns to model the many kinds of interactions existing in the data. These interactions are manifested through the varying ratings users give to items. The different rating levels reflect many factors underlying the data including: varying items' popularity, different ways users interact with the rating widget, temporal dynamics affecting both users and items, and finally truly different tastes in music. By devising a detailed model for explaining and capturing the factors driving observed ratings, we can reason on which items fit a user, the ultimate goal of a recommender system.

Our system is described in an incremental manner, where the gradually added components are designed to capture more patterns present in the data set. Although we focus on a specific data set, the phenomena we describe and model are common to many recommendation data sets. Furthermore, the gradual description of the model also parallels the structured processes we typically follow when developing a recommendation model.

II. BACKGROUND

Broadly speaking, recommender systems are based on two different strategies. The content analysis (CA) approach (aka, content filtering) creates a profile for each user or product to characterize its nature. The resulting profiles allow programs to associate users with matching products. Of course, content based strategies require gathering external information that might not be available or easy to collect.

A known successful realization of CA is the *Music Genome Project*, which is used within the internet radio service Pandora.com. Each song in the Music Genome Project is scored on hundreds of distinct musical characteristics by a trained music analyst. These attributes (or, "genes") capture not only the musical identity of a song, but also many significant qualities that are relevant to understanding the musical preferences of listeners. Other music CA approaches analyze the audio content for characterizing tracks and establishing item-to-item similarities [2]. For example, Mel-Frequency Cepstral Coefficients are often used to generate feature vectors that can be used to find other items with acoustic resemblance [3], [4]. This, of course, requires a significant signal processing and analysis effort on all the available songs. In other domains, textual attributes, cultural information, social tags, and other kinds of web-based annotations are employed for characterizing the items in the inventory. For example, a movie profile could include attributes regarding its genre, the participating actors, its box office popularity, etc. Typically in CA, a major challenge is to find the best attributes that describe the items. Nonetheless, it is often impossible to define attributes that describe all the different types of relations between the items. In music, for example, it is difficult to capture the social context of songs which is often attributed to a mixture of evasive concepts such as the lyrics meaning, the way the artist dresses, her hometown, ethnicity, and countless other factors.

Collaborative filtering (CF), which is our focus in this work, is an alternative to CA. CF relies only on past behavior of users, e.g., their previous transactions or product ratings, thus overcoming the need for creating 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, yet it can address aspects of the data that are often elusive and very difficult to profile using CA.

Algorithms based on collaborative filtering typically suffer from the cold-start problem when encountering items with little rating information [5]. Several hybrid approaches are suggested for merging CA and CF; see [6]–[8] for some of the more recent approaches. They allow relying on item attributes when rating information is not sufficient, while still enjoying the improved accuracy CF offers as more ratings are gathered.

In order to establish recommendations, CF systems need to relate two fundamentally different entities: items and users. There are two primary approaches to facilitate such a comparison, which constitute the two main techniques of CF: *the neighborhood approach* and *latent factor models*. Neighborhood methods focus on relationships between items or, alternatively, between users. An item–item approach [9], [10] models the preference of a user to an item based on ratings of "neighboring" items by the same user. The neighbors of a product are other products that tend to be scored similarly when rated by the same user. For example, consider the well-known song "Smells Like Teen Spirit" on the well-known album "Nevermind" by the American grunge rock group "Nirvana." Its neighbors might include other "Nirvana" albums, similar Alternative Rock artists (e.g., Pearl Jam), or other popular music from the 1990s. To predict a particular user's rating for "Nevermind," we would look for the album's nearest neighbors that were actually rated by that user. A dual to the item-item approach is the user-user approach, which identifies like-minded users who can complement each other's missing ratings.

Latent factor models represent an alternative approach that tries to explain the ratings by characterizing both items and users as vectors in a space of tens to hundreds of latent factors, inferred from the pattern of ratings. In a sense, such factors comprise a computerized analogue to the aforementioned human created song genes. In music,

factors discovered by the decomposition might measure obvious dimensions such as genres, tempo or pitch or less well-defined dimensions such as the emotions provoked by the song or the social target audience.

One of the most successful realizations of latent factor models is based on *matrix factorization*; see, e.g., [11]. These methods have become very popular in recent years by combining good scalability with predictive accuracy. In addition, they offer considerable flexibility in modeling diverse real life situations. This paper will describe various aspects and recent developments concerning matrix factorization methods.

Recommender systems rely on various types of input. Most convenient is high quality *explicit feedback*, where users directly report their interest in products. For example, Netflix collects star ratings for movies and TiVo users indicate their preferences for TV shows by hitting thumbs-up/down buttons. The Yahoo! Music data, which will be described shortly, also contains explicit feedback in the form of 0–100 ratings. Still, we should mention that explicit feedback is often not available. In such cases, recommenders infer user preferences from the more abundant *implicit feedback*, which indirectly reflect opinion through observing user behavior. Types of implicit feedback include purchase history, browsing history, search patterns, or even mouse movements. For example, a user who has listened to many songs of an artist implicitly gives a positive feedback for that artist.

Regardless of the kind of feedback, system accuracy can greatly improve by considering also the context of the provided feedback. For example, was the rating provided to a song suggested by the system, or followed listening to a song chosen by the user? Or, did the user listen to the song just before rating it or not? Such effects are known to bear much impact on the provided rating values. Of a particular relevance to the Yahoo! Music data, which this work addresses, Marlin *et al.* [12] have shown strong selection effects leading users to deliberately rate more often the items they actually like. In a later work, Marlin and Zemel [13] suggested models for coping with such effects. The various effects influencing user selection are strongly related to the sparseness patterns found in the data, to which we now turn.

In a typical rating data set, most users rate only a small fraction of the items, thereby leaving most user-item relations unknown. Thus, a key characteristic of collaborative filtering is the need to analyze a sparse and highly nonuniform data set. The reader should note that unlike most other sparse problems, the unknown values should not assume any particular default value (e.g., 0) but rather are truly missing. Such sparseness raises challenges such as the risk of overfitting users (or items) for which only a few ratings are available, and the inability to use standard analysis tools, which rely on having a full knowledge of the data set. Furthermore, it is desired that CF methods exploit the data sparseness for making computations more scalable. Hence, direct implementations of the aforementioned neighborhood methods are generally less scalable as they represent all possible item–item (or, user–user) relations. Similarly, matrix factorization methods which model the full user–item matrix by imputing suitable values into the missing entries would not scale well. Indeed, when addressing large scale data sets, one should look for models that treat only the known ratings and thus scale linearly with data size.

Another challenge a recommender faces is the dynamic nature of the recommendation domain. Item perception and popularity are constantly changing as new items are introduced into the system and as a result of various social phenomena. Similarly, user inclinations are evolving, leading them to ever redefine their taste [14]. Thus, modeling temporal dynamics should be key when designing recommender systems. However, this raises unique challenges. In a system that models a complex interaction of numerous products and customers, many different characteristics are shifting simultaneously. Those shifts are often quite delicate and are associated with a few data instances. We will pay special attention to modeling these temporal changes, which our experiments show to be crucial to model accuracy.

While this work concentrates on scalable solutions for improved recommendation accuracy, the reader should note that recommendation systems strive to optimize additional metrics. For example, systems typically present the user a list of recommended items, hence full list optimization should also be considered, frequently as a postprocessing phase. Here, not only the relevance of each single item matters, but also considerations like the diversity of the item list and its ability to cover different genres in order to increase the chance that the user will find at least one item of interest. Further, users mostly benefit from novel recommendations for items they are not aware of, which tend to be the less popular items. Hence, a desired metric is serendipity. Finally, going even further from quantitative metrics, it is highly desired that the recommendation mechanism will be transparent and accompanied with a short description to why a specific product was recommended to the user. This helps in improving the users' trust in the system and their ability to put recommendations in the right perspective. For more on the different aspects on which recommendation systems are evaluated the reader is referred to [15]–[17].

III. THE YAHOO! MUSIC DATA SET

A. Yahoo! Music Radio Service

Yahoo! Music[1] was one of the first providers of personal internet music radio stations, with a database of

[1]new.music.yahoo.com.

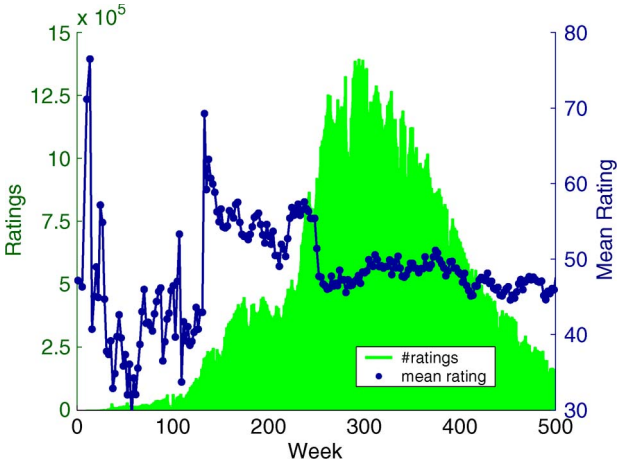

Fig. 1. ***Yahoo! Music data set: Number of ratings and mean rating score versus time in weeks.***

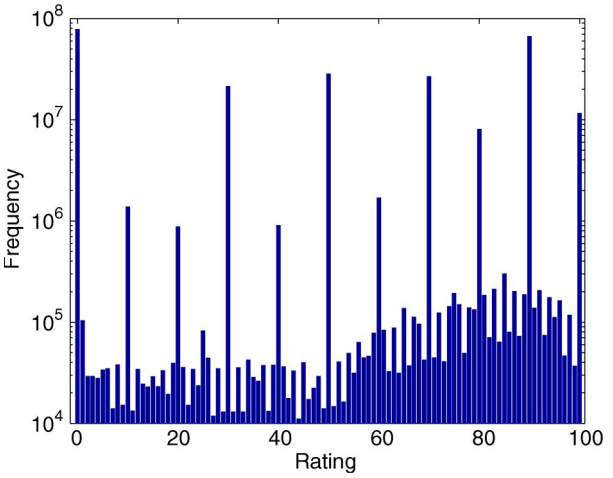

Fig. 2. ***Ratings distribution.***

hundreds of thousands of songs. As a pioneer in online music streaming, it influenced many subsequent services. Yahoo! Music used to be the top ranked online music site in terms of audience reach and total time spent. The service was free with some commercial advertising in between songs that could be removed by upgrading to a premium account. Users could rate songs, artists, albums and even genres on a five-star system, or using a slider interface. These ratings were used by Yahoo! Music to generate recommendations that match the user's taste, based on either the taxonomy of items or on recommendations of other users with similar musical tastes.

B. Ratings Data Set

This work is based on a data set sampled from the Yahoo! Music database of ratings collected during 1999–2010.[2] The data set comprises 262 810 175 ratings of 624 961 music items by 1 000 990 users. The ratings include one-minute resolution timestamps, allowing refined temporal analysis. Each item and each user has at least 20 ratings in the whole data set. The available ratings were split into train, validation and test sets, such that the last six ratings of each user were placed in the test set and the preceding four ratings were used in the validation set. The train set consists of all earlier ratings (at least 10). The total sizes of the train, validation, and test sets are therefore 252 800 275, 4 003 960, and 6 005 940, respectively. Fig. 1 depicts the weekly number of ratings and the weekly mean ratings score versus the number of weeks that passed since the launch of the service on 1999.

The ratings are integers between 0 and 100. Fig. 2 depicts the distribution of ratings in the train set using a logarithmic vertical scale. The vast majority of the ratings are multiples of ten, and only a minuscule fraction are not. This mixture reflects the fact that several interfaces ("widgets") were used to rate the items, and different users had different rating "strategies." While different widgets have different appearances, scores have always been stored internally at a common 0–100 scale. We possess only the 0–100 internal representation, and do not know the exact widget used for creating each rating. Still, the popularity of a widget used to enter ratings at a "1"- to "5"-star scale is reflected by the dominance of the peaks at 0, 30, 50, 70, and 90 into which star ratings were translated.

An interesting aspect of the data is the fact that widgets have been altered throughout the years. Fig. 3 depicts the

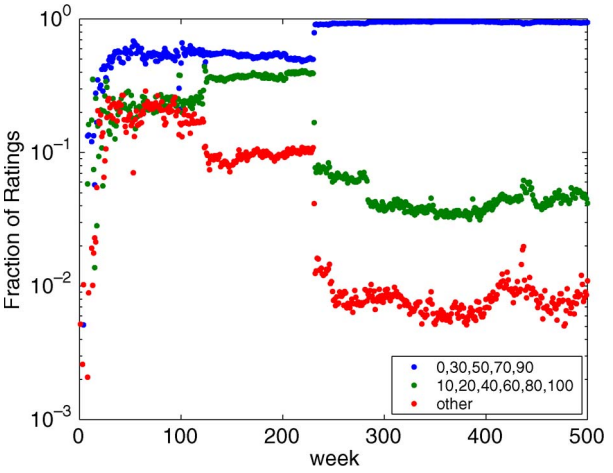

Fig. 3. ***Relative frequency of the three groups of ratings as a function of time.***

[2]Publicly available at http://webscope.sandbox.yahoo.com/catalog.php?datatype=c, with thousands of downloads currently on record.

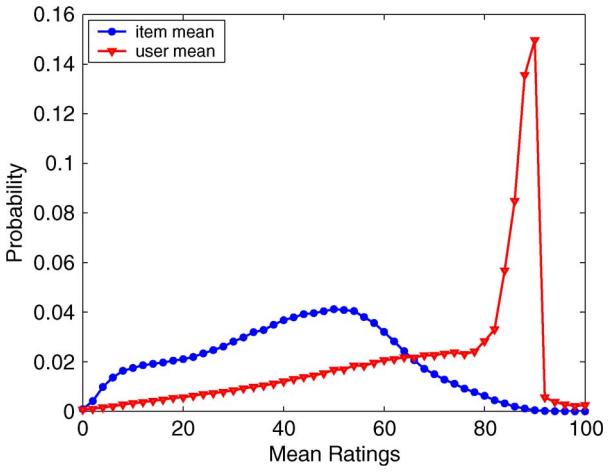

Fig. 4. ***The distributions of item and user mean ratings.***

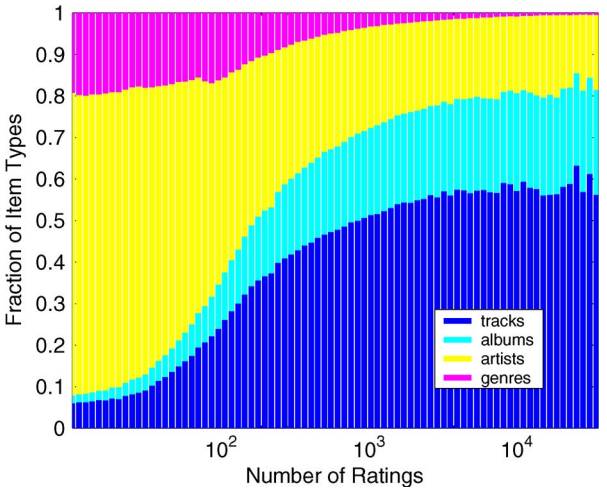

Fig. 6. ***The fraction of ratings the four item types receive as a function of the number of ratings a user gives.***

relative frequency of each of the three types of ratings. In the first group are the ratings corresponding to the five dominant peaks of Fig. 2 (0, 30, 50, 70, and 90). The second group includes the remaining peaks (10, 20, 40, 60, 80, and 100), and the third group contains the remaining ratings (those not divisible by 10). Abrupt changes in the relative frequencies of the three groups may be clearly observed on the 125th week as well as on the 225th week. These dates are also associated with a dramatic change in mean rating, as can be observed in Fig. 1.

We calculated the mean rating of each user, as well as the mean rating of each item. Fig. 4 depicts these two distributions. The location of the modes (at 89 and 50, respectively), as well as the variances of the two distributions are quite distinct. In addition, the distribution of the mean user ratings is significantly more skewed.

Different rating behavior of users accounts for the apparent difference between the distributions. It turns out that users who rate more items tend to have considerably lower mean ratings. Fig. 5 substantiates this effect. Users were binned according to the number of items they rated, on a linear scale. The graph shows the median of the mean ratings in each bin, as well as the interquantile range in each bin plotted as a vertical line. One of the explanations for this effect is that "heavy" raters, those who explore and rate tens of thousands of items, tend to rate more items that do not match their own musical taste and preferences, and thus the rating scores tend to be lower.

A distinctive feature of this data set is that user ratings are given to entities of four different types: *tracks*, *albums*, *artists*, *and genres*. The majority of items (81.15%) are tracks, followed by albums (14.23%), artists (4.46%), and genres (0.16%). The ratings however, are not uniformly distributed: Only 46.85% of the ratings belong to tracks, followed by 28.84% to artists, 19.01% to albums and 5.3% to genres. Moreover, these proportions are strongly dependent on the number of ratings a user has entered. Heavier raters naturally cover more of the numerous tracks, while the light raters mostly concentrate on artists; the effect is shown in Fig. 6. Thus, unlike the train set, the validation and test sets, which equally weight all users, are dominated by the many light-raters and dedicate most of their ratings to artists rather than to tracks; see Table 2. Interestingly, the rating distribution of each of the types is very similar to that exhibited by the set of all items, depicted in Fig. 2.

All rated items are tied together within a taxonomy. That is, for a track we know the identity of its album, performing artist and associated genres. Similarly we have

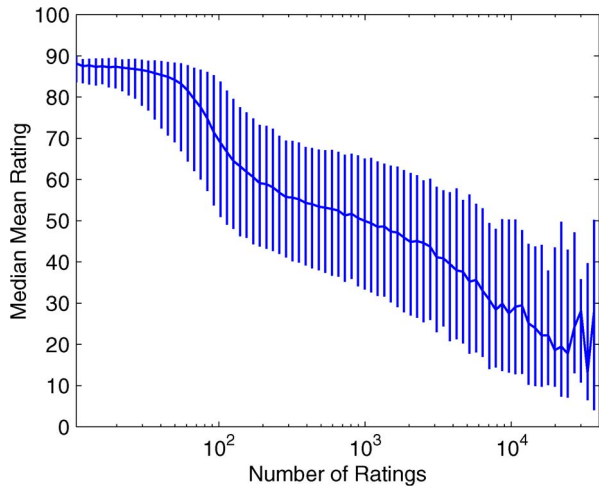

Fig. 5. ***Median of user ratings as a function of the number of ratings issued by the user. The vertical lines represent inter-quartile range.***

artist and genre annotation for the albums. There is no genre information for artists, as artists may switch between many genres in their career. We show that this taxonomy is particularly useful, due to the large number of items and the sparseness of data per item (mostly attributed to "tracks" and "albums").

Although this section focuses on the Yahoo! Music data set, many of the characteristics described here are not limited to this data set. A hierarchy of item categories, for example, is a very common feature relevant to most web recommender systems. Indeed, e-commerce sites, e.g., those selling books, movies or electronics, tend to arrange their items within a taxonomy. Our experience is that other recommender system data sets (like the Netflix data set [18]) also exhibit power law distributions of both number of ratings per item and number of ratings per user, producing effects similar to that of Fig. 4. A nonstationary distribution of ratings as depicted in Fig. 1 is also a common phenomenon, which usually stems from the change in the supply of items, from upgrades of web interfaces or from social, political or economic trends affecting the taste and inclination of the users; see, e.g., [14]. On the other hand, we are not aware of other recommender system data sets where items attached to different levels of the taxonomy can be rated. While this phenomenon is more relevant to music data sets, one could reasonably imagine movie recommenders asking users to rate full genres, or book recommenders asking users to rate book authors.

IV. NOTATION

We reserve special indexing letters for distinguishing users from items: for users u, and for items i. A rating r_{ui} indicates the rating given by user u to item i. We distinguish predicted ratings from known ones using the notation $\hat{r}_{ui}$ for the predicted value of r_{ui}.

For tracks, we denote by $\mathrm{album}(i)$ and $\mathrm{artist}(i)$ the album and the artist of track i respectively. Similarly, for albums, we denote by $\mathrm{artist}(i)$ the artist of album i. Tracks and albums in the Yahoo! Music data set may belong to one or more genres. We denote by $\mathrm{genres}(i)$ the set of genres of item i. Lastly, we denote by $\mathrm{type}(i)$ the type of item i, with $\mathrm{type}(i) \in \{\mathrm{track}, \mathrm{album}, \mathrm{artist}, \mathrm{genre}\}$.

We are training our models in order to minimize the squared error, which is a common measure in rating-based recommenders. In particular, the accuracy of the model is measured by its root mean squared error (RMSE) on the test set defined as

$$\sqrt{\frac{\sum_{(u,i)\in \mathrm{Test}} \left(\hat{r}_{ui} - r_{ui}\right)^2}{|\mathrm{Test}|}}. \tag{1}$$

V. BIAS MODELING

A. The Importance of Biases

In the context of rating systems, biases model the portion of the observed signal that is derived either solely by the user or solely by the rated item, but not by their interaction. For example, a user bias may model a user's tendency to rate higher or lower than the average rater, while an item bias may capture the extent of the item's popularity. Fig. 4 illustrates the fact that the mean ratings of various users and items are quite diverse, suggesting the importance of modeling these two types of biases.

A general framework for capturing the bias of the rating by user u to item i is described as

$$b_{ui} = \mu + B_i + B_u \tag{2}$$

where μ is the overall mean rating value (a constant), and B_i and B_u stand for item and user biases, respectively.

Since components of the user bias are independent of the item being rated, while components in the item bias are independent of any user, they do not take part in modeling personalization, e.g., modeling the musical taste of a user. After all, ordering items by using only a bias model (2) necessarily produces the same ranking for all users, hence personalization—the cornerstone of recommendation systems—is not achieved at all. Yet, there is plenty of evidence that much of the observed variability of ratings is attributed to biases. Hence, properly modeling biases would effectively amount to cleaning the data from patterns unrelated to personalization purposes. This will allow the personalization part of the model (e.g., matrix factorization) to be applied to signals much more relevant to personalization, where users and items do interact.

We investigated adding different bias components that capture various patterns present in the data set. Accordingly, in this section, we present a rich model for both the item and user biases, which accounts for the item taxonomy, user rating sessions, and items' temporal dynamics. The model adheres to the framework of (2). In the following we will gradually develop its B_i and B_u components. The predictive performance of the various components of the bias model is discussed in Section VIII.

B. A Basic Bias Model

The most basic bias model captures the main effects associated with users and items [19]. Following bias template (2), we set the item bias B_i as a distinct parameter associated with each item denoted by b_i, and similarly the user bias B_u as a user-specific parameter b_u. This gives rise to the model

$$b_{ui} = \mu + b_i + b_u. \tag{3}$$

C. Taxonomy Biases

We start enhancing our bias model by letting item biases share components for items linked by the taxonomy. For example, tracks in a good album may all be rated somewhat higher than the average, or a popular artist may have all her songs rated a bit higher than the average. We therefore add shared bias parameters to different items with a common ancestor in the taxonomy hierarchy. We expand the item bias model for tracks as follows:

$$B_i^{(0)} = b_i + b_{\mathrm{album}(i)} + b_{\mathrm{artist}(i)} + \frac{1}{|\mathrm{genres}(i)|}\sum_{g\in\mathrm{genres}(i)} b_g. \quad (4)$$

Here, the total bias associated with a track i sums both its own specific bias modifier (b_i), together with the bias associated with its album $(\mathrm{album}(i))$ and its artist $(\mathrm{artist}(i))$, and the mean bias associated with its genres $(1/|\mathrm{genres}(i)|)\sum_{g\in\mathrm{genres}(i)} b_g$.

Similarly for each album we expand the bias model as follows:

$$B_i^{(0)} = b_i + b_{\mathrm{artist}(i)} + \frac{1}{|\mathrm{genres}(i)|}\sum_{g\in\mathrm{genres}(i)} b_g. \quad (5)$$

One could view these extensions as a gradual accumulation of the biases. For example, when modeling the bias of album i, the start point is $b_{\mathrm{artist}(i)} + (1/|\mathrm{genres}(i)|)\sum_{g\in\mathrm{genres}(i)} b_g$, and then b_i adds a residual correction on top of this start point. Similarly, when i is a track another track-specific correction is added on top of the above. As bias estimates for tracks and albums are less reliable, such a gradual estimation allows basing them on more robust initial values.

Note that such a framework not only relates items to their taxonomy ancestors, but (indirectly) also to other related items in the taxonomy. For example, a track will get related to all other tracks in its album, and to lesser extent to all other tracks by the same artist.

Also, note that while artists and genres are less susceptible to the sparsity problem, they also benefit from this model as ratings to tracks and albums also influence the biases of their corresponding artist and genre.

The taxonomy of items is also useful for expanding the user bias model. For example, a user may tend to rate artists or genres higher than songs. Therefore, given an item i the user bias is

$$B_u^{(0)} = b_u + b_{u,\mathrm{type}(i)} \quad (6)$$

where b_u is the user specific bias component and $b_{u,\mathrm{type}(i)}$ is a shared component of all the ratings by user u to items of type $\mathrm{type}(i)$.

D. Rating Sessions Modeling

Ratings are marked by a date and a timestamp with resolution down to minutes. We used this information for modeling temporal dynamics of both items and users. We start by modeling user sessions. It is common for users to listen to many songs and rate them one after the other. A rating session is therefore a set of consecutive ratings without an extended time gap between them. In our implementation user sessions are separated by at least five hours idle (no rating activity). There are many psychological phenomena that affect ratings grouped in a single session. These effects are captured by user session biases.

One example is the fact that the order in which the songs were listened by the user might determine the ratings scores, a phenomenon known as the drifting effect [20]. Users tend to rate items in the context of previous items they rated. If the first song a user hears is particularly good, the following items are likely to be rated by that user lower than the first song. Similarly, if the user did not enjoy the first song, the ratings of subsequent songs may shift upwards. The first song therefore may serve as a reference rating to all the following ratings. However, with no absolute reference for the first rating, the user sometimes find it hard to rate, and some users tend to give it a default rating (e.g., 70 or 50). Consequently, all the following ratings in that same session may be biased higher or lower according to the first rating. Another source for session biases is the mood of the user. A user may be in a good/bad mood that may affect her ratings within a particular session. It is also common to listen to similar songs in the same session, and thus their ratings become similar.

The Yahoo! Music data set exhibits another form of session bias, where longer sessions tend to have a lower mean rating. This is not surprising, given the lower average rating for "heavier" users, as summarized in Fig. 5. But more importantly, this claim is correct even on a per-user basis, namely for each user longer sessions tend to have a significantly lower mean rating. To show this we calculated for each session the difference, Δ, between the mean rating of the session to the mean rating of the corresponding user. Averaging Δ over all sessions and plotting it as a function of the sessions' length results in Fig. 7. The error-bars represent a 0.95 confidence interval of the estimate of the mean value of Δ for each length. The latter was binned on a logarithmic scale. The figure shows that long sessions comprising 100 ratings or more are on average about two points lower than the mean rating of the user, whereas the shortest sessions, comprising a single rating, are on average six points higher than the mean rating of the user.

To take such effects into account, we added a session bias term to our user bias model. We denote by

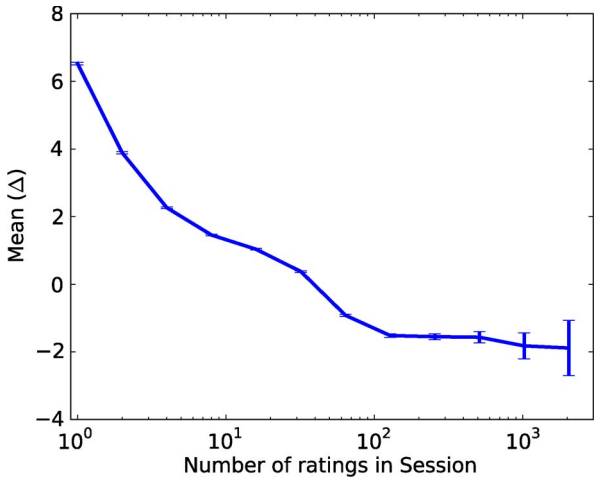

Fig. 7. *The difference between the mean rating of a session to the mean rating of the corresponding user as a function of session length, averaged over all sessions.*

session(u, i) the rating session of the rating r_{ui}, and expand our user bias model to include session biases

$$B_u^{(1)} = B_u^{(0)} + b_{u,\text{session}(i,u)}. \tag{7}$$

The session bias parameter $b_{u,\text{session}(i,u)}$ models the bias component common to all ratings of u in the same session she rated i.

E. Items Temporal Model

The popularity of songs may change dramatically over time. While users' temporal dynamics seem to follow abrupt changes across sessions, items' temporal dynamics are much smoother and slower, thus calling for a different modeling approach. We follow here an approach suggested by Piotte and Chabbert [21].

Given item i and the time t since i's first rating, we define a time dependent item bias as a linear combination of n temporal basis functions $f(t) = (f_1(t), f_2(t), \ldots, f_n(t))^T$ and expand the item bias component to be

$$B_i^{(1)} = B_i^{(0)} + c_i^T f(t) \tag{8}$$

where $c_i \in \mathbb{R}^n$ is an item specific vector of coefficients.

Both $f(t)$ and c_i are learned from data using the standard RMSE-minimizing stochastic gradient descent (SGD), which is also used for learning the other model components; see Section VII-A. In practice, a two-week course time resolution is sufficient for the rather slow changing item temporal dynamics, therefore the basis functions are only estimated at a small number of points

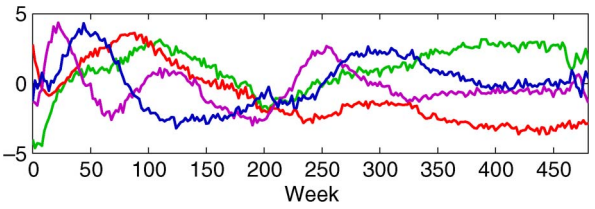

Fig. 8. *Items temporal basis functions $\{f_i(t)\}_{i=1}^4$ versus time since an item's first rating measured in weeks.*

and can be easily learned. This process does not guarantee an orthogonal or normalized basis, however, it finds a basis that fits the patterns seen in the data set.

We have found that a basis of four functions is sufficient to represent the temporal dynamics of item biases in our data set. Fig. 8 depicts the learned basis functions $\{f_i(t)\}_{i=1}^4$. Since the coefficients of the basis function can be either positive or negative, it is hard to give a clear interpretation to any specific basis function. However, an interesting observation is that basis functions seem to be have high gradients right after an item was released, indicating more dynamic temporal effects in this time period. It is also interesting to note that after a long time period (above 360 weeks), the temporal basis functions converge into relatively steady values. This indicates that at a longer perspective, items seem to have either a positive or a negative bias, with much less temporal dynamics.

F. Full Bias Model

To summarize, our complete bias model, including both enhanced user and item biases is (for a track i)

$$b_{ui} = \mu + b_{u,\text{type}(i)} + b_{u,\text{session}(i,u)} + b_i + b_{\text{album}(i)} + b_{\text{artist}(i)} + \frac{1}{|\text{genres}(i)|}\sum_{g\in\text{genres}(i)} b_g + c_i^T f(t_{ui}) \tag{9}$$

where t_{ui} is the time elapsed from i's first rating till u's rating of i.

Learning the biases is performed by SGD together with the other model components as described in Section VII-A. The extended bias model dramatically reduced the RMSE even before any personalization components were added into the model (see results in Section VIII). Biases were able to absorb much of the effects irrelevant to personalization. Such a "cleaning" proved to be a key for accurately modeling personalization in later stages.

VI. PERSONALIZATION MODEL

Our initial personalization model is based on a classical matrix factorization approach. Each user u is associated with a user-factor vector $p_u \in \mathbb{R}^d$, and each item i with an

item-factor vector $q_i \in \mathbb{R}^d$. Predictions are done using the rule

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i \tag{10}$$

where b_{ui} is the bias model (9), and $p_u^T q_i$ is the personalization model which captures user's u affinity to item i. In this section, we expand this basic personalization model to encompass more patterns observed in the data.

A. Taxonomy in Personalization

Musical artists often have a distinct style that can be recognized in all their songs. Similarly, artists style can be recognized across different albums of the same artist. Therefore, we introduce shared factor components to reflect the affinity of items linked by the taxonomy. Specifically, for each artist and album i, we employ a factor vector $v_i \in \mathbb{R}^d$ (in addition to also using the aforementioned q_i). We expand our item representation for tracks to explicitly tie tracks linked by the taxonomy

$$\tilde{q}_i \stackrel{\text{def}}{=} q_i + v_{\text{album}(i)} + v_{\text{artist}(i)}. \tag{11}$$

Therefore, q_i represents the difference of a specific track from the common representation of all other related tracks, which is especially beneficial when dealing with less popular tracks.

Similarly, we expand our item representation for albums to be

$$\tilde{q}_i \stackrel{\text{def}}{=} q_i + v_{\text{artist}(i)}. \tag{12}$$

One may also add shared factor parameters for tracks and albums sharing the same genre, similarly to the way genres were exploited for enhancing biases. However, our experiments did not show an RMSE improvement by incorporating shared genre information. This indicates that after exploiting the shared information in albums and artists, the remaining information shared by common items of the same genre is limited.

B. Session Specific User Factors

As discussed earlier, much of the observed changes in user behavior are local to a session and unrelated to longer term trends. Thus, after obtaining a fully trained model (hereinafter, "Phase I") we perform a second phase of training, which isolates rating components attributed to session-limited phenomena. In this second phase, when we reach each user session, we try to absorb the session specific signal in a separate component of the user factor. To this end we expand the user representation into

$$\tilde{p}_u = p_u + p_{u,\text{session}} \tag{13}$$

where the user representation $\tilde{p}_u$ consists of both the original user factor p_u and the session factor vector $p_{u,\text{session}}$. We learn $p_{u,\text{session}}$ by fixing all other parameters and making a few (e.g., three) SGD iterations only on the ratings given in the current session in order to learn $p_{u,\text{session}}$. After these iterations, we are able to absorb much of the temporary per-session user behavior into $p_{u,\text{session}}$, which is not explained by the model learned in Phase I. We then move to a final relaxation step, where we run one more iteration over all ratings in the same session, now allowing all other model parameters to change and shed away any per session specific characteristics. Since $p_{u,\text{session}}$ already captures much of the per-session effects of the user factor, the other model parameters adjust themselves accordingly and capture possible small changes since the previous rating session. After this relaxation step, we reset $p_{u,\text{session}}$ to zero, and move on to the next session, repeating the above process. Since we discard $p_{u,\text{session}}$ after iterating through each session, there is no need to store session vectors for every user and session in the data set, which makes the described method memory efficient.

VII. MODEL LEARNING AND TUNING

A. Model Learning

Our final prediction model takes the following form:

$$\hat{r}_{ui} = b_{ui} + \tilde{p}_u^T \tilde{q}_i \tag{14}$$

where b_{ui} is the detailed bias model as in (9), $\tilde{q}_i$ is our enhanced item factor representation as described in (11) and (12), and $\tilde{p}_u$ is defined in (13).

As previously alluded, learning proceeds by stochastic gradient descent (SGD), where all learned parameters are L2-regularized. SGD visits the training examples one-by-one, and for each example updates its corresponding model parameters. More specifically, for training example (u, i), SGD lowers the squared prediction error $e_{ui}^2 = (r_{ui} - \hat{r}_{ui})^2$ by updating each individual parameter θ by

$$\Delta\theta = -\eta \frac{\partial e_{ui}^2}{\partial \theta} - \lambda\theta = 2\eta e_{ui} \frac{\partial \hat{r}_{ui}}{\partial \theta} - \lambda\theta \tag{15}$$

here η is the learning rate and λ is the regularization rate. The purpose of L2-regularization is to reduce the model complexity thereby improving its generalization power

into unseen examples. Hence, the constant λ controls the amount of penalty on the L2-norm of the learned parameters.

Our data set spans over a very long time period (a decade). In such a long period musical taste of users slowly drifts. We therefore expect model parameters to change with time. We exploit the fact that the SGD optimization procedure gradually updates the model parameters while visiting training examples one by one. It is a common practice in online learning to order training examples by their time, so when the model training is complete, the learned parameters reflect the latest time point, which is most relevant to the test period. Since we perform a batch learning, which includes several sweeps through the data set, we need to enhance this simple technique.

We loop through the data in a cyclic manner: we visit user-by-user, whereas for each user we first *sweep forward* from the earliest rating to the latest one, and then (after also visiting all other users) we *sweep backward* from the latest rating to the earliest one, and so on. This way, we avoid the otherwise discontinuous jump from the latest rating to the first one when starting a new sweep. This allows parameters to slowly drift with time as the user changes her taste. The process always terminates with a forward iteration ending at the latest rating.

B. Hyper-Parameter Tuning

We used a distinct learning and regularization rates for each type of learned parameter. For example, item bias and personalization model parameters may require very different learning and regularizations rates. As a result, our model employs over 20 hyperparameters. This grants us the flexibility to tune learning rates such that, e.g., parameters that appear more often in a model are learned more slowly (and thus more accurately) Similarly, employing a different regularization parameter for each type of model parameter allows a better control of model complexity.

Tuning the hyper-parameters of a model is a costly procedure: partial derivatives with respect to the hyper-parameters can not be analytically computed and the objective function is usually not convex. Thus, some form of hyper-parameters search is required. Every step in the search requires running the SGD algorithm on the entire training data set, with the validation data set used for optimizing the values of the hyper-parameters.

We resorted to the Nelder–Mead simplex search algorithm [22], a widely used algorithm with excellent results on real world scenarios [23] including CF [21]. We implemented a parallel version of the algorithm [24], gaining an order of magnitude speedup of the process. We were using a five-thread parallel process, which tends to converge after about 20 iterations. A single Nelder–Mead iteration requires two executions of the model per thread. The search was conducted in logarithmic space, starting at a point where all hyper-parameters are set to 10^{-3}. Such an automatic hyperparameters optimization process was vital to the development of a rich and modular model.

VIII. RESULTS

A. RMSE Analysis

To isolate the contribution of each component in the model, we measured the RMSE of our test predictions as we gradually add components to model: we first add the bias components, and then personalization model components. The results are presented in Table 1. The table also states the SGD training time measured on an Intel 2.13 GHz Xeon (E7320) CPU.

The first and most basic model is a constant predictor. In the case of the RMSE cost function, the optimal constant predictor would be the mean train rating, $\hat{r}_{ui} = \mu$; see row 1 of the table. Row 2 presents the basic bias model $\hat{r}_{ui} = \mu + b_i + b_u$ (3). Row 3 reports the results after adding taxonomy terms, which mitigate data sparseness by capturing relations between items of the same taxonomy; see Section V-C. We then added the user session bias of Section V-D. This gave a significant reduction in terms of RMSE as reported in row 4. We believe that modeling session biases in users' ratings is key in explaining ratings behavior in domains like music in which users evaluate and rate multiple items at short time frames. In row 5 we add the item temporal bias from Section V-E. This term captures changes in item biases that occur over the lifespan of items since their first ratings. This bias is especially useful in domains in which item popularity easily changes over time such as in music, or data sets in which the rating history is long. The result in row five reflects the RMSE of our final bias model (defined in Section V-F), when no personalization is yet in place.

We move on to personalized models, which utilize a matrix factorization component of dimension 50. The model of (10) yields RMSE of 22.9235 (row 6). By adding taxonomy terms to the item factors, we were able to reduce this result to 22.8254 (row 7). Finally, in row 8 we report the full prediction model including user session factors (as in Section VI-B). The relatively large drop in RMSE, even

Table 1 RMSE of the Evolving Model. Adding Model Components Is Reducing the RMSE

#	Model Name	RMSE	Time (min.)
1	Mean Score	38.0617	0
2	Items and Users Bias	26.8561	5
3	Taxonomy Bias	26.2553	9
4	User Sessions Bias	25.3901	14
5	Items Temporal Dynamics Bias	25.2095	17
6	MF	22.9533	57
7	Taxonomy	22.7906	72
8	Final	22.5918	114

Table 2 RMSE per Item Type for the Three Personalized Models. We Also Report the Fraction of Each Item Type in the Test Data Set

	Track	Album	Artist	Genre
%Test	28.7%	11.01%	51.61%	8.68%
MF	27.1668	24.5203	20.9815	15.7887
Taxonomy	26.8899	24.3531	20.8766	15.7965
Final	26.85	24.1854	20.566	15.4801

when the model is already fully developed, highlights the significance of temporal dynamics at the user factor level.

Let us consider the effect of the taxonomy on the RMSE results of each item type. Table 2 breaks down the RMSE results per item type of the three personalized models. It is clear that incorporating the taxonomy is most helpful for the sparsely rated tracks and albums. It is much less helpful for artists, and becomes counterproductive for the densely rated genres.

We further investigate the performance of our model as more factors are added. Fig. 9 depicts the RMSE versus factors dimensionality. Since we carefully tune the regularization terms, there is no overfitting even as the dimensionality reaches 500. However, there are clearly diminishing returns of increasing dimensionality, with almost no improvement beyond 100 dimensions. Note that the RMSE gain given by the taxonomy terms and session factors remains steady even as the dimensionality increases. The graph indicates a need for at least 100 dimensions in order to accurately explain people's music preferences.

B. Error Rate Analysis

The RMSE metric is a favorable evaluation criteria thanks to its elegance and mathematical tractability. However, it can be quite detached from the items' ranking task, which is often the ultimate goal of a recommender system. For example, even a perfectly ranked solution can score arbitrarily badly on an RMSE scale. We therefore also evaluate the performance of the model by using a metric that was promoted in the KDD-Cup'11 challenge.[3] We pair each highly rated item (scored not less than 80) in the test set with a randomly drawn item not rated by the user. We use the model to rank the two items for the relevant user. The fraction of wrongly ranked pairs ("error rate") is the metric value. This metric evaluates the model ability to suggest the "right" item to the user. Henceforth, we refer to this metric as *ErrorRate*. Note that ErrorRate generalizes also to items not rated by the user, and does not constrain the evaluation only to the subset of items actually rated by the user.

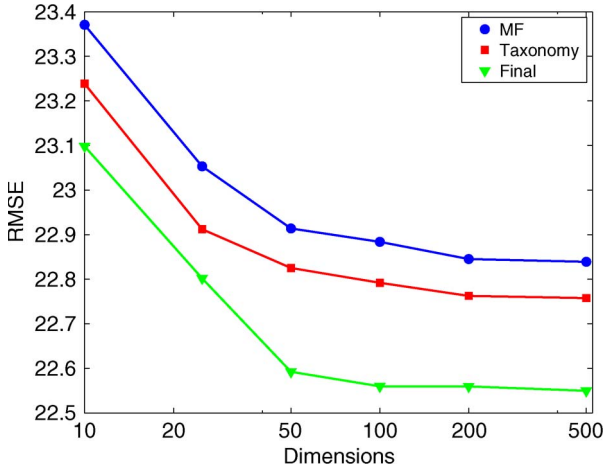

Fig. 9. *RMSE versus dimensionality of factors (d). We track regular MF, MF enhanced by taxonomy, and the final model.*

An appealing property of ErrorRate is that it can be easily adapted to diminish effects attributed to the tendency of favoring universally popular items. Ranking a highly rated item above a random item may merely reflect the ability to separate popular items from less popular ones. Thus, we modify *ErrorRate* in a way that better measures the personalization power of the models. For each highly rated item in the test set we pick a competing item (unrated by the user) with probability proportional to the item's popularity (popularity is defined as the total number of high ratings received over the whole population). This way, the rated and the randomly picked items are coming from the same distribution. Henceforth, we refer to this metric as *ErrorRatePop*. ErrorRatePop emphasizes not only the ability to accurately predict items that the user will like, but also a deeper notion of personalization separating the specific user from the crowd, which is related to the serendipity goal desired by recommenders (see Section II).

When evaluating under the error rate metrics, we need to amend the training procedure to account for the facts that 1) actual rating values are disregarded; 2) nonrated items are also considered. In order to minimize the difference from the modeling process described so far, we still train exactly the same model. However, we consider only user-item pairs with high rating values (80) and assume the fictitious rating value of 100 for these pairs. All other user-item pairs are disregarded. In addition, for each user, we pair each train rating (scored 100) with a newly introduced train rating for a randomly-picked item, with an assigned score of 0. Naturally, for the ErrorRate metric the random item is picked uniformly across all items, while for the ErrorRatePop metric the random item is picked with probability proportional to its popularity. Since timestamps are not available for the randomly drawn items, we discard the modeling of temporal effects.

We evaluate our models using both metrics. Table 3 depicts results for a 100-D model. We also provide a baseline model which predicts that the item highly rated by the user is the one more popular (i.e., with more high ratings) over the whole training data set. Note that the results under ErrorRatePop are more than twice higher

[3]http://kddcup.yahoo.com.

Table 3 Error-Rate Analysis: Popularity Effects and Taxonomy Have a Material Effect in Reducing Error Rate

	ErrorRate	ErrorRatePop
baseline	16.99%	42.87%
no taxonomy	3.26%	7.80%
with taxonomy	2.69%	5.83%

than those of ErrorRate. This reflects the high predictive power of popularity: the model is much more accurate in identifying the loved items, when benefiting from the general popularity trends of items. However, we would argue that ErrorRatePop is a better measure of the actual personalization power of the algorithm. This observation is clearly reflected in the results of the baseline model. Under the ErrorRate metric the baseline model, which obviously does not learn any personalization patterns, performs much better than under ErrorRatePop.

We isolate the contribution of the taxonomy by also training a taxonomy-oblivious model [see (10)]. We observe that under both metrics, taxonomy enables a reduction in error rate that is much more pronounced than in the RMSE case. By modeling taxonomy relations we reduce ErrorRatePop from 7.80% to 5.83%, indicating the high utility of incorporating available taxonomy structure into recommender systems. The effect of dimensionality on the error rate is illustrated in Fig. 10. For a the taxonomy aware model we see that increasing the dimensionality, even beyond 250, benefits the ErrorRatePop results (unlike the case of the RMSE metric).

C. Further Analysis

Fig. 11 depicts the item factor vectors for some of the more popular artists and genres based on a 2-D model ($d = 2$). We see that most items are located within a 90° sector. At the right hand side, we see a concentration of Hip-Hop, Rap and R&B artists and genres, while various Rock groups and subgenres of Rock are located on the left hand side of the sector. The fact that the two groups are orthogonal, means that there is little correlation between people who listen to Hip-Hop and people who listen to Rock. In other words, a user's tendency towards Hip-Hop does not convey any information about her taste in Rock. These two genres (Hip-Hop and Rock) are the most popular genres in our data set. Therefore, a 2-D system places them as the main two extremes. Fig. 11 emphasizes the need for more than two dimensions when modeling music preferences. Since the two dimensions represent Rock and Hip-Hop, there are no dimensions left for other genres. Therefore they are constrained to lie between these two groups. We thus see a concentration of Pop genres and artists at an angle that is between Rock and Hip-Hop. Similarly, genres such as Jazz, Electronic/Dance, Latin, and an artist such as Bob Marley (Reggae) are placed somewhere between Hip-Hop and Rock. In higher dimensional factorization models, all these items become mostly orthogonal to each another.

The norm of an item vector determines how suitable that item would be to a user in the same direction. For example, the vector for the Industrial Rock group Nine Inch Nails is at an angle between Nirvana and Metallica, but with a much higher magnitude. However, the latter two have much larger item biases. This means that while Nirvana and Metallica tend to get higher ratings in general (as indicated by their higher item biases), Nine Inch Nails will get higher inner product values with user vectors in its direction. Therefore, the ratings for Nine Inch Nails are more dependent on a user's musical taste (the personalization component), whereas Metallica and Nirvana are more popular in general.

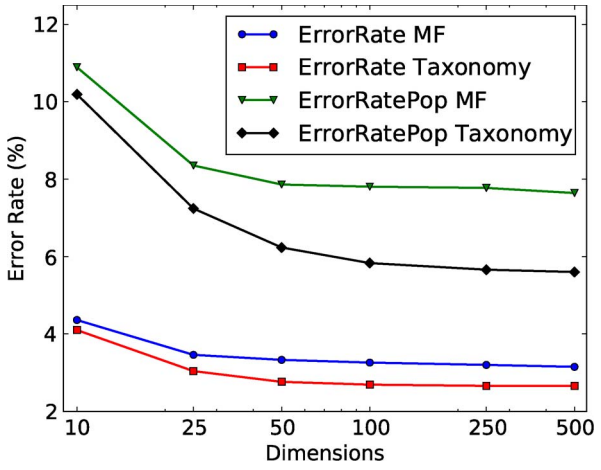

Fig. 10. *Error-rate versus dimensionality of factors (d). We track regular MF and MF enhanced by taxonomy under ErrorRate and ErrorRatePop metrics.*

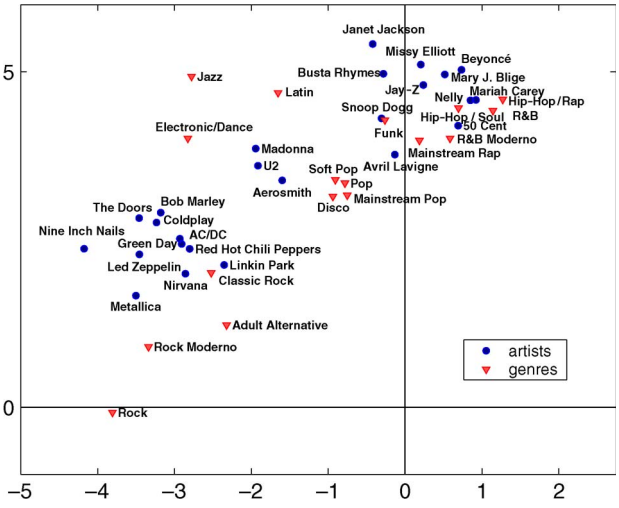

Fig. 11. *Visualization of some of the popular genres and artists item vectors using a 2-D matrix factorization model.*

As we noted above, most of the items are placed within a 90° sector. We further verified this by computing the inner product between all possible item pairs when using 50-D factor vectors and found that 96.36% of them were nonnegative. Hence, negative correlations between items ("users who like an item dislike another one") are rare. This indicates that in music, personalization is mostly based on modeling patterns of positive correlations between items. The prominence of this finding prompted us to compare it to other domains. We repeated the same experiment using the Netflix movies data set [18]. We have found that negative correlations between movies are more likely than in music. While the majority of inner products between movie factor vectors are nonnegative, a non-negligible fraction (32.35%) is of negative inner products.

Common wisdom supported by previous analysis of rating data sets (e.g., the Netflix data set [18]) suggests that users and items with more ratings are better modeled. Consequently, superior performance is expected for such users. Our results, however, demonstrate the opposite pattern, where both items and users with fewer train ratings exhibit substantially lower RMSE. Fig. 12 shows the mean RMSE of users as a function of the number of their train set ratings, binned into 200 bins. Contrary to our expectations, the RMSE is increasing as users rate more. In order to analyze this phenomenon, we also depict the mean standard deviation of test ratings for the corresponding users. The test standard deviation is a measure of the difficulty of the prediction task, and serves as a lower bound for the RMSE of a model predicting a constant value per user. The fact that the mean RMSE and the mean standard deviation follow a very similar trend clearly demonstrates that the increase of RMSE is due to the increase of rating variability for users that rate more.

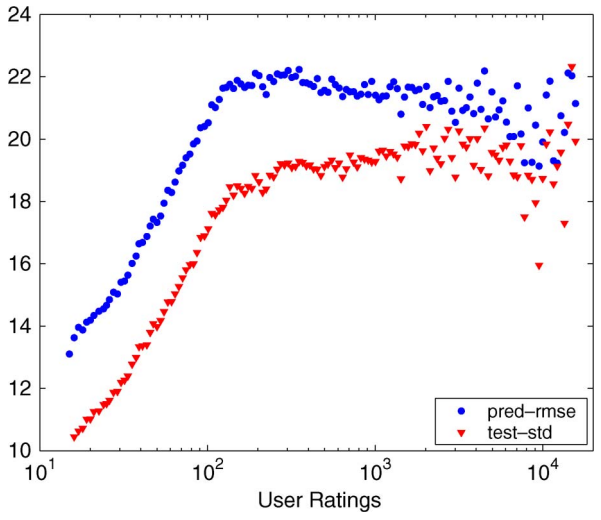

Fig. 12. ***Mean RMSE and test ratings standard deviations of users versus the number of their train set ratings.***

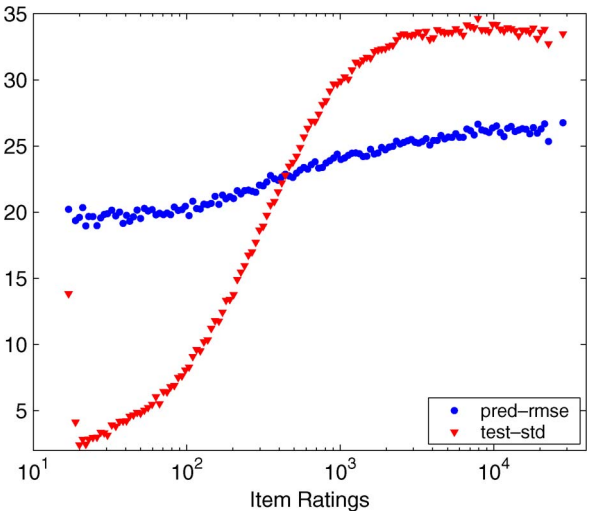

Fig. 13. ***Mean RMSE and test ratings standard deviations of items versus the number of their train-set ratings.***

Furthermore, it is obvious that users with many ratings are better modeled in the sense that a larger fraction of their rating variance is explained. An even more pronounced effect is observed when analyzing the items, as shown in Fig. 13. Whereas the mean standard deviation rises sharply with the number of train ratings, the slope of the mean RMSE is substantially more moderate, and the two cross for items with several hundred test ratings. Hence, while modeling high-variance heavily rated items is more challenging, the model is utilizing the added ratings in order to explain a growing fraction of the rating variability.

Last, we investigate the rate of performance degradation as the model is applied to events further into the future. To this end we associated each test example with the time elapsed since the last train set rating of the corresponding user. We then sorted the test ratings in an ascending order of the elapsed time and divided them into bins of 10 000 ratings each.

Fig. 14 depicts the mean RMSE and the elapsed time for each bin. Ratings with a zero elapsed time, which mostly correspond to user sessions artificially split between train and test set, were excluded from this analysis, as they are less relevant for most real recommender system. The plateau on the left part of the figure suggests that the performance of the model is stable for about three months since the time it was trained, whereupon the RMSE of its predictions gradually increases. Thus, the model needs updating only once in a month or so in order to exhibit uniform performance.

IX. CONCLUDING REMARKS

Media recommendation systems have amassed much interest in both research and industrial communities.

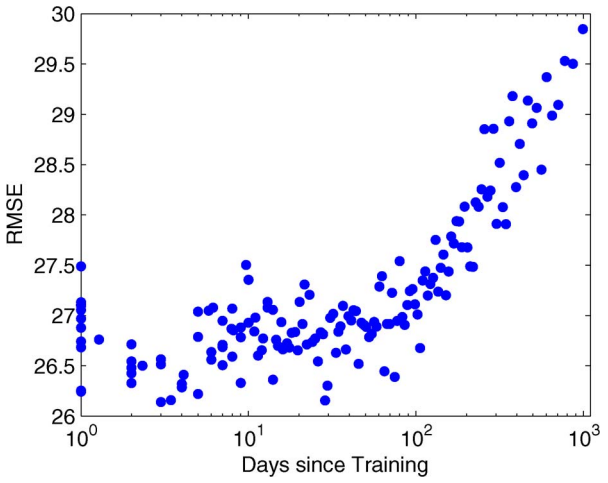

Fig. 14. ***RMSE versus number of days elapsed since last user's training example.***

Such systems allow users to cope with the ever growing product selection, and let media websites personalize their versatile inventory for the needs of each user. We have explored some of the challenges and main approaches used by media recommenders, while focusing on a concrete music recommendation example.

This work introduced a large scale music rating data set, which is currently the largest publicly available data set of its kind. We have chosen to model the data by CF methods, which can excel without relying on item identities or attributes. Our results show that detailed modeling of the various effects that naturally occur in this data through CF lead to improved predictive performance.

The music rating data set is characterized by an unusually high number of items (not only of users), representing a decade of Yahoo! Music service. Items are multityped, so users rate both tracks, albums, artists, and genres. The items are arranged within a music taxonomy, which we showed to be crucial in combating the sparse ratings associated with them. Many phenomena can be observed in the data, reflecting the life cycle of the Yahoo! Music service, usage patterns of different users, the evolution of song and artist popularity as well as psychological artifacts of the rating process. The data set also provides fine timestamps for each performed rating, thereby allowing a delicate temporal analysis. We have utilized this information for performing user rating session analysis. While there are differences between different media and services, we believe that the methods and insights illustrated in this paper on the music domain are broad enough to generalize to other domains.

This paper could not cover all considerations relevant to building a comprehensive recommender. We would like to highlight some additional important aspects, which deserve future research. An online deployment of the system would raise new challenges. One is evaluating the interplay between the recommender and the user feedback within the same environment. Another challenge is an ongoing update of the model, in order to incorporate new user feedback in a timely manner. A third challenge is devising a scalable method for matching each user with the few items he is predicted to like, without requiring the evaluation of each possible user–item pair.

While proper modeling improves recommendation relevance, not less important is a careful collection and interpretation of the various signals provided by the users. Recommender designers can greatly benefit by integrating various types of user feedback, thereby enriching user representation and helping in modeling users new to the system. For example, we believe there is much value in addressing additional kinds of user feedback, like collecting "play song" and "skip song" events. A major research challenge would be combining diverse signals within a single model. ■

REFERENCES

[1] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds., Recommender Systems Handbook. New York: Springer-Verlag, 2011.

[2] X. Amatriain, J. Bonada, À. Loscos, J. L. Arcos, and V. Verfaille, "Content-based transformations," *J. New Music Res.*, vol. 32, 2003.

[3] J.-J. Aucouturier and F. Pachet, "Music similarity measures: What's the use?" in *Proc. 3rd Int. Symp. Music Inf. Retrieval*, 2002, pp. 95–114.

[4] B. Logan, "Mel frequency cepstral coefficients for music modeling," in *Proc. Int. Symp. Music Inf. Retrieval*, 2000. [Online]. Available: http://ismir2000.ismir.net/

[5] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations," in *Proc. 25th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2002, pp. 253–260.

[6] D. Agarwal and B.-C. Chen, "Regression-based latent factor models," in *Proc. SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2009, pp. 19–28.

[7] Z. Gantner, L. Drumond, C. Freudenthaler, S. Rendle, and L. Schmidt-Thieme, "Learning attribute-to-feature mappings for cold-start recommendations," in *Proc. Int. Conf. Data Mining*, 2010, pp. 176–185.

[8] A. Gunawardana and C. Meek, "Tied Boltzmann machines for cold start recommendations," in *Proc. 2nd ACM Conf. Recommender Syst.*, 2008, pp. 19–26.

[9] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, 2003.

[10] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. Conf. World Wide Web*, 2001, pp. 285–295.

[11] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[12] B. M. Marlin and R. S. Zemel, "Collaborative filtering and the missing at random assumption," in *Proc. 23rd Conf. Uncertainty Artif. Intell.*, 2007, pp. 19–28.

[13] B. M. Marlin and R. S. Zemel, "Collaborative prediction and ranking with non-random missing data," in *Proc. 3rd ACM Conf. Recommender Syst.*, 2009, pp. 5–12.

[14] Y. Koren, "Collaborative filtering with temporal dynamics," in *Proc. SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2009, pp. 447–456.

[15] Z. Abbassi, S. Amer-Yahia, L. V. Lakshmanan, S. Vassilvitskii, and C. Yu, "Getting recommender systems to think outside the box," in *Proc. 3rd ACM Conf. Recommender Syst.*, 2009, pp. 285–288.

[16] M. Ge, C. Delgado-Battenfeld, and D. Jannach, "Beyond accuracy: Evaluating recommender systems by coverage and serendipity," in *Proc. 4th ACM Conf. Recommender Syst.*, 2010, pp. 257–260.

[17] S. M. McNee, J. Riedl, and J. A. Konstan, "Being accurate is not enough: How accuracy metrics have hurt recommender systems," in

Proc. Extended Abstr. Human Factors Comput. Syst., 2006, pp. 1097–1101.

[18] J. Bennett and S. Lanning, "The Netflix prize," in *Proc. KDD Cup Workshop*, 2007, pp. 3–6.

[19] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2008, pp. 426–434.

[20] M. Kendall and K. D. Gibbons, *Rank Correlation Methods*. New York: Oxford Univ. Press, 1990.

[21] M. Piotte and M. Chabbert, *The pragmatic theory solution to the Netflix Grand Prize*, 2009.

[22] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer J.*, vol. 7, no. 4, pp. 308–313, 1965.

[23] M. Wright, "Direct search methods: Once scorned, now respectable," in *Numerical Analysis*, D. Griffiths and G. Watson, Eds. Reading, MA: Addison-Wesley, 1995, pp. 191–208.

[24] D. Lee and M. Wiswall, "A parallel implementation of the simplex function minimization routine," *Comput. Econ.*, vol. 30, pp. 171–187, 2007.

ABOUT THE AUTHORS

Gideon Dror received the B.Sc. (*cum laude*) and Ph.D. degrees in physics from the Tel-Aviv University, Tel-Aviv, Israel, in 1985 and 1991, respectively.

From 1991 to 1993, he held a research position at the Nuclear Physics Department, Tel-Aviv University. He is an Associate Professor at the School of Computer Sciences of the Academic College of Tel-Aviv-Yaffo, Israel, where he also serves as the Head of the artificial intelligence program. Since 2010, he has been on academic leave, with Yahoo! Research, Haifa, Israel. His research interests include machine learning applications in community question answering services, recommender systems, activity recognition, marketing, medicine, psychology, natural language processing, computer vision, high-energy physics, and bioinformatics.

Noam Koenigstein received the B.Sc. degree in computer science (*cum laude*) from the Technion—Israel Institute of Technology, Haifa, Israel, in 2007 and the M.Sc. degree in electrical engineering from Tel-Aviv University, Tel-Aviv, Israel, in 2009. Currently he is working toward the Ph.D. degree in the School of Electrical Engineering, Tel-Aviv University.

In 2011, he joined the Xbox Live Machine Learning research team. His research interests include large-scale multimedia information retrieval and recommender systems, with specific focus on the music information retrieval.

Yehuda Koren received the Ph.D. degree in computer science from The Weizmann Institute, Rehovot, Israel, in 2003.

He joined Yahoo! Research, Haifa, Israel, in September 2008. Prior to this, he was a principal member of AT&T Labs-Research.

Dr. Koren was awarded the Netflix Grand Prize and also best paper awards at the 2005 IEEE Symposium on Information Visualization (INFOVIS), the 2009 Conference on Knowledge Discovery and Data Mining (KDD), and the 2011 ACM Recommender Systems (RecSys). He serves at senior program committees of conferences like KDD, International Conference on Data Mining (ICDM), RecSys, and Conference on Information and Knowledge Management (CIKM), and is on the editorial board of the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING.