# Informative Ensemble of Multi-Resolution Dynamic Factorization Models

Tianqi Chen*, Zhao Zheng, Qiuxia Lu, Xiao Jiang, Yuqiang Chen, Weinan Zhang
Kailong Chen and Yong Yu
Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai 200240 China
{tqchen, zhengzhao,luqiuxia,jiangxiao,yuqiangchen,wnzhang,chenkl,yyu}@apex.sjtu.edu.cn

Nathan N. Liu†, Bin Cao, Luheng He and Qiang Yang
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{nliu,caobin,luhenghe,qyang}@cse.ust.hk

## ABSTRACT

The Yahoo! music rating data set in KDD-Cup 2011 raises a number of interesting challenges: (1) It contains significantly larger number of users/items than all existing benchmark data sets. (2) The data covers a lengthy time period of more than 8 years. (3) Both date and time information are available for both training and test data. (4) The items form a natural hierarchy consisting of 4 types of items: genres, artists, albums and tracks. To capture the rich temporal dynamics within the data set, we design a class of time-aware hybrid matrix/tensor factorization models, which adopts time series based parameterizations and models user/item drifting behaviors at multiple time granularitys. We also incorporate the taxonomical structure into the item parameters by introducing sharing parameters between ancestors and descendants in the taxonomy. Finally, we have found that different types of models or parameter settings often work more(or less) effectively for users/items with certain characteristics. To more effectively combine multiple models, we design an *informative* ensemble learning framework, which augments model predictions by an additional set of *meta features* to represent the training instances for ensemble learning.

## Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval—*Information Filtering*

## General Terms

Algorithms, Experimentation

## Keywords

Collaborative Filtering, Recommender Systems

## 1. INTRODUCTION

Recommender systems have become an indispensable tool for helping users tackle with information overload as new content (e.g., news, products) are growing at an explosive rate. Collaborative filtering (CF) is one of the most promising technology for recommender systems. It works by discovering the correlation between users and items based on observed user preferences (i.e., ratings, clicks, etc.) so that unobserved user preferences can be interpolated from the observed ones. For example, the well known user based algorithm first finds the similarities between users based on their past ratings, then a target user's rating on a new item can be predicted from the ratings on that item from other similar users, also known as neighborhood. Thanks to the widely publicized Netflix prize competition, there has been a surging interests on CF algorithm design in the research community in recent years.

In contrast to the Netflix prize competition which deals with movie domain, this year's KDD-Cup[4] provides access to one of the largest music ratings data set, which contains nearly 3 times more ratings than the Netflix movie ratings data set. In addition to the differences in data size, we also noted several other interesting new challenges raised by the Yahoo! music data set:

- The data set spans a very long time period of more than 6,000 days and there are a significant number of users that have been active in the system for multiple years. People's tastes in music is arguably much more diverse and unpredictable than in movies due to the much larger number of choices available. In the mean time, the popularity of genres and artists is also fast changing. The effect of temporal dynamics has to be carefully taken into account to cope with various drifting and transient characteristics of users/items over such a long period of time.

- Each rating in both the training and test data set is associated with both date and time information. While previous work[7] has demonstrated the value of considering date information, there has been no previous work that jointly considers both date and time information in a single model. In the Netflix movie ratings systems, a user rarely rates movies during multiple time periods within the same day. However, as music services can be used by people through out a day, it is actually quite common to observe multiple ratings at different times of the same day in the Yahoo! music ratings data set. Intuitively, people's mood can naturally change over the day and people often prefer different types of music depending on his current context/status (e.g., at home vs. at work).

*The two groups from Hong Kong University of Science and Technology and Shanghai Jiao Tong University have contributed equally to the team's solution, and the author ordering does not indicate differences in contributions.

†Nathan N. Liu and Tianqi Chen are leaders of the team.

Therefore, considering time in addition to date as an additional context dimension can lead to more accurate modeling of user behaviors as we will demonstrate.

- Unlike traditional data sets which contains a single set of homogeneous items, the Yahoo! music data set consists of 4 types of items: genres, artists, albums and tracks, which are naturally linked together as a directed acyclic graph (DAG) based on predefined taxonomical relations. Intuitively, a user's preference for a particular track can be highly influenced by whether he likes the singer or the genre of the song. Similarly, if a use hates a particular artist, he can hardly rate any of his songs highly. To capture such correlated user preference over linked items, we also design a downward parameter sharing scheme such that the parameters of an item lower in the taxonomy will depend on the parameters of its ancestors, but not vice versa.

- The data set contains huge number of items, which is nearly 30 times more than that of the Netflix data set. As a result, although the number of ratings is much larger, the data sparsity is actually much more severe than the Netflix data set. In the mean time, this also leads a user/item population with highly diverse characteristics, which can be hardly served using a single model. As we will show, different models or parameter settings peform differently on particular user/item segments. To more flexibly fuse multiple models so as to serve different user/item segments differently, we design an *informative ensemble learning* strategy, which augments the model predictions with an additional set of *meta features* describing various user/item characteristics and then train a nonlinear model to make predictions based on this representation.

To solve the problems, we have designed various kinds of models targeting different features of the data set. We also implement a toolkit capable of handling large-scale data and implement different variants of model in one framework. Based on extensive large scale experiments, we find that the incorporating date, time and taxonomy information into the matrix factorization can lead to huge improvement over the basic matrix factorization and the informative ensemble of a collection of model can further significantly improve upon the best single model. Our final solution obtained an RMSE of 21.2634 on the held out test set, which achieved the 3rd place in Track 1 of KDD-Cup 2011.

# 2. PRELIMINARIES

In this section, we will give a concise overview of the matrix factorization model, which forms the foundation of our KDD-CUP solution. We will also describe some of its recent extensions specifically designed for collaborative filtering problems. Before formally describing the models, we first define our notational conventions. Suppose we have $m$ users and $n$ items. Generally, we use $u$, $v$ to denote users and $i$, $j$ to denote items. Day and time indices are denoted by $d$ and $t$ respectively. The ratings are arranged in a $m \times n$ matrix $R = r_{ui}$. In the matrix, some cells have values, the others are empty. We use $S$ to denote the set of $(u, i)$ indices for which ratings are observed.

## 2.1 Matrix Factorization for Collaborative Filtering

Matrix factorization is one of the state of the art model for large scale collaborative filtering as having been demonstrated by its success in the Netflix prize competition[8]. In its basic form, every user $u$ and item $i$ is associated with a vector $p_u, q_i \in \mathbb{R}^k$ and a scalar $a_u$ and $b_i$ respectively. The vectors $p_u$ and $q_i$ are generally referred to as the user and item factors where as $a_u$ and $b_i$ are referred to as the user and item biases. Under this model, user $u$'s rating on item $i$ is predicted via the following equation:

$$\widehat{r}_{ui} = f(p_u^T q_i + a_u + b_i) \tag{1}$$

where the function $f(\cdot)$ is a warping function that can map the a real value to a certain range [10]. In this work, we adopt the sigmoid function defined below:

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{2}$$

which can enforce that the predicted ratings ranges between 0 and 1, which can be then be easily mapped to the Yahoo! music data set's rating scale by multiplication with 100.

The model parameters are generally learnt by solving the following regularized least squares problem:

$$\min_{a_*, b_*, p_*, q_*} \sum_{(u,i)\in K}(r_{ui} - f(p_u^T q_i + a_u + b_i))^2 + \lambda_1(\sum_u ||p_u||^2 + \sum_i ||q_i||^2) + \lambda_2(||a||^2 + ||b||^2) \tag{3}$$

Here the constant $\lambda_1$ and $\lambda_2$ are parameters guarding the extent of regularization. When we choose not to regularize bias terms, we can set $\lambda_2$ to 0.

## 2.2 Integrate Neighborhood Information

The traditional neighborhood methods focus on computing the relationships between items or, alternatively, users. They are most effective at detecting very localized relationships and base predictions on a few very similar neighbors, but it may fall short when there is no or few observed ratings within the neighborhood of limited size. In contrast, the latent factor model is effective at capturing global information and have much better generalization capability due to its capability to more abstractly represent user/iems via learnable parameters.

The neighborhood based prediction model can be easily combined with the matrix factorization model additively:

$$\widehat{r}_{ui} = f\Big(a_u + b_i + p_u^T q_i + |N(u,i;k)|^{-\frac{1}{2}} \sum_{j\in N(u,i;k)} w_{ij}(r_{uj} - \bar{r}_u)\Big) \tag{4}$$

Here the set $N(u, i; k)$ consists of all the items that are selected as k-nearest neighbors of the item $i$ and have been rated by the user $u$. Traditional neighborhood methods rely on some arbitrary similarity metric such as Cosine or Pearson Correlation Coefficient to define the parameters $w_{ij}$. In this work, we treat $w_{ij}$ as free parameters which are learnt together along with the matrix factorization model parameters as first suggested by [1][6]. During computation, we only need to store and update the parameters for k-nearest neighbors of each item instead of all the item pairs, which results in $k \times n$ parameters where $n$ is number of items. The k-nearest neighbors are precomputed using a map-reduce cluster(Pearson Coefficient as metric). Then we precompute $N(u, i; k)$ on a single machine for each rating record. With the precomputed information, we can train the model using method described in Section 6 efficiently.

## 2.3 Utilizing Implicit Feedback

In general, implicit feedbacks can refer to any types of actions users performed on items other than ratings. Implicit feedback is a less precise indicator of user preferences but is generally more abundant and easier to obtain. For music recommendation, ideally the user's listening history such as how many times he listened to a song can be a very useful type of implicit feedback. Unfortunately,

such information is not available in the KDD-Cup data set. Instead, we consider another type of simple implicit feedback: whether a user rated an item or not. This type of implicit feedback allows us to utilize the test data, which consists of no ratings, in addition to the training data. To incorporate the information of implicit feedbacks, we adjust our estimation function as follows:

$$\widehat{r}_{ui} = f\Big(a_u + b_i + (p_u^T + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j^T) q_i\Big) \quad (5)$$

where $R(u)$ denote the set of items that are rated by the user $u$. Every item $j$ is assigned with a feature vector $y_j$ with the same dimension $k$ as the user/item factors $p_u, q_i$. $|R(u)|^{-\frac{1}{2}}$ is an empirical parameter for normalization of implicit feedbacks.

Equation 5 shows implicit feedback extension to basic matrix factorization. We point out that the extension to other models( e.g. neighborhood model ) is straightforward. In the rest part of the paper, we will also show extensions over basic matrix factorization for simplification.

We also use another kind of implicit feedback called "local implicit feedback". We don't include all the items rated by user into $R(u)$. Instead, we restrict $R(u)$ to the set of items user rate within current rating minute, excluding current item to be predicted. Due to the strong locality of the data set, the items user rated in nearby time(in same session) will provide more information than items user rated long time ago. The set of local implicit feedback is small so that it's possible to be directly optimized by stochastic gradient descend without using the speedup tricks for implicit feedback. It also yields similar performance gain as implicit feedback.

## 2.4 Model Learning

A very effective algorithm for solving the above least squares problem on large scale data set is the stochastic gradient descent algorithm, which simply randomly sweeps through the training ratings in $S$ one by one and takes a small gradient descent step on the relevant parameters along the direction that minimizes the error on each rating. We define $e_{ui} = r_{ui} - \widehat{r}_{ui}$ when $f(x) = x$. When $f(x) = sigmoid(x)$, we define $e_{ui} = (r_{ui} - \widehat{r}_{ui})\widehat{r}_{ui}(1 - \widehat{r}_{ui})$. The model parameters are updated based on the following equations:

$$\begin{aligned}
p_u &\leftarrow p_u - \gamma\big(\lambda_1 p_u - e_{ui}(p_u^T + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j^T)\big) \\
q_i &\leftarrow q_i - \gamma(\lambda_1 q_i - e_{ui} p_u) \\
y_j &\leftarrow y_j - \gamma(\lambda_2 y_j - |R(u)|^{-\frac{1}{2}} e_{ui} p_u) \\
w_{ij} &\leftarrow w_{ij} - \gamma(\lambda_2 w_{ij} - e_{ui}(r_{uj} - \bar{r}_u)) \\
a_u &\leftarrow a_u - \gamma(\lambda_3 a_u - e_{ui}) \\
b_i &\leftarrow b_i - \gamma(\lambda_3 b_i - e_{ui})
\end{aligned}$$

where the parameter $\gamma$ is the learning rate and $\lambda_1$, $\lambda_2$, $\lambda_3$ are regularization parameters for different type of parameters.

# 3. DYNAMIC FACTORIZATION MODEL

This section discusses a variety of techniques for utilizing the available date and time information associated with each rating. The key of incorporating temporal dynamics into the matrix factorization framework is to let the user/item factors and biases become time dependent. In the following two subsection, we discuss several schemes we attempted to design date-time dependent versions of user/item factors $p_u(d,t), q_i(d,t)$ and biases $a_u(d,t), b_i(d,t)$.

## 3.1 Date-Time Dependent User/Item Biases

The user bias $a_u$ in the basic matrix factorization is used to capture the general tendency of a user to assign high or low ratings, which is a rather person dependent effect. For example, there are very picky users who rarely give high ratings on items and there are casual users who likes to rate highly on most items they find acceptable. On the other hand, the item bias $b_i$ is used to capture the overall popularity of an item.

The date can have the following effects on the user/item biases as first suggested in [7]: Firstly, a user may change their rating scale over time as he becomes more adept at use rating to express his personal preferences or becomes more picky in the music listened to. Secondly, item popularity may change overtime, which is especially true for the highly dynamic music domain where new artists and genres quickly rise and fall whereas a small number of classics may remain popular overtime.

In addition to date, the time may also affect music ratings in the several ways. Firstly, a user may rate differently during different hours. For example, in the day time, as one is often busy with work at hand and less willing to spend time to rate items, a user may mostly assign low ratings for songs that he finds really annoying as a way to filter out bad songs. Secondly, different songs may be more(or less) popular during different hours. For example, dance music may be more preferred at night whereas light music may be generally more preferred in the morning.

### *3.1.1 Basic Date/Time Dependent Bias Model*

One simplest approach to design date-time dependent biases is to assume date and time have independent effect and then simply assign a separate bias value to each date and time point. An important decision in this scheme is the granularity at which to treat data and time. Our design is based on discretization of the involved 6,000 days and 24 hours of a day into equal sized bins such as every week and every 30 minutes and then designate a user/item bias for each date bin and time bin, which leads to the following formulation:

$$a_u(d,t) = a_u + a_{u,Bin(d)} + a_{u,Bin(t)} \quad (6)$$

where $Bin(d)$ and $Bin(t)$ denote the bin index associated with the particular date and time.

### *3.1.2 Piecewise Linear Function based Bias Model*

The previous bin based bias model is rather coarse and has the following disadvantages. Firstly, biases in different bins are learnt separately and the underlying temporal order of the bins are totally ignored. In a word, it essentially treats date and time as categorical information and can not reflect whether two bins are consecutive or far apart. Secondly, different date and time values falling into the same bin are forced to take on the same value regardless of whether they are near the beginning or end of the bin boundaries. It will be more desirable to have a model that considers temporal ordering while being able to more smoothly interpolate between discrete date and time points. To achieve this, we also design a more flexible piecewise linear time series based model for modeling the biases. In particular, we designate $K_a$ and $K_b$ equally distanced date and time "knots" $d_0, d_1, ..., d_{K_a}$ and $t_0, t_1, ..., t_{K_b}$ along the time line. We let each user and item have a bias value at each of these date and time points. Then given any particular date and time $d$ and $t$, the bias values are interpolated from the bias parameter of the preceding and succeeding date and time knots:

$$\begin{aligned}
a_u(d,t) &= a_u + \frac{(d - d^-)a_u(d^-) + (d^+ - d)a_u(d^+)}{\delta_{date}} \\
&\quad + \frac{(t - t^-)a_u(t^-) + (t^+ - t)a_u(t^+)}{\delta_{time}} \quad (7)
\end{aligned}$$

where $d^-$, $d^+$ denote the pair of consecutive knots between which the date value $d$ falls into. $t^-$ and $t^+$ are defined similarly. The value $\delta_{date}$ and $\delta_{time}$ denote the distance between consecutive date and time knots. It can be seen that under this model, the user and item biases are modeled by a piece wise linear function of date and time which are controlled by $K_a$ and $K_b$ knots respectively.

#### 3.1.3 Tensor based Bias Model

There are two major drawbacks of the previous two models for modeling date-time dependent user/item biases. Firstly, they are not capable of extrapolating into unseen date value or unseen hours. More specifically, the bias at any date later than the date of the latest rating or earlier than the first rating in the training data set can not be predicted at all. A second drawback is that they do not consider the interaction effects between date and time, such as songs with happy mood are more popular in the night during Christmas seasons. In this subsection, we describe a tensor factorization based framework for modeling date-time dependent user/item biases. The key idea is to treat the collections of date-time dependent biases $a_u(d,t)$ as being represented as a 3 dimensional tensor with each dimension corresponding to user, date and time respectively.

$$a_u(d,t) = w_u^T u(d) + w_u^T v(t) + u(d)^T v(t) \tag{8}$$

Here $u(d)$ and $v(t)$ are vectors corresponding to the latent factors of each date and time point and are treated in the same way as the date-time dependent biases via a piecewise linear function to model each dimension of the factors as a time series.

$$u_{d,k} = \frac{(d-d^-)u_{d^-,k} + (d^+-d)u_{d^+,k}}{\delta_{date}} \tag{9}$$

The time factors $v(t)$ are modeled similarly and the detailed formulas are omitted due to space limitation.

## 3.2 Date-Time Dependent User/Item Factors

Unlike the biases, the user/item factors captures the interaction between users and items, which we refer to as *second-order effect*. A date-time dependent user factor $p_u(d,t)$ allow us to capture user's changing preferences over different types of items over date and time. For example, a user may be into Dance/Rock music when he is young but may gradually start to like Jazz/Classical music as he grow older. Also, a user may prefer different types of music during working hours versus during night time. Similarly, the date-time dependent item factor $q_i(d,t)$ can enable the modeling of how an item may appeal to users with different characteristics over time. Furthermore we also assume the effect of date and time can be separated additively. This leads to the following parametrization:

$$\widehat{r}_{ui}(d,t) = f\Big(a_u + b_i + \dot{p}_u^T(d)\dot{q}_i + \ddot{p}_u^T(t)\ddot{q}_i + p_u^T q_i\Big) \tag{10}$$

Note that rather than replacing the original static user factor $p_u$ by a date-time dependent version as have been used in [7], we have created another two sets of separate factors $\dot{p}(d), \dot{q}_i$ and $\ddot{p}(t), \ddot{q}_i$ to capture date and time dependent second order effects. This has the flexibility of allowing us to model date-time dependent second order effects with factors of a smaller dimension than the static user/item factors $p_u, q_i$. This can significantly reduce the number of additional free parameters incurred by introducing date-time dependent factors and is an effective strategy to counter overfitting as we have found in experiments. The date-time dependent factors are modeled using the following techniques.

#### 3.2.1 Piecewise Linear Time Series

One way to model date-time dependent factors is via multidimensional piecewise linear time series. We assume that the user/item factors are less subject to abrupt changes and are expected to be varying more smoothly over time than the user/item biases. We therefore use two separate parameters $K'_a$ and $K'_b$ to set the number of knots on the time line for date-time dependent user/item factors. The resulting user factor is shown as follows:

$$\begin{aligned} p_u(d,t) &= p_u + \frac{(d-d^-)p_u(d^-) + (d^+-d)p_u(d^+)}{\delta_{date}} \\ &\quad + \frac{(t-t^-)p_u(t^-) + (t^+-t)p_u(t^+)}{\delta_{time}} \end{aligned} \tag{11}$$

Empirically, we found that a much smaller number of knots are needed for factors compared with the biases.

#### 3.2.2 Time Centered Factor

We also add a center decay-style factor. The user factor is given as follows

$$p_u(t) = p_u + e^{-\beta_u|t-c(u)|}p_u^{(c)} \tag{12}$$

Here $c(u)$ is the center point of user $u$'s time line and $p_u^{(c)}$ is the time centered user factor. The factors at other time points are interpolated from the centered user factor via an exponential decay function with an decay rate parameter $\beta_u$. $\beta_u$ and $p_u^{(c)}$ is trained by the data set. This center decay style time factor helps fix the limitation of the piecewise linear-style factor with one knot because interpolation mainly focus on two sides( beginning and end ). And time centered factor can help improve the modeling over center of time.

## 3.3 Time Dependent Neighborhood Model

In addition to the factors and biases, the neighborhood based component of our model can also take time into account by emphasizing more on the user's more recent ratings with an exponentially decay time weighting function. The idea is shown as follows

$$\begin{aligned} \widehat{r}_{ui} = & f\Big(a_u + b_i + p_u^T q_i \\ & + |N(u,i;k)|^{-\frac{1}{2}} \sum_{j \in N(u,i;k)} e^{-\alpha_u|\Delta t_j|}(r_{uj} - \bar{b}_u)\Big) \end{aligned} \tag{13}$$

$\Delta t_j$ is the amount of time between the time of $r_{uj}$ and the prediction time. $\alpha_u$ is initialized by 0 and trained by the data set. This setting can make the recent history contribute more influence over the prediction, and yield better prediction.

## 3.4 Session Locality

A quite common phenomenon that we have found in the KDD-CUP data set is the presence of *rating sessions*, in which users rate multiple items consecutively with only several minutes or less time apart. In addition, we also noted that consecutive ratings within the same session often have the same value most. We refer to this continuity in user's rating behavior as *session locality*.

To test if we can exploit the session locality effect to improve prediction accuracy on the test data, we did some analysis and find that 72% of the ratings in test set actually happened within 1 minute of some ratings in the training and validation set. To model session locality, we introduce a new *session bias*. More specifically, for each 1 minute interval where a user has a rating in both training and test data, we designate a bias $c_{u,s}$, where $s$ denote the index of such intervals extracted for user $u$. Despite the simplicity of this idea, we have found it to work surprisingly well.

## 3.5 Multi Resolution Dynamic Models

A critical design choice in both our date-time dependent factor and bias modeling is the number of knots, which controls the granularity at which the model deal with time varying user/item characteristics. Models with large number of knots can capture transient behaviors of users/items more effectively but may be subject to overfitting whereas a small number of knots can capture slowly drifting behaviors. As can be seen, there is a natural trade off between granularity and generalization. Even more trickier is the fact that the optimal granularity for different users and items can be quite different and a particular setting with the best overall accuracy can be suboptimal for some users and items. To tackle this difficulty, our solution is to train a collection of dynamic factorizations with increasing number of knots, each of which can capture temporal dynamics with different granularity. The final prediction will then be based on the combination of this collection of models rather than any single one. We refer to this framework as multi-resolution dynamic factorization model. In our experiments, we have found that the simple average of a collection of 8 models with number of knots ranging from 2 to 64 for the factors and 20 to 600 for the biases can improve upon the best single model's RMSE by over 0.3. In addition to the simple averaging method for combining different models, we also designed a much more effective ensemble learning method for model combination, which we will describe in section 7.

# 4. INCORPORATING TAXONOMICAL INFORMATION

One interesting property of Yahoo! music data set is that items belong to multiple categories, namely tracks, albums, artists, and genres. The relations among these items naturally form a directed acyclic graph (DAG) structure, where artists are above albums, which then contains tracks, whereas genres can annotate all the other three items. How to utilize this taxonomical structure thus raises another interesting challenge.

The categories of items and their parent-child relations can have the following effect. Firstly, the user's rating behavior on different types of items may be intrinsically different. Secondly, a user's preference over two closely tied items in the taxonomy is expected to be correlated. For example, a user who favors an artist is more likely to assign high ratings to his songs. Similarly, a user who hates a genre rarely give good ratings on songs in that genre. In the following subsections, we describe several further enhancements to the proposed matrix factorization model in order to exploit these two effects caused by taxonomy.

## 4.1 Category and Artist Bias

We first examine the user's rating behaviors on different categories of items. In particular, we examine the distribution of rating values on each of the four types of items, which are plotted in Figure 1. It can be clearly seen that users do exhibit highly distinct rating behaviors across item types. In particular, we can note ratings on genres are much more polarized than ratings on other types of items. To model this effect, we augment the user bias $a_u$ with 4 additional bias parameters for track, album, artist and genre respectively.

We also try to model the fact that many users may have their favorite artists, which leads to high rating over these artists' tracks and albums. We model this phenomenon by introducing a user-artist bias to the bias term. But it's not practical to add the bias term for every user artist pair. Since this can result in too many parameters to estimate. So we first find a potential set of possible

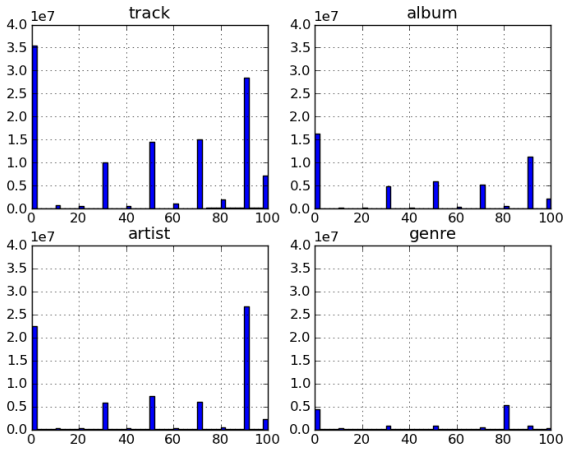


**Figure 1: The Distribution of Ratings Scores on Each Category of Items**

user artist pair from the data. We extract user artist pairs when a user rated an artists' tracks and/or albums more than a certain number of times and introduce a user artist bias when the pair are in this set. By this way, we can only estimate the parameters for potential user-artist pairs. This way also ensures we have enough data to estimate the parameters.

The extended predictor with these two new types of taxonomy induced biases is shown as follows

$$\widehat{r}_{ui} = f\Big(b_i + a_u + a_{u,Cat(i)} + a_{u,Art(i)} + p_u^T q_i\Big) \qquad (14)$$

where $Cat(i)$ and $Art(i)$ denote the category and artist index of item $i$ respectively.

## 4.2 Taxonomy Neighborhood

Traditional approach using neighborhood information requires pre-computation of k-nearest neighborhood set. Because Yahoo! Music data set provide the taxonomy information. It's natural to make use of taxonomy information to build the neighborhood set. During our experiment, we find users' rating over artist have strong correlation with their ratings over albums and tracks of the artist. So we try to add artist-track, artist-album neighborhood information. The idea is shown as follows:

$$\widehat{r}_{ui} = f\Big(a_u + b_i + p_u^T q_i + w_i(r_{u,Art(i)} - \bar{b}_u)\Big) \qquad (15)$$

## 4.3 Taxonomy Aware Predictor

One observation we made about the KDD-Cup data set is that the rating sparsity within different categories of items are highly different. From the Table 1 we can see that on average genres tend to receive the most ratings, followed by artists and then albums with tracks tend to have the least number of ratings on average. Such skewed data sparsity over different item will imply that user preference over more frequently rated items such as genres and artists can be estimated more robustly.

One way to incorporate the correlations between connected items is to directly let the predictor $\widehat{r}_{ui}$ to depend on item $i$'s parameters as well as the parameters associated with its ancestors. In particular, let $\check{r}_{ui}(d,t)$ denote our original design of the predictor (14) and

**Table 1: Rating Statistics Across Different Item Categories**

| Category | Num of Items | Total Num of Ratings | Avg Num of Ratings |
|---|---|---|---|
| Genre | 992 | 13,829,235 | 13,940 |
| Artist | 27,888 | 74,985,515 | 2,688 |
| Album | 88,909 | 48,485,593 | 545 |
| Track | 507,172 | 119,503,892 | 235 |

let $\mathcal{A}_i$ denotes the set of ancestors of item $i$ in the taxonomy, we design the taxonomy aware predictor based on the original predictor as follows:

$$\widehat{r}_{ui}(d,t) = \omega \cdot \check{r}_{ui}(d,t) + \frac{1-\omega}{|\mathcal{A}_i|} \cdot \sum_{j \in \mathcal{A}_i} \check{r}_{uj}(d,t) \tag{16}$$

where the parameter $\omega \in [0,1]$ controls the relative importance of the target item itself and its ancestors' information. With a smaller $\omega$ value, the predicted rating on an item will depend more on the predicted user ratings over its ancestors. Based on this formulation, an item's parameters are automatically coupled with the parameters of its ancestors. Thus when updating parameters based on the error on the target item, both the factors of the item itself and its ancestors will be updated.

# 5. COST SENSITIVE MODEL LEARNING VIA IMPORTANCE SAMPLING

Our analysis on the composition of the training and test data of the Yahoo! music data set reveals that all the ratings of a particular user in the test data are dated on or after the last day of their rating in the training and validation data. Furthermore, we also find that the validation data are all dated later than the training data as well. This indicates that the evaluation of this track actually emphasizes more on predicting each user's latest preferences. Given the lengthy period of time covered by the training data, it is important to inform the model learning strategy to focus more on the latest data rather than equally treating both historical and recent data. It should be noted that model has already incorporated some mechanisms for supporting date-time awareness, which should be able to remove the global influence of certain non-stationary data characteristics within particular time periods. However, our training objective is still trying to maximizing the model's accuracy over all training data. Therefore, we should adapt the training objective to make it *cost-sensitive*.

One common method to implement cost-sensitive model learning is simply to assign a weight to each training rating instance based on its recency and uses a weighted sum of prediction errors in the objective function[12]. Intuitively, each user's more recent ratings should be assigned a larger weight whereas his older ratings should have a smaller weight. As incorporating such instance weights did not change the additive form of the objective, the resulted stochastic gradient update rule can be easily adapted by scaling the step length $\gamma$ with an instance dependent weight $w_{ij}$. Unfortunately, we find this simple technique does not work well with the stochastic gradient descent algorithm and does not led to improved performance on the test set.

The basic stochastic gradient descent algorithm can be regarded as randomly sample a rating instance from a uniform distribution over the available training data in each update step. This inspires us to design an alternative strategy for cost sensitive learning based on *importance sampling* using a nonuniform distribution over the training data[13, 11]. In particular, we let the probability of each rating being sampled to be proportionate to its recency based weight, which naturally lets the algorithm more frequently update its parameters based on more recent data. While there exists many possible design choices on the recency weight, we nevertheless find the following simple strategy to work very well. More specifically, we sample the validation data for each user 3 times more often than the ratings in the training data. Using this simple importance sampling technique, we have achieved RMSE improvement ranging between 0.1 to 0.3 for various different models.

# 6. IMPLEMENTATION

In implementing the ideas proposed in previous sections, we face two major problems: (1) There are so many variants of models we want to experiment with. (2) Yahoo! music data set is so big that we must design a scalable solution. In the next two sections, we will discuss how to solve these problems.

## 6.1 Feature-based Matrix Factorization

We have already mentioned many variants of matrix factorization models in the previous sections. Instead of implementing each variant one a time, we design a toolkit to solve the following abstract model in Equation 17

$$\begin{aligned} y = & f\Big(\mu + \left(\sum_j b_j^{(g)}\gamma_j + \sum_j b_j^{(u)}\alpha_j + \sum_j b_j^{(i)}\beta_j\right) \\ & + \left(\sum_j p_j\alpha_j\right)^T \left(\sum_j q_j\beta_j\right)\Big) \end{aligned} \tag{17}$$

The input consists of three kinds of features $< \alpha, \beta, \gamma >$, we call $\alpha$ user feature, $\beta$ item feature and $\gamma$ global feature. $\alpha$ describes the user aspects that's related to user preference. $\beta$ describes the item properties. $\gamma$ describes some global bias effects. Figure 2 shows the idea of the model. We can find most of the models we described in the previous sections can fit into this abstract framework. Similar idea has been proposed before by libFM[9]. Compared with their approach, our model divides the features into three types, while there is no distinction of features in libFM. This difference allows us to include global feature that doesn't need to be taken into factorization part, which is important for bias features such as user day bias, neighborhood based features. The division of features also gives hints for model design. For global features, we shall consider what aspect may influence the overall rating. For user and item features, we shall consider how to describe user preference and item property better. Basic matrix factorization is a special case of Equation 17. For predicting user item pair $< u, i >$, we can define

$$\gamma = \emptyset, \alpha_h = \begin{cases} 1 & h = u \\ 0 & h \neq u \end{cases}, \beta_h = \begin{cases} 1 & h = i \\ 0 & h \neq i \end{cases} \tag{18}$$

If we want to integrate neighborhood information, simply redefine $\gamma$ as Equation 19. Here $index$ is a map that maps the possible pairs in k-nearest neighborhood set to consecutive integers.

$$\gamma_h = \begin{cases} \frac{r_{uj} - \bar{r}_u}{\sqrt{|N(u,i;k)|}} & h = index(i,j), j \in N(u,i;k) \\ 0 & otherwise \end{cases} \tag{19}$$

We can also include time dependent user factor by defining new $\alpha$. Taxonomy information can be integrated into $\beta$. We have opened the source code of our toolkit[1], see the footnote for the link. Using the toolkit, we can implement most of the described ideas simply by generating features.

[1] http://apex.sjtu.edu.cn/apex_wiki/svdfeature

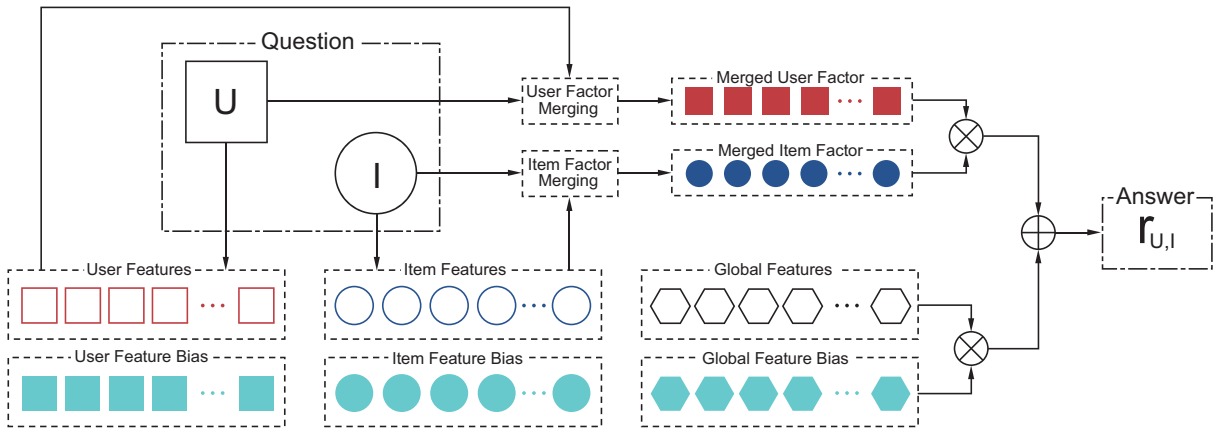


**Figure 2: Feature-based matrix factorization**

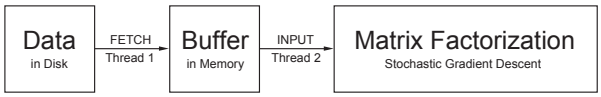


**Figure 3: Execution pipeline**

## 6.2 Input Buffering and Execution Pipeline

Because our training data can be extremely large in real applications, we can't load all of data into memory. In our approach, we store the data into a buffer file in hard-disk. The data is shuffled before storing. Then the training program linearly iterate over the buffer and update the model for each training sample. This approach allows us to do training as long as the model fit into memory.

Storing data into hard-disk can solve the problem of large training data size. However, it introduces additional cost of hard-disk reading. To minimize the cost of I/O, we use a pre-fetching strategy: An independent thread is created to fetch the data in hard-disk into a memory queue. At the same time, the training thread reads the data from memory queue and update the model. The procedure is shown in Figure 3.

This pipeline style of execution removes the burden of I/O from training thread. As long as I/O speed is similar to(or faster than) training speed, the cost of I/O is negligible. With input hard-disk buffering and pipeline execution, we can train a single model with 22.17 RMSE(model 11,Table 3) over test data using less than 2GB memory in 1 day.

# 7. INFORMATIVE ENSEMBLE LEARNING

As we pointed earlier, different model design choices and parameter settings may have their respective pros and cons. The single model with the best overall performance over a large population of users and items may not be the most accurate on every of the instances. The different models often have their own advantages under different conditions. For example, the empirical study carried by Cremonesi et. al.[3] demonstrated that for the matrix factorization model to perform well on tail items (i.e., items with few ratings) a larger dimensionality $K$ is often more desirable, whereas large $K$ often leads to overfitting on the head items. Similarly, the increased parametric complexity of a dynamic factorization model with finer temporal resolution can handle transient user/item characteristics well but are also more prone to overfitting. On the other hand, dynamic models with coarser temporal resolution can handle drifting behaviors well but not transient behaviors. The usage of taxonomical information is faced with similar trade off. For an item with sufficient ratings, its item factor may be learnt more effectively without imposing any correlation or dependence on the factors of its ancestors, whereas rarely rated ones should be able to benefit more from absorbing information from the ancestors. These forms the basic motivation for using the ensemble approach to combining multiple models, which has been demonstrated to be highly effective in the Netflix prize competition already[5].

In particular, we consider the stacking approach to ensemble learning, which first learn a set of models (i.e., component models) using the original training data and then learn another regression model with the component models' predictions as input features using an additional set of validation data[2]. Once the regression model for model combination is learnt, the component models are then retrained using the combined training and validation. In the end, the ensemble regression model is used to combine component models' predictions on the test data to make the final predictions.

Traditional methods for ensemble learning for collaborative filtering [5] only use the component models' predictions as the input features. Each training instance is of the format $(< \widehat{r}^0_{ui}, \widehat{r}^0_{ui}, ..., \widehat{r}^n_{ui} >, r_{ui})$, where $\widehat{r}^t_{ui}$ denote the prediction for $(u, i)$ pair by the $t$-th model. In this work, we have devised a slightly more complicated stacking approach by augmenting the aforementioned representation with an additional set of $p$ meta features, which leads to the following instance representation$(< f_1, f_2, ..., f_p, \widehat{r}^0_{ui}, \widehat{r}^0_{ui}, ..., \widehat{r}^n_{ui} >, r_{ui})$. We then learn a nonlinear regression model using gradient boosted regression tree and neural network. The meta features can describe different user/item characteristics and other more complicated metrics derived based on user, item and the training data. The purpose of incorporating meta features is to inform the stacking models about how different user/item segments or special conditions can be determined under which different subsets of models can be more effectively combined. As a result, we refer to our ensemble learning framework as *informative ensemble*.

We have used a collection of total 16 models with different parameter settings and design choices for building the ensemble model. The models are in three types in general:

- 100NN neighborhood model without factor: Although this type of model doesn't work well alone, it complements other models well.
- Factorization model using basic time bias and piecewise linear time bias: These type of model works well when there

**Table 2: Features Used in the Informative Ensemble Learning**

| ID | Feature Description |
|---|---|
| 1 | Number of ratings of the user |
| 2 | Number of ratings of the item |
| 3 | Mean rating of the user |
| 4 | Mean rating of the item |
| 5 | Variance of the ratings of the user |
| 6 | Variance of the ratings of the item |
| 7 | Number of days on which the user have ratings in the training data |
| 8 | Number of days between the user's first and last rating in the training data |
| 9 | Whether the user has any ratings on any ancestors of the item in the training data |
| 10 | Whether the user has any ratings within 5 minutes on the same day as the target rating |
| 11 | Whether the user has any ratings in the same hour as the target rating |
| 12 | Number of available neigborhood information(How many items $u$ rate in $N(u, i; k)$) |

is enough data to estimate the bias parameters. However, it doesn't work well when the time slot corresponds to few training data .

- Factorization model using tensor time bias: Complements with previous style of time bias. Can work even when user didn't rate in the corresponding time slot.

During our work, we have designed a total of 12 meta features, which are described in Table 2. Some of the meta-features are designed to show possible difference in models. For example, 100NN model may work well when many neighborhood information available, tensor bias model may work better in test data far from training data. These intuitive explanation justifies why our informative ensemble can improve the performance of our final result.

Both Gradient Boosted Regression Tree (GBRT) and Neural Networks (NN) were used for training the regression model. For GBRT, we set the maximum depth of each tree to be 6 and the number of trees to be 40, which achieved a test RMSE of 21.2989. The NN model consists of one hidden layer with 50 units with sigmoid activation functions, which obtained a test RMSE of 21.3257. Our final solution is a simple average of the predictions by the two ensemble model, which achieved a final performance of 21.2634.

# 8. CONCLUSION

In this paper, we describe the design of matrix factorization models and ensemble learning methods for accurate rating prediction on the Yahoo! music data used for KDD-Cup 2011. We study different extensions of the matrix factorization to handle temporal dynamics and taxonomical information. We also propose a meta-feature based ensemble learning framework for combining multiple models. Experimental results on KDD-Cup data sets demonstrated the effectiveness of the various techniques proposed.

# 9. ACKNOWLEDGEMENT

The team is supported by grants from NSFC-RGC joint research project HKUST 624/09 and 60931160445. We greatly appreciate the computing support from Professor Lin Gu.

# 10. REFERENCES

[1] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM*, pages 43–52, 2007.

[2] L. Breiman. Stacked regressions. *Mach. Learn.*, 24:49–64, July 1996.

[3] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, pages 39–46, 2010.

[4] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The Yahoo! Music Dataset and KDD-Cup'11. In *KDD-Cup Workshop 2011*, 2011.

[5] M. Jahrer, A. Töscher, and R. Legenstein. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 693–702, New York, NY, USA, 2010. ACM.

[6] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[7] Y. Koren. Collaborative filtering with temporal dynamics. In *Proc. of SIGKDD 2009*, 2009.

[8] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[9] S. Rendle. Factorization machines. In *Proceedings of the 10th IEEE International Conference on Data Mining*. IEEE Computer Society, 2010.

[10] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Neural Information Processing Systems*, 2007.

[11] V. S. Sheng and C. X. Ling. Roulette sampling for cost-sensitive learning. In *ECML*, pages 724–731, 2007.

[12] K. M. Ting. Inducing cost-sensitive trees via instance weighting. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, PKDD '98, pages 139–147, London, UK, 1998. Springer-Verlag.

[13] B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 435–, Washington, DC, USA, 2003. IEEE Computer Society.

# APPENDIX

In this appendix, we summarize our experiments and discuss the effectiveness of the methods. To show the effectiveness of each component, we add one component at a time, until we get our best single method. Table 3 shows the result of the experiments. The number of latent factors is fixed at 64. We left the some slot empty because we didn't submit corresponding test prediction. To train the models for test prediction, we use importance sampling by default. We can find the biggest performance gain was achieved by the time aware modeling. The neighborhood information, implicit feedback and category information also give improvement. And these improvements do not conflict with each other so they can be composite together to give a better predictor. Our best single predictor submitted is given by model 12 in Table 3 with 512 latent factors.

**Table 3: Performances of Different Single Models**

| ID | Description | Reference Section | Validation RMSE | Test RMSE |
|---|---|---|---|---|
| 1 | basic matrix factorization | section 2.1 | 21.84 | - |
| 2 | 1 + item day bias (binsize=40 days,no time bin) | section 3.1.1 | 21.74 | - |
| 3 | 2 + user day bias (binsize=1 day, no time bin) | section 3.1.1 | 21.09 | - |
| 4 | 3 + category and artist bias | section 4.1 | 21.04 | 23.01 |
| 5 | 4 + user tensor-bias (binsize=30 days) and factor (binsize=150 days) | section 3.1.3 | 20.37 | 22.56 |
| 6 | 4 + user piece-wise linear bias and factor over day(1 bin) | section 3.1.2,3.2.1 | 20.80 | 22.83 |
| 7 | 6 + taxonomy aware predictor($\omega = 0.1$) | section 4.3 | 20.73 | 22.68 |
| 8 | 7 + taxonomy neighborhood | section 4.2 | 20.59 | 22.58 |
| 9 | 8 + session locality | section 3.4 | 20.38 | 22.44 |
| 10 | 9 + local implicit feedback | section 2.3 | 20.15 | 22.28 |
| 11 | 10 + 10-nearest neighborhood | section 2.2 | 20.04 | 22.17 |
| 12 | 11 + time center factor | section 3.2.2 | 19.97 | 22.12 |
| 13 | add time dependency to neighborhood in 11 | section 3.3 | 19.94 | - |

**Table 4: Component Models Used in the Ensemble**

| ID | Description | Validation RMSE |
|---|---|---|
| 1 | matrix factorization with category and artist bias (100 latent factors) | 21.45 |
| 2 | matrix factorization with category and artist bias (200 latent factors) | 21.37 |
| 3 | 1 + category-artist bias+taxonomy aware predictor ($\omega = 0.1$) | 21.33 |
| 4 | 1 + category-artist bias+taxonomy aware predictor ($\omega = 0.2$) | 21.56 |
| 5 | 2 + category-artist bias+taxonomy aware predictor ($\omega = 0.1$) | 21.22 |
| 6 | 2 + category-artist bias+taxonomy aware predictor ($\omega = 0.2$) | 21.44 |
| 7 | 2 + tensor date bias (binsize 10 days) and piecewise date linear factor(binsize 150 days) | 20.58 |
| 8 | 2 + tensor date bias (binsize 20 days) and piecewise date linear factor(binsize 300 days) | 20.67 |
| 9 | 2 + tensor date bias (binsize 40 days) and piecewise date linear factor(binsize 600 days) | 20.86 |
| 10 | 7 + implicit feedback | 20.44 |
| 11 | 2 + tensor date-time bias (binsize 10 days, 15 mins) and piecewise date-time linear factor(binsize 150 days, 2 hours) | 20.39 |
| 12 | 2 + tensor date-time bias (binsize 20 days, 30 mins) and piecewise date-time linear factor(binsize 300 days, 4 hours) | 20.44 |
| 13 | 2 + tensor date-time bias (binsize 40 days, 1 hour) and piecewise date-time linear factor(binsize 600 days, 8 hours) | 20.69 |
| 14 | 12 + implicit feedback | 20.26 |
| 15 | model 12 in Table 3, 512 latent factors | 19.91 |
| 16 | 100-nearest neighbor information, with basic time bias, no latent factor | 21.61 |