Combining Factorization Model and Additive Forest for Collaborative Followee Recommendation

Tianqi Chen, Linpeng Tang, Qin Liu, Diyi Yang, Saining Xie, Xuezhi Cao, Chunyang Wu, Enpeng Yao, Zhengyang Liu, Zhansheng Jiang, Cheng Chen, Weihao Kong, Yong Yu

ACMClass@SJTU Team, Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai 200240 China

{tqchen,TLP, lqhl, yangdiyi, xiesaining, cxz, chunyang, yaoenpeng,liuzhengyang,chencheng, jzhsh1735, kongweihao, yyu}@apex.sjtu.edu.cn

ABSTRACT

Social networks have become more and more popular in recent years. This popularity creates a need for personalization services to recommend tweets, posts (information) and celebrities organizations (information sources) to users according to their potential interest. Tencent Weibo (microblog) data in KDD Cup 2012 brings one such challenge to the researchers in the knowledge discovery and data mining community. Compared to traditional scenarios in recommender systems, the KDD Cup 2012 Track 1 recommendation task raises several challenges: (1) Existence of multiple, heterogeneous data sources; (2) Fast growth of the social network with a large number of new users, which causes a severe user cold-start problem; (3) Rapid evolution of items' popularity and users' interest.

To solve these problems, we combine feature-based factorization models with additive forest models. Specifically, we first build factorization models that incorporate users' social network, action, tag/keyword, profile and items' taxonomy information. Then we develop additive forest models to capture users' activity and sequential patterns. Because of the additive nature of such models, they allow easy combination of the results from previous factorization models. Our modeling approach is able to utilize various side information provided by the challenge dataset, and thus alleviates the cold-start problem. The new temporal dynamics model we have proposed using an additive forest can automatically adjust the splitting time points to model popularity evolution more accurately. Our final solution obtained an MAP@3 of 0.4265 on the private leader board, giving us the first place in Track 1 of KDD Cup 2012.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Experimentation

Keywords

Social Recommendation, Factorization Model, Additive Forest

1. INTRODUCTION

With the growth of social networking sites such as Twitter, Facebook and Tecent Weibo, social networks have become part of people's daily life. People spend lots of time on social networks to share their feelings and opinions with friends and get the latest information on their idols. However, the popularity of social networks also brings information overload; users need to pick out the pieces of information they want from those they do not need. This creates a need for personalization services. Recommender systems that can recommend friends, tweets and other information to users according to their potential interest can help users to find what they need and improve the user experience. Tencent Weibo data[10] in KDD Cup 2012 brings one such challenge. The task is to predict which users (or information sources) one user might follow. Recommendation in social networks are different from traditional recommendation tasks. Compared to traditional scenarios in recommender systems, social network recommendation raises several challenges:

- Existence of multiple, heterogeneous data: Besides the training log data, the track1 data set in KDD Cup 2012 has social network, user profile, user action, item taxonomy and other information such as tag and keywords. These data can not be directly used as training data, but they give extra information about users' profile and items' properties. How to utilize the rich information available in social network to enhance recommendation becomes an important issue.

- Fast growth of the social network with many new users; social networks grow fast, and new users are joining everyday. In the track1 dataset, more than 70% users in the test query do not have any training log history. This creates a severe cold-start problem for the recommendation task.

- The rapid evolution of the social network: the users' interest in a social network changes frequently, as do item popularity. We find that a model trained from one weeks' data can have much lower prediction accuracy for latter weeks than for the same week. This rapid evolution property poses a challenge for time-aware modeling.

To solve these challenges, we combine two kinds of useful models: feature-based matrix factorization and additive forest. We use feature-based matrix factorization model to incorporate side information such as users' social network, action, keyword/tag and items' taxonomy information. We also develop additive forest models

to incorporate users' profile, activity and sequential patterns. The two kinds of models each have their own advantages : the factorization models are good at handling sparse matrix data (such as users' following information, keywords, etc.) and additive forest models are more suitable for characterizing continuous features (such as timestamp, user's age, etc.). The two models complement each other and the combination produces more accurate predictions. With respect to the three challenges, our solution gives the following contributions: (1) Our model fully utilize the heterogeneous data sources available; (2) We solve the cold start problem by user modeling via profile and social network information; (3) We develop a new time-aware model that can better capture items' popularity evolution.

Our final solution obtained an MAP@3 of 0.4265 on the private leader board, which achieved the first place in Track 1 of KDD Cup 2012.

The remainder of the paper is organized as follows. We introduce the general modeling methods used in our solution in Section 2. We then show how to incorporate different kinds of information available to build specific models in Section 3. Then experimental results are presented in Section 4. And finally, we conclude this paper in Section 5.

2. MODELING METHODS

In this section, we will introduce the various modeling methods we have used in the competition. We will first discuss the loss function we have used, and then three modeling methods that we found very useful—feature-based matrix factorization, bilinear models and tree-based models.

2.1 Loss Function

In all the modeling methods in this section, for a user-item pair (u, i), we will try to give a predicted rating $\hat{r}_{ui}$ that indicates the preference of u towards i. The loss function specifies how close our predictions are to the actual result, and all learning methods try to minimize the loss on the training set.

2.1.1 Classification

Since each record in the training set only has a binary result (accept or reject), it is natural to view it as classification problem. We use $r_{ui} \in \{-1, +1\}$ to denote whether user u accepts the recommendation of item i. The 0-1 classification loss is presented in the following equation

$$L_{ui}^{0-1} = \delta(r_{ui}\hat{r}_{ui}),\ \ \delta(x) = \begin{cases} 0 & x \geq 0 \\ 1 & x < 0 \end{cases} \tag{1}$$

Where δ is the 0-1 decision function. Since δ is not differentiable, we usually replace it by a convex surrogate loss function, which we denote by $\mathcal{C}$. There are two popular choices of surrogate loss functions. One is hinge loss, which corresponds to max-margin optimization:

$$\mathcal{C}(x) = max(1 - x, 0) \tag{2}$$

Another is logistic loss, which corresponds to maximum likelihood estimation of maximum entropy model:

$$\mathcal{C}(x) = \ln(1 + e^{-x}) \tag{3}$$

We note that these surrogate loss function can also be used for the ranking loss in later subsections. We will keep the notation $\mathcal{C}$ for surrogate loss function in our discussions in the following subsections.

2.1.2 Pairwise Ranking

In this task, we need to rank the top-3 items for each user, so the task is actually a learning to rank task. There has been lots of previous work on collaborative ranking[11][13]. Most pairwise ranking models try to minimize the number of reverse-pairs. So for each user u, if she rates i higher than j, we expect that the model will give $\hat{r}_{ui} > \hat{r}_{uj}$. The loss function for u is:

$$L_u^0 = \sum_{(i,j):r_{ui}>r_{uj}} \delta(\hat{r}_{ui} - \hat{r}_{uj}) \tag{4}$$

which is equivalent to maximizing the empirical AUC (Area under curve).

(4) can be very difficult to optimize because of its non-continuity. So we smooth it with

$$L_u = \sum_{(i,j):r_{ui}>r_{uj}} \mathcal{C}(\hat{r}_{ui} - \hat{r}_{uj}) \tag{5}$$

When a user has n positive ratings and m negative ratings, by (5) we need mn pairs to evaluate the pairwise ranking loss. This can be costly when a user has many ratings. In this case, we just sample a subset of the pairs.

Pairwise ranking models outperform classification models significantly in our experiments. This may be partially explained by the fact that this method automatically rebalances the data which contains far more negative ratings than positive ones. Note that when we sample a pair, it has exactly one positive rating and one negative rating.

2.1.3 LambdaRank: Listwise Ranking

Although pairwise ranking can effectively optimize the AUC, it may not be suitable for optimizing many other ranking measures. In KDD Cup 2012 Track 1, for example, MAP@3 is used as the ranking measure, which emphasizes the items ranked at the top of the list. LambdaRank [1] is a listwise learning method that can effectively optimize such complex measures. Specifically, the loss function becomes

$$L_u = \sum_{(i,j):r_{ui}>r_{uj}} \Delta_{ij}\mathcal{C}(\hat{r}_{ui} - \hat{r}_{uj}) \tag{6}$$

where Δ_{ij} is the difference in the ranking measure of two ranking lists obtained by the current predictions. Let L_{ij} be the ranking list by current predictions where we put i before j (we may swap the positions of i and j if i is ranked after j), and L_{ji} be the list where we put i after j. Δ_{ij} is just the difference in the ranking measure between L_{ij} and L_{ji}. For our task, we define Δ_{ij} as difference of MAP (mean average precision, MAP@∞), instead of MAP@3. This is because MAP@3 ignores the items not on the top-3 list, and will miss potential useful pairs. MAP LambdaRank gives nonzero Δ_{ij} for all pairs, and is still closer to MAP@3 than the pairwise approach.

2.2 Feature-based Matrix Factorization

Factorization models are an important class of model for collaborative filtering. The plain matrix factorization model[9] predicts $\hat{r}_{ui}$ by

$$\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i \tag{7}$$

where $\mathbf{p}_u, \mathbf{q}_i$ are both low-dimensional vectors (typically of 64 or 128 dimensions), denoting the latent factors of user u and item i, respectively. Combined with any of the loss functions proposed above, we can obtain a baseline method for the contest.

In the data set, aside from the training set, we also have many kinds of side information about the users, the items and their interactions. We use a general methodology to incorporate them into our model. Take user modeling for example. We can group the users into many classes using their profiles and behaviors, and each user may belong to a subset of the classes. For example, the users with ages from $13-18$ belong to the "adolescent class", the female users belong to the "female class", and the users who have followed Kaifu Lee belong to the "Kaifu Lee's fans class". Use $C(u)$ to denote the classes u belongs to, we can build a class aware factorization model by adding the following user latent factor:

$$\mathbf{p}'_u = \sum_{c \in C(u)} \alpha_c^{(u)} \mathbf{p}_c \tag{8}$$

where $\alpha_c^{(u)}$ is u's feature weight coefficient to class c, and p_c is the latent factor for the category (to be learned).

We may roughly distinguish between two kinds of user-related data available in the data sets. The first kind is categorical features, such as gender/age/taxonomy, and one user has at most one non-zero coefficient in a group of features. The second kind is the keyword-like features, and a user can have many non-zero coefficients in a group. For example, the items a user has followed form such a group of features. For such keyword-like features, it is good practice to normalize the coefficient vector α to ensure $\|\alpha\|_2 = 1$. In this way, Equation 8 encompasses the famous "user feedback" model, which is also known as SVD++[7].

In addition to the user related features, we can also incorporate item features and dyadic features (i.e., the features for the pair (u, i)), and build a full feature-based matrix factorization model[2] as follows:

$$\hat{r}_{ui} = \left(\sum_{c \in C(u)} \alpha_c^{(u)} \mathbf{p}_c \right)^T \left(\sum_{c \in C(i)} \beta_c^{(i)} \mathbf{q}_c \right) + \sum_{c \in C(u,i)} \gamma_c^{(u,i)} g_c \tag{9}$$

Here α, β, γ refer to the user/item/dyadic feature coefficients, respectively; p, q refer to the latent factors of user/item features; and g refers to the dyadic bias term. We have implemented a general solver for this model and built specific models via incorporation of useful features.

2.2.1 *Efficient Parameter Training*

To update the model, we use the following update rule to do stochastic gradient descent training

$$\mathbf{p}_c \leftarrow \mathbf{p}_c + \eta \left(\hat{e}\alpha_c \left(\sum_{j \in C(i)} \beta_j \mathbf{q}_j \right) - \lambda_1 \mathbf{p}_c \right) \tag{10}$$

$$\mathbf{q}_c \leftarrow \mathbf{q}_c + \eta \left(\hat{e}\beta_c \left(\sum_{j \in C(u)} \alpha_j \mathbf{p}_j \right) - \lambda_2 \mathbf{q}_c \right) \tag{11}$$

$$g_c \leftarrow g_c + \eta \left(\hat{e}\gamma_c - \lambda_3 g_c \right) \tag{12}$$

Here $\hat{e} = -\partial_{\hat{y}} l(y, \hat{y})$ the negative gradient of the loss function over predicted value. η is the learning rate and the λs are regularization parameters that define the strength of regularization.

The time complexity of the stochastic gradient descent training is $O(N\bar{K})$, where N is number of instances in the training procedure and $\bar{K}$ is number of nonzero features in each instance. In this task, we can usually have a large $\bar{K}$(about $20-100$) introduced by each user since we need to incorporate users' follows, actions, and keywords.

To speed up training with many user features, we use a speedup technique for user feedback information. The update of each $\mathbf{p}_c$ after one step without regularization is

$$\Delta\mathbf{p}'_c = \eta\hat{e}\alpha_c \left(\sum_{j \in C(i)} \beta_j \mathbf{q}_j \right) \tag{13}$$

Using the definition of $\mathbf{p}'_u$ in Equation 8, the resulting difference in $\mathbf{p}'_u$ is given by

$$\Delta\mathbf{p}'_u = \eta\hat{e} \left(\sum_{c \in C(u)} \alpha_c^2 \right) \left(\sum_{j \in C(i)} \beta_j \mathbf{q}_j \right) \tag{14}$$

Given a group of samples with the *same user*, to get a new $\mathbf{p}'_u$, we do not need to update each $\mathbf{p}_c$. Instead, we only need to update $\mathbf{p}'_u$ using Equation 14. Furthermore, there is a relation between $\Delta\mathbf{p}'_u$ and $\Delta\mathbf{p}_c$

$$\Delta\mathbf{p}_c = \frac{\alpha_c}{\sum_{k \in C(u)} \alpha_k^2} \Delta\mathbf{p}'_u \tag{15}$$

These relations can be used to reduce the complexity of the training algorithm to $O(N(\bar{K} - \bar{K}_u) + |U|\bar{K}_u)$. Where $\bar{K}_u$ is the average number of user features and $|U|$ gives number of users in the training set. This can greatly reduce the computation cost when a large amount of user information is incorporated into the model.

2.3 Bilinear Models

The bilinear model is also a powerful method for modeling tabular data[12]. It can be incorporated into feature-based matrix factorization via global feature definition, but it can be viewed as an alternative to the factorization model, so we discuss it in an independent section. It is actually closely related to matrix factorization models. Bilinear models give

$$\hat{r}_{ui} = x_u^T W y_i \tag{16}$$

where x_u, y_i are the feature vectors of u and i, respectively, and W is the parameter matrix in the model.

In the matrix factorization model, $\hat{r}_{ui} = p_u \cdot q_i$. Let $w_{ui} = p_u \cdot q_i$, and let

$$x_u = (0, \ldots, 0, \underset{u\text{-th}}{1}, 0, \ldots, 0),$$

$$y_i = (0, \ldots, 0, \underset{i\text{-th}}{1}, 0, \ldots, 0),$$

then $x_u^T W y_i = w_{ui} = p_u \cdot q_i$, so bilinear models actually encompass matrix factorization models. Actually, matrix factorization models can be seen as a restricted form of bilinear models where the parameter matrix is required to be low rank. Compared to matrix factorization models, bilinear models have more predictive power, but also have more parameters and can lead to over-fitting. We only use bilinear models when W is not large.

2.4 Additive Forest Models

2.4.1 *Tree-based Models*

Tree-based models partition the feature space along the axes into a set of rectangles, and fit a simple model (in our case, a constant) in each one. A popular tree-based method called CART, is described in [5], Chapter 9.

Compared to factorization models, tree-based models can capture complex and sudden changes in the interaction between features and response variables automatically. For example, when incorporating the user age into the matrix factorization model, we

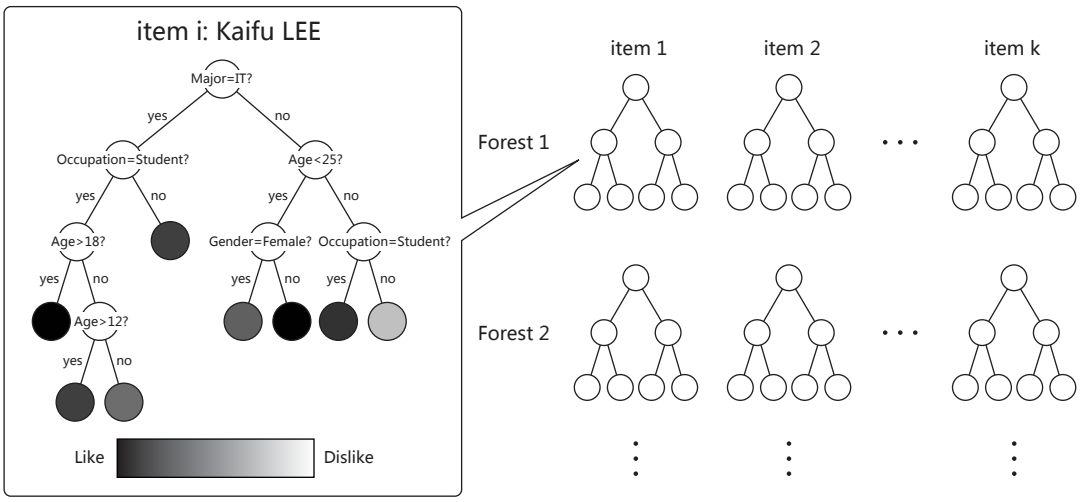


Figure 1: Illustrative Example of Additive Forest

have to manually partition the feature into many intervals and set up a user feature latent factor for each age group. Too many age groups may lead to too many parameters and over-fitting, while too few age groups might fail to fully capture the correlation and utilize the feature. Tree-based models automatically solve this issue; each rectangle is further partitioned if and only if the partition results in enough loss reduction and has enough supporting samples.

In addition, tree-based models only use a few parameters compared to matrix factorization or bilinear models, and we can easily control the complexity of the tree by its depth and number of supporting samples on each leaf. So tree-based models hardly ever over-fit, which can be a huge advantage compared to other models.

2.4.2 Additive Tree Methods

In many cases, a single tree is not powerful enough to efficiently describe our data. In particular, suppose we have data points generated by a linear model with m input variables $x_1, \cdots, x_m$. The output variable $y = x_1 + x_2 + \cdots x_m$. Although y can be perfectly modeled by linear regression, when using a single tree to fit the data, the tree size needs to grow exponentially to m in order for the model to be accurate.

Additive trees augment the power of single tree models by modeling the response with the sum of a series of trees.

$$y = \sum_{s=1}^{S} f_s(x) \tag{17}$$

where each f_s represents a tree model.

In the above example, if tree f_s partitions the input space only w.r.t x_s, then we just need m trees, each of size $O(1/\epsilon)$ to recover the data with relative error ϵ. So additive tree models can be much more efficient than a single tree model.

2.4.3 Additive Forest

In our scenario, there are complex user-item interactions. For example, a basketball star may be followed by lots of young men, while a handsome pop singer may be more favored by young women. It is hard to capture all these kinds of information using a single tree. We use an additive forest model to resolve the problem. Specifically, our model can be expressed by the following equation

$$\hat{r}_{ui} = \sum_{s=1}^{S} f_{s,root(i,s)}(x_{ui}) \tag{18}$$

Here $root(i,s)$ is the root for all ratings related to i in the s-th tree. For each s, the set $\mathcal{F}_s = \{f_{s,r} | \exists i, r = root(i,s)\}$ forms a forest with roots specified by $root(i,t)$. A special case is when $root(i,s) = i$, then our model becomes a sum of a series of forests with instances for each item to form trees. Figure 1 gives an illustrative example of an additive forest model.

$$\hat{r}_{ui} = \sum_{s=1}^{S} f_{s,i}(x_{ui}) \tag{19}$$

This model can capture the specific properties of each item, and build separate trees for each of them. It also allows different tree size according to the number of supporting instances, which can be tuned by the building algorithm. We also emphasize that we can define $root(i,s)$ to be the taxonomic parents of each items; this allows us to make use of taxonomy information and mine the shared properties between items in the same category. When $root(i,s)$ are same for all items, then we fall back to the additive tree model.

2.4.4 Optimization of Tree-based Models

In this section we will briefly discuss how to optimize the tree-based models, including additive tree and additive forest, using Newton's method. Readers are referred to [4] for the details (see the LogitBoost section). The basic criterion for optimizing such models is also *maximum likelihood*, as in feature-based matrix factorization models. For simplicity here we first consider the regression/classification problems, and ranking problems will be discussed later. The optimization objective is (see [5]):

$$\min \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \alpha |T| \tag{20}$$

where $l(\cdot,\cdot)$ denotes the loss function (for example, for logistic loss $l(y,\hat{y}) = \ln(1+e^{-y\hat{y}})$) and $|T|$ denotes the size of the trees/forests which controls model complexity. .

To exactly optimize (20), even for a single tree model, would be NP-Hard, so we use a greedy algorithm instead. Each time we

find a node and choose a split line across some axis that would reduce the loss most. So we may gradually grow the tree until the leaves reach a maximum depth or the support of each leaf is below a specified threshold. Then we shrink the tree by merging the nodes that reduce the loss least, until the smallest loss reduction is larger than α. This process will greedily optimize (20).

Now the central problem is, given a node and some axis (feature), how to efficiently choose the split-line that would reduce the loss most. We use Newton's method for this purpose. Suppose the support for the current node is

$$(x_1, y_1, \hat{y}_1), \cdots, (x_n, y_n, \hat{y}_n)$$

with $x_1 < x_2 < \cdots < x_n$, where x_i is the feature value, y_i the response value and $\hat{y}_i$ the current predicted value. Let

$$g_i = \frac{\partial}{\partial \hat{y}_i} l(y_i, \hat{y}_i), \quad h_i = \frac{\partial^2}{\partial \hat{y}_i^2} l(y_i, \hat{y}_i)$$

Denote $l_i(d) = l(y_i, \hat{y}_i + d)$, then by Taylor expansion

$$l_i(d) = l(y_i, \hat{y}_i) + g_i d + \frac{1}{2} h_i d^2 + o(d^2) \tag{21}$$

Now assume we have $x_s \in [x_k, x_{k+1})$, and we wish to minimize $\sum_{i=1}^{k} l_i(d_1) + \sum_{i=k+1}^{n} l_i(d_2)$. Using Newton's method we have

$$d_1^* = -\frac{\sum_{i=1}^{k} g_i}{\sum_{i=1}^{k} h_i}, \quad d_2^* = -\frac{\sum_{i=k+1}^{n} g_i}{\sum_{i=k+1}^{n} h_i} \tag{22}$$

with the approximate loss reduction $\sum_{i=1}^{k} \frac{1}{2} h_i d_1^2 + \sum_{i=k+1}^{n} \frac{1}{2} h_i d_2^2$. So we only need to sweep through the data points in the order of x_i, keeping a record of $\sum_{i=1}^{k} g_i$, $\sum_{i=1}^{k} h_i$, $\sum_{i=k+1}^{n} g_i$ and $\sum_{i=k+1}^{n} h_i$. This can be done in $O(n)$ time, assuming the data points are pre-sorted.

Assuming there are N samples in total with K features, since additive tree/forest models typically have constant depth ($4 \sim 10$) in our experiments, the total time for constructing a tree (or forest) would be $O(NK)$. We may further accelerate the algorithms by a subsampling technique. When constructing additive trees, in each iteration, we only sample a subset of the data points to construct the tree. This gives the stochastic gradient boosting method[3]. If we use ρ for sample ratio, then the time for constructing a tree is only $O(\rho NK)$. Finally, for ranking problems, we compute g_i, h_i from the pairwise ranking or LambdaRank as in section 2.1.2 and 2.1.3.

3. INFORMATIVE MODELS

We have discussed the general modeling techniques in the previous section. In this section, we show how to apply these techniques to build specific models to utilize available information.

3.1 Social Network

One of the most important user preference information sources in the provided dataset is users' social network data. Users can choose to follow friends or celebrities. This information may correlate with which recommendations the user will accept in the future. For example, a user who follows lots of football stars may be more likely to accept of recommendations of other football stars. There are two kinds of modeling approach to use this data. The first approach is a bilinear model:

$$\hat{r}_{ui} = \frac{1}{\sqrt{|F(u)|}} \sum_{j \in F(u)} W_{ij} + b_i \tag{23}$$

Where $F(u)$ is the set of items followed by u. It directly models the correlation between i and j by W_{ij}. The second approach is a factorized user feedback model:

$$\hat{r}_{ui} = \left(\frac{1}{\sqrt{|F(u)|}} \sum_{j \in F(u)} \mathbf{p}_j \right)^T \mathbf{q}_i + b_i \tag{24}$$

This model can be viewed as a factorized version of the bilinear model. It contains fewer parameters than the bilinear model and can learn the latent topic of items through the factorized parameters. This model works better than the bilinear model in our experiments.

Besides users' follow data, we can also get users' action data. A user can retweet, @ and comment on other users. The difference is that a user's action has counting information. Fanatic football fans may retweet football stars many more times than other persons they follow. We take this into account, and include users' action information in our model

$$\hat{r}_{ui} = \left(\frac{1}{\sqrt{|F(u)|}} \sum_{j \in F(u)} \mathbf{p}_j + \frac{1}{\|\alpha_u\|_2} \sum_{j \in A(u)} \alpha_{u,j} \mathbf{y}_j \right)^T \mathbf{q}_i + b_i \tag{25}$$

Where $A(u)$ is the set of items user u has actions with. $\alpha_{u,j}$ is the number of actions user u has over j. We use ℓ^2 norm to normalize the weight. The model in Equation 25 is a special case of Equation 9, and we can train the model using the methods discussed in section 2.2.1.

3.2 Age and Gender

The age and gender of a user could also indicate his/her interests to us. For example, a teenage boy usually likes sports and is more likely to follow a sports star, while a teenage girl may like music better and is more likely to follow the music stars. So users from different age and gender groups might have different preferences. We could use this information by introducing latent factors for these groups into the user latent factors, but as the number of groups is quite small, we find it is better to directly use the bilinear models.

We first partition the user ages into k bins, and then split each bin by gender, so there are in total $2k$ groups. Define $ag(u) : u \mapsto \{1, \cdots, 2k\}$ that maps u to the group he/she belongs to. We use $\mathbf{e}_{ag(u)}$ to denote a base vector with 1 on the $ag(u)$-th component and 0 on every other component. Use β_i to denote the coefficient vector for item i's latent factors. Note that in addition to a latent factor for each item, we also have latent factors for the keywords of the items (refer to section 3.3). So β_i would have non-zero components that correspond to its identity and its keywords, and possibly other item features. Then the bilinear model utilizing the user's age/gender information is as follows:

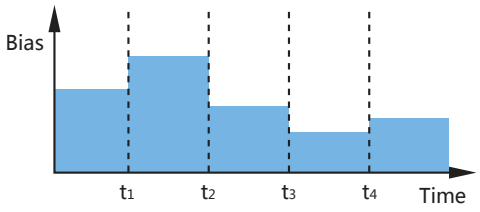
$$\hat{r}'_{ui} = \hat{r}_{ui} + \mathbf{e}_{ag(u)} W \beta_i \tag{26}$$

Here W is the parameter matrix defined in Section 2.3.

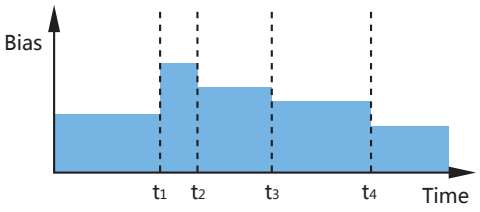
In the bilinear modeling of age and gender, we use the same partition by age and gender for all items. However, since different items may be favored by different kinds of people, a uniform partition may not be perfect for all the items. We use additive forest model to solve this problem (for simplicity we omit the item keyword feature in this model),

$$\hat{r}'_{ui} = \hat{r}_{ui} + \sum_{s=1}^{S} f_{s,root(i,s)}(x_{ag(u)}) \tag{27}$$

Where $x_{ag}(u)$ is a two dimensional feature vector that contains the age and value of u. This model can cut and select related age inter-



(a) Temporal Model via Item Time Bin



(b) Temporal Modeling via K-piece Step Function

Figure 2: Comparison of Two Temporal Models

vals and genders for each item, and overcome the disadvantage of bilinear model. In our final model we first use (26) in the factorization model and then boost (27) with additive forest models.

3.3 Keywords and Tags

The data sets also provide us with a lot of descriptive keywords for each user and item. For each user, we know the tags she put on herself and the keywords extracted from her tweets. Additionally, for each item, we know the keywords extracted from her description given by the Tencent Weibo officials. These keywords and tags describe the characteristics of the users and the items, so we also introduce them to the latent factors. Take the user latent factor for example,

$$\mathbf{p}'_u = \mathbf{p}_u + \frac{1}{\|w_u\|_2} \sum_{j \in K(u)} w_{u,j} \mathbf{y}_j \tag{28}$$

where $K(u)$ denote the set of keywords for user u, and $w_{u,j}$ the weight of keyword j for u. As we have mentioned, there are in total 4 sets of such keyword/tag features, and we add all of them to the user/item latent factors in the style of (28).

3.4 Taxonomy Information

In the provided data set, we also know the taxonomy information for each item. Specifically, we know i belongs to a depth-4 hierarchical category system, so $i \in c^1(i) \subset c^2(i) \subset c^3(i) \subset c^4(i)$. Naturally, we expect that items in the same category would have similar latent factors. We address this issue by setting up latent factors for the categories and sharing these latent factors among all items in the category[6]. So the latent factor for item i is:

$$\mathbf{q}'_i = \mathbf{q}_i + \mathbf{q}_{c^1(i)} + \mathbf{q}_{c^2(i)} + \mathbf{q}_{c^3(i)} + \mathbf{q}_{c^4(i)} \tag{29}$$

Note that the solution for this system is clearly undetermined because we could subtract a constant vector from the latent factor of a category and add that constant to the latent factors of all the items in that category, and the predictions would remain the same. This problem is solved by the ℓ^2 regularization on all the latent factors.

3.5 Temporal Dynamics

Temporal information plays a very important role in collaborative filtering models. Users' preference and items' popularity can change with time. Since users' behavior information is extremely sparse in this task, it is hard to model users' preference change. However, we can model items' popularity change using time-aware model,

$$\hat{r}'_{ui}(t) = \hat{r}_{ui} + b_{i,binId(t)} \tag{30}$$

Where $binId$ is a function that maps timestamp into corresponding time bin. This approach follows the methodology in Koren[8], and showed significant performance improvement in last year's KDD Cup[2]. The basic idea is to learn a localized item temporal bias in each time bin to capture the recent popularity of items. However, this approach is not perfect: the time bin size is fixed and needs to be tuned each specific dataset.

In the real world scenario, the popularity change of items can be different. Consider the following example: a football star may suddenly became popular in a recent week due to his perfect play in the European Cup; a pop star stays hot for long time and her popularity has not changed recently; a politician was rocked by a scandal a month ago and becomes less popular this month. To better model their popularity patterns, we need to use weeks as time bin unit for the football star, while months are more desirable for the politician. This example shows that we need to use different model settings for different items. With these observations, we propose a new temporal dynamics model as follows

$$\hat{r}'_{ui}(t) = \hat{r}_{ui} + f_i(t) \tag{31}$$

Where $f_i(t)$ is a k-piece step function. In the training step, we need to choose the function f_i to optimize the loss function. We compare the new approach with the old one in Figure 2. Figure 2(a) gives an example of a time bin model, with a predefined time bin size. Figure 2(b) shows an example of a new k-piece step function. The splitting points are adjusted during the training process to optimize the loss function. This makes our new model more flexible than approaches with a predefined bin size.

To further improve the model, we change the single step function to a sum of S step functions

$$\hat{r}'_{ui}(t) = \hat{r}_{ui} + \sum_{s=1}^{S} f_{s,i}(t) \tag{32}$$

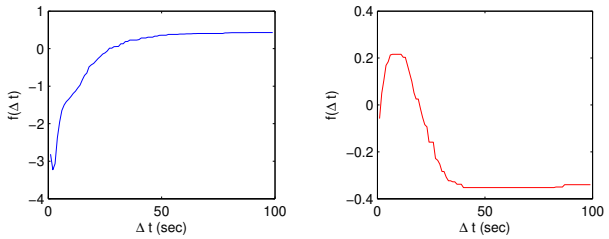
This new model can express more complex temporal patterns than the traditional fixed-bin approach as (30). It can be viewed as a special case of the additive forest model introduced in Section 2.4.3. We can use gradient boosting to find a good solution in functional space.

3.6 User Activity

Different types of users may have different kinds of microblog activity. For example, high school students may tend to use microblogs on the weekend since they are occupied by courses on weekdays. Our assumption is that the time activity pattern of users' microblog usage correlates to their identities and preferences. Unfortunately, the only timestamp information we can get is users' accept/reject activity, which is quite sparse. Nevertheless, we make use of this information to build a model that uses activity patterns. Specifically, we calculate the proportion of each users' activity which occurs during the weekends.

$$x_u = \frac{|R^{weekend}(u)|}{|R(u)|} \tag{33}$$

Where $R^{weekend}(u)$ is the set of user u's accept/reject actions during the weekend, and $R(u)$ is the set of all u's actions. Similarly, we also build a histogram of users' activity over the 24 hours in



(a) $\Delta t = t_{next} - t_{curr}$ (b) $\Delta t = t_{curr} - t_{prev}$

Figure 3: Single Variable Pattern Functions Learned

a day. These feature is then added into the additive forest model to improve the prediction. This modeling approach provides a minor improvement in the performance. However, we believe that if we can access more activity information such as timestamp of tweet actions, user activity modeling could potentially give more improvement.

3.7 User Sequential Patterns

Users' behaviors in a social network are not identical nor independent. Instead, they are series of related actions. Some users may continually reject recommendations until they find a satisfying one. Users' patience may also be changed by how many undesirable recommendations they have received recently. To capture these sequential patterns, we utilize an additive tree model

$$\hat{r}'_{ui}(t) = \hat{r}_{ui} + \sum_{s=1}^{S} f_s(x_{seq}) \tag{34}$$

The feature x_{seq} is intended to capture the local sequential feature of this recommendation, and it includes the time difference between the current recommendation record and the previous/next k records.[1] We also include each users' average click frequency within a session (a session is defined to be a time interval when a user has continual actions) into x_{seq}. The advantage of a tree-based model is that it automatically learns the non-linear relation of these sequential features to the acceptance rate.

Intuitively, the sequential features can capture cases when a user has found an item she likes, accepts it and stops processing recommendations. The features might also capture whether a user's current patience (reflected in her click frequency) will affect her acceptance rate. Since these sequential patterns are not item specific, we use additive trees (rather than additive forests) in our model for them.

To give illustrative examples of relationships between sequential features and prediction. We train several single variable predictors that take one sequential feature Δ as input and output $f(\Delta t)$

$$\hat{r}'_{ui}(t) = \hat{r}_{ui} + f(\Delta t), \;\; f(\Delta t) = \sum_{s=1}^{S} f_s(\Delta t) \tag{35}$$

We plot two of the single variable functions learned in Figure 3. Here t_{curr}, t_{next}, t_{prev} are timestamps of current, next and previous action respectively. Figure 3(a) can be intuitively explained by the possibility that some users may stop exploration when they find satisfying items. Figure 3(b) shows an interesting relationship; it may be interpreted as when users click too fast or too slow, it is less likely to be an acceptance. It may also related to a user's current patience. We also tried to include sequential features in matrix factorization models or linear models. However, the tree-based approach turns out to be the most natural and effective way to handle these features in our experiments.

[1] In our experiments we include the time difference between current record and the next 4 records the previous 2 records.

3.8 Final Joint Predictor

For completeness, we give the equation of our final predictor in this section. Our final predictor is as follows.

$$\begin{aligned}\hat{r}_{ui} = &\left(\sum_{c\in C(u)} \alpha_c^{(u)} \mathbf{p}_c\right)^T \left(\sum_{c\in C(i)} \beta_c^{(i)} \mathbf{q}_c\right) \\ &+ \sum_{c\in C(u,i)} \gamma_c^{(u,i)} g_c + \sum_{s=1}^{S} f_{s,root(s,i)}(x_{ui})\end{aligned} \tag{36}$$

This is a combination of feature-based matrix factorization and an additive forest model. The detailed configuration of features are already given by previous subsections. To train this model, we can first train the factorization part and keep the prediction as base line, then use gradient boosting to train the additive forest part. This model allows us to utilize the advantages of the two models, and make accurate predictions.

4. EXPERIMENTS

4.1 Experiment Setup

As it is time consuming to submit the predictions to the result of one method, we find it very important to make a reasonable validation set at the early stage of the competition. Note that the training set is collected from October 11th, 2011 to November 11th, 2011, and the test set from November 11th to November 30th. So there are in total 30 days of training data. Since the test comes right after the whole training set, we also make the validation set right after our private training set. Specifically, we divide the 30 days evenly into 5 periods (so 6 days for each period) and take the first 4 as the training set $\mathcal{S}$ and the last period as the validation set $\mathcal{V}$. In the experiments, we first train the models on the training set, use the validation result to determine the learning parameters (mainly learning rate and number of rounds), and then apply the same parameters to the models trained on the whole data set $\mathcal{S} + \mathcal{V}$ and get the predictions on the test set.

We conduct our experiments in an incremental way. We add each type of information to our model, and train a joint model using all the information. For the factorization models, we use feature-based matrix factorization to train the joint model. For the models that include an additive forest, we use the factorization model as a starting point and add forests to improve the results. For all the factorization models, the number of latent factor is set to 128. The learning rate is set to be sufficiently small for the model to converge. All the regularization parameters are set to 0.004. We use pairwise rank to train factorization models and use LambdaRank in additive forest training. Classification loss is not as good as learning to rank methods and is not used in our experiments.

We have tried ensemble methods but failed to improve the final joint model. So we do not include ensemble methods in our experiments. This is an interesting fact, it indicates that in this task, how to better utilize information is more important than combine different predictors using the same piece of information.

4.2 Results and Discussions

Table 1 shows the results of different methods on the data set. We can find that the results on the validation set are largely consistent with those on the test set. If one method improves the results of

No.	model	validation MAP@3	public MAP@3	private MAP@3
1	item bias	39.0%	34.6%	34.0%
2	1 + user follow + user action	42.1%	36.7%	35.8%
3	2 + user age/gender	43.5%	38.0%	37.2%
4	3 + user tag/keyword	44.2%	38.5%	37.6%
5	4 + item taxonomy	44.4%	38.7%	37.8%
6	5 + temporal dynamics	44.7%	39.0%	37.9%
7	6 + user activity/gender/age(addtive forest)	44.9%	39.1%	38.0%
8	7 + user sequential patterns	51.4%	44.2%	42.7%

Table 1: MAP@3 of different methods

the other on the validation set, it usually improves on the test set also, although the improvement is often less significant. This is expected. Since the social network is an evolving system, the more distant the future, the more difficult it will be to predict the user behavior.

Our method significantly improves the experimental results. We emphasize that the incremental approach here does not give an exact measurement of the impact of each method since the information of different models may correlate with each other, and it is extremely hard to increase the performance of later models by even a slight amount. The experimental results show that user modeling greatly helps the prediction. This is due to the sparsity of users' training information.

It is worth mentioning that our proposed new temporal dynamics model gives 0.3% improvement on MAP@3, while all our other trials of traditional time-aware models fail to yield improvement. This shows our model is potentially better than previous time-aware collaborative filtering methods. We will further study its properties in future work.

The combination of additive forest and factorization models yields the best performance we can get. While each of them is powerful enough as a stand alone model, they complement each other; feature-based matrix factorization can easily handle sparse matrix information such as a user social network, while an additive forest is good at dealing with continuous features such as age and timestamps. While it is possible to build a complex factorization model that can replace additive forest via feature engineering, we believe that combination of the two models is a more natural way to produce state-of-art results in this task.

5. CONCLUSION

In this paper, we describe our solution to followee recommendation to the Tecent Weibo dataset for KDD Cup 2012. We study different extensions of factorization models to handle information such as social network, user action and item taxonomy. We develop additive forest models to incorporate user profile, activity and sequential behavior patterns. We also propose a new time-aware model to capture items' popularity changes. Our final solution combines feature-based matrix factorization and additive forest models, which gives us the advantages of both models. Experimental results on track1 data sets demonstrated the effectiveness of the various techniques and models proposed by this paper.

Acknowledgement

The team is supported by grants from NSFC-RGC joint research project 60931160445.

6. REFERENCES

[1] C. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010.

[2] T. Chen, Z. Zheng, Q. Lu, X. Jiang, Y. Chen, W. Zhang, K. Chen, Y. Yu, N. Liu, B. Cao, L. He, and Q. Yang. Informative ensemble of multi-resolution dynamic factorization models. In *KDD-Cup Workshop*, 2011.

[3] J. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[4] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[5] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer Series in Statistics, 2001.

[6] N. Koenigstein, G. Dror, and Y. Koren. Yahoo! music recommendations: modeling music ratings with temporal dynamics and item taxonomy. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 165–172, New York, NY, USA, 2011. ACM.

[7] 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.

[8] Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 447–456, 2009.

[9] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42, August 2009.

[10] Y. Niu, Y. Wang, G. Sun, A. Yue, B. Dalessandro, C. Perlich, and B. Hamner. The Tencent Dataset and KDD-Cup'12. In *KDD-Cup Workshop*, 2012.

[11] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009.

[12] J. Tenenbaum and W. Freeman. Separating style and content with bilinear models. *Neural computation*, 12(6):1247–1283, 2000.

[13] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. Cofi rank - maximum margin matrix factorization for collaborative ranking. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1593–1600, Cambridge, MA, 2008. MIT Press.