General Functional Matrix Factorization Using Gradient Boosting

Tianqi Chen TQCHEN@APEX.SJTU.EDU.CN
Shanghai Jiao Tong University, China

Hang Li HANGLI.HL@HUAWEI.COM
Huawei Noah's Ark Lab, Hong Kong

Qiang Yang QYANG@CSE.UST.HK
Huawei Noah's Ark Lab, Hong Kong

Yong Yu YYU@APEX.SJTU.EDU.CN
Shanghai Jiao Tong University, China

Abstract

Matrix factorization is among the most successful techniques for collaborative filtering. One challenge of collaborative filtering is how to utilize available auxiliary information to improve prediction accuracy. In this paper, we study the problem of utilizing auxiliary information as features of factorization and propose formalizing the problem as general functional matrix factorization, whose model includes conventional matrix factorization models as its special cases. Moreover, we propose a gradient boosting based algorithm to efficiently solve the optimization problem. Finally, we give two specific algorithms for efficient feature function construction for two specific tasks. Our method can construct more suitable feature functions by searching in an infinite functional space based on training data and thus can yield better prediction accuracy. The experimental results demonstrate that the proposed method outperforms the baseline methods on three real-world datasets.

1. Introduction

Matrix factorization has been proved to be one of the most successful approaches to collaborative filtering. It assumes that the users' preference matrix $\mathbf{Y} \in \mathbb{R}^{n \times m}$ can be factorized into a product of two low rank matrices $\mathbf{U} \in \mathbb{R}^{d \times n}$ and $\mathbf{V} \in \mathbb{R}^{d \times m}$ as $\hat{\mathbf{Y}}(i,j) = \mathbf{U}_i^T \mathbf{V}_j$, and conducts collaborative filtering by exploiting $\mathbf{U}$ and $\mathbf{V}$. In real world scenarios, many types of auxiliary information are available besides the matrix $\mathbf{Y}$, and the use of them may enhance the prediction accuracy. For example, information such as time and location can help better predict users' preferences based on the contexts, while information such as users' ages and gender can help make better prediction on the basis of user profiles. From the viewpoint of learning, the use of auxiliary information can overcome the sparsity of the matrix data.

The use of auxiliary information in matrix factorization has been studied, various models (Agarwal & Chen, 2009; Rendle et al., 2011; Weston et al., 2012) have been proposed. In these models, auxiliary information is encoded as features and factorization of the matrix amounts to linearly mapping the feature vectors in the feature space into a latent space. The key question here is how to encode useful features to enhance the performance of the model. Ideally one would like to automatically construct "good" feature functions in the learning process.

To the best of our knowledge, little work has been done on automatic feature function construction in matrix factorization. In this paper, we try to tackle the challenge by employing general functional matrix factorization and gradient boosting. In our method, feature functions are automatically constructed via searching in a functional space during the process of learning. The learning task is effectively and efficiently carried out with the gradient boosting algorithm. The contribution of the paper is as follows. (1) We propose

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

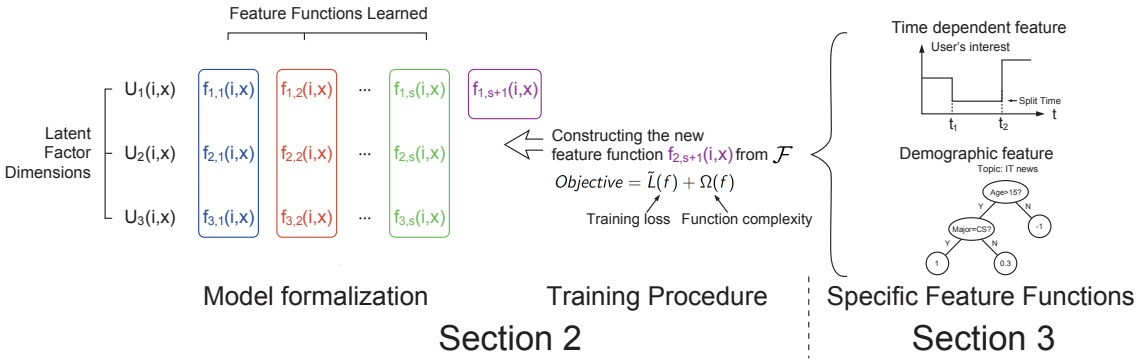


Figure 1. Outline of Our Methods

a general formalization of functional matrix factorization. (2) We introduce an efficient algorithm to solve the optimization problem using gradient boosting. (3) We give two specific feature construction algorithms using time and demographic information. (4) We empirically demonstrate the effectiveness of the proposed method on three real-world datasets.

The rest of the paper is organized as follows. Section 2 gives the general description of our model. Section 3 describes two specific algorithms for feature function construction. Experimental result is provided in Section 4. Related work is introduced in Section 5. Finally, the paper is concluded in Section 6.

2. General Functional Matrix Factorization

2.1. Model

Matrix factorization with auxiliary information can be represented by the following equation, where the auxiliary information is denoted as $\mathbf{x}$.

$$\hat{\mathbf{Y}}(i,j,\mathbf{x}) = \mathbf{U}^T(i,\mathbf{x})\mathbf{V}(j,\mathbf{x}) \tag{1}$$

Note that for simplicity we always assume that there is an extra constant column in both $\mathbf{U}$ and $\mathbf{V}$ to model the bias effect. Further generalization can be considered, which involves the sum of multiple factorizations. To simplify the discussion, we restrict our model to a single factorization. The latent factor model in Equation (1) assumes that prediction $\hat{\mathbf{Y}}$ is a function of user i, item j and auxiliary information $\mathbf{x}$ in the current context. The matrix $\mathbf{U}$ is constructed by linear combination of features. The kth dimension of $\mathbf{U}$ is calculated by the following equation[1]

$$\mathbf{U}_k(i,\mathbf{x}) = \sum_s \alpha_{k,s} f_{k,s}(i,\mathbf{x}), \;\; f_{k,s} \in \mathcal{F} \tag{2}$$

[1]The indicator function can be used to encode user/item index in the model.

The matrix $\mathbf{V}$ can be formalized similarly, while in many applications it seems to be sufficient to define one of them in a functional form (either $\mathbf{U}$ or $\mathbf{V}$). $\mathcal{F}$ is the set of candidate feature functions (space of functions) which is closed under scaling and can be infinite. The novelty of our model lies in that it introduces functions as model parameters, which allows us to construct "good" feature functions $f_{k,s} \in \mathcal{F}$ in learning of the latent factor model. To simplify the notations, we will also use $\hat{\mathbf{Y}}_{ij}$ for $\hat{\mathbf{Y}}(i,j,\mathbf{x})$, $\mathbf{U}_{ki}$ for $\mathbf{U}_k(i,\mathbf{x})$, and $\mathbf{V}_{kj}$ for $\mathbf{V}_k(j,\mathbf{x})$.

One very important factor in our method is the definition of the functional space $\mathcal{F}$. As shown in Fig. 1, we will give two examples of feature functions. The first one is a user-specific k-piece step function that captures users' interest change over time; the second one is a user-independent decision tree to model users with their properties. The details will be introduced in Section 3.

Relationship with Existing Models

General functional matrix factorization contains existing models as its special cases in which certain constraints are imposed on the model. The model degenerates to the conventional factorization models using auxiliary information, when the feature functions $f_{k,s}$ are predefined and fixed. The model becomes that of functional matrix factorization in (Zhou et al., 2011), when the feature functions in all the latent dimensions are assumed to be the same decision tree.

2.2. Training

The objective function of our method is defined in the following equation

$$\mathcal{L} = \sum_{i,j} l(\mathbf{Y}_{ij}, \hat{\mathbf{Y}}_{ij}) + \sum_{k,s} \Omega(f_{k,s}) \tag{3}$$

where l is a differentiable convex loss function that measures the difference between the prediction $\hat{\mathbf{Y}}$ and

Algorithm 1 GFMF Training

repeat
  update $\mathbf{U}$:
  for $k = 1$ **to** d **do**
    collect statistics for $\tilde{L}(f)$ in Equation (5)
    construct function f^* to optimize $\tilde{L}(f) + \Omega(f)$
    $\mathbf{U}_k(i, \mathbf{x}) \leftarrow \mathbf{U}_k(i, \mathbf{x}) + \epsilon f^*(i, \mathbf{x})$
  end for
  update $\mathbf{V}$:
  for $k = 1$ **to** d **do**
    update each $\mathbf{V}_k$ in the same way for $\mathbf{U}$
  end for
until converge

the target $\mathbf{Y}$, and the summation is over all the observed entries. The second term Ω measures the complexity of the model (i.e., the feature functions) to avoid over-fitting. With the objective function, a model employing simple and useful feature functions will be selected as the best model. Because the model includes functions as parameters, we cannot directly use traditional methods such as stochastic gradient descent to find the solution. We propose exploiting gradient boosting to tackle the challenge. More specifically, we construct the latent dimensions additively: in each step, we pick a latent dimension k and try to add a new feature function f to the dimension. During the step, search over the functional space $\mathcal{F}$ is performed to find the feature function f that optimizes the objective function.

For a general convex loss function l, the search of f can be hard. We approximate it by using the quadratic loss function to simplify the search objective. We define

$$g_{ij} = \frac{\partial}{\partial \hat{\mathbf{Y}}_{ij}} l(\mathbf{Y}_{ij}, \hat{\mathbf{Y}}_{ij}),\ h_{ij} = \frac{\partial^2}{\partial^2 \hat{\mathbf{Y}}_{ij}} l(\mathbf{Y}_{ij}, \hat{\mathbf{Y}}_{ij}) \quad (4)$$

Suppose that we add a new feature $f(i, \mathbf{x})$ to $\mathbf{U}_k(i, \mathbf{x})$ while fixing the rest of parameters. The first part of the objective function can be approximated by Taylor expansion as:

$$\begin{aligned} L(f) =& \sum_{i,j} l(\mathbf{Y}_{ij}, \hat{\mathbf{Y}}_{ij} + f(i, \mathbf{x})\mathbf{V}_{kj}) \\ =& \sum_{i,j} l(\mathbf{Y}_{ij}, \hat{\mathbf{Y}}_{ij}) + \sum_{i,j} (g_{ij}\mathbf{V}_{kj}) f(i, \mathbf{x}) \\ &+ \frac{1}{2} \sum_{i,j} (h_{ij}\mathbf{V}_{kj}^2) f^2(i, \mathbf{x}) + o(f^2(i, \mathbf{x})) \\ =& \tilde{L}(f) + o(f^2(i, \mathbf{x})) \end{aligned} \quad (5)$$

We use $\tilde{L}(f) + \Omega(f)$ to find the best feature function f. The general procedure is shown in Algorithm 1, where ϵ is a shrinkage parameter to avoid over-fitting.

Convergence of the Training

The choice of h_{ij} does not necessarily require that the second order gradient is utilized to approximate the loss function. The algorithm converges as long as the following inequality holds

$$l(\mathbf{Y}_{ij}, \hat{\mathbf{Y}}_{ij} + \Delta y) \leq l(\mathbf{Y}_{ij}, \hat{\mathbf{Y}}_{ij}) + g_{ij}\Delta y + \frac{1}{2} h_{ij} \Delta y^2 + o(\Delta y^2) \quad (6)$$

Proof. Denoting the upper bound as $\tilde{L}(f)$, we have $L(\epsilon f^*) + \Omega(\epsilon f^*) \leq \tilde{L}(\epsilon f^*) + \Omega(\epsilon f^*) \leq \tilde{L}(0) + \Omega(0) = L(0)$. Here we assume $0 \in \mathcal{F}$ and $\Omega(0) = 0$. Thus the objective function decreases at each step. [2] □

Time Complexity

The time complexity for calculating the sufficient statistics in the general algorithm is $O(kN)$, if we buffer the predictions of observed pairs in the training data, where N represents the number of observed pairs in the matrix. Furthermore, the time complexity of the training algorithm depends on the specific setting of feature function learning. We will provide the time complexity of the specific learning algorithms in Section 3, which is of log-linear order.

Integration with Existing Methods

The training algorithm is an iterative optimization procedure like coordinate descent. This allows us to easily integrate our approach with existing methods for learning a latent factor model: to use our update algorithm for the feature function construction and to use stochastic gradient descent or coordinate descent to optimize the other parts of the model.

2.3. Segmentation Function Learning

An important part of the training algorithm is to learn the feature function f in each step. In this section, we consider a general method for learning a class of feature functions. It first segments the data into several parts using auxiliary information, and then learns an independent parameter in each segment

$$f(i, \mathbf{x}) = \beta_{p(i,\mathbf{x})} \quad (7)$$

The function $p(i, \mathbf{x})$ defines the cluster index of an instance, and a weight β is assigned to each cluster. As-

[2] This proof holds up to $o(\Delta y^2)$. To make the loss strictly decrease, we can set h_{ij} to be a uniform upper bound of the second order gradient to remove this error term.

sume that there are $|C|$ clusters and the instance set of each cluster is defined as $I_c = \{(i, j, \mathbf{x})|p(i, \mathbf{x}) = c\}$. The complexity of feature function is defined as

$$\Omega(f) = \gamma|C| + \frac{1}{2}\lambda\sum_{c=1}^{|C|}\beta_c^2 \tag{8}$$

The first term represents the number of segments; the second term represents the norm of weights in each cluster. For this problem, the optimal β can be found analytically for a given segmentation as follows

$$\beta_c^* = -\frac{\sum_{i,j,\mathbf{x}\in I_c} g_{ij}\mathbf{V}_{kj}}{\sum_{i,j,\mathbf{x}\in I_c} h_{ij}\mathbf{V}_{kj}^2 + \lambda} \tag{9}$$

the value will be mainly determined by λ when the second order term is small. Putting β^* back to $\tilde{L}(f)$, we obtain the objective function as

$$\tilde{L}(f)+\Omega(f) = \tilde{L}(0)-\frac{1}{2}\sum_{c=1}^{|C|}\frac{\left(\sum_{i,j,\mathbf{x}\in I_c} g_{ij}\mathbf{V}_{kj}\right)^2}{\sum_{i,j,\mathbf{x}\in I_c} h_{ij}\mathbf{V}_{kj}^2 + \lambda}+\gamma|C| \tag{10}$$

We can use Equation (10) to guide the search for a segmentation function. The basic motivation of the segmentation function is to make the latent dimensions of data in the same segments as similar as possible. One of the most widely used segmentation functions is decision tree (Friedman et al., 2001). However, for specific types of segmentation functions, we can use specific algorithms rather than decision tree. We will use the general methodology in this section to guide the feature function constructions in Section 3.

3. Specific Feature Functions

3.1. Time dependent Feature Function

Time is very important auxiliary information in collaborative filtering. One important reason is that the users' preferences change over time. To model users' local preferences, following the methodology in (Koren, 2009), we can define an indicator function for each localized time bin as feature function; the model can be written as

$$\hat{\mathbf{Y}}(i, j, t) = \mathbf{U}_{i,binid(t)}^T\mathbf{V}_j \tag{11}$$

The idea is to learn a localized latent factor in each time bin to capture the time dependent preference of the user. The time bin size is fixed and needs to be predefined. In real world scenarios, different users may have different patterns of preference evolution. Some users may change their interest frequently, while others may like to stick to their interest. The patterns

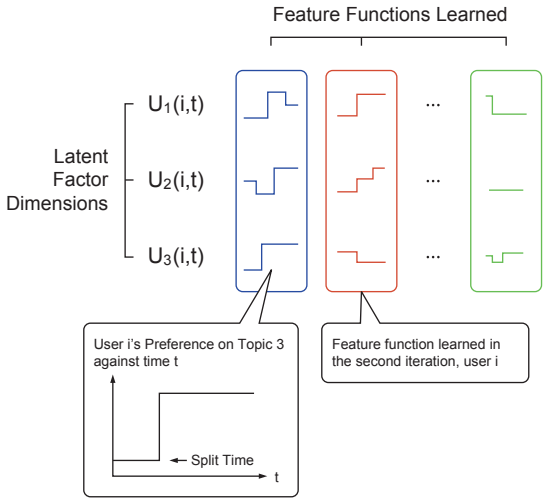


Figure 2. Illustration of Time-dependent Feature Function

of changes for different topics may also differ; a user might have constant favor on classical music for a long time while changing her preference on pop music very often. These patterns may be hard to capture with a fixed size of time bins. We can utilize our model to learn more flexible feature functions over time. Specifically, we define the user latent factor as

$$\mathbf{U}_k(i, t) = \sum_s f_{k,s,i}(t) \tag{12}$$

where each $f_{k,s,i}(t)$ is a *user-specific* $|C|$-piece step function in t defined by segmentations on time. Fig. 2 gives an illustration of the model. Note that time bin indicator function can also be viewed as a special type of our time feature function. The difference is that our method decides the number of segments, changing points, and function values based on training data, which offers more flexibility. Using the result in Section 2.3, the objective for discovering time segmentation can be written as follows

$$\begin{aligned}\underset{t_1,\cdots t_{|C|-1}}{\text{argmin}} \; &-\frac{1}{2}\sum_{c=1}^{|C|}\frac{\left(\sum_{i,j,t\in I_c} g_{ij}\mathbf{V}_{kj}\right)^2}{\sum_{i,j,t\in I_c} h_{ij}\mathbf{V}_{kj}^2 + \lambda} + \gamma|C| \\ \text{where } I_c &= \{(i, j, t)|t_{c-1} \le t < t_c\} \\ t_0 &= -\infty, t_{|C|} = +\infty\end{aligned} \tag{13}$$

The optimal solution of the objective can be found using a dynamic programming algorithm in quadratic time complexity. We use a faster greedy algorithm; it starts from all instances as independent segments, then greedily combines segments together when an improvement of objective function can be made, until a local minimum is reached.

Algorithm 2 Tree Split Finding with Missing Value

Require: I: instance set, n_p: number of properties
  $gain \leftarrow 0$
  $G_{all} \leftarrow \sum_{i,j \in I} g_{ij}\mathbf{V}_{kj}$, $H_{all} \leftarrow \sum_{i,j \in I} h_{ij}\mathbf{V}_{kj}^2$
  for $p = 1$ **to** n_p **do**
    $I_p \leftarrow \{(i,j,x) \in I | x_p \neq missing\}$
    {enumerate default goto right}
    $G_{left} \leftarrow 0, H_{left} \leftarrow 0$
    for (i,j,x) in sorted(I_p, ascent order) **do**
      $G_{left} \leftarrow G_{left} + g_{ij}\mathbf{V}_{kj}$
      $H_{left} \leftarrow H_{left} + h_{ij}\mathbf{V}_{kj}^2$
      $G_{right} \leftarrow G_{all} - G_{left}, H_{right} \leftarrow H_{all} - H_{left}$
      $gain \leftarrow \max(gain, \frac{G_{left}^2}{H_{left}+\lambda} + \frac{G_{right}^2}{H_{right}+\lambda} - \frac{G_{all}^2}{H_{all}+\lambda})$
    end for
    {enumerate default goto left}
    $G_{right} \leftarrow 0, H_{right} \leftarrow 0$
    for (i,j,x) in sorted(I_p, descent order) **do**
      $G_{right} \leftarrow G_{right} + g_{ij}\mathbf{V}_{kj}$
      $H_{right} \leftarrow H_{right} + h_{ij}\mathbf{V}_{kj}^2$
      $G_{left} \leftarrow G_{all} - G_{right}, H_{left} \leftarrow H_{all} - H_{right}$
      $gain \leftarrow \max(gain, \frac{G_{left}^2}{H_{left}+\lambda} + \frac{G_{right}^2}{H_{right}+\lambda} - \frac{G_{all}^2}{H_{all}+\lambda})$
    end for
  end for
  output split and default direction with max gain

Time Complexity

Let m_i to be the number of observed entries in the i-th row. The time complexity of the greedy algorithm for user i is $O(m_i \log m_i)$, where the log-scale factor comes from the cost for maintenance of the priority queue. The overall training complexity for one iteration over the dataset is $O(kN \log m)$. As for the space complexity, addition of two step functions can be combined into a single step function, which allows us to store $\mathbf{U}_k(i,t)$ efficiently and do not need to buffer $\mathbf{U}$ for each instance.

3.2. Demographic Feature Function

Data sparsity is a major challenge for collaborative filtering. The problem becomes especially severe for handling new users, who may have only a few or even no records in the system. One important type of auxiliary information that can be used to deal with this problem is user demographic data such as age, gender and occupation. The information can be relatively easily obtained and has strong indication to user preferences. For example, a young boy is more likely to favor animation, and a student of computer science may have an interest in IT news. To learn useful feature functions from the user demographic data for latent factor model construction, we use $\mathbf{x}$ to represent the demographic information, and define the latent factor to be

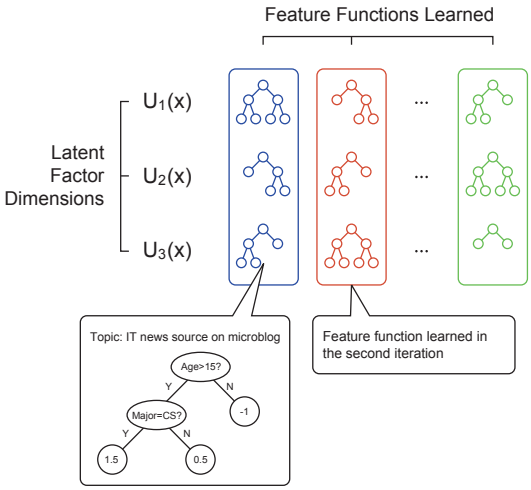


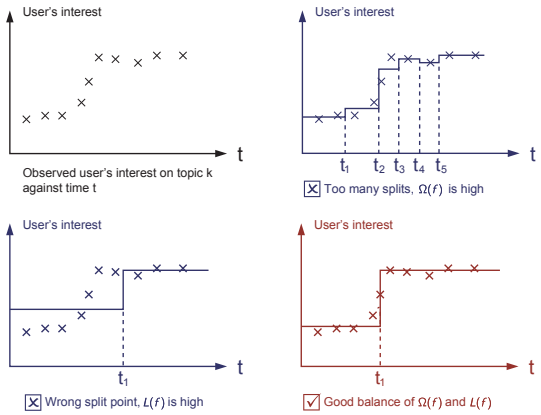
Figure 3. Illustration of Demographic Feature Function

$$\mathbf{U}_k(i,\mathbf{x}) = \sum_s f_{k,s}(\mathbf{x}) \tag{14}$$

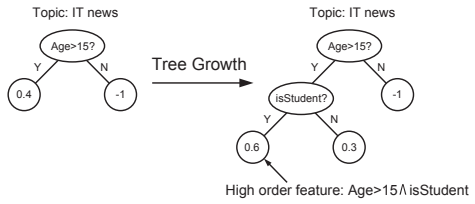
where each $f_{k,s}(\mathbf{x})$ is a *user-independent* feature function. We define as the feature function set $\mathcal{F}$ the set of regression trees over user properties $\mathbf{x}$. Fig. 3 gives an intuitive explanation of the model. We place in the leaf nodes example values that are used to illustrate the structure of the trees, whereas in the real feature trees they are calculated using Equation (9). The advantage of using trees as features is that it is possible to select useful attributes, create high-order combinations of attributes, and automatically segment continuous attributes to construct meaningful features.

Next, we describe the tree learning algorithm. The tree construction is also guided using Equation (10) in Section 2.3. In each round, we greedily enumerate the attributes and choose to make a split on the attribute that can give the maximum loss reduction. The process is repeated until some stopping condition is met (e.g maximum depth), and then a pruning process is carried out to merge the nodes with less loss reduction. One problem that we may face here is the existence of missing values. That is, not all the attributes are available for each user. To solve this problem, we decide a *default direction* (either left child or right child) for each node and adopt the default direction for classification when a missing value is encountered. The splitting construction algorithm also enumerates default directions to find the best split; the algorithm is shown in Algorithm 2[3].

[3]All features are assumed to be numerical here, and categorical attributes are encoded in an indicator vector



(a) Time dependent feature construction



(b) Tree construction

Figure 4. Illustration of Feature Construction

TIME COMPLEXITY

The time complexity for one iteration over the dataset is $O(kDN_p \log(n) + kN)$. D is the maximum depth of the tree and N_p is the number of observed entries in the user attribute matrix. We note that the computation only depends on the size of observed entries. We also note that the complexity for tree construction (first part) does not depend on the number of observed entries N. This is because we can make statistics for each user before tree construction and thus have the construction step only depend on the number of users. Similarly, we only need to buffer $\mathbf{U}$ for each user.

3.3. Advantage of Our Method

Next, we will discuss why our method can enhance the prediction accuracy. As previously explained, user preference patterns are dynamic and hard to capture. Fig. 4 illustrates our method of feature construction. In the time dependent case, different users may have different interest changing patterns. Our method can effectively detect the dynamic changes in interest over some topics along the time axis. The regularization in our method will also lead to selection of a simple model when no change occurs in user preference. In the user demographic case, our method can automatically create high order features and build the tree based complex feature functions. The goal of optimization is to find the feature functions which are simple and have good fit to the data; automatically constructing high order features and automatically detecting change points make this possible and give our model the potential of achieving better results.

Table 1. Statistics of Datasets

DATASET	#USER	#ITEM	#OBSERVED
ML-1M	6,040	3,952	1 M
YAHOO! MUSIC	1 M	624 K	252 M
TENCENT MIROBLOG	2.3 M	6,095	73 M

4. Experiments

4.1. Dataset and Setup

We use three real-world datasets in our experiments. The statistics of the datasets are summarized in Table 1. The first dataset is the Movielens dataset ml-1M, which contains about 1 million ratings of 3,952 movies by 6,040 users with timestamps. We use five-fold cross validation to evaluate the performance of our method and the baseline methods. The second dataset is the Yahoo! Music Track1 dataset[4], which contains 253 million ratings by 1 million users over 624 thousand tracks from the Yahoo! Music website. The data is split into training, validation, and test sets by time. We use the official training and validation datasets to train the model and the test set to make evaluation. For the two datasets, we use RMSE (root mean square error) as evaluation measure. The first two datasets are utilized to evaluate the performance of our method with time dependent feature function construction.

The third dataset we use is the Tencent Microblog[5]. It is one of the largest Twitter-like social media services in China. The dataset contains the recommendation records of 2.3 million users over a time period of about two months. The item set is defined as celebrities and information sources on Tencent Microblog. The dataset is split into training and testing data by time. Furthermore, the test set is split into public and private set for independent evaluations. We use the training set to train the model and use the public test set to conduct evaluation. We also separate 1/5 data from the training data by time for parameter selection. The dataset is extremely sparse, with only on average two positive records per user. Furthermore, about 70% of users in the test set never occur in the training set. User demographic information (age and gender) and

[4] http://kddcup.yahoo.com/datasets.php
[5] http://kddcup2012.org/c/kddcup2012-track1/data

Table 2. Results on Movielen in RMSE

METHOD	RMSE ($d = 32$)	RMSE ($d = 64$)
MF	0.8519± 0.0008	0.8497± 0.0010
TIMEMF	0.8474± 0.0009	0.8450± 0.0011
GFMF	0.8436± 0.0013	0.8417± 0.0011
SVD++	0.8473± 0.0006	0.8456± 0.0005
TIMESVD++	0.8423± 0.0007	0.8410± 0.0006
GFMF++	0.8393± 0.0010	0.8377 ± 0.0008

social network information is available in the dataset. For this dataset, we use $MAP@k$ (mean average precision) as evaluation measure. The dataset is exploited to evaluate the performance our method with demographic feature construction.

In all the experiments, the parameter λ of our model is heuristically set to 2. We use the validation set (in ml-1M via cross validation) to tune the meta parameters for the baseline models and our model.

4.2. Results on Movielen

We compare six methods on the ml-1M dataset. The compared methods are as follows:

- MF is the basic matrix factorization model that does not utilize auxiliary information.
- TIMEMF uses predefined time bins to model users' preference changes over time. The size of time bins is tuned through cross validation.
- GFMF is our model that learns time dependent feature functions.
- SVD++ is the matrix factorization model with implicit feedback information (Koren, 2008).
- TIMESVD++ uses the same time feature function as TIMEMF, and it also utilizes implicit feedback information.
- GFMF++ utilizes the GFMF part to train a time dependent model and also uses implicit feedback information.

We train the baseline models and our model, when the latent dimensionality d is 8, 16, 32 and 64, and find that the results are consistent. We show the results in two parameter settings n Table 2. Both TIMEMF and SVD++ give 0.004 reduction over MF in terms of RMSE. GFMF is able to give a further 0.003 reduction of RMSE. Finally, the GFMF++ model that combines implicit feedback information into our model achieves the best result. We can see that our model brings more than 75% improvement compared with TIMEMF, which uses predefined time bins. This is due to the fact that our method adapts the time bin size and time split points for each user and can capture users' preference evolution more accurately.

Table 3. Results on Yahoo! Music in RMSE

METHOD	RMSE ($d = 64$)	RMSE ($d = 128$)
SVD++	23.06	23.02
TIMESVD++	22.88	22.84
GFMF++	22.80	22.77

4.3. Results on Yahoo! Music

We also investigate the performance of our method on the Yahoo! Music dataset. We compare three methods on this dataset, with the same model definitions as in the previous section. For a time stamp that does not appear in the training set, we assign a constant feature to it, which is equal to the feature of the corresponding last time bin in the training set. We add user and item time bias terms to all the models to offset the bias effect and focus on modeling of latent dimensions. We compare the methods with different latent dimensions (ranging in 8, 16, 32, 64, 128), and obtain consistent results. We report the results for $d = 64$ and $d = 128$.

The experimental results are shown in Table 3. The comparison shows GFMF++ gives 40% more RMSE reduction than TIMESVD++ over SVD++. This indicates that our method can learn good feature functions in a large real-world dataset.

4.4. Results on Tencent Microblog

We compare six methods on the Tencent Microblog dataset. We use logistic loss to train all the models. Due to the extreme sparsity of the dataset, the performance of matrix factorization is similar to the bias model and thus is not reported. Because this is a social network dataset, it is interesting to see the impact of social information as well, and thus we also include social-aware models in the experiment.

- ITEM BIAS is the basic model that only includes a bias term to measure the popularity of each item.
- DEMOMF is a latent factor model that utilizes predefined feature functions on age and gender. We segment ages into intervals in advance and use an indicator function to represent the age feature of each interval.

Table 4. Results on Tencent Microblog in MAP

METHOD	*MAP*@1	*MAP*@3	*MAP*@5
ITEM BIAS	22.58%	34.65%	38.26%
DEMOMF	24.07%	36.38%	40.00%
GFMF	24.48%	36.82%	40.43%
SOCIALMF	24.67%	37.16%	40.80%
SOCIALDEMOMF	25.41%	37.98%	41.60%
SOCIALGFMF	25.70%	38.28%	41.90%

- GFMF is our model that learns demographic feature functions during the training process.
- SOCIALMF follows the approach of social-aware matrix factorization (Jamali & Ester, 2010) that utilizes social network information to construct user latent factors.
- SOCIALDEMOMF and SOCIALGFMF are integrated models of two demographic-aware models with SOCIALMF respectively.

We train the model, when the latent dimensionality d is 8, 16, 32, and 64. The results for all the models are optimal when $d = 32$, and thus we only report those results here, as shown in Table 4. From the results, we can first find that the bias effect is strong in this dataset. However, improving the performance is still possible using social network and user demographic information. The performances of GFMF are consistently better than DEMOMF. DEMOMF improves upon the bias model in terms of *MAP*@1 by about 1.5%, while GFMF is able to further improve the performance by about 0.4%. This means GFMF can learn good feature functions to utilize the user demographic information. The results of SOCIALMF and GFMF are similar, which means user demographic information is as important as social information in this task. Finally, SOCIALGFMF utilizing both the social and user demographic information achieves the best performance among all the methods.

5. Related Work

This work is concerned with matrix factorization (Salakhutdinov & Mnih, 2008; Koren et al., 2009; Lawrence & Urtasun, 2009), which is the among most popular approaches to collaborative filtering. There are many variants of factorization models to incorporate various auxiliary information, including time information (Koren, 2009), attribute information (Agarwal & Chen, 2009; Stern et al., 2009), context information (Rendle et al., 2011), and content information (Weston et al., 2012). These models make use of predefined feature functions to encode the auxiliary information. In (Zhou et al., 2011) a latent factor model is proposed for a different task of guiding user interview, which can be viewed as a special case of our model. Their method cannot be applied to solve the problems in our experiments because of its specific model formulation (e.g., it cannot handle time dependent features). Our method is unique in that it simultaneously learns the feature functions and the latent factor model for CF which is, to our best knowledge, is the first work in the literature.

From the perspective of infinite feature space, one related general framework is (Abernethy et al., 2009), which implicitly defines features using kernel functions. Our method allows explicit definition and automatic construction of features, and is scalable to deal with a large amount of data. Our method employs gradient boosting (Friedman, 2001), which is equivalent to performing coordinate descent in the functional space. Gradient boosting has been successfully used in classification (Friedman et al., 2000) and learning to rank (Burges, 2010). To our knowledge, this is the first time it is used in matrix factorization.

6. Conclusion

In this paper, we have proposed a novel method for general functional matrix factorization using gradient boosting. The proposed method can automatically search in a functional space to construct useful feature functions based on data for accurate prediction. We also give two specific algorithms for feature function construction on the basis of time and demographic information. Experimental results show that the proposed method outperforms the baseline methods on three real-world datasets. As future work, we plan to explore other feature function settings. We are also interested in implementation of the method in a distributed environment to scale up to larger problems. Finally, it is also interesting to apply our method to other problems than collaborative filtering.

Acknowledgement

We thank the anonymous reviewers for their very helpful reviews. Yong Yu is supported by grants from NSFC-RGC joint research project 60931160445. Qiang Yang is supported in part by Hong Kong CERG projects 621010, 621307, 621211, and NSFC-RGC project N_HKUST624/09. We also appreciate the discussions with Wei Fan, Zhengdong Lu, Weinan Zhang, and Fangwei Hu.

References

Abernethy, J., Bach, F., Evgeniou, T., and Vert, J.P. A new approach to collaborative filtering: Operator estimation with spectral regularization. *Journal of Machine Learning Research*, 10:803–826, 2009.

Agarwal, D. and Chen, B.C. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pp. 19–28, New York, NY, USA, 2009. ACM.

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

Friedman, J., Hastie, T., and Tibshirani, R. Additive logistic regression: a statistical view of boosting. *The annals of statistics*, 28(2):337–407, 2000.

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

Friedman, J.H. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pp. 1189–1232, 2001.

Jamali, M. and Ester, M. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pp. 135–142, New York, NY, USA, 2010. ACM.

Koren, Y. 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, pp. 426–434, New York, NY, USA, 2008. ACM.

Koren, Y. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 447–456, 2009.

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

Lawrence, N.D. and Urtasun, R. Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 601–608, Montreal, June 2009. Omnipress.

Rendle, S., Gantner, Z., Freudenthaler, C., and Schmidt-Thieme, L. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th ACM SIGIR Conference on Reasearch and Development in Information Retrieval*. ACM, 2011.

Salakhutdinov, R. and Mnih, A. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20*, pp. 1257–1264. MIT Press, Cambridge, MA, 2008.

Stern, D.H., Herbrich, R., and Graepel, T. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pp. 111–120, New York, NY, USA, 2009. ACM.

Weston, J., Wang, C., Weiss, R., and Berenzweig, A. Latent collaborative retrieval. In *International Conference on Machine Learning (ICML)*, Edinburgh, Scotland, June 2012.

Zhou, K., Yang, S.H., and Zha, H. Functional matrix factorizations for cold-start recommendation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pp. 315–324, New York, NY, USA, 2011. ACM.