Group Based Multi-feature Latent Factor Model for Recommendation System in Social Network

Guangyu Lin
Dept. of Electrical Computer Engineering
Software Engineering and Systems
The University of Texas
Austin, Texas, 78741
Email: guangyu.lin9276@hotmail.com

***Abstract*—Social Network Services (SNS) are becoming more and more popular in recent years with huge amount of information growing exponentially. The popularity appeals to the need of accurate recommendation system for SNS users. In this paper, the author presents a recommendation system with time-aware latent factor model for Tencent Weibo, which is one of the most popular Chinese social networking website. The recommendation system is built according to the data set provided by KDD Cup 2012 and compared with other recommendation systems proposed by other KDD Cup competitors in accuracy and efficiency.**

Keywords: Latent Factor Model, Recommendation Systems, Social Network Services (SNS)

I. Introduction

Online social networking services have become tremendously popular in recent years, with popular social networking sites like Facebook, Twitter, and Tencent Weibo adding thousands of enthusiastic new users each day to their existing billions of actively engaged users.

Tencent Weibo, starting from April 2010, has become one of the largest micro-blogs platform for sharing in China with hundreds of millions of users. Some of them are active users, while others don't check their micro-blogs so often or even become bot users. Everyday, there are over 40 million messages generated on Tencent Weibo.

Now Tencent Weibo wants to help its users to gain better user experience by expanding their view of information and recommending new tweets, celebirties, news (and etc.) that users might be interested potentially. At the mean time, Tencent Weibo doesn't want to flood a huge volume of information to users and definitely wants to improve the recommendation system from the feedback that whether a user has followed the item that has been recommended by the system.

The object of this paper is to present a recommendation system with the data provided by Tencent Weibo. For a given item, which could be a user, a group and an organization, and a given item, the recommender system should be able to tell whether the item should be recommended to the user. The data set consists of users profiles, items descriptions, records of users actions, users follow history, keyword sets for each user, and the feedbacks of previous active usersrecommendations. In this project, I will set up a training model, abstract and process useful data, and design a prediction algorithm.

A. Survey of related works

Before I start the project, I have gone through papers and slides presented by prior KDD Cup 2012 competitors and winners. In the solution presented by Team ACM-Class@SJTU, who has won the first prize in KDD Cup 2012, they used a factorization model combined with additive forest to predict the result. [1] They calculated the feature-based matrix with some user features and item features, and an additional bias matrix with information of global feature related to interaction between users and items. Afterwards, they trained the model via stochastic gradient descent. Moreover and the most important, they built up an additive forest by learning via gradient boosting algorithm from the property features of user information. With this combination approach, the team has gained roughly 45% accuracy in prediction results. In the paper "Scorecard with Latent Factor Models for User Follow Prediction Problem", the author from FICO built up the model based on Latent Factor Model with FICO's internal software tools. [2] The team made a great improvement in prediction by dividing the users into 15 groups based on age and gender combination and the data set was trained seperately from different groups. The competitors from Shanda Innovations presented a

novel approach called context-aware ensemble of Multifaceted Factorization Models for recommendations. [3] The greatest contribution they made was pre-processing the training data and removing noisy data in negative samples by eliminating "omitted" records which were not helpful in predicting users' interests. They seperated the data set by positve and negative records and trained them seperately.

After studying the materials, the author come up with a new prediction model based on Latent Factor Model which combines with some useful factors presented in the papers and my newly add features in data pre-processing and predicting phase. In the following part of this section, more details in data pre-processing, model training and other approaches and techniques will be presented.

B. Data Pre-processing

The data we used in this project is sponsored by KDD Cup 2012 Track 1 (in Table II). The data files includes a training file (1.9G), user social network information, user profiles, and etc.

TABLE I
DATA FILE

File Name	Description
rec_log_train.txt (1.9G)	Training dataset
rec_log_test.txt (989.1M)	Testing dataset
user_profile.txt (58.5M)	User Profile Data
item.txt (1.2M)	Item data
user_action.txt (227.7M)	User action data
user_sns.txt (776.4M)	User follow history
user_key_word.txt (191.2M)	User key word data

To better utilize these data within limited time, I decide to pre-process these files and filter out data with useful information.

In the data set, some users have used Tencent Weibo for a long time or quite often and there are a good amount of information about them, whereas some users are new to Tencent Weibo and we do not know much about them.

Here we define the users that we have much information about as **active users**, and the ones we do not know much about as **inactive users**. We will establish a criterion of active users, based on which we divide the whole users into two sets: active users and inactive users. We only adopt the information of active users in the training process. From the feedbacks of the previous recommendation, we can see that some items are recommended to many different users but are seldom accepted. We can get rid of this kind of items in the training process because we can assume that all users do not like them. After excluding inactive users and noisy items, we shrink the data set being trained, which improves the speed and the efficiency of the training process. On the other hand, due to that fact that users interest changes as time goes by, we train data for different time intervals.

C. Training Model

We aim to obtain an $m \times n$ matrix R, where m is the number of users and n is the number of items, and let R_{ij} represent the interests of the ith user to the jth item. Initially, we will quantify R_{ij} based on the data referring to the ith user and the jth item, as well as the data reflecting the ith user's actions and follow history. Since compared with the size of matrix R the information provided by the data set is very limited, very few blanks of R can be filled in at the beginning. My goal is to estimate all the elements of matrix R. The model we use to estimate R is **Latent Factor Model (LFM)**. We will estimate R by the product of an $m \times k$ matrix P and a $k \times n$ matrix Q. k is a parameter set by us and can be viewed as the number of classes that we partition the items into. However we dont care about what the k classes are. We can view P_{ij} as the interests of the ith user to the jth class and regard Q_{ij} as the weight that the jth item belongs to the ith class. The main idea is to find a matrix P and the matrix Q such that the difference between the initial R and $P \times Q$ is minimized, and to estimate R as $P \times Q$. **Gradient Descent** can be a proper way to compute P and Q.

D. Prediction

For noisy items, we will definitely not recommend to any users. For an active user u and a non-noisy item i, we will look at R_{ij} from matrix R and see whether R_{ij} exceed the threshold we establish. If it does we will recommend item i to user u, otherwise we will not do the recommendation. For an inactive user v, and a non-noisy item j, we will determine whether we will recommend j to v by observing the attributes of the active users who v have actions with to j.

II. LATENT FACTOR MODELS

In this section, we will give a very brief overview of the Latent Factor Model, from which my model is derived. Latent factor models are widely used in solving Top-N recommendation problems. The main idea of latent factor model is feature-based matrix factorization, i.e. to approximate a rating R_{ui} by the inner product of a

user latent factor and an item latent factor. Also, for the most cases, we need to add a bias to adjust the model.

A. Preliminary Model

The basic form of latent factor model is showed below:

$$\hat{r}_{ui} = b_i^I + b_u^U + p_u^T q_i$$

Here b_u^U is the global user-based adjustment and b_i^I is the global item-based adjustment. p_u is the latent factor for user and q_i is the latent factor for item. For the problem described in this project, using the basic latent factor model directly does not perform well on prediction. So the author modified the basic model by introducing keyword latent factors and utilizing social network information, which could make the model more precise and more predictive. Given a latent factor model, a standard way to learn the latent factors is to minimizing the loss function, which describes the deviation from the original values and the predicted values. Here in this paper, we use a loss function, which calculates the accumulation of squared errors between the original values (ratings) given in the training set and the predicted values.

B. Keyword Model

In the data given by Track 1, we can get the keywords of every user extracted from his action, associated with the weight of each keyword to the user. Since almost every user has keywords, we want to make these data useful in the basic model. If the user has a high weight on certain keyword, it is very likely that the user will follow the items that are related to this keyword. Therefore, we proposed **Keyword Latent Factor** and derived my **Keyword Model** by replacing the latent factor of a user by the weighted sum of keyword latent factors that user related to.

$$p_u = \sum_{k \in K(u)} w_{uk}^K p_k^K$$

In this expression, p_k^K is the latent factor for keyword k, and $K(u)$ is the set of keywords for user u. w_{uk}^K is the weight of keyword k to user u.
So we can get our Keyword Model.

$$\hat{r}_{ui} = b_i^I + b_u^U + (\sum_{k \in K(u)} w_{uk}^K p_k^K) q_i$$

C. Social Network Model

Another important factor that may affect a user's preference is the social network influence. In the data given by Track 1, the user_sns dataset tells us the set of users each user has followed and the user_action dataset tells us the set of users each user has retweeted, @ and commented. For example, if a user follows many sports fans, he may be very likely to accept the recommendation of a sport star. Or if a user retweeted and commented many moments of a user, he may behave very similar to that user. The social network influence can help us to solve the cold start problem. If we find that a user has very limited information to use, we can refer to users that have social influence on him to make recommendation. Therefore, we proposed the **Social Influence Latent Factor** and derived my **Social Network Model** by replacing the latent factor of a user by the weighted sum of social influence latent factors that the user related to.

$$p_u = \sum_{s \in S(u)} w_{us}^S p_s^S$$

In this expression, p_s^S is the latent factor for user s, and $S(u)$ is the set of users has social influence on user u. w_{us}^S is the weight of user s to user u, which combined the data get from user_sns and user_action dataset. The definition of weight will be given in data preprocessing part.
So we can get our Social Network Model.

$$\hat{r}_{ui} = b_i^I + b_u^U + (\sum_{s \in S(u)} w_{us}^S p_s^S) q_i$$

D. Group Based Model

Apart from the above factors, the age and gender of a user can also potentially affect a user's preference. For example, a 20-year-old boy may like basketball and would like to follow NBA stars, while a girl with the same age may like beauty products and would like to follow famous fashion brands. And a man who is above 40 years old could be interested in military and politics and a woman at that age would probably like house decoration and would like to follow some designers in that field. We can see that users from different groups may have different preferences and if we treat these groups as one, the recommendation we made would be preferable to one group but not the other. Therefore we decided to divide users into different groups according to their age and gender, and training each model separately. In this way, every group has its own model in the end.

E. Combined Model

The final step is to combined all the models above into one, that is to incorporate keywords, social influence and group adjustment into one model.

$$\hat{r}_{ui} = b_i^I + b_u^U + (\sum_{k \in K(u)} w_{uk}^K p_k^K + \sum_{s \in S(u)} w_{us}^S p_s^S) q_i$$

To incorporate the group information, we use this model to train each groups.

III. Preprocessing

In order to utilize the information we have and filter out noise data from the training set, we designed a series of steps to pre-process the data before training. In this section, we will give some detials on these pre-processing techniques and reasons.

A. Step 1: filter dummy users

In step 1, we filter out all the "dummy" users. In the training data set, some users either accept all the recommendations of items or reject all of them. It is not that they are interested in the recommended item so they follow it, but they just want to follow something. Information of this kind of users is worthless for the model training. Because these decisions made by the user were not based on their preference of items and we cannot gain any valid information from training these data to model the relationship between users and recommended items. These data are noise and may hurt the accuracy of model training. They are one type of "dummy" user we should remove. Another type of "dummy" users is someone who is neither following nor followed by any other users. These inactive users are isolated in the social network and their data are also useless to train the model. Removing data of inactive user will shrink the training data set and speed up the training process. After filter out these two kinds of "dummy" user, the number of the records in rec_log_train decreases from 73,209,277 to 63,551,881.

B. Step 2: time-based session slicing

The training dataset consists of 73,209,277 recommendation records from 1,392,873 users on 4,710 items. Among these records, there are 67,955,449 negative records and only 5,253,828 positive ones, which means 92.82% of the training ratings are negative samples. This made us wonder whether all these negative records are valuable and not hurtful to the accuracy of our subsequent training results.

As a result, we found that the training dataset only shows whether the user accepted the recommendation or not. If not accepted, it will still be recorded as a "rejection" and this may not necessarily mean that the user really rejected the recommendation. For example, the user may be attracted to something interesting on the Weibo website, and during these sessions, the recommended items were omitted by users. So removing these "omitted" records from the training dataset is necessary and should be done as an important step of preprocessing.

In order to do so, we use the following "session analysis". We define **session** as a continuous period of browsing Tencent Weibo by a particular user. Therefore, each user will have many sessions, and we need to slice these sessions using a reasonable threshold.

For each user u, we denote $t_0(u)$, $t_1(u)$, $t_2(u)$,. . . , $t_m(u)$ as the timestamps of the recommended records in ascending order. Then the time interval between two adjacent timestamp can be calculated as:

$$\Delta t_s(u) = t_{s+1}(u) - t_s(u)$$

where $0 \leq s < m$. When $\Delta t_s(u) < 3600$ sec, we denote it as $\Delta \hat{t}_s(u)$.

The threshold for session slicing is then:

$$\tau(u) = \frac{1}{2} \cdot (\tau_0 + \frac{\sum_{s=0}^{m} \Delta \hat{t}_s(u)}{|\Delta \hat{t}(u)|})$$

where parameter t_0 is set to 90 seconds in our experiment.

If $\Delta t_s(u)$ is larger than this threshold, the training records on time $t_{s+1}(u)$ and $t_s(u)$ will be separated to different sessions.

We have computed this threshold for many users, and they all fall into the range of 50s to 300s. Then after experimenting with different thresholds ranging from 45s to 1800s, we found that the sizes of the preprocessing results using these thresholds have very small differences (they are all around 14% of the original data), and the data records in them are also more or less the same ones. So we have chosen **45s** as the threshold. Now that we have the sessions partitioned for each user, we do the following things as the session analysis.

For each session,

- We discard the whole session if it contains only negative records, or it contains more than 90% negative records. This means the user is very likely

not paying attention to the recommendations whatsoever. So we only consider the session as "valid" if:

$$\frac{N_{pos}}{N_{total}} > 10\%$$

where N_{pos} is the number of positive records in the current session and N_{total} is the total number of records in the current session.

- If the session is valid, it must have some positive records. We denote T_1 as the timestamp of its first positive record, T_2 as the timestamp of its last positive record, and T_r as the timestamp of any data record r in this session. Then, we only considered r as a valid training data if it simultaneously satisfies the following two conditions:

$$T_r - T_1 \geq 3$$

and

$$T_2 - T_r \leq 2$$

This is because in this session, the user might not have noticed the recommendations until then he started to reject some and accept some. So we have allowed **3** rejections before the first acceptance and discarded all records before them. Similarly, the user might start ignoring the recommendations again after rejecting two recommendations in a row and never accepted recommendations any more in the current session, so we have allowed **2** rejections after the last acceptance and discarded all records after them in the current session.

The preprocessing results after the step 1-3 are as follows:

We have picked out 9,933,395 records from the original 73,209,277 recommendation records, which is approximately 13.6%, making all the subsequent training possible to be conducted within the short period of time that we are given for this project. Among them, 5,980,185 records are negative and 3,953,210 records are positive. As a result, the ratio of number of negative records to positive records is reduced from 13:1 to 3:2, and experimental results show that these preprocessing steps have effectively enhanced the accuracy of our training results.

C. Step 3: user group division

After filtering out useless or noisy data in the original training data set, we assign the remaining data into 11 groups based on users' age and gender in the third step. A users age and gender have a strong impact on his/her preference. It is more likely people who are about the same age with the same gender to share the similar preference because they may have more similar experience. And on the other hand, people from different age and gender groups are tend to have different preference. To utilize this observation, we separate data into 11 groups and train them separately later. After carefully analyzing the distribution of age in our data set, we first split the users into 5 age groups, which are 0-15, 15-20, 20-25, 25-32, and above 32. And then we separate each group to two groups by users gender. For users without information about their age or gender, we assign them to the undefined group, the 11st group.

D. Step 4: matrix construction

In this part, we will explain how to construct two kinds of matrix, user-keyword matrix K and user-user matrix S, which are used in the model. Since users are split into 11 groups, there are 11 user-keyword matrixes and 11 user-user matrixes accordingly.

1) user-keyword matrices (K):

In the file user_key_word.txt of the dataset, each user has a list of weighted keywords. Keywords is in the form "kw_1:$weight_1$;kw_2:$weight_2$; ... kw_n:$weight_n$". The greater the weight, the more interested the user is with regards to the keyword.

To represent users' interests towards different kinds of keyword, we construction the user-keyword matrixes, with a user in each row, and a keyword in each column. if user i has keyword j, K(i, j)= weight; otherwise, K(i, j) = 0.

There are approximately 250, 000 keywords in the dataset. To reduce matrix dimension and speed up calculation, we only extract top 5 keywords for each user. Thus we managed to reduce the keyword number to about 120, 000.

2) user-social matrices (S):

The user-user matrixes S are used to measure the unidirectional social influence of one user to another. $S(i, j)$ represents the social influence of user j to user i. To quantify the social influence, we assume that the more frequently user i take a social action on user j, the more close his relationship is to user j. Specifically, the matrixes are constructed using the data of users' follow history and social actions (@, comment, retweet).

Considering the impact of different factors, we assign a weight to each factor according to their importance, with "follow" the most important one. The weight of "follow", "at", "comment" and "retweet" are 5, 2, 0.2

and 1 respectively. Each entry in the matrix is calculated by the following formula.

$$S(i,j) = 5 \times Follow(i,j) + 2 \times N_{at}(i,j)$$
$$+0.2 \times N_{comment}(i,j) + 1 \times N_{retweet}(i,j)$$

$Follow(i,j)$: user i followed user j? 1: 0
$N_{at}(i,j)$: the number of times user i "at" user j
$N_{comment}(i,j)$: the number of times user i "comment" on user j
$N_{retweet}(i,j)$: the number of times user i "retweet" user j

IV. Training

A. Cost Function

After formulating our final model, we can derive our loss function for each group.

$$C_g = \sum_{(u,i)\in T_g} (\hat{r}_{ui} - r_{ui})^2 + \lambda_1 \sum_{k\in K(u)} \left\|p_k^K\right\|^2$$
$$+\lambda_2 \sum_{s\in S(u)} \left\|p_s^S\right\|^2 + \lambda_3 \sum_i \|q_i\|^2$$
$$+\lambda_4 \sum_i (b_i^I)^2 + \lambda_5 \sum_u (b_u^U)^2$$

In this cost function, C_g is the cost for group g, T_g is the training set for group g. And other variables are defined before.
From this cost function, we can see that this is a function that has many local minimum. For example, when all latent factors and bias are equal to 0, this function can reach its local minimum. So in order to get satisfactory results, initialization of latent factors and the parameter tuning are very important.

B. Latent Factor Training

In the training step, we use **Gradient Descent** to train the latent factors.

1) Gradient Descent: Gradient descent is a first-order optimization method for minimizing an objective function that is written as a sum of differentiable functions. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. [4] The basic form of gradient descent is show as follow:

$$w_{n+1} \leftarrow w_n - \alpha_n \Delta F(x_n)$$

where α is the learning rate and $\Delta F(x_n)$ is the derivative of the loss function. In each iteration, update the weight w on the basis of the gradient of the loss function. If α is proper and the initial value w_0 is close enough to the optimum, this algorithm can achieve linear convergence. In another word, we can get a local minimal value of the loss function after iterations.

2) Training Step: Therefore, during each step, we update the latent factors and bias by the following expressions.

$$p_k^K \leftarrow p_k^K - \alpha(\sum_{(u,i)\in T_g \cap u\in U(k)} (\hat{r}_{ui} - r_{ui}) w_{uk}^K q_i^T$$
$$+\lambda_1 p_k^K)$$

$$p_s^S \leftarrow p_s^S - \alpha(\sum_{(u,i)\in T_g \cap u\in U(s)} (\hat{(r)}_{ui} - r_{ui}) w_{us}^S q_i^T$$
$$+\lambda_2 p_s^S)$$

$$q_i \leftarrow q_i - \alpha(\sum_{u\in U} (\hat{r}_{ui} - r_{ui})(\sum_{k\in K(u)} w_k^K p_k^K + \sum_{s\in S(u)} w_s^S p_s^S)^T$$
$$+\lambda_3 q_i)$$

$$b_i^I \leftarrow b_i^I - \alpha(\sum_{u\in U} (\hat{r}_{ui} - r_{ui}) + \lambda_4 b_i^I)$$

$$b_u^U \leftarrow b_u^U - \alpha(\sum_{i\in I} (\hat{r}_{ui} - r_{ui}) + \lambda_5 b_s^S)$$

In these expressions, $(u,i) \in T_g \cap u \in U(k)$ means that the keyword k's latent factor is updated only with the training data of users who have weight on this keyword. Similarly, $(u,i) \in T_g \cap u \in U(s)$ means that the social influence s's latent factor is updated only with training data of users who have social relation with this user. U is the set of users, and I is the set of items.

V. Recommendation

In this part, we will talk about how to decide which items should be recommended to each user. For this purpose, it suffices to solve the problem of designing an algorithm, which aims at choosing at most 3 items from the candidates listed in testing dataset that may be recommended to each user, and recommending them to each user.

A. Testing Dataset

The testing dataset provided by KDD cup 2012 track 1 is given in the file rec_log_test.txt. The format of this file is:

$$(UserId)\backslash t(ItemId)\backslash t(Result)\backslash t(timestamp)$$

This file is pre-processed by KDD cup 2012 to make it easy to use. There is no repeated recommended (UserId, ItemId) pairs in this file. All the lines are sorted in ascending order by timestamp, and are divided into two sets: any line with timestamp smaller than 1321891200 belongs to public set, and equals to or larger than 1321891200 belongs to private set. The value of "Result" field in each line is set to be 0, which means whether the user accepts the item or not is to be determined. Now we can state our task more specifically. In this file, a user may appear in many (UserId, ItemId) pairs, from which we should choose at most 3 items from the users (UserId, ItemId) pairs to recommend to this user.

B. Recommendation Scheme

The main idea of my recommendation scheme is **User Classification**, which follows the same strategy as we preprocess user data to classify users. Consider a user-item tuple listed in rec_log_test.txt. The first step is to classify user into two groups: **active user group** (users who belong to one of the 11 groups we defined before) and **inactive user group** (users who dont belong to any one of the 11 groups). For active user, further classify her into 11 smaller groups which is we defined before; for inactive user, we dont have so many data related to him, so an alternative method is to consider the influence that some active users have on this inactive user.

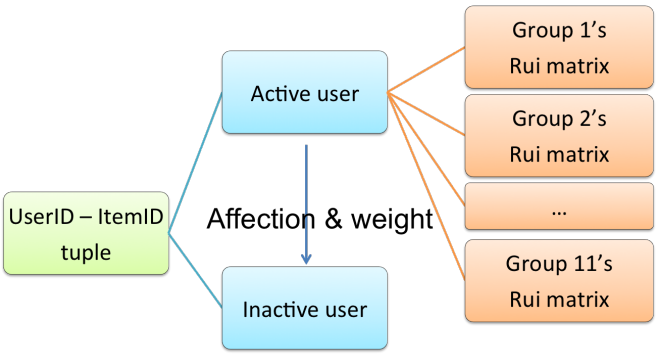

Fig. 1. Recommendation Scheme

From training part, we get the rating of each active users to each items. The data is stored in matrix in for each groups. With this data, the main recommend scheme is as follow:

(1) For each active user u

- Get user u's profile from user_profile.txt and find out which group g user u belongs to
- Use userId and itemId as index to get the rating value user u to item i from group gs rating matrix
- Sort all the items related to user u by their ratings in descending order
- Recommend 3 highest rated items to user u, and discard the rest

(2) For each inactive user u

- Get all the active users that affect user u. This relation can be derived from user_action.txt and user_sns.txt files.
- For each active user, do the steps in (1) to get his rating for this item i
- Calculate the weighted sum of active user ratings for this item, and set it to be the rating of inactive user u for item i. The weight is the same as $S(i,j)$ defined in the preprocessing part.

$$R_{ui} = \sum_{i \in itemset} \sum_{u \in activefolloweeset} W_u * R_{ui}$$

- Sort all the items related to user u by their ratings in descending order
- Recommend 3 highest rated items to user u and discard the rest

VI. Experiments

The whole procedure is shown in Fig 2.

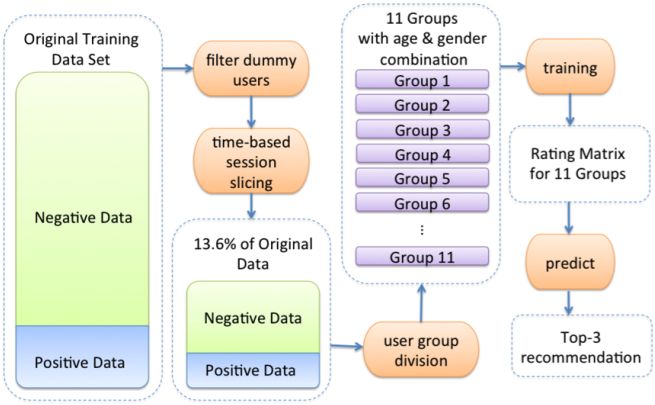

Fig. 2. Procedure of data preprocessing, training and prediction

My programs is run on Mac OS Yosemite system. With CPU 2.6GHz Intel Core i5 and memory 16G and disk 256G.

- The preprocessing programs are written in python, and the total running time is around 4 hours.
- The training program is written in C. We haven't use any other library and implement our own sparse matrix addition, subtraction and multiplication. For each model, the C program will run in 10-20 minutes and the data produced by the training step is around 60G large. And will be used for recommendation part.
- For the recommendation, the program is written in python and the running time is around 5 minutes.

A. Evaluation metric

We need a metric to evaluate my recommendation result. Since my project comes from KDD cup 2012 track 1, we adopt the metric given on its website. Suppose that m ordered items - which means the former the item is listed, the more highly the item is recommended - are recommended to a user, and the user clicks some (one or more or none) of them to follow. We adopt the definition of **average precision** in IR [5] [6], the average precision at n for this user is:

$$ap@n \equiv \frac{\sum_{k=1...n} P(k)}{\#\ of\ clicked\ in\ m\ items}$$

In this formula, if the denominator equals to zero, the result is set to be zero; $P(k)$ represents the precision at cut-off k in the item list, in other words, the ratio of number of clicked items up to the position k over the number k, and $P(k)$ is set to be 0 when kth item is not clicked by the user; according to the requirement of this competition, the default number of items that should be recommended to each user can be 0-3.
For example, if 5 items were recommended to the user, and the user clicked #1,#3 and #4, then

$$ap@3 \equiv \frac{\frac{1}{1} + \frac{2}{3}}{3} \approx 0.56$$

The average precision for N users at position n equals to the average of the average precision of each user, i.e.

$$AP@n \equiv \frac{\sum_{k=1...n} ap@n_i}{N}$$

B. Preprocessing

After preprocessing, we shrink down the training set to 9M and got 11 groups of different size (In Table III).

TABLE II
GROUP DATA

Group	Gender and Age	Size
1	Male, 0-15	75,265
2	Female, 0-15	110,383
3	Male, 15-20	105,343
4	Female, 15-20	84,148
5	Male, 20-25	171,801
6	Female, 20-25	163,662
7	Male, 25-32	92,506
8	Female, 25-32	88,953
9	Male, 32+	62,685
10	Female, 32+	59,297
11	unknown, unknown	211,50

Since LFM will predict more accurate with more data and efficiently, so I try my best to divide the group evenly. We have taken into consideration of both social behaviors for different groups and the limited time and memory we have. For example, we can see that the age ranges for group 3-8 are quite small. Because we think that for people in this range may have totally different behavior. People from 15 to 20 years old may be still in high school or just entering colleges while people between 20 and 25 are more mature and have more opinion about life and society, and people from 25 to 32 will be those who already enter the society, whose life could be totally different from the other two groups. And also since most of the Weibo users lies in this age range, so by dividing like this, we can avoid to oversizing groups that could not load into the memory.

C. Latent Factor Training

In experiment, we choose 128 as the number of latent factors for users and items. And we do 100 gradient descent for each model. To make the result satisfactory, the initialization of p^K, p^S and q is very important. We first initialized these latent factors with random 1 and -1, then normalized every p^K, p^S and q by dividing 11 (which is $\sqrt{128}$). In this way, the 2-norm of each latent factor is no larger than 1 and the $\hat{r}_{ui}$ calculated from this can be mostly in $[-1, 1]$. And for the bias we initialize them as zero.

D. Parameter Tuning

For every model, the learning rate α and the regularization parameter λ_1 to λ_5 need to fit the size of data.

Since the largest model has 172k users. The learning rate for this model should be smaller than other groups to make it converge, since difference of each step may be very large and the cost may jump away from the local minimum. And we also need to adjust every λ to avoid overfitting. Since λ_1 to λ_3 are all for the latent factors, to simplify, we let them be the same (λ). And for the same reason, we let λ_4 and λ_5 be the same (λ_{bias}).
We tuned the parameter for each model by following steps:

- run the training with and test whether the cost function and the error is declining at each gradient descent step as in Fig 3. And find the largest α that makes it descend.
- To tune the λ and λ_{bias}, we first choose a reasonable pair of values to get the rating matrix of a model. Then we use this matrix to do recommendation and get the MAP@3 value for active users in that group. And then change another pair to maximize the MAP@3.

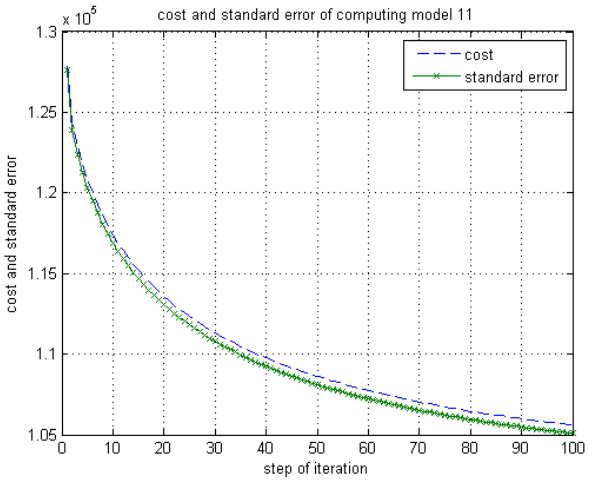

Fig. 3. Convergence for Model 11

Table IV shows my tuned parameters and the MAP@3 for each model.

TABLE III
PARAMETER FOR EACH MODEL

Group	α	λ	λ_{bias}	MAP@3
1	0.00004	0.003	0.002	0.431
2	0.00003	0.003	0.001	0.432
3	0.00003	0.003	0.001	0.452
4	0.00004	0.003	0.002	0.446
5	0.00002	0.004	0.001	0.434
6	0.00001	0.004	0.001	0.433
7	0.00003	0.003	0.002	0.391
8	0.00003	0.002	0.002	0.401
9	0.00005	0.003	0.002	0.366
10	0.00005	0.003	0.002	0.390
11	0.00005	0.004	0.002	0.446

From this table, we can see that Group 9 has the lowest MAP@3, which strongly affect the total MAP@3. The group 9 is consist of male above 32. In common sense, it is reasonable, because the interest for people in this group could be more volatile than other groups. Since male at this age may be someone just start their career and also could be someone who may anticipate a change in their future. So model for this group has lower MAP@3 than other groups.

E. Recommendation Results

We user the **click dataset** to evaluate my results. The click dataset is the dataset shows which items each user has clicked in real. It is given in the file KDD_Track1_solution.csv. The format of this file is:

$$(UserId)\backslash t(Clicks)\backslash t(Indicator)$$

The indicator can be “Public” and “Private” for public and private board in the contest.

Before parameter tuning, the MAP@3 for public and private combined dataset is 41.7%, and after tuning, we improved MAP@3 to 42.2%. For my final model, the MAP@3 is 43.2% for public dataset and 41.0% for private dataset. Here we compare our result for combined dataset with the 3 highest ranked results gotten by all the teams participating in that competition.
Fig 4 shows the comparison of their results and my result in public dataset. We can see that my result is better than the FICO group and slightly lower than the other two groups.

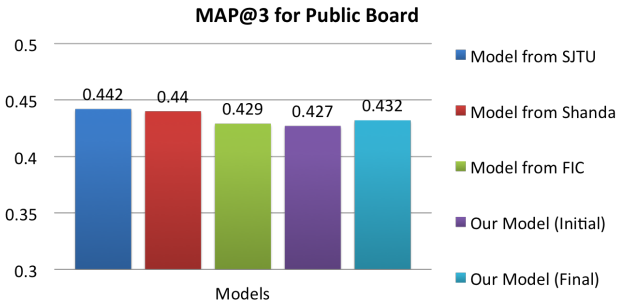

Fig. 4. Comparison of Different Results for Public Dataset

Fig. 5 shows the comparison in private dataset. My result is slightly lower than the top three groups.

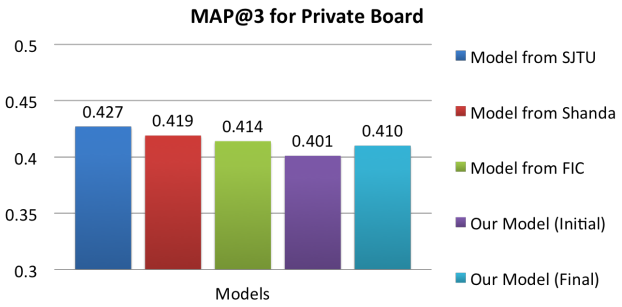

Fig. 5. Comparison of Different Results for Private Dataset

Although my result cannot beat the first two competitors, we bring up some improvements.

- My model cuts 90% useless data from the original data, while the teams participating in the KDD cup 2012 track 1 competition only cut a most 50% of the original data.

- My model has high accuracy under limited computation resource. It takes at most less than 1 hour to calculate the largest rating matrix, and it takes at most 5G memories to calculate all the result.

VII. Conclusion & Future Work

A. Conclusion

In this paper, the author proposed a solution for items recommendation problems described in KDD Cup 2012 Track 1 with the data set sponsored by Tencent Weibo and designed a time-aware combined model based on latent factor model. We studied the basic form of latent factor model and different extensions of matrix factorization as well as the achievements of predecessor competitors. Based on LFM, we significantly improved the prediction accuracy by introducing user-keyword latent factors and utilizing the information of users social network and activities. The MAP@3 of all groups is improved by about 9% after integrating these implicit feedbacks. Also, we pre-processed the original training data set with a series of steps to balance the positive and negative training data and cur off nearly 90% data by utilizing users profile information, users SNS information, as well as the timestamp of each piece of recommendation. In final evaluation, we achieved a 0.432 MAP@3 for Public Board and 0.410 MAP@3 for Private Board.

B. Future Work

Due to limited computing resources, we optimized the training data by cutting off 90% of the original data. By utilizing more information, we may achieve a higher score on MAP@3. The method we used to minimize the loss function and optimize the final model in this paper is Gradient Descent, which is not guaranteed to find the global optimal solution. In another word, the initial weights of keyword latent factors, user SNS latent factors and item latent factors may have an impact on the final result after steps of gradient descent. Moreover, the values of learning parameter α and regularization parameter λ are also very sensitive to my model and experiments. In this project, we tuned these parameters in limited time, but it still can be improved by more scientific induction and tests.

Acknowledgment

Foremost, I would like to express the deepest appreciation to Prof. Byron C Wallace, who has shown the attitude and the substance of genius: he continually and persuasively conveyed a spirit of adventure in regard to research and scholarship. His patience, and profound knowledge consistently motivate me to be insightful to the problems and he always put forward the constructive suggestion and innovative idea to my project. Without his supervision and constant instruction, this project would not have been possible.

Appendix I: How To Run the Code

In this part, we will show how to run the code step by step. To run the code, we recommend the environment as follow:

- RAM: 16G or more
- Disk Space: 70GB or more
- gcc version: 4.2.1
- python version: 2.x

TABLE IV
FOLDER STRUCTURE

Path	Description
./predata	folder for preprocessing data
./preprocessing	python scripts for preprocessing
./track1	original data get from KDD Cup 2012 website
./training	C program for training
./predict	python scripts for recommendation

My code is included in the uploaded file on CCLE. And you can also download it from my github repo:

https://github.com/gl8429/dataMining

C. Preprocessing

First make sure that **track1** is under the same folder with **preprocessing**

- $ **python step1.py** *#filter out dummy user*
- $ **python step2.py** *#filter out dummy user*
- $ **python step3.py** *#time-based session slicing*
- $ **python step4.py** *#user group division*
- $ **python step5.py** *#generate keyword matrix*
- $ **python step6.py** *#generate user social network matrix*
- $ **python step7.py** *#generate training matrix* R_{ui}

After running above steps, we will get matrixes for every group:

$$Rui\_\{1-11\}\_nm.txt$$
$$S\_\{1-11\}\_n\_nu.txt$$
$$S\_\{1-11\}\_nu\_n.txt$$
$$user\_key\_word\_\{1-11\}\_ij.txt$$
$$user\_key\_word\_\{1-11\}\_ji.txt$$

If you want to save time running the preprocessing please refer to Google Drive: link
Download these data and then put it under the same folder with the **training**.

D. Training

In training, make sure that **training** folder is under the same folder with **predata**.

- $ **sudo purge** *# clear the disk cache so that there is enough* memory for running the program
- $ **make**
- $ **./a.out [group-id]** *# group-id should be from 1 through 11*

After training every model, we generate rating matrixes for every group.

$$model\{1-11\}.csv$$

E. Recommendation

The predict script is under **predict** folder. If you only want to get the predict result for all the users (both inactive and active users included), just do the following steps:

- $ **python generate_inactive_relation_active.py**
- $ **python predict.py**
- $ **python MAP.py**

If you want to get the predict result for the active users belong to a certain group, do the following steps

- $ **python generate_inactive_relation_active.py** *#if you run this command in 1, there is no need to run it again*
- $ **python one-group-predict.py [group-id]** *# here group-id should be one of 1 to 11*
- $ **python one-group-MAP.py [group-id]** *# here group-id should be one of 1 to 11*

If you want to get the predict result for "Public" solution (please see "KDD_Track1_solution.csv" to see which clicks are "Public")

- $ **python generate_inactive_relation_active.py** *# if you run this command in 1, there is no need to run it again*
- $ **python predict-public.py**
- $ **python MAP-public.py**

If you want to get the predict result for "Private" solution (please see "KDD_Track1_solution.csv" to see which clicks are "Priavte")

- $ **python generate_inactive_relation_active.py** *#if you run this command in 1, there is no need to run it again*
- $ **python predict-private.py**
- $ **python MAP-private.py**

The MAP result will be printed on the screen.

REFERENCES

[1] Chen, Tianqi, et al. "Combining factorization model and additive forest for collaborative followee recommendation." KDD CUP (2012).
[2] Zhao, Xing. "Scorecard with latent factor models for user follow prediction problem." KDD-Cup Workshop. 2012.
[3] Chen, Yunwen, et al. "Context-aware ensemble of multifaceted factorization models for recommendation prediction in social networks." KDD-Cup Workshop. 2012.
[4] Bottou, Lon. "Large-scale machine learning with stochastic gradient descent."Proceedings of COMPSTAT'2010. Physica-Verlag HD, 2010. 177-186.
[5] http://en.wikipedia.org/wiki/Information_retrieval
[6] http://sas.uwaterloo.ca/stats_navigation/techreports04WorkingPapers/2004-09.pdf

[7] T. Chen, Z. Zheng, Q. Lu, X. Jiang, Y. Chen, W. Zhang, K. Chen, Y. Yu, N. N. Liu, B. Cao, L. He, and Q. Yang. Informative ensemble of multi-resolution dynamic factorization models. In In the KDD-CUP 2011 Workshop of 17th ACM SigKDD conference., San Diego, CA, USA., 2011.

[8] M. Jahrer and A. Toscher. Combining predictions for accurate recommender systems. KDD10 Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 693702, 2010.

[9] M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In RecSys, pages 135142, 2010.

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