Scorecard with Latent Factor Models for User Follow Prediction Problem

Xing Zhao
FICO
3661 Valley Centre Drive
San Diego, CA 92130, USA
xingzhao@fico.com

ABSTRACT

This paper describes team mb73's solution to Track 1 of KDD Cup 2012. Using FICO Model Builder software we generated many predictive features and latent factor models. These features were then ensembled together using the Model Builder Scorecard trainer. Session features derived from timestamps are found to be strong predictors. Effective latent factor models are built using information such as followed items, keywords, tags and user actions. Interactions among features are detected, and feature pairs with cross binnings are introduced within the final scorecard model to capture these interactions.

Categories and Subject Descriptors

I.2.6 [Machine Learning]: Engineering applications

General Terms

Algorithm

Keywords

Recommendation system, collaborative filtering, model builder, scorecard

1. INTRODUCTION

The Track 1 problem of KDD Cup 2012 is to predict which user another user might follow in Tencent Weibo [5]. Tencent Weibo is one of the largest micro blogging websites in China. This is another recommendation challenge, similar to last year's Yahoo music recommendation [2] and the Netflix movie recommendation [1] problems.

Such recommendation problems are usually solved by the collaborative filtering (CF) approach, which relies only on information about the behavior of users in the past. There are two primary methods in CF: the neighborhood approach and latent factor modeling. Both of these methods have been proven to be successful for the recommendation problem. A key step in CF is to combine these models. Methods such as regression, gradient boosting decision trees, and neural networks have been used to ensemble CF models [4, 7], while our solution uses a scorecard to create the final ensemble.

This year's problem is different in several ways from the music and movie recommendation challenges. The scope of the problem is large. There is wealth of auxiliary information on each user and each item. For example, each user possesses age, gender, tags and keywords, as well as a history of followed items, and a history of connections with other users. The measurement is average precision [8], which is quite different from traditionally used RMSE. These make this year's problem special and interesting to solve.

In our solution, we generate many predictive features as well as latent factor models and interaction terms. Finally we train a scorecard model to ensemble these features, interaction terms, and latent factor models. We find that the scorecard model is an effective tool to inspect, learn, and continuing innovating the model, as well as providing a strong formula with which to ensemble all the constituent elements. This is a marked and novel departure from scorecard's usual application in credit risk modeling.

The rest of the paper is organized as the following. Section 2 introduces scorecard and outlines how the problem is solved. The details of feature creation are described in Section 3. Different latent factor models are created in Section 4. Results and Discussions are summarized in Section 5.

2. SCORECARD

2.1 Introduction

The most common prediction task is to estimate a function *f* from predictors to target domain. For example, linear regression model is a linear combination of predictors:

$$f = w_0 + \sum_{i=1}^{I} w_i x_i,$$

where x_i is the value of the ith predictor, and w_i the regression weight to be trained.

Compared with regression models, scorecard model has a binning step to divide each predictor space into bins, and then assign score weight to each bin.

$$f = w_0 + \sum_{i=1}^{I} f_i(x_i),$$

where $f_i(x_i)$ is the predictor score:

$$f_i(x_i) = \sum_{j=1}^{J_i} w_{ij} b_{ij}(x_i)$$

$$b_{ij}(x_i) = \begin{cases} 1 & \text{if value of } x_i \text{ belongs to the jth bin} \\ 0 & \text{else} \end{cases}$$

w_{ij} : the score weights associated with bin j for predictor x_i.

b_{ij} : the dummy indicator variables for the bins of predictor x_i.

Bins for all predictors are generated using various binning algorithm, which also support categorical predictors. Missing values can be handled easily by using a missing value bin. Given enough bins for a predictor x_i, the above predictor score function $f_i(x_i)$ is flexible to approximate any general function based on a single predictor, and the scorecard model is the sum of such functions. The complete, compact representation of a model by its bin definitions and weights makes the scorecard a popular, transparent, and easily understood model formulation. And the scorecard's ability to approximate general functions makes it a strong predictive tool.

The scorecard is trained to optimize the bin weights such that the model's prediction error is minimized. Scorecards can be used to predict both continuous and binary targets. For continuous targets, the error function is usually RMSE. For binary targets, common objective functions include maximizing Kullback–Leibler divergence or Bernoulli likelihood. Details for the scorecard method can be found in the Model Builder product document [9].

Scorecard is much more powerful than regression models, and still simple enough to be interpretable. By looking into the bin weights for the predictors, lots of insight can be gained for the data.

In practice, there can be interactions among predictors, and scorecard has several techniques to deal with such interactions.

One approach is to use segmented scorecard which is similar to a decision tree. When using a traditional decision tree as a predictive model, the leaf node typically returns the dominant outcome (classification) or mean value (regression). With segmented scorecard, the leaf node is associated with a scorecard model. The well-known FICO score, which is the most widely used score by US's largest banks to make credit and loan approval decisions, uses this segmented scorecard approach. By placing individual scorecards in the leaf nodes of a decision tree, the whole ensemble model can capitalize on interactions among the predictors.

Another approach to capturing interactions is to use cross binnings between a given predictor pair (x_{i1} , x_{i2}).

$$f_{i1,i2}(x_{i1}, x_{i2}) = \sum_{j=1}^{J_{i1i2}} w_{i1i2,j} b_{i1i2,j}(x_{i1}, x_{i2}) \qquad (2.0)$$

$$b_{i1i2,j}(x_{i1}, x_{i2}) = \begin{cases} 1 & \text{if values of } x_{i1} \text{ and } x_{i2} \text{ belongs to the jth cross bin} \\ 0 & \text{else} \end{cases}$$

$w_{i1i2,j}$: weight for cross bin j for predictor pair (x_{i1} , x_{i2})

$b_{i1i2,j}$: indicator variables for cross bins of predictor pair (x_{i1} , x_{i2})

Here the predictor pair is divided into many cross bins. For example, if x_{i1} has M bins and x_{i2} has N bins, then there are MxN cross bins for the (x_{i1} , x_{i2}) pair.

When there are many predictors, it is not practical to simply include all possible pairs in the model. Thus, finding the predictor pairs with strong interactions is critical. Model Builder provides a function to detect pairwise interactions efficiently. The key idea is to iterate through all predictor pairs and train scorecard model using only the predictor pair as described in (2.0). This model is then compared to scorecard model with two additive (non-crossed) predictors described in (2.1):

$$f_{i1}(x_{i1}) + f_{i2}(x_{i2}) = \sum_{j=1}^{J_{i1}} w_{i1j} b_{i1j}(x_{i1}) + \sum_{j=1}^{J_{i2}} w_{i2j} b_{i2j}(x_{i2}) \qquad (2.1)$$

If the variable pair has strong interaction, the model in (2.0) will have notably stronger performance than model (2.1). Thus these differences in model performance can be used to measure the degree of any pairwise interaction. Once predictor pairs with strong interactions are found, such pairs can be included in the scorecard model with other predictors:

$$f = w_0 + \sum_{i=1}^{I} f_i(x_i) + \sum_{(i1,i2)} f_{i1,i2}(x_{i1}, x_{i2}) \qquad (2.2)$$

The above scorecard model captures important pairwise interactions among predictors. This approach has been used to detect and model interactions among predictors in our solution to the Track1 problem.

2.2 Problem Setup

The Track 1 task is to predict whether a user will "follow" a given recommended item. The training data has 70M records spanning 31 days. Test1 data has 19M records for the 10 days immediately following the training data. Test2 data has 15M records for the next 9 days after test1. Predictions of test1 are used for the public leader board, and predictions for test2 are used for the final competition evaluation.

We first performed data cleaning for the training data, removing duplicate items for the same user. This is important because the same cleaning step has been done for the given test data. We then removed "dummy" users whose recommended items were either universally followed, or universally ignored. Such uninteresting records will not help the model training. After cleansing the data as described, 36M records remained in the training data.

Then we split the training data to two parts, where train1 consists of the first 22 days with 26M records, and train2 contains the final 9 days with 10M records. Initially we used train1 data for feature generation and latent factor model training. Train2 data is used as the validation data for latent factor model, as well the training data for the scorecard model. Later it is noticed that the latent factor model is more predictive with more training data. Thus train2 data is further split into train2_a and train2_b based on user_id, where train2_a has 1M records and train2_b has 9M records. In the rest of the paper, we refer to train1+train2_b as T and train2_a as T2. Now we have T with 35M records for latent factor model training and feature generation. T2 with 1M records in 9 days is for scorecard training and latent factor model validation. This kind of data split is to keep T2 close to test2, and also have as much data as possible in T for latent factor model training and feature generation.

2.3 Average Precision

The Track 1 task uses the average precision at 3 for all users (MAP@3) as the performance measure [8]. This objective is special because it only considers the top 3 scores for a given user. In addition, a user with more followed items is less important given the way that average precision is defined. We find that the MAP objective can be addressed indirectly by introducing proper weights for the training records. The key is to balance the weight of follow records and recommend records for each user, and also adjust the user's importance based on user's total follow records.

This can be achieved with the following training record weight w_{ui} for user u and item i:

y_{ui}: the target value of train record. $y_{ui}=1$ is a follow record. $y_{ui}=0$ is a recommend record.

H_u: the count of follow records for user u in training data

L_u: the count of recommend records for user u in training data.

$$w_{ui} = \begin{cases} 1/H_u & \text{if follow record} \\ 1/L_u & \text{if recommend record} \end{cases} \quad (2.3)$$

The above record weights w_{ui} are used in the objective function for both scorecard training and latent factor models training.

Our solution can then be described as several steps. First we generate many predictive features from the raw data. Then we train latent factor models using T and generate their model scores on T2. Finally, we train scorecard model on T2 using all features and model scores.

3. PREDICTIVE FEATURES

Finding strong predictive features is critical to all machine learning techniques, including scorecard model development. In this section, we introduce the features that are used in the scorecard model.

3.1 Item Popularity Features

Item popularity is a basic predictive feature generated as the following:

$follow_i$: count of follow records for item i in T

$recommend_i$: count of recommend records for item i in T

$ratio_i$: $follow_i$ /$recommend_i$

The Track 1 problem has rich information, including the user age and gender. Intuitively, the popularity of items for users of different gender may be different. Similarly, users from different age groups may express interests in very different items. We divide the users into different age groups, and each big age group is further split to two gender groups. Totally we have 15 such user groups. Then we have user group based item popularity features:

$follow_{i,ug}$: count of follows for item i and user group ug in T

$recommend_{i,ug}$: count of recommend records for item i and user group ug in T

$ratio_{i,ug}$: $follow_{i,ug}$ / $recommend_{i,ug}$

Another useful feature is the difference between $follow_{i,ug}$ and $recommend_{i,ug}$ after normalization:

$follow\_sum_{ug}$: sum of $follow_{i,ug}$ for all items

$recommend\_sum_{ug}$: sum of $recommend_{i,ug}$ for all items

$follow\_norm_{i,ug} = follow_{i,ug}/ follow\_sum_{ug}$

$recommend\_norm_{i,ug} = recommend_{i,ug}/ recommend\_sum_{ug}$

$diff_{i,ug}$: $follow\_norm_{i,ug} – recommend\_norm_{i,ug}$

$follow_i$, $ratio_i$, $ratio_{i,ug}$ and $diff_{i,ug}$ are predictors included in the final model.

3.2 Session Features

Track 1 data also provides timestamps on the train/test records. Usually several records are recommended to a user at the same time. Usually several records are recommended to a user at the same time. We define records recommended to the same user at the same time as a “batch”. We also define “session” as a sequence of batches to the same user. If there are several hours between two adjacent batches, we consider a new session is started. Most users only have one session, and users with multiple sessions are the returning users. The following session related features are generated and found predictive:

pre_batch: the seconds between the current batch and the previous batch.

next_batch: the seconds between the current batch and the next batch.

session_start: the seconds from session start to the current batch.

session_end: the seconds from the current batch to session end.

session_start_batches: the number of batches from session start to the current batch.

session_end_batches: the number of batches from current batch to session end.

Among these features, next_gap is the strongest predictor. Intuitively after following an item the user may spend some time looking into the item, thus there is usually no further recommendation immediately after a follow. For example, if there is another batch one second after the current batch, the odds that there is a follow in the current batch are less than 0.1.

For this next_gap feature, around one third of records have missing value because they are the last batches in the session. Scorecard models handle missing value nicely by creating a bin to capture such records, and the scorecard training will find the best bin weight for the missing values.

As explained in Section 2, Model Builder can detect pairwise interactions among predictors. Strong interactions have been found for pair (next_batch, pre_batch), (next_batch, session_end_batches) and (next_batch, session_start_batches). These feature pairs are included in the scorecard model as described in (2.2) to capture the pairwise interactions among session features.

3.3 Keyword Features

In Track 1 data, each user has a set of weighted keywords which are extracted from user’s tweet/retweet/comment. We want to measure the correlation between a keyword and an item. When a user follows an item, the keyword in user’s keyword set is connected to the item. The keyword that is more frequently connected to an item probably has stronger correlation with that item. The following features are generated based on such keyword and item correlation.

Ku: set of keywords for user u.

w_{ku} : weight of keyword k for user u, where $k \in Ku$

$follow_{i,k}$: the sum of keyword weights for all train records when keyword k is connected to item i.

$$\sum_{ui\in T} w_{ku} \quad st. \quad k \in K_u, y_{ui} = 1$$

$\text{recommend}_{i,k}$: the sum of keyword weights for all train records when keyword k is not connected to item i.

$$\sum_{ui\in T} w_{ku} \quad st. \quad k \in K_u, y_{ui} = 0$$

follow_max_{ui}: the max of $\text{follow}_{i,k}$ on Ku given user u and item i.

k_max_{ui}: the keyword where the above max is found.

follow_diff_{ui}: follow_max_{ui}- $\text{recommend}_{i,\text{k\_max}}$

Several other keyword related features are also predictive by themselves, but do not strongly contribute to the final model, because the final model has keyword-based latent factor models which are very strong predictors. The above follow_max_{ui} and follow_diff_{ui} features help the final model, because they use the max operation which is different from the latent factor model.

Another feature included is based on simple keyword matching.

$\text{weights\_matched}_{ui}$: sum of the weights of the matching keywords between a user and an item.

For this feature, more than 95% records do not have any matching keywords between the user and the item. Though this predictor is very weak, it is different from other keyword predictors and helps the final model slightly.

3.4 Other Features

Here are some of the other features that are included in the final model.

Similar to user group based item popularity using the training records, we generate the user group based item popularity using the user_sns data which has the history of user followed items.

$\text{follow\_sns}_{i,ug}$: count of user_sns records for item i in user group ug

$\text{follow\_sns\_sum}_{ug}$: sum of $\text{follow}_{i,ug}$ for all items

$\text{follow\_sns\_norm}_{i,ug} = \text{follow}_{i,ug}/ \text{follow\_sum}_{ug}$

Feature $\text{follow\_sns\_norm}_{i,ug}$ is included in the model.

Using the timestamps of train/test records, we generate the day and hour features in Beijing time. The odds for users to follow are slightly different on different days of the week and different hours of the day. These two features are weak predictors in the final model.

Item category information is used to generate the following feature:

F_u: user u's followed items in user_sns data and training data T.

p_i: item i's direct parent category

$FP_{u,i}$: items in F_u that have the same direct parent as item i

$$FP_{ui} = \{ j \in F_u, st\ p_j = p_i \}$$

The size of FP_{ui} is the feature included in the final model.

3.5 KNN Features

The most common approach to CF is based on neighborhood models. User-oriented KNN models estimate unknown item ratings based on the known item ratings of similar users. Item-oriented KNN models are based on known ratings made by the same user on similar items [3].

In our item KNN models, the preference of user u on item i is measured by:

$$r_{ui} = \sum_{j\in S(N(u),k)} w_{ij}$$

where N(u) is the set of items that user u followed. S(N(u), k) includes the top k items in N(u) that are similar to item i. w_{ij} is the similarity between item i and item j.

We use Jaccard index as the item similarity measure

$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|}$$

where N(i) is the set of users who follow item i. Since there are only 6k items, the above item similarity can be calculated and cached, thus saving a lot of computation time.

For Track 1 data, we find that the above item KNN models help the final model very little. Since there is only small overlap between N(i) and N(j) for most items in Track 1 data, it is likely that the Jaccard index based item similarity is not very reliable. We also tried item similarity based on item latent factor distance, but it did not help much.

We experienced computation difficulties using the user KNN models. Given an item and a user, there are many users that have followed the item. The user similarities between these users and the given user have to be calculated on the fly. Since there are almost 2 million users in this problem, the user similarity cannot be calculated and reused efficiently, and computing the Jaccard index based similarity takes too long. As a workaround, we tried using the total number of actions in the user_action data as user similarity. Such user KNN models are predictive by themselves but do not help the final model, probably because there are other latent factor models which already capture the user_action data.

4. LATENT FACTOR MODELS

Models based on collaborative filtering approaches often exploit latent information for hidden structures of users and items from the given data. Latent factor models have been widely used in CF to predict ratings. The key idea is to approximate a rating r_{ui} by the inner product of a user latent factor and an item latent factor. A standard method to learn the latent factors is to minimize the sum of squared errors with regularization applied to prevent overfitting.

For the Track1 problem, directly using latent factors for users does not work very well. Introducing latent factors for user keywords, user tags, user followed items and user related users turn out to be the key for more predictive models.

4.1 Latent Factor for Users

In this basic latent factor model, the prediction of user u's preference on item i is formulated as the following:

$$\hat{r}_{ui} = u + u_u + u_i + p_u^T q_i$$

Here u is the global adjustment, u_u the user based adjustment, u_i the item based adjustment, p_u the user latent factor, q_i the item latent factor.

The above model parameters are trained by minimizing the following cost function:

$$\sum_{(u,i)\in T} w_{ui} (r_{ui} - \hat{r}_{ui})^2 + \lambda (\|p_u\|^2 + \|q_i\|^2)$$

Here T is the training sample, r_{ui} the binary target, λ the regularization parameter. As explained in Section 2, w_{ui} is the record weight introduced in (2.3) to address MAP@3. Stochastic gradient-decent is used to minimize the cost function. We evaluate the MAP using the validation sample T2, and stop the training once the MAP measure no longer improves on the T2 sample.

The above model is the weakest latent factor model for several reasons. First, most users only have very limited follow records, thus there is not enough information to train the user latent factor well. Second, the users in the training dataset do not overlap much with the users in the validation dataset. Those new users only in the validation/test dataset do not have the user latent factor available from the training. Thus, this model only helps the frequently returning users who are both in the training and validation/test dataset. But frequent users are only a small portion of the population. In addition, frequent users usually have more follow records, thus their average precisions are low and their contribution to MAP is small.

4.2 AFM

This model is called asymmetric factor model because it uses only item-dependent parameters. It was first mentioned by Paterek in [6].

$$\hat{r}_{ui} = u + u_u + u_i + q_i^T \left(\frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} q_j^0 \right)$$

Here I(u) are the followed items by user u in the user_sns data. q^o is the latent factor for user followed item, and $1/\sqrt{|I(u)|}$ is for normalization. Here the user latent factor is expressed through the user's followed items. There are 40M records in user_sns data for followed items, and most users have records of followed items. The latent factors of these 6k followed items q^o are adequately trained with 35M records using the following cost function:

$$\sum_{(u,i)\in T} w_{ui}(r_{ui} - \hat{r}_{ui})^2 + \lambda\left(\sum_{j\in I(u)} \|q_j^0\|^2 + \|q_i\|^2 \right)$$

This AFM model is the most predictive simple latent factor model.

4.3 Latent Factor for Keywords and Tags

The above AFM model represents each user through their followed items. Similarly we can represent the user through their keywords or tags.

All users have weighted keyword information, and there are around 250K unique keywords. We can build effective models by introducing latent factors for user keywords.

$$\hat{r}_{ui} = u + u_u + u_i + q_i^T \left(\sum_{k\in K(u)} w_k p_k \right)$$

Here p_k is the user latent factor for keyword k, K(u) the keywords for user u, w_k the keyword weight. Compared with traditional user item latent factor model, here the user latent factor is expressed through user's keyword latent factors.

The above model parameters are trained by minimizing the cost function:

$$\sum_{(u,i)\in T} w_{ui}(r_{ui} - \hat{r}_{ui})^2 + \lambda\left(\sum_{k\in K(u)} \|p_k\|^2 + \|q_i\|^2 \right)$$

This keyword based latent factor model is very predictive because all users have keyword information. There are 35 M records in the training data used to train latent factors for 250K keywords, making the keyword latent factor trained with enough data.

Similarly, we have the tag based latent factor models.

$$\hat{r}_{ui} = u + u_u + u_i + q_i^T \left(\frac{1}{\sqrt{|Tag(u)|}} \sum_{t\in Tag(u)} p_t \right)$$

Here p_t is the user latent factor for tag t, Tag(u) the tags for user u, and $1/\sqrt{|Tag(u)|}$ for normalization.

The tag based latent factor model is not as predictive as keyword model, because only around one third users have tags.

4.4 Latent Factor for Action Users

There are about 10M records in user_action.txt, detailing all the exchanges among pairs of users. Similarly we can introduce latent factors for these inter-user actions.

$$\hat{r}_{ui} = u + u_u + u_i + q_i^T \left(\frac{1}{\sqrt{|A(u)|}} \sum_{t\in A(u)} p_t \right)$$

Here p_j is the latent factor for user j, and A(u) is set of users that user u has actions with.

In the training, there are around 600k action users involved, each with a user latent factor. With factor = 20, there are 12M parameters for these user latent factors. Given 35M training records, it is easy to overfit. Thus a bigger regularization parameter is used in the cost function. This model is found to be more predictive by itself than the keyword latent model, which is somewhat surprising. Putting into the ensemble though, its contribution is not as much as the keyword model.

4.5 User Group Based Item Adjustment

All of the models discussed thus far include a u_i term representing the average item popularity for all users. In the Track 1 problem, each user possesses age and gender information. Intuitively, any item may express different popularity for users with different age or gender. We thus introduce user group based item adjustment into the latent factor models. For example, the AFM model is enhanced to the following:

$$\hat{r}_{ui} = u + u_u + u_{i,ug(u)} + q_i^T \left(\frac{1}{\sqrt{|I(u)|}} \sum_{j\in I(u)} q_j^0 \right)$$

As explained in Section 3, 15 user groups with different age and gender have been introduced for all users. Here ug(u) is the user group index for user u, and $u_{i,ug(u)}$ is the user group based item adjustment. Instead of 6K item adjustment parameters, now we have 6K*15 item adjustment parameters in the model. These user group based item adjustments are used for all other latent factor models also.

4.6 Combo Latent Factor Models

Here is a combo model including keywords, tags and followed items:

$$\hat{r}_{ui} = u + u_u + u_{i,ug(u)} + q_i^T \left(\frac{1}{\sqrt{|I(u)|}} \sum_{j\in I(u)} q_j^0 + \sum_{k\in K(u)} w_k p_k + \frac{1}{\sqrt{|Tag(u)|}} \sum_{t\in Tag(u)} p_t \right)$$

This is the best latent factor model we found, achieving 0.384

MAP on the public leader board. We also tried adding the action user latent factors into the above combo model, but our bigger model achieves a weaker MAP. It is likely that we do not have the right training parameters. Ideally different step size and regularization should be used for different type of parameters with careful tuning, but we did not have enough time to try this.

Models based on other combinations such as (followed items + keywords), (followed items + tags) and (keywords + tags) are also built and included in the scorecard ensemble.

We use 20 as default factors for all the above models. In our testing, larger factors does not improve model performance much. There are two combo models with factor 100 and factor 150 included in the final ensemble. We also train models without the records weights. Models trained without the record weights are not as good in terms of MAP, but they help the final scorecard because of their differences. All these latent factor models are used to generate model scores on T2. These model scores and features from Section 3 are the predictors for the final scorecard model. In the scorecard model, all latent factor model scores are divided into 40 bins based on score percentile. Most features have around 20 bins. The scorecard model is trained using the Bernoulli likelihood objective function, which is the same as the maximum likelihood used in logistic regression.

5. RESULTS and DISCUSSIONS

Without latent factor models, scorecard model using features only has 0.390 MAP on public board. Latent factor models improve the MAP to 0.422. There are also three feature pairs included in the scorecard model to handle interactions among session features. These feature pairs improves the MAP from 0.422 to 0.428. The following is a table showing our model improvements through time, with the key predictors added to the model.

Table 1. Model Improvements

Date	MAP	Descripiton
3/18	0.340	Item popularity and session features
3/29	0.377	User group based item features
3/30	0.381	Item follow feature from user_sns
4/13	0.390	KNN features, keyword, tag features
4/30	0.404	Keyword,tag latent models
5/1	0.411	AFM
5/30	0.422	More latent models
5/31	0.425	Cross binnings (next_batch, pre_batch),
6/1	0.428	More cross binnings

The above model MAP performance is based on public leader board. As can be seen, our early models only include features. Most of the latent factor models were added in the last month. The cross binning were added in the last two days. After introducing latent factor models, lots of features created early are no longer helpful in the scorecard model. We should have started latent factor models early, which take more time and effort for training and tuning.

Our final scorecard model has 18 generated features, 3 feature pairs, and 16 scores from latent factor models. The strongest single predictor is the combo latent factor model score which has MAP 0.384 on the public board. The improvement from the best single predictor's 0.384 to final ensemble's 0.428 shows that the scorecard works well combining all these features and latent factor model scores, as well as capturing interactions among the features.

Our solution is ranked 5th on the private board, and 4th on the public board. The top two teams have 0.441 and 0.439 MAP on the public board, showing that there are still lots of room for our solution to improve. It has been reported that parameter tuning is important for latent factor models. Such models in our solution probably can be further improved with careful tuning. Another idea is to train latent factor models on different segments, which we did not have time to implement. Techniques to more directly optimize to the MAP@3 measure remains another interesting topic for further investigation.

6. ACKNOWLEDGMENTS

This work is supported by the FICO Model Builder product development team. I would like to thank Peter Hamilton and Andrew Flint for their helpful feedback on a draft of this paper.

7. REFERENCES

[1] J. Bennett and S. Lanning. 2007. The netflix prize. *Proc. KDD Cup and Workshop*

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

[3] G. Linden, B. Smith, and J. York. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1): (Jan. 2003), 76-80

[4] Y. Koren. 2009. *The bellkor solution to the netflix grand prize*. Tech. report

[5] Yanzhi Niu, Yi Wang, Gordon Sun, Aden Yue, Brian Dalessandro, Claudia Perlich, Ben Hammer, 2012. The Tencent Dataset and KDD-Cup'12. *KDD-Cup Workshop*

[6] A. Paterek. Improving regularized singular value decomposition for collaborative fltering. 2007. *Proc. KDD Cup workshop at SIGKDD'07, 13th ACM Int. Conf.on Knowledge Discovery and Data Mining*, 39-42

[7] M. Piotte and M. Chabbert. 2009. *The Pragmatic Theory solution to the Netflix Grand prize*. Tech. report

[8] Mu Zhu. 2004. Recall, Precision and Average Precision, Working Paper

[9] FICO, 2012, *FICO Model Builder 7*. http://www.fico.com/en/Products/DMTools/Pages/FICO-Model-Builder.aspx