A Two-Stage Ensemble of Diverse Models for Advertisement Ranking in KDD Cup 2012

Kuan-Wei Wu, Chun-Sung Ferng, Chia-Hua Ho, An-Chun Liang, Chun-Heng Huang, Wei-Yuan Shen, Jyun-Yu Jiang, Ming-Hao Yang, Ting-Wei Lin, Ching-Pei Lee, Perng-Hwa Kung, Chin-En Wang, Ting-Wei Ku, Chun-Yen Ho, Yi-Shu Tai, I-Kuei Chen, Wei-Lun Huang, Che-Ping Chou, Tse-Ju Lin, Han-Jay Yang, Yen-Kai Wang, Cheng-Te Li, Shou-De Lin, Hsuan-Tien Lin
Department of Computer Science and Information Engineering, National Taiwan University
{r00922007, r99922054, r99922033, b97902004, b97902082, r00922024, b98902114, r00942050, b97902083, r00922098, r00922048, b97902085, b97902053, b97902048, b98901134, r00943037, b98902039, b97902014, b97902050, b97901187, b97902081, d98944005}@ntu.edu.tw, { sdlin, htlin}@csie.ntu.edu.tw

ABSTRACT

This paper describes the solution of National Taiwan University for track 2 of KDD Cup 2012. Track 2 of KDD Cup 2012 aims to predict the click-through rate of ads on Tencent proprietary search engine. We exploit classification, regression, ranking, and factorization models to utilize a variety of different signatures captured from the dataset. We then blend our individual models to boost the performance through two stages, one on an internal validation set and one on the external test set. Our solution achieves 0.8069 AUC on the public test set and 0.8089 AUC on the private test set.

1 Introduction

Track 2 of KDD Cup 2012 is a competition for search advertising. The task of the competition is to predict the click-through rate (CTR) of ads in a web search engine given its logs in the past. The dataset, which is provided by Tencent, includes a training set, a test set and files for additional information. The training set contains 155,750,158 instances that are derived from log messages of search sessions, where a search session refers to an interaction between an user and the search engine. During each session, the user can be impressed with multiple ads; then, the same ads under the same setting (such as position) from multiple sessions are aggregated to make an instance in the dataset. Each instance can be viewed as a vector (#click, #impression, DisplayURL, AdID, AdvertiserID, Depth, Position, QueryID, KeywordID, TitleID, DescriptionID, UserID), which means that under a specific setting, the user (UserID) had been impressed with the ad (AdID) for #impression times, and had clicked #click times of those. In addition to the instances, the dataset also contains token lists of query, keyword, title and description, where a token is a word represented by its hash value. The gender and segmented age information of each user is also provided in the dataset.

The test set contains 20,297,594 instances and shares the same format as the training set, except for the lack of #click and #impression. The test set is generated with log messages that come from sessions latter than those of the training set. Detailed information about the dataset and KDD Cup 2012 can be founded in [17].

The goal of the competition is to predict the click-through rate (#click / #impression) for each instance in the test set. The goodness of the predictions is evaluated by the area under the ROC curve (AUC), which is equivalent to the probability that a random pair of a positive sample (clicked ad) and a negative one (unclicked ad) is ranked correctly using the predicted click-through rate. That is, an equivalent way of maximizing the AUC is to divide each instance into (#click) of positive samples and (#impression-#click) negative samples, and then minimize the pairwise ranking loss of those samples using the predicted click-through rate [10].

During the competition, teams are allowed to upload the predictions on the test set. The AUC calculated on a fixed 42% of the test set (public test set) is shown on the leaderboard; the remaining 58% of the test set (private test set) is used to determine the final standings. Teams are allowed to select up to 5 submissions to be evaluated on the private test set before the end of the competition.

The paper describes the system proposed by National Taiwan University team. According to the leaderboard online,[1] the system reaches the best performance on both of the public and the private sets.

The paper is organized as follows. Section 2 describes the framework of our system. Section 3 discusses all individual models, which tackle the task of the competition from several different perspectives. Section 4 and Section 5 then introduce how we combine the individual models using the validation set and the test set, respectively, to form the final system. Finally, we conclude in Section 6.

2 Framework

We first provide an overview of our proposed system. Then, we discuss a key step in building the system: generating the internal validation set. The set not only is used for parameter and model selection, but also plays an important role in the blending stage. Finally, we show our efforts in another key step: generating meaningful features for training the individual models.

2.1 System Overview

The proposed system can be divided into three stages: generating individual models, blending with the validation set, and ensemble learning with the test set. In the first stage, we apply several different approaches to capture different concepts and learn a diverse set of models. Diversity here represents different perspective for modeling the task. Our experience in earlier KDD Cups [6, 15] shows

[1] Team "Catch up" on `http://www.kddcup2012.org/c/kddcup2012-track2/leaderboard`

that enhancing the diversity could boost the performance when the models are appropriately aggregated — that is, the folklore rule of two heads being better than one. The aggregation happens during the second and the third stages to form the final solution. In the second stage, we apply non-linear blending methods to combine the results from the first stage to improve the prediction performance and to further increase the diversity of our model set. We also conduct re-blending, which feeds the aggregated models into the non-linear blending methods again. Such a validation-set blending step is known to be crucial in many recent data mining competitions [23, 6, 15]. Then, in the final stage, we pick predictions from the second stage, along with another re-blending loop, based on the leaderboard results to generate the final solution. The final solution thus aggregates all our models and utilizes the information from the leaderboard. Figure 1 illustrates the flow of the system. This 3-stage framework has also been exploited successfully for the Netflix Prize [23] and KDD Cup 2011 [6].

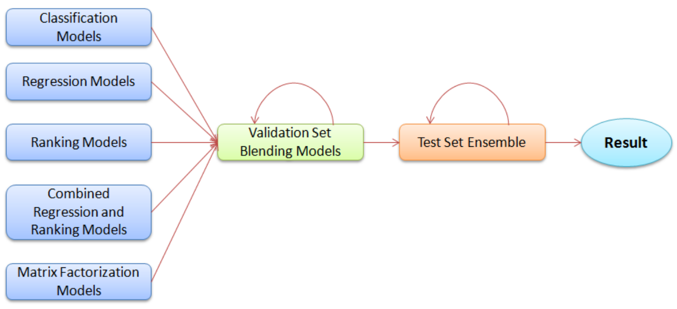
Classification Models
Regression Models
Ranking Models
Combined Regression and Ranking Models
Matrix Factorization Models
Validation Set Blending Models
Test Set Ensemble
Result

Figure 1: Overview of Proposed System

2.2 Validation Set

The validation set is important in data mining systems to select and combine models [23, 6, 15, 14]. In this competition, the training set comes from earlier time period than that of the test set. Thus, we naturally want to use the later data in the training set for validation. However, the training and test sets do not contain time information, and it is non-trivial to directly obtain such a validation set.

We take a simple alternative for generating the validation set. We randomly sample 1/11 of the training instances as the validation instances.[2] Table 1 shows the data statistics of the validation and the sub-training sets.

Table 1: Statistics of the sub-training and validation sets

	#instance	#click	#impression
sub-training	136,041,984	7,479,132	214,556,140
validation	13,597,121	738,501	21,026,739

We have looked at other different methods to generate the validation set, but none of them appears more robust then the simple instance-based sampling. For example, we have tried impression-based sampling; that is, we randomly extract 1/11 of the impressions to make the validation set.

[2]In the early stage of the competition, the official site showed that the training and test sets respectively cover the logs from the first 50 days and the last 5 days in all 55 days. This is why we extract 1/11 training instances to the validation set. The remain 10/11 is called the sub-training set.

The generated validation set contains 7,722,457 users, and 7,182,467 of them are also in the sub-training set. However, in the test set, there are only 3,263,681 users, and only 1,379,733 of them are in the training set. The statistics show a drastic difference between such a validation set (with respect to the sub-training set) and the test set (with respect to the whole training set). In particular, such a validation set does not contain enough cold-start users. Although the simple instance-based sampling also suffer from the same problem (possibly because of the lack of time information), but the ratio of cold-start users appears closer to the statistics of the training and test sets.

The validation set also plays an important role in our 3-stage framework. In the first stage, we train all our individual models twice. We first train a model on the sub-training set (sub-model) and predict on the validation set. Then, we retrain a model on the full training set (full model) with the same setting as the model trained on sub-training and make predictions on the test set. Then, in the second stage, we take the predictions from the sub-models and train with the validation set to aggregate the full models.

2.3 General Features

Before introducing each individual model, we first describe features used by our individual models here. Each of our individual models may use different set of features. The detailed list of our feature set is provided in Appendix A.

2.3.1 *Categorical Features*

The dataset contains information on UserID, AdID, user's gender, ad's position, etc. We can take those fields as categorical features directly. We use AdID, QueryID, KeywordID, UserID, and ad's position as categorical features in our models.

2.3.2 *Basic Sparse Features*

As mentioned above, we are provided with a set of categorical features such as UserID, AdID, user's gender, ad's position. We have also tried expanding those categorical features into binary features. For example, there are 22,023,547 different UserIDs in the training data, and we expand UserID as a 22,023,547-dimensional binary feature vector. We expand the following categorical features into binary features: AdID, AdvertiserID, QueryID, KeywordID, TitleID, DescriptionID, UserID, DisplayURL, user's gender, user's age, depth of session, position of ad and combination of position and depth. We also expand query's tokens, title's tokens, description's tokens and keyword's tokens into binary features. That is, if a token occurs in title, query, description or keyword, the corresponding value in the feature vector will be 1, or 0 otherwise. For some of our models, we only construct the expanded features for those IDs with clicks. For those IDs without any clicks, we do not generate any binary indicator for them. Eventually we have features with more than ten million dimensions, but only a few of them are non-zero. That is, the feature vectors are sparse.

2.3.3 *Click-Through Rate Features*

Click-through rate features are simple but very useful in our models. For each categorical feature, we compute the average click-through rate as an additional one-dimensional feature. Take AdID as an example. For each AdID, we compute the average click-through rate for all instances with the same AdID, and use this value as a single feature. This feature represents the estimated click-through rate given its category. We compute this kind of feature for AdID, AdvertiserID, depth, position and (depth−position)/depth.

However, we observe that some categories come with only a few or even no instances. Computing the click-through rate directly for those categories would result in inaccurate estimations because of the insufficient statistics. Thus, we apply smoothing methods during click-through rate estimation. We mainly use a simple additive smoothing

$$\text{pseudo-CTR} = \frac{\#\text{click} + \alpha \times \beta}{\#\text{impression} + \beta}, \tag{1}$$

and we name it pseudo click-through rate (pseudo-CTR). In our experiments, we set α as 0.05 and β as 75. We generate pseudo-CTR features for AdID, AdvertiserID, QueryID, KeywordID, TitleID, DescriptionID, UserID, DisplayURL, user's age, user's gender and (depth−position)/depth.

2.3.4 *ID Raw Value Features*

We observe that the numerical value of IDs contain some information. For instance, we observe that #impression decreases when the value of KeywordID increases. We suspect that IDs values may contain time information if they were sequentially assigned. To exploit this information, we use the raw value of IDs as features directly. We have this kind of feature for TitleID, QueryID, KeywordID, DescriptionID and UserID.

To capture the trends in the raw ID values more generally, we quantize IDs by their value into 10000 categories. We generate this kind of grouped-ID value feature for AdID, AdvertiserID, QueryID, KeywordID, TitleID, DescriptionID, and UserID.

2.3.5 *Other Numerical Features*

Other numerical features that we use include numerical value of depth, numerical value of position and the relative position, which is (depth−position)/depth. Another kind of numerical feature we take is called the length feature. In particular, we calculate the number of tokens for each QueryID, KeywordID, TitleID and DescriptionID, and use a numerical value to represent it. A weighted version is also applied, where each token is weighted by their idf value instead of 1 above. We also use #impression of AdID, AdvertiserID, ad's position, session's depth, relative position as features in some of our models.

2.3.6 *Token's Similarity Features*

The dataset provides the tokens of each QueryID, KeywordID, TitleID and DescriptionID. To utilize these information, we try to compare the similarity between query, keyword, title and description, and use these combinations of similarities as a 6-dimensional feature for our models. We have two methods to compute the similarity. The first utilizes the cosine similarity between tf-idf vector of tokens directly as in [20]. The second adopts topic model to extract latent features. We use LDA model implemented by [18] to do this. In our experiments, we extract 6 and 20 topics from LDA with $\alpha = 0.5$ and other default parameters. Then, we compute topic's cosine similarity between each type of IDs as one feature. So eventually, we have 3 similarity feature sets, each of them are 6-dimensional.

3 Individual Models

In this section we introduce several different approaches for click-through rate prediction, which can be treated as solutions in the domain of classification, regression and ranking. Table 2 provides a comparison among each kind of individual models. The detailed information about each individual model's setting, input features and performance on the validation set and test set is listed in Appendix B.

Table 2: The best performance (on the public test set) for each kind of individual models

Model	Validation AUC	Public Test AUC
Naïve Bayes	0.8108	0.7760
Logistic Regression	0.8152	0.7888
Ridge Regression	0.7539	0.7351
SVR	0.8135	0.7705
RLR	0.7442	0.7220
RankNet	0.7941	0.7577
CRR	0.8174	0.7818
Regression Based MF	0.8077	0.7775
Ranking Based MF	0.8246	0.7968

3.1 Classification Models

Here, we try to exploit binary classification models to solve the problem. For each training instance, we first split it into (#click) positive samples and (#impression-#click) negative samples. Then we train a classifier to discriminate positive samples from negative ones. We explore two models here, naïve Bayes and logistic regression.

3.1.1 *Naïve Bayes*

Naïve Bayes is a simple probabilistic classification model which is based on Bayes' Theorem. It exploits a strong independence assumption to efficiently handle a large dataset. We adopt the multinomial model for each feature in naïve Bayes. After getting a probabilistic classifier, we directly use values of the estimated conditional probability as the ranking criteria. We mainly test on the raw categorical features and use a simple leave-one-feature-out method repeatedly to select relevant features. AdID, ad's position, UserID and QueryID are four useful categorical features we identified. Besides, a query and title's tf-idf cosine similarity feature with discretized to 10 equal intervals is also found to be useful.

To avoid the insufficient statistics in estimating the conditional probability per feature, two kinds of smoothing techniques as in [25] are applied:

1. Additive smoothing: Add-one smoothing and Laplace smoothing improve AUC in the validation set, but not in the test set.
2. Back-off method: Good-Turing estimation and Katz smoothing, provide the best AUC in both validation set and test set.

Table 3 compares between different smoothing techniques for the naïve Bayes model.

Furthermore, we design a simple leave-out heuristic to avoid using features with zero estimated conditional probability. The main idea is that if a categorical feature leads to zero estimated probability during the calculation in an instance, then we leave this categorical feature out for calculating the overall conditional probability of the instance. The heuristic decrease the AUC gap between the validation and the test sets, and reaches competitive AUC on the test set. The best naïve Bayes model includes more features than models in Table 3 with feature selection and Good-Turing smoothing technique. The validation AUC of this model is 0.8108 and the public test AUC is 0.7760.

Table 3: A comparison between different smoothing techniques for naïve Bayes

Method	Validation AUC	Public Test AUC
Additive smoothing	0.7905	0.7612
Good-Turing	0.8060	0.7736

3.1.2 *Logistic Regression*

Logistic regression is widely used for binary classification. Given input data $\boldsymbol{x}$ and weights $\boldsymbol{w}$, it models the classification problem by the following probability distribution

$$P(y = \pm 1 | \boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + \exp(-y(\boldsymbol{w}^T \boldsymbol{x}))}, \tag{2}$$

where y is the class label. Assume the training instances are $(\boldsymbol{x}_i, y_i), y_i \in \{1, -1\}, i = 1, \ldots, N$, we consider a maximum likelihood problem with (2). To avoid overfitting, we add a regularization term $\boldsymbol{w}^T \boldsymbol{w}/2$. That is, we solve the following optimization problem for logistic regression.

$$\min_{\boldsymbol{w}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + C \sum_{i=1}^{N} \log(1 + e^{-y_i(\boldsymbol{w}^T \boldsymbol{x}_i)}), \tag{3}$$

where $C > 0$ is the regularization parameter.

The clicking probability in the logistic regression model is $1/(1 + \exp(-(\boldsymbol{w}^T \boldsymbol{x})))$, which is taken as the ranking criteria. To solve the optimization problem (3), we adopt a large-scale linear solver LIBLINEAR [9], which can learn from large data efficiently. We take the default parameters of LIBLINEAR.

In our experiments, we randomly sample 10% of the official training data with replacements to train the logistic regression model. We tried larger sample sizes but the performance did not improve. We thus stayed with the smaller sample size (while paying the price of some instability). For the logistic regression model, we have tried the following steps to enhance performance and diversity.

- Feature selection:
  We modify the sparse features of expanded id's tokens in Section 2.3.2. We only use those indicators for frequent query tokens. That is, we set a parameter $t > 0$ and only use query tokens that appear more than t times. We set $t = 20$ in our models.
- Different sampling strategy:
  Originally we sample the instances directly, but it is not reasonable because each instance may contains multiple impressions. Therefore, we try impression-based sampling as in Section 2.2 to sample on impressions rather than instances.
- Separate users to UserID 0 and others:
  Users with UserID=0 represent unknown users, and those users may behave differently to others. So we try to separate users into the group with UserID=0 and others. Then, we train two separate models for these two groups and combine the results. In this model, we use different sampling ratio on the two group. Small sampling ratio on the users with UserID=0 is enough. Larger sampling ratio on the other group can enhance the performance.
- Train on instances with positive clicks only:
  We try to train a model on the training instances with positive clicks only and find that the performance of this model is rather comparable with other models.

Including all the steps above leads to the best result on the public test set for the model, achieving 0.8152 on the validation set and 0.7888 on the public test set.

3.2 Regression Models

For the regression models, we use CTR $= \frac{\#\text{click}}{\#\text{impression}}$ as the value to be predicted. The idea behind is that the regression models will give higher CTR to those instances that are more likely to be clicked. Although regression models do not aim to optimize AUC, they turn out to work well for this criteria. We use two different regressions models, ridge regression and support vector regression.

3.2.1 *Ridge Regression*

Ridge Regression minimizes the following objective function:

$$\min_{\boldsymbol{w}} \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{w} + \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2,$$

where N is the number of instances; $\boldsymbol{x}_i, y_i$ are the feature vector and the label. This well-studied algorithm has a close form solution, that can be obtained efficiently.

We include one ridge regression model in our final solution, which is trained on the whole training data. This model reaches 0.7539 on the validation and 0.7351 on the public test set.

3.2.2 *Support vector regression*

Support vector regression (SVR) is proposed by Vapnik [24], which is an extension from support vector classification [2]. SVR solves the following optimization problem.

$$\min_{\boldsymbol{w}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + C \sum_{i=1}^{N} \xi_\epsilon(\boldsymbol{w}; \boldsymbol{x}_i, y_i), \tag{4}$$

where $C > 0$ is the regularization parameter, and

$$\xi_\epsilon(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = \max(|\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i| - \epsilon, 0)^2 \tag{5}$$

is the squared ϵ-insensitive loss function associated with $(\boldsymbol{x}_i, y_i)$. Function $\phi(\cdot)$ maps $\boldsymbol{x}$ to higher dimension space, and we take the degree-2 polynomial mapping here.

To solve the degree-2 polynomial SVR problem efficiently, we adopt an extension of LIBLINEAR that directly expands features without using a kernel [5]. The data of this competition is accordant to the situations for using primal solvers of LIBLINEAR illustrated in Appendix L of [9]. We also use the number of impressions as instance weights.

The degree-2 polynomial expansion gives about 0.35% improvement on AUC compared to linear SVR when using only pseudo-CTR as the features. For both cases, we set ϵ in (5) to be zero and use the default parameters of the primal L2-loss SVR solver in LIBLINEAR. This model is used in our final ensemble and blending stage as SVR-1 in Appendix B.

The best degree-2 polynomial SVR model we used is based on a similar setting of SVR-1. But we discard the pseudo-CTR of AdvertiserID and KeywordID. We then add the token's similarity features as describing in Section 2.3.6. This model is our SVR-2 predictor. It improves the result from 0.7990 to 0.8135 on the validation and from 0.7610 to 0.7705 on the test set. For both models, the total training procedure of LIBLINEAR costs about two hours for the whole training set.

3.3 Ranking Models

For the ranking models, we divide each aggregated training instances into (#click) positive samples and (#impression-#click) negative samples and try to minimize pairwise 0/1 loss, which is equivalent to maximizing AUC. We think this kind of model is more suitable for the task, because it optimizes AUC directly. We exploit two pairwise ranking models here, rank logistic regression (RLR) and RankNet.

3.3.1 *Rank Logistic Regression*

As reported in KDDCup 2009 [14], we try to minimize the pairwise logistic loss. That is, we optimize

$$\min_{\boldsymbol{w}} \frac{1}{N^+N^-} \sum_{i=1}^{N^+} \sum_{j=1}^{N^-} \log\left(1 + \exp(-\boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j))\right),$$

where N^+ is the number of positive samples, N^- is the number of negative samples, $\boldsymbol{x}_i$ has positive label and $\boldsymbol{x}_j$ has negative label.

We solve this problem by stochastic gradient descent (SGD). The result is 0.7442 on the validation set and 0.7220 on the public test set.

3.3.2 *RankNet*

RankNet [3] builds a ranking model based on neural network. Given pair of instance $(\boldsymbol{x}_i, \boldsymbol{x}_j)$, our goal is to minimize the cross entropy loss function

$$C_{ij} \equiv C(\hat{r}_{ij}) = -\overline{P}_{ij} \log P_{ij} - (1 - \overline{P}_{ij}) \log(1 - P_{ij})$$

where $\overline{P}_{ij}$ is the target value and $P_{ij} \equiv (e^{\hat{r}_{ij}})/(1 + e^{\hat{r}_{ij}})$.

Because a pair of instances always contains a positive and a negative sample, $\overline{P}_{ij}$ is always 1, so the loss function becomes $C_{ij} = -\hat{r}_{ij} + \log(1 + e^{\hat{r}_{ij}})$, where

$$\hat{r}_{ij} = \zeta(\hat{r}_i - \hat{r}_j). \quad (6)$$

In (6), $\hat{r}_i$ denotes the model prediction on instance $\boldsymbol{x}_i$, and ζ is a scaling scalar.

We use two layers neural network with an additional bias term. The output for instance $\boldsymbol{x}_i$ is

$$\hat{r}_i = \sum_{j=1}^{H} \boldsymbol{w}_j^{(2)} \tanh(\sum_k {\boldsymbol{w}_{jk}^{(1)}}^T \boldsymbol{x}_{ik}),$$

where $\boldsymbol{x}_i$ is the feature vector of the i-th instance.

For this model, we normalize the features into [0,1], and apply early stopping and weight decay regularization to avoid over-fitting. The model is trained by SGD with random pairs. The model reaches 0.7941 on the validation set and 0.7577 on the public test set.

3.4 Combining Regression and Ranking

We also explore another model that combines the ranking loss and the regression loss during learning.

3.4.1 *SGD Support Vector Machine*

According to [22], minimizing an objective function that combines ranking loss and regression loss (CRR) usually leads to strong performance on ranking metrics. We use the package suite of fast incremental algorithms for machine learning (sofia-ml) [21] that uses SGD to exploit such an idea. Let N^+ denotes the number of positive samples, N^- denotes the number of negative samples, $\boldsymbol{x}_i$ denotes positive instances and $\boldsymbol{x}_j$ denotes negative samples. We solve the combined objective function

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} + \frac{p}{N^+N^-}\sum_{i=1}^{N^+}\sum_{j=1}^{N^-} H_{ij}(\boldsymbol{w}) + \frac{(1-p)}{N}\sum_{k=1}^{N} L_k(\boldsymbol{w}) \quad (7)$$

by using *sgd-svm* solver and *combined-ranking* loop type, where $H_{ij}(\boldsymbol{w}) = max(0, 1 - \boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j))$, and $L_k(\boldsymbol{w}) = max(0, 1 - y_k(\boldsymbol{w}^T\boldsymbol{x}_k))$ and p is the probability of considering the ranking loss. We also experiment on different solver and loop type combinations and the setting above leads to the best performance.

We include four combine regression and ranking models in our final solution, all of them use the same parameters but with different features. We also use leave-one-feature-out strategy for feature selection for this model. The model achieves 0.8174 on the validation set and 0.7818 on the public test set.

3.5 Matrix Factorization Models

Matrix Factorization (MF) is a common technique for collaborative filtering (CF). This competition task can be modeled as a recommendation problem solvable by CF, because similar users may click the similar ads. Therefor, we apply MF to exploit latent information from data. We utilize this kind of model on regression problems as in Section 3.2, and on ranking problems as in Section 3.3.

3.5.1 *Regression Based Matrix Factorization*

Considering the traditional CF approach, we can treat ads as items. We first divide all our features as user's features and item's features, and then we can apply feature-based matrix factorization model as in [7] on this problem.

Using features in factors, the prediction of feature-based matrix factorization on instance $\boldsymbol{x}_i$ can be represented as

$$\begin{aligned}&\hat{r}(\boldsymbol{x}_i)\\ &= \mu + (\sum_j \boldsymbol{w}_j^{(u)}\boldsymbol{\alpha}_j + \sum_j \boldsymbol{w}_j^{(i)}\boldsymbol{\beta}_j) + (\sum_j \boldsymbol{p}_j\boldsymbol{\alpha}_j)^T(\sum_j \boldsymbol{q}_j\boldsymbol{\beta}_j),\end{aligned} \quad (8)$$

where μ is global bias, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the user features and item features. $\boldsymbol{w}$ models bias of the features, $\boldsymbol{p}$ and $\boldsymbol{q}$ represent factors of the features. We use pseudo-CTR of user, expanded UserID, age, gender and raw value as user's features. For item features, we use pseudo-CTR (ad, advertisement, query, title, keyword, description and URL), binary expanded features (depth, position, ad, advertiser and query), raw value features (title, query and title) and number of tokens in query, title, keyword and description.

We minimize the least squared error between our prediction $\hat{r}$ and target r, where r is CTR of the training instance.

$$\min_{\theta} \sum_{i=1}^{N} (r(\boldsymbol{x}_i) - \hat{r}(\boldsymbol{x}_i))^2 + \frac{\lambda}{2}\|\theta\|^2 \text{ subject to (8)}, \quad (9)$$

where θ represents the parameters of the model and λ is the regularization parameter.

Training procedure is done by modifying the package *SVD-Feature* [7] with SGD. The best performance of this model is 0.8077 on the validation set and 0.7775 on the public test set.

3.5.2 Ranking Based Matrix Factorization

The basic idea of this model is from the Factorization Machine [19]. We modify the objective function to optimize the pair-wise ranking loss. We choose the model because it provides a powerful way to make use of explicit data to overcome the sparsity of the implicit data. For this model, all training instances are expanded into positive and negative samples as in Section 3.3.

The output of this model is similar with regression based MF in Section 3.5.1 except that not all features possess bias term or interact with other features, and we didn't group features into user's feature and item's features in this model. Thus features can freely interact with any other features. The prediction of each instance $\hat{r}(\boldsymbol{x}_i)$ can be obtained by

$$\hat{r}(\boldsymbol{x}_i) = \sum_{j=1}^{A} \boldsymbol{w}_j \boldsymbol{\alpha}_j + \sum_{j=1}^{B} \sum_{k=j+1}^{B} \langle \boldsymbol{p}_j, \boldsymbol{p}_k \rangle \boldsymbol{\beta}_j \boldsymbol{\beta}_k, \tag{10}$$

the first term represents features' bias, and the second term is the sum of the inner product between every pair of parameter vector from different features.

In our model, any features can belong to $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ or both of them. We use all-pairs AUC with weighted instance as our optimization criterion to fit the evaluation metric of track 2. The optimization criterion is:

$$\min_{\theta} \sum_{i=1}^{N^+} \sum_{j=1}^{N^-} L(\hat{r}(\boldsymbol{x}_i) - \hat{r}(\boldsymbol{x}_j)) + \frac{\lambda}{2} \|\theta\|^2 \text{ subject to (10)}, \tag{11}$$

where θ represents the parameters of the model. We take the logistic loss $L(\delta) = \ln(1/(1 + \exp(-\delta)))$ here, and λ is regularization parameter.

We use SGD to train the model. In each step, we random sample a positive and a negative sample as a pair to update the parameters. And in each iteration, we randomly sample one million pairs for SGD to optimize. In this model, we only use tokens of query, title, description and keyword as pair-wise interaction features, all other features are only trained as bias. This is the best individual model we have, and can achieve 0.8246 on the validation set and 0.7968 on the public test set.

4 Validation Set Blending

The goal of validation set blending is to combine each single predictors, to boost the performance and to enhance the diversity for next stage ensemble. Validation set blending methods work similar to individual models, except that the most of features come from the individual models' predictions. We discover that adding some extra features, which are not from individual model's prediction, can enhance the performance a little bit. In addition, we also found that re-blending with predictions from blending models can exploit the additional enhancement, and we named it two-level blending models. Feature selection and score transformation are also very useful techniques for blending models. The performance of each blending models are shown is Table 4, including results for 1-level and 2-level blending. The detailed features and parameters of these models are listed in the Appendix B.

4.1 Feature Selection

As discuss in Section 2.2, we could not sample a good validation set without time information, and hence we do observe that for some models, the performance is very different between our validation set and the public test set. To avoid wrongly including overfitted models in blending, some of our blending models use a simple criteria to select input features. If the difference between AUC on validation set and AUC on test set is larger than a threshold, we discard that model before blending. We take 0.031$\tilde{0}$.035 as the threshold.

Table 4: The best performance of each validation set blending models

Model	Public Test AUC
1-level SVR	0.8038
1-level LambdaMart	0.8051
1-level RankNet	0.8058
2-level SVR	0.8031
2-level CRR	0.8050
2-level RF-LambdaMart	0.8059
2-level RankNet	0.8062

4.2 Score Transformation

The distribution of each individual model's outputs can be very different. As in [15], applying score transformation before blending may boost the performance for some of the models. We have tried some transformation methods:

- raw score: no score transformation
- normalized score: features are scaled into a specific interval by simple normalization
- ranked score: sort all instances in validation and test set according to the model's output score, and then scale the scores into [0,1] linearly

For the two-level blending models, we will use score transformation once again before the second level of blending.

4.3 Blending Models

4.3.1 RankNet

Inspired by [13] we also use RankNet to blend models. The RankNet blending model is the same as model in Section 3.3.2, except that the input features are predictions from each single models. Feature selection and ranked score transformation are applied in this model. We also add additional features and other blending models prediction into our RankNet blending model. All our RankNet models are trained by 20 nodes single layer neural network. Our best validation set blending model is generated from this method, which achieves 0.8062 AUC on the public test set. More detailed information about model's setting is listed in Appendix B.

4.3.2 Support Vector Regression

We also use degree-2 polynomial SVR models which is introduced in Section 3.2.2. Our validation set is much smaller than the training set, and the training procedure here using LIBLINEAR takes only 10 minutes. The short training time enables us to do parameter selection on C and γ. Thus , we use grid search to find parameters as suggested in [12]. We have three one-level blending models, and one two-level blending models from SVR. All models use #impression as their instance weights and the parameter ϵ is always set to be zero. The two-level SVR model overfits our validation set severely, which can be easily seen from the difference of it's improvement from VB-1 to VB-9 on validation set and the public test set. It reaches 0.8038 AUC on the public test set.

4.3.3 *Combined Regression and Ranking*

Because of the promising result from combined ranking and regression (CRR) individual models, we also adopt CRR to validation set blending. Our strategy is two-level validation set blending, that is, we take the prediction on validation set from other blending model as feature. We include one CRR blending model in our final solution. The model consist of blending feature and additional features. We use the same parameters as CRR individual models, except that we strengthen the model regularization and decrease SGD iteration due to the smaller size of validation data. As shown in Appendix B, the CRR blending model gives 0.0018 improvement on the public test set.

4.3.4 *LambdaMart*

For another model of validation blending, we apply LambdaMART, as proposed in [4]. We choose LambdaMART due to its recent success on Yahoo! Learning to Rank Challenge. The unique properties of the model enjoy a few advantages. First, LambdaMART is a form of gradient boosted regression tree, which leads to promising performance [26]. Second, the model respects the non-smooth ranking objective without directly calculating the gradient. Last, we can easily tune parameters to prevent overfitting validation data via bagging.

To implement LambdaMART, we use the package provided in [11], with slight modifications to meet the ranking objective. We also incorporate bagging for tree training for speedup. And bagging also helps us to avoid overfitting.

Inspired by [16], we implement another variant that uses initialized residual obtained from Random Forest as the target. We hope to benefit from the generalization ability of Random Forest (RF), while running at a much faster speed for effective model checking. We first train a Regression Random Forest ($Regression_{RF}$) and then initialize the target of LambdaMART as:

$$target_i = r_i - Regression_{RF}(\boldsymbol{x}_i), for\ i = 1 \cdots N. \quad (12)$$

After training, we then sum the predictions from both models as

$$\hat{r}(\boldsymbol{x}_i) = \hat{r}'(\boldsymbol{x}_i) + Regression_{RF}(\boldsymbol{x}_i) \quad (13)$$

where $\hat{r}'(\boldsymbol{x}_i)$ represents the predict value of LambdaMART on instance $\boldsymbol{x}_i$. Note that the λ updates in LambdaMART happens on pair misplacements, so with a fair residual as target, we can save significant amount of updates, leading to much faster convergence.

The RF-initialized LambdaMART improves the performance slightly and converges much faster, dropping to around 300 iterations from 1,500 iterations to terminate. This model can achieve 0.8059 on the public test set.

5 Test Set Ensemble

In order to improve leaderboard (test set) performance, we further combine individual and blending models using ensemble methods. We first apply the test set blending technique to linearly combine several selected models. The technique uses leaderboard results to estimate the linear blending weight of each model, which has been used for optimizing RMSE in previous competitions [23, 6]. We adapt the same technique to optimize AUC. Since we can select five entries as final entries, we also include several uniform average results as a backing for the case that test set linear blending overfits public leaderboard result.

5.1 Test set Linear Blending

The test set linear blending technique proposed by [23] minimizes the square error using ridge regression. To adapt the technique for AUC, we transform AUC to a square-error-like format.

It is known that AUC is the same as pairwise ranking accuracy. Here we assume that each instance has the same number of impressions because the true impression count of testing instances are hidden. Let n be the number of instances in the test set, $\mathbf{y} \in \mathbb{R}^n$ be the true CTR of instances, and $\mathbf{x} \in \mathbb{R}^n$ be a predicted ranking, where $x_i \in [1, n]$ is the rank of the i-th instance, and $x_i \neq x_j$ for any i, j. We define pairwise ranking prediction $\hat{x}_{i,j} = sign(x_i - x_j)$ for each pair (i, j) and use $\hat{\mathbf{x}} \in \{1, -1\}^{\binom{n}{2}}$ to denote the pairwise ranking prediction for all pairs. Similarly, $\hat{\mathbf{y}} \in \{1, 0, -1\}^{\binom{n}{2}}$ denotes the optimal pairwise ranking. Then, we have

$$\text{AUC} = c_1 \hat{\mathbf{x}}^T \hat{\mathbf{y}} + c_2 = -c_3 \|\hat{\mathbf{x}} - \hat{\mathbf{y}}\|^2 + c_4 \quad (14)$$

for constants c_1, c_2, c_3, and c_4. Equation (14) suggests that minimizing the square error between $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ is equivalent to maximizing AUC.

Let $\hat{\mathbf{x}}^{(m)}$ be the predicted pairwise rankings from the m-th model, and $\hat{X} = [\hat{\mathbf{x}}^{(1)}, \cdots, \hat{\mathbf{x}}^{(M)}]$. The optimal weights $\mathbf{w}$ for linear blending these M models obtained from ridge regression is $(\hat{X}^T \hat{X} + \lambda I)^{-1} \hat{X}^T \hat{\mathbf{y}}$, where λ is the regularization parameter. Although the true $\hat{\mathbf{y}}$ is unknown, we may use the leaderboard result (AUC) to estimate $\hat{X}^T \hat{\mathbf{y}}$. In addition, each element in $\hat{X}^T \hat{X}$ can be calculated in the same way as calculating AUC.

The ideal ensemble prediction is $\hat{\mathbf{x}}^* = \hat{X}\mathbf{w}$. But this is a real-valued pairwise ranking prediction, and we have to convert it to a ranking prediction $\mathbf{x}^*$. Unfortunately, the conversion is not easy because the pairwise ranking prediction may not induce a linear ordering, i.e. the pairwise order is not transitive. The problem of finding a linear ordering with minimum pairwise mis-rankings with respect to a pairwise ranking is NP-complete [8]. Thus we adopt the approximation approach proposed by [1] which takes the pairwise ranking prediction $\hat{\mathbf{x}}^*$ as the comparison function of the popular QuickSort algorithm. The result of the QuickSort algorithm is a linear order of instances, and the ranking prediction $\mathbf{x}^*$ is constructed according to this linear order.

Note that the approximation introduces some randomness into the result, so we did multiple rounds of QuickSort and average the ranking predictions of each round for stability. Moreover, we found out that adding more models into the ensemble sometimes results in worse performance. The reason may be the hidden impression counts or the difficulty of approximation. In practice, we apply this blending method on a manually selected set of high-performance and diverse models.

5.2 Uniform Average

In addition to test set linear blending, we also tried the simplest blending method: Uniformly average the ranking predictions. Given that we have five final entries, at first we thought this is an insurance against the failings of test set linear blending. But it turns out to be better than test set linear blending on the manually selected set of models. Furthermore, we selected the top-five models according to the public leaderboard result and took uniform average of them as our final submission. In the end this is the highest performance model of ours.

Table 5: Performance of the best single model, validation set blending model and test set ensemble model

Model	Public Leaderboard AUC
Best individual model (Rank_MF-5)	0.7968
Best Validation Set Blending (VB-12)	0.8062
Best Test Set Ensemble (TE-3)	0.8069

6 Conclusion

In this paper, we introduce our solution for track 2 of KDD Cup 2012. We model users' behavior on click-through rate by using linear and non-linear models and aggregate all of them. For each single model, we provide various kinds of features combination to capture users' behavior from different perspective with competitive performance. Several ensemble methods are designed to combine models and boost the performance. We believe the success of our solution is based on capturing various information in the data and utilizing those information effectively. Table 5 provides a comparison among the best single model, the best validation set blending model and the best test set ensemble model.

7 Acknowledgment

We thank the organizers for holding this interesting competition. We also thank National Taiwan University for providing a stimulating research environment. This work was supported by National Science Council, National Taiwan University and Intel Corporation under Grants NSC 100-2911-I-002-001, 101R7501, NSC 100-2628-E-002-010, NTU 10R70837, 10R70838 and 10R70839.

8 References

[1] N. Ailon and M. Mohri. An efficient reduction of ranking to classification. In *COLT*, 2008.

[2] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, 1992.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML*, 2005.

[4] C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An overview, 2010.

[5] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear SVM. *JMLR*, 2010.

[6] P.-L. Chen, C.-T. Tsai, Y.-N. Chen, K.-C. Chou, C.-L. Li, C.-H. Tsai, K.-W. Wu, Y.-C. Chou, C.-Y. Li, W.-S. Lin, S.-H. Yu, R.-B. Chiu, C.-Y. Lin, C.-C. Wang, P.-W. Wang, W.-L. Su, C.-H. Wu, T.-T. Kuo, T. G. McKenzie, Y.-H. Chang, C.-S. Ferng, C.-M. Ni, H.-T. Lin, C.-J. Lin, and S.-D. Lin. A linear ensemble of individual and blended models for music rating prediction. In *JMLR W&CP*, volume 18, 2011.

[7] T. Chen, Z. Zheng, Q. Lu, W. Zhang, and Y. Yu. Feature-based matrix factorization. Tech. report, Apex Data & Knowledge Management Lab, Shanghai Jiao Tong University, 2011.

[8] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *JAIR*, 10, 1999.

[9] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9, 2008.

[10] T. Fawcett. Roc graphs: Notes and practical considerations for researchers. Tech. report, 2004.

[11] Y. Ganjisaffar, R. Caruana, and C. Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *SIGIR*, 2011.

[12] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Tech. report, National Taiwan University, 2003.

[13] M. Jahrer and A. Töscher. Collaborative filtering ensemble for ranking. In *JMLR W&CP*, volume 18, 2011.

[14] H.-Y. Lo, K.-W. Chang, S.-T. Chen, T.-H. Chiang, C.-S. Ferng, C.-J. Hsieh, Y.-K. Ko, T.-T. Kuo, H.-C. Lai, K.-Y. Lin, C.-H. Wang, H.-F. Yu, C.-J. Lin, H.-T. Lin, and S.-D. Lin. An ensemble of three classifiers for kdd cup 2009: Expanded linear model, heterogeneous boosting, and selective naïve bayes. In *JMLR W&CP*, volume 7, 2009.

[15] T. G. McKenzie, C.-S. Ferng, Y.-N. Chen, C.-L. Li, C.-H. Tsai, K.-W. Wu, Y.-H. Chang, C.-Y. Li, W.-S. Lin, S.-H. Yu, C.-Y. Lin, P.-W. Wang, C.-M. Ni, W.-L. Su, T.-T. Kuo, C.-T. Tsai, P.-L. Chen, R.-B. Chiu, K.-C. Chou, Y.-C. Chou, C.-C. Wang, C.-H. Wu, H.-T. Lin, C.-J. Lin, and S.-D. Lin. Novel models and ensemble techniques to discriminate favorite items from unrated ones for personalized music recommendation. In *JMLR W&CP*, 2012.

[16] A. Mohan, Z. Chen, and K. Q. Weinberger. Web-search ranking with initialized gradient boosted regression trees. In *JMLR W&CP*, volume 14, 2011.

[17] Y. Niu, Y. Wang, G. Sun, A. Yue, B. Dalessandro, C. Perlich, and B. Hamner. The Tencent dataset and KDD-Cup'12. 2012.

[18] X.-H. Phan and C.-T. Nguyen. GibbsLDA++: A C/C++ implementation of latent Dirichlet allocation (LDA), 2007.

[19] S. Rendle. Factorization machines. In *ICDM*, 2010.

[20] G. Salton, A. Wong, and C. S. Yang. Readings in information retrieval. chapter A vector space model for automatic indexing. Morgan Kaufmann, 1997.

[21] D. Sculley. Large scale learning to rank. In *NIPS Workshop on Advances in Ranking*, 2009.

[22] D. Sculley. Combined regression and ranking. In *SIGKDD*, pages 979–988, 2010.

[23] A. Töscher and M. Jahrer. The BigChaos solution to the Netflix grand prize. Tech. report, 2009.

[24] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[25] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *SIGIR*, 2001.

[26] Z. Zheng, H. Zha, K. Chen, and G. Sun. A regression framework for learning ranking functions using relative relevance judgments. In *SIGIR*, 2007.

APPENDIX

A Feature List

A.1 Categorical Features

ID	Feature Description
cat_User	categorical feature for UserID
cat_Query	categorical feature for QueryID
cat_Ad	categorical feature for AdID
cat_Keyword	categorical feature for KeywordID
cat_Position	categorical feature for ad's position

Table 6: Categorical Features

A.2 Basic Sparse Features

ID	Feature Description
binary_Ad	binary expanded AdID
binary_Advertiser	binary expanded AdvertiserID
binary_Query	binary expanded QueryID
binary_Keyword	binary expanded KeywordID
binary_Title	binary expanded TitleID
binary_Description	binary expanded DescriptionID
binary_User	binary expanded UserID
binary_Url	binary expanded Display Url
binary_Gender	binary expanded user's gender
binary_Age	binary expanded user's age
binary_Position	binary expanded ad's position
binary_Depth	binary expanded session's depth
binary_PositionDepth	binary expanded (position,depth)
binary_QueryTokens	binary expanded query's tokens
binary_TitleTokens	binary expanded title's tokens
binary_DescriptionTokens	binary expanded description's tokens
binary_KeywordTokens	binary expanded keyword's tokens

Table 7: Basic Sparse Features

A.3 Click-Through Rate Features

ID	Feature Description
aCTR_Ad	average click-through rate of AdID
aCTR_Advertiser	average click-through rate of AdvertiserID
aCTR_Depth	average click-through rate of session's depth
aCTR_Position	average click-through rate of ad's position
aCTR_RPosition	average click-through rate of relative position (depth−position)/depth
pCTR_Ad	pseudo click-through rate of AdID
pCTR_Advertiser	pseudo click-through rate of AdvertiserID
pCTR_Query	pseudo click-through rate of QueryID
pCTR_Title	pseudo click-through rate of TitleID
pCTR_Description	pseudo click-through rate of DescriptionID
pCTR_User	pseudo click-through rate of UserID
pCTR_Keyword	pseudo click-through rate of KeywordID
pCTR_Url	pseudo click-through rate of Display Url
pCTR_RPosition	pseudo click-through rate of relative position (depth−position)/depth
pCTR_Gender	pseudo click-through rate of user's gender
pCTR_Age	pseudo click-through rate of user's age

Table 8: Click-Through Rate Features

A.4 ID Raw Value Features

ID	Feature Description
value_Title	value of TitleID
value_Query	value of QueryID
value_Keyword	value of KeywordID
value_Description	value of DescriptionID
value_User	value of UserID

Table 9: ID Raw Value Features

A.5 Grouped ID Value Features

ID	Feature Description
group_Ad	quantized AdID
group_Advertiser	quantized AdvertiserID
group_Title	quantized TitleID
group_Query	quantized QueryID
group_Keyword	quantized KeywordID
group_Description	quantized DescriptionID
group_User	quantized UserID

Table 10: Grouping ID Value Features

A.6 Other Numerical Features

ID	Feature Description
num_Position	Numerical value of ad's position
num_Depth	Numerical value of ad's position
num_RPosition	Numerical value of relative position (depth−position)/depth
num_Query	Number of tokens in query
num_Title	Number of tokens in title
num_Keyword	Number of tokens in keyword
num_Description	Number of tokens in description
num_idf_Query	Sum of tokens' idf values in query
num_idf_Title	Sum of tokens' idf values in title
num_idf_Keyword	Sum of tokens' idf value in keyword
num_idf_Description	Sum of tokens' idf value in description
num_Imp_Ad	Number of impressions for AdID
num_Imp_Advertiser	Number of impressions for Advertiser
num_Imp_Depth	Number of impressions for session's depth
num_Imp_Position	Number of impressions for ad's position
num_Imp_Rposition	Number of impressions for relative position (depth−position)/depth

Table 11: Other Numerical Features

A.7 Token's Similarity Features

ID	Feature Description
similarity_tfidf	similarity computed by tf-idf vector between query, title, description and keyword (6 features)
similarity_topic_6	Topic similarity between query, title, description and keyword (6 features, form 6 topics LDA)
similarity_topic_20	Topic similarity between query, title, description and keyword (6 features, from 20 topics LDA)

Table 12: Token's Similarity Features

B Predictor List

In this section we list all predictors which are included in our final submission. VAUC indicates the model's performance on our own validation set, and TAUC means the performance on public leaderboard.

B.1 Naïve Bayes Predictors

- NB-1: VAUC=0.7905, TAUC=0.7612
  additive sommthing and leave-out heuristic
  features: cat_User, cat_Query, cat_Ad, cat_Keyword, cat_Position
- NB-2: VAUC=0.8060, TAUC=0.7736
  Good-turing
  features: cat_User, cat_Query, cat_Ad, cat_Keyword, cat_Position
- NB-3: VAUC=0.8108, TAUC=0.7760
  Good-turing, quantized numerical features, use query and title's consine similarity_tfidf only
  features: cat_User, cat_Query, cat_Ad, cat_Position, similarity_tfidf

B.2 Logistic Regression Predictors

- LR-1: VAUC=0.8144, TAUC=0.7853
  random sample 10% of instances, expand binary features for ID with clicks only, using default parameters in LIBLINEAR
  features: similarity_tfidf, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_User, binary_User, binary_Ad, binary_Query, value_User, value_Query, num_Query, num_Title, num_Description, num_Keyword, binary_Gender, binary_Age, binary_PositionDepth, binary_QueryTokens
- LR-2: VAUC=0.8144, TAUC=0.7890
  random sample 10% of instances, expand binary features for ID with clicks only, Feature selection on biny_QueryTokens, using default parameters in LIBLINEAR
  features: similarity_tfidf, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_User, binary_User, binary_Ad, binary_Query, value_User, value_Query, num_Query, num_Title, num_Description, num_Keyword, binary_Gender, binary_Age, binary_PositionDepth, binary_QueryTokens
- LR-3: VAUC=0.8129, TAUC=0.7860
  users separate, divide instances into positive and negative samples before sampling, random sample 10% user0's instances and 10% other user's instance, expand binary features for ID with clicks only, Feature selection on biny_QueryTokens, using default parameters in LIBLINEAR
  features: similarity_tfidf, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_User, binary_User, binary_Ad, binary_Query, value_User, value_Query, num_Query, num_Title, num_Description, num_Keyword, binary_Gender, binary_Age, binary_PositionDepth, binary_QueryTokens
- LR-4: VAUC=0.8181, TAUC=0.7876
  divide instances into positive and negative samples before sampling, random sample 10% , expand binary features for ID with clicks only, Feature selection on biny_QueryTokens, using default parameters in LIBLINEAR
  features: similarity_tfidf, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_User, binary_User, binary_Ad, binary_Query, value_User, value_Query, num_Query, num_Title, num_Description, num_Keyword, binary_Gender, binary_Age, binary_PositionDepth, binary_QueryTokens
- LR-5: VAUC=0.8152, TAUC=0.7888
  user separate, divide instances into positive and negative samples before sampling ,random sample 10% user0's instances and 30% other user's instance, expand binary features for ID with clicks only, Feature selection on biny_QueryTokens, using default parameters in LIBLINEAR
  features: similarity_tfidf, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_User, binary_User, binary_Ad, binary_Query, value_User, value_Query, num_Query, num_Title, num_Description, num_Keyword, binary_Gender, binary_Age, binary_PositionDepth, binary_QueryTokens
- LR-6: VAUC=0.8011, TAUC=0.7612
  train on instances with click>0 only, using default parameters in LIBLINEAR
  features: num_RPosition, num_Query, num_Title, num_Description, num_Keyword, binary_User, binary_Advertiser, binary_Ad, binary_Query, binary_QueryTokens, binary_TitleTokens, binary_DescriptionTokens, binary_KeywordTokens

B.3 Ridge Regression Predictors

- RR-1: VAUC=0.7539, TAUC=0.7351
  degree-2 polynomial expansion on real value features, $\lambda = 1$
  features: binary_Age, binary_Gender, num_Imp_Ad, num_Imp_Advertiser, aCTR_Ad, aCTR_Advertiser, num_Depth, num_Position, num_RPosition, num_Query, num_Title, num_Description, num_Keyword, num_idf_Query, num_idf_Title, num_idf_Description, num_idf_Keyword

B.4 Support Vector Regression Predictors

- SVR-1: VAUC=0.7990, TAUC=0.7610
  degree-2 polynomial expansion
  features: pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_Title, pCTR_Description, pCTR_User, pCTR_Keyworyd, pCTR_Url, pCTR_Gender, pCTR_Age, pCTR_RPosition

- SVR-2: VAUC=0.8135, TAUC=0.7705
  degree-2 polynomial expansion
  features: pCTR_Ad, pCTR_Query, pCTR_Title, pCTR_Description, pCTR_User, pCTR_Url, pCTR_Age, pCTR_RPosition, similarity_topic_6, similarity_topic_20

B.5 Rank Logistic Regression Predictors

- RLR-1: VAUC=0.7442, TAUC=0.7220
  $\eta = 0.001$, $\zeta = 1$
  features: binary_Age, binary_Gender, num_Imp_Ad, num_Imp_Advertiser, aCTR_Ad, aCTR_Advertiser, num_Depth, num_Position, num_RPosition, num_Query, num_Title, num_Description, num_Keyword, num_idf_Query, num_idf_Title, num_idf_Description, num_idf_Keyword

B.6 RankNet Predictors

- RN-1: VAUC=0.7941, TAUC=0.7577
  10 hidden nodes, $\eta = 0.01$, $\lambda = 0$, $\zeta = 1$
  features: num_RPosition, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_Title, pCTR_Description, pCTR_User, pCTR_Keyword, pCTR_Url, num_Query, num_Title, num_Description, num_Keyword

B.7 Combine Regression and Ranking Predictors

- SGD_SVM-1: VAUC=0.8160, TAUC=0.7768
  $\lambda = 0.0001, p = 0.5, iterations = 50000000$
  also use square terms of raw id features
  features: binary_Ad, binary_Advertiser, binary_Query, binary_Title, binary_Description, binary_User, binary_Keyword, binary_Age, binary_Gender, binary_PositionDepth, num_Depth, num_Position, num_RPosition, num_Query, num_Title, num_Description, num_Keyword, num_idf_Query, num_idf_Title, num_idf_Description, num_idf_Keyword, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_Title, pCTR_Description, pCTR_User, pCTR_Keyword, pCTR_Url, value_Query, value_Title, value_Title, value_Description, value_user

- SGD_SVM-2: VAUC=0.8163, TAUC=0.7785
  $\lambda = 0.0001, p = 0.5, iterations = 50000000$
  features: SGD_SVM-1's features, square term of pCTR features

- SGD_SVM-3: VAUC=0.8175, TAUC=0.7817
  $\lambda = 0.0001, p = 0.5, iterations = 50000000$
  also use square terms of raw id features
  features: binary_Ad, binary_Advertiser, binary_Query, binary_Title, binary_Description, binary_User, binary_Keyword, binary_Age, binary_Gender, binary_PositionDepth, num_Depth, num_Position, num_RPosition, num_Query, num_Title, num_Description, num_Keyword, num_idf_Query, num_idf_Title, num_idf_Description, num_idf_Keyword, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_Title, pCTR_User, value_Query, value_Title, value_Title, value_Description, value_user, binary_Url, group_Ad ,group_Advertiser, group_Title , group_Query, group_Keyword, group_Description, group_User, square term of pCTR features

- SGD_SVM-4: VAUC=0.8174, TAUC=0.7818
  $\lambda = 0.0001, p = 0.5, iterations = 50000000$
  features: SGD_SVM-3's features, similarity_topic_6

B.8 Regression Based Matrix Factorization Predictors

- Reg_MF-1: VAUC=0.8013, TAUC=0.7711
  $\eta = 0.005$, $size\_of\_latent\_factor = 64$, $\lambda = 0.04$
  features: num_RPosition, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_Title, pCTR_Description, pCTR_User, pCTR_Keyword, pCTR_Url, num_Query, num_Title, num_Description, num_Keyword, binary_User, binary_Advertiser, binary_Ad, binary_Query

- Reg_MF-2: VAUC=0.8077, TAUC=0.7775
  $\eta = 0.005$, $size\_of\_latent\_factor = 64$, $\lambda = 0.04$
  features: num_RPosition, pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_Title, pCTR_Description, pCTR_User, pCTR_Keyword, pCTR_Url, num_Query, num_Title, num_Description, num_Keyword, binary_User, binary_Advertiser, binary_Ad, binary_Query, binary_Position, binary_Depth

B.9 Ranking Based Matrix Factorization Predictors

- Rank_MF-1: VAUC=0.8069, TAUC=0.7712
  $\eta = 0.01$, $\lambda = 0.0001$, $size\_of\_latent\_factor = 10$, $iterations = 30$
  features: pCTR_Ad, pCTR_Advertiser, pCTR_Title, pCTR_Keyword, binary_User, binary_Ad, binary_Query, num_Query, num_Title, num_Description, num_Keyword, binary_QueryTokens, binary_TitleTokens, binary_DescriptionTokens, binary_KeywordTokens

- Rank_MF-2: VAUC=0.8139, TAUC=0.7853
  $\eta = 0.01$, $\lambda = 0.0001$, $size\_of\_latent\_factor = 10$, $iterations = 40$
  features: pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_Title, pCTR_Description, pCTR_User, pCTR_Keyword, value_Title, value_Query, value_Keyword, value_Description, value_User, binary_Ad, binary_Advertiser, binary_Query, binary_Title, binary_Description, binary_User, binary_Keyword, binary_Position, binary_QueryTokens, binary_TitleTokens, binary_DescriptionTokens, binary_KeywordTokens
- Rank_MF-3: VAUC=0.8227, TAUC=0.7920
  $\eta = 0.01$, $\lambda = 0.0001$, $size\_of\_latent\_factor = 10$, $iterations = 100$
  features: pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_User, value_Title, value_Query, value_Keyword, value_Description, value_User, binary_Ad, binary_Advertiser, binary_Query, binary_Title, binary_Description, binary_User, binary_Keyword, binary_Position, binary_QueryTokens, binary_TitleTokens, binary_DescriptionTokens, binary_KeywordTokensi, num_Query, num_Title, num_Description, num_Keyword, binary_Age, binary_Gender
- Rank_MF-4: VAUC=0.8236, TAUC=0.7935
  $\eta = 0.01$, $\lambda = 0.0001$, $size\_of\_latent\_factor = 10$, $iterations = 100$
  features: pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_User, value_Title, value_Query, value_Keyword, value_Description, value_User, binary_Ad, binary_Advertiser, binary_Query, binary_Title, binary_Description, binary_User, binary_Keyword, binary_Position, binary_QueryTokens, binary_TitleTokens, binary_DescriptionTokens, binary_KeywordTokensi, num_Query, num_Title, num_Description, num_Keyword
- Rank_MF-5: VAUC=0.8246, TAUC=0.7968
  $\eta = 0.005$, $\lambda = 0.0001$, $size\_of\_latent\_factor = 10$, $iterations = 80$
  features: pCTR_Ad, pCTR_Advertiser, pCTR_Query, pCTR_User, value_Title, value_Query, value_Keyword, value_Description, value_User, binary_Ad, binary_Advertiser, binary_Query, binary_Title, binary_Description, binary_User, binary_Keyword, binary_Position, binary_QueryTokens, binary_TitleTokens, binary_DescriptionTokens, binary_KeywordTokensi, num_Query, num_Title, num_Description, num_Keyword

B.10 Validation Set Blending Predictors

Following models are trained on validation set, so we use *VAUC here to distinguish them from other models.

- VB-1: *VAUC=0.8321, TAUC=0.8029
  blending by poly-2 SVR
  features: SVR-2, NB-3, SGD_SVM-3, Reg_MF-2, LR-2, Rank_MF-4
- VB-2: *VAUC=0.8326, TAUC=0.8038
  blending by poly-2 SVR, scaling features by normalized score to [0,1]
  features: SVR-2, NB-3, SGD_SVM-3, SGD_SVM-4, Reg_MF-2, LR-2, Rank_MF-4
- VB-3: *VAUC=0.8322, TAUC=0.7995
  blending by poly-2 SVR
  features: NB-1, RR-1, RLR-1, Reg_MF-1, Reg_MF-2, LR-1, LR-2, Rank_MF-2, Rank_MF-3
- VB-4: *VAUC=0.8376, TAUC=0.8051
  blending by LambdaMart
  features: NB-1, RR-1, RLR-1, Reg_MF-1, Reg_MF-2, LR-1, LR-2, Rank_MF-2, Rank_MF-3
- VB-5: *VAUC=0.8357, TAUC=0.8058
  blending by RankNet, scaling features by rank score, also use square of id raw value without score transformation, $\lambda = 0$, $\zeta = 1$
  features: NB-1, RR-1, RLR-1, Reg_MF-1, Reg_MF-2, LR-1, LR-2, Rank_MF-2, Rank_MF-3, AUV_MF-4, value_Title, value_Query, value_Keyword, value_Description, value_User
- VB-6: *VAUC=0.8271, TAUC=0.7934
  blending by RankNet, scaling features by ranked score, $\eta = 1e^{-10}$, $\lambda = 2e^{-3}$, $\zeta = 2$
  features: SVR-1, NB-2, NB-3, SGD_SVM-1, SGD_SVM-2, LR-6, RN-1, Rank_MF-1
- VB-7: *VAUC=0.8330, TAUC=0.8049
  blending by RankNet, scaling features by rank score, $\eta = 1e^{-10}$, $\lambda = 2e^{-3}$, $\zeta = 2$
  features: NB-1, RR-1, RLR-1, Reg_MF-1, Reg_MF-2, LR-1, LR-2, Rank_MF-2, Rank_MF-3, AUV_MF-4
- VB-8: *VAUC=0.8336, TAUC=0.8043
  blending by RankNet, scaling features by ranked score, $\eta = 1e^{-10}$, $\lambda = 2e^{-3}$, $\zeta = 2$
  features: NB-1, NB-2, NB-3, RR-1, RLR-1, Reg_MF-1, Reg_MF-2, LR-1, LR-2, Rank_MF-1, Rank_MF-2, Rank_MF-3, AUV_MF-4, SGD_SVM-3
- VB-9: *VAUC=0.8348, TAUC=0.8031
  two-level blending by poly-2 SVR, scaling features by normalized score to [-1,1]
  features: VB-1, VB-2, VB-3, VB-6, VB-7
- VB-10: *VAUC=0.8319, TAUC=0.8050
  two-level blending by CRR, only use square of raw ID value features
  features: VB-7, value_Title, value_Query, value_Keyword, value_Description, value_User

- VB-11: *VAUC=0.8356, TAUC=0.8059
  two-level blending by LambdaMART on Random Forest's residual
  features: NB-1, NB-2, NB-3, RR-1, RLR-1, Reg_MF-1, Reg_MF-2, LR-1, LR-2, LR-3, Rank_MF-1, Rank_MF-2, Rank_MF-3, AUV_MF-4, SGD_SVM-3, SGD_SVM-4, VB-9
- VB-12: *VAUC=0.8357, TAUC=0.8062
  two-level blending by RankNet, scaling features by ranked score, also use square of id raw value as features, $\eta = 1e^{-10}$, $\lambda = 1e^{-3}$
  features: NB-1, RR-1, RLR-1, Reg_MF-1, Reg_MF-2, LR-1, LR-2, LR-4, LR-5, Rank_MF-2, Rank_MF-3, AUV_MF-4, Rank_MF-5, VB-1, VB-4, VB-8, value_Title, value_Query, value_Keyword, value_Description

B.11 Test Set Ensemble Predictors

- TE-1: TAUC=0.8065
  manual diversity feature selection uniform average
  features: VB-4, Vb-5, VB-7, VB-9, VB-10, VB-11, VB-12
- TE-2: TAUC=0.8063
  manual diversity feature selection leader board ensemble
  features: VB-4, Vb-5, VB-7, VB-9, VB-10, VB-11, VB-12
- TE-3: TAUC=0.8069
  uniform average average of top five models
  features: VB-5, VB-11, VB-12, TE-1, TE-2