# Report on Expedia Ranking Competition

You Hu$^{[2631052, yhu310]}$, Wenchen Lai$^{[2643117, wli310]}$, and Chang Tian$^{[11653205, ctn930]}$

Vrije University Amsterdam, DMT Group 53

**Abstract.** Having a metadata which portraits user choices, user attributes, hotel attributes and competitors' attributes. We aim at exploring pattern relations between website data and hotel recommendation list and presenting reasonable ranking of hotels list for each specific search query. The data preparation contains filling NAN scheme, feature normalization and feature extraction. During tests, LambdaMART outperforms than other regression or classification models such as Random Forest, Neural Network, Logistic Regression and so on and proves its benefits with a result score 0.37240.

**Keywords:** Feature Engineering · LambdaMART · Recommendation Systems

## 1 Introduction

This project is about how to reasonably present hotel lists when users query. It originates from one similar Kaggle competition called Expedia Hotel Searches. Among many machine learning models used in previous competition, LambdaMART[1] performs better than others through combing MART model and LambdaRank models[2][3][4]. LambdaRank defines differentiable loss and induces gradient descent for loss by converting common IR metrics into continuous values. While MART models try to fit loss gradient by adding and combining more weaker learning models to form one strong trained model. During this process, MART will also give more weights to error prone samples for faster convergence. Our project utilizes the benefits of above combination learner called LambdaMART to predict hotel lists.

## 2 Exploratory Data Analysis and Feature Engineering

Training set contains approximately 5 million records and 54 features. 4 features('position', 'booking_bool', 'click_bool' and 'goss_booking_usd') are only provided in training set. Since our task is to predict which hotel is most likely to be booked, booking_bool and click_bool will be used to calculate our target. Exploring the data, we will find that about 95% of records are neither booked nor clicked, 31% of search (one 'srch_id' means one search) do not book any hotel and 94% of booking search (the user books a hotel in a search) just click once.

The findings show our dataset is unbalanced and for a classification problem it may lead to the prediction to be neither booked nor clicked more likely. To deal with it, we choose lambdaMart model (more details in section 3.1) which is not much sensitive to unbalanced dataset.

There are 50 features both in train set and test set. After calculating 'booking_bool' correlation with these 50 features, we will find the most important feature is 'prop_location_score2' and it is most correlated to 'prop_location_score1'. 50 features can be separated into 3 categories: user data, hotel data and competitors data. More exploratory data analysis and feature engineering about these 3 categories features will be discussed in following 3 subsections. Our work mainly refers to [2] and [3].

## 2.1 Missing Data

All competitors' features have missing value and Table 1 also shows user's and hotel's features with missing value and their percentages. We set customized schemes for each categorical missing data as follows:

*Hotel descriptions* According to Figure 1, we find that users are unlikely to book or click hotels whose review score or other comment scores are missing because of unpredictable results. So in order to guide our model to notice this, we sets missing values into a worse scenario values matching original value range. Also, we create one extra column to indicate which values are filled manually to make sure model will be mislead much by given values.

*User's data* After calculating the correlation matrix (Figure 2), we found 'visitor_location_country_id', 'prop_country_id', 'site_id' and 'srch_destination_id' are more related to user's historical features. Thus, we fill all missing user historical data based on the assumption that people coming from same city or place will have similar shopping experiences and feelings for one hotel. So we group them by 'visitor_location_country_id' and use each group's mean values as default values for missing data. And their shopping abilities will not vary much. As for 'orig_destination_distance', people coming from same country and heading to same destination will have similar distance.

*Competitors' data* Since there is no significant difference of click rate or book rate between related competitors label 0 data and label missing value data, i.e NaN data. So we set all the missing values into 0.

| | | | |
|---|---|---|---|
| visitor_hist_starrating | 94.920364 | prop_review_score | 0.148517 |
| visitor_hist_adr_usd | 94.897735 | prop_location_score2 | 21.990151 |
| orig_destination_distance | 32.425766 | srch_query_affinity_score | 93.598552 |

**Table 1.** Percentage of missing values except competitors data

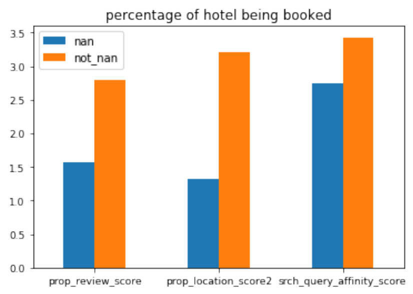


**Fig. 1.** Percentage of hotel being booked between NaN and not NaN values

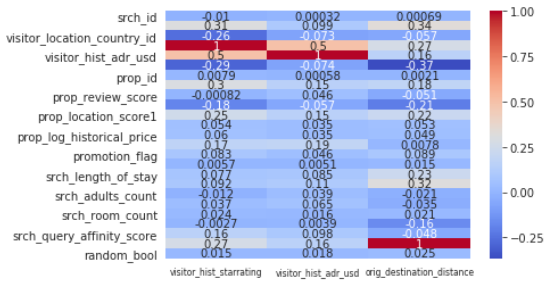


**Fig. 2.** Correlation Matrix of user's features with missing data

## 2.2 Feature Extraction

For user data, 'year', 'month', 'day', 'hour', 'quarter' and 'dayofweek' can be extracted from 'date_time' and we only choose 'month' because 'month' has the highest correlation with 'booking_bool'. Another features we can find are the popular destination and country. We create a column to record if the user's destination id equals to the top 2 popular destination id. We also generate 'hist_price_diff', 'starrating_diff' and 'usd_diff' by the formula below.

$$hist\_price\_diff = abs(exp(prop\_log\_historical\_price) - price\_usd)$$

$$starrating\_diff = abs(visitor\_hist\_starrating - prop\_starrating)$$

$$usd\_diff = abs(visitor\_hist\_adr\_usd - price\_usd)$$

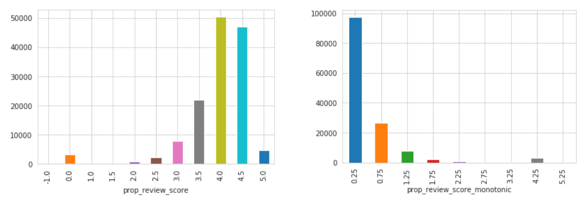
For hotel data, the probability of a hotel being clicked or booked can be calculated. Some hotels are not in training set but in test set and these two values are set to 0.

$$click\_prob = click\_times(prop\_id)/presentation(prop\_id)$$

$$booking\_prob = booking\_times(prop\_id)/presentation(prop\_id)$$

Some hotel's features are non-monotonic as Figure 3(a) shows. Take 'prop_review_score' as example. It reveals users do not tend to book the hotel with higher score. The hotel with a popular score are more likely to be booked. We plot bar chart or histogram for these non-monotonic features and get their popular scores by observation. Then we can extract

$$prop\_review\_score\_monotonic = abs(prop\_review\_score - popular\_score)$$



(a) non-monotonic (b) monotonic

**Fig. 3.** non-monotonic features (prop_review_score)

## 2.3 Normalization

When considering the competitiveness of one item, we should not only score it by the absolute value of each attribute, but also by the relative position among a group of items with the same conditions. The price of the room is the most important attribute that effects to competitiveness. Therefore, we normalized the price in multiple dimensions. The normalization follows the rule below:

$$X_i' = \frac{X_{MAX} - X_i}{X_{MAX} - X_{MIN}} \tag{1}$$

where, $X_i'$ is the normalized value of attribute $x$ of item $i$,$X_{MAX}$ and $X_{MIN}$ are the maximum and minimum value in this group. $X_i$ is the value of item $i$, therefore the maximum and minimum values will be scaled to 0 and 1. For the normalization of price, the dimensions and the meaning of normalization are listed below:

- Overall: To get the rank of price in all items.
- Query: To measure the competitiveness among one search query.
- Hotel: To compare the price to historical price.
- Month: To reduce the season effect.
- Destination: To measure the competitiveness among one destination/place.

Although when we calculated the correlations between those normalized attributes there was no significant correlation shown(all lower than 0.01), we still believe the normalization is useful for tree models.

## 3 Modeling

### 3.1 LambdaMART

The Random-forest model aims to classify the samples into two groups and assign each of them a possibility to be clicked. However it is not directly for our goal in theory. Therefore, we choose LambdaMART for ranking which is the boosted tree version of LambdaRank[1]. The basic idea of LambdaMART is consisted of 2 parts:Multiple Additive Regression Tree(MART) and LambdaRank.

**MART** MART is the model that follows the idea of boosting. There are three main features: using decision tree for prediction; multiple trees used;every tree getting slight improvement from previous one. MART can be viewed as performing gradient descent in function space using regression trees. The goal of MART is to find the model $f(x)$ meets the equation below:

$$\hat{f}(x) = \arg\min_{f(x)} E[L(y, f(x))|x] \tag{2}$$

As it is an additive model, after training it will be a sum of sub models:

$$\hat{f}(x) = \hat{f}_M(x) = \sum_{m=1}^{M} f_m(x) \tag{3}$$

Here,

$$\hat{f} = \hat{f}_M : R^d \mapsto R$$

is the target function of the model; $\boldsymbol{x} \in R^d$ is sample which is consist of $d$ features;$\hat{f}(x) \in R$ is the prediction; $M$ represents the number of iteration which is also the number of tree in the model; $f_m : R^d \mapsto R$ is the week model in training process.

Therefore, remaining unsolved problem is to determine the direction of fitting in iterations. Assume $m$ times iterations have been done, which means there are $m$ trained trees, set those trees as $\hat{f}_m(x) = \sum_{i=1}^{m} f_i(x)$. Therefore, the target for the $m+1$ iteration is

$$\delta\hat{f}_{m+1} = \hat{f}_{m+1} - \hat{f}_m = f_{m+1} \tag{4}$$

And then, introduce loss function $L = L((x,y), f) = L(y, f(x)|x)$ for getting $\delta\hat{f}_{m+1}$ and measure the difference between predictions and true values. The target turns to that the difference should be reduced, which means:

$$\delta L_{m+1} = L\left((x,y), \hat{f}_{m+1}\right) - L\left((x,y), \hat{f}_m\right) < 0 \tag{5}$$

Considering that

$$\delta L_{m+1} \approx \frac{\partial L\left((x,y), \hat{f}_m\right)}{\partial \hat{f}_m} \cdot \delta\hat{f}_{m+1} \tag{6}$$

If we set

$$\delta \hat{f}_{m+1} = -g_{im} = -\frac{\partial L\left((x,y),\hat{f}_m\right)}{\partial \hat{f}_m} \tag{7}$$

There will always be $\delta L_{m+1} < 0$. Therefore, this $\delta \hat{f}_{m+1}$ is the target of fitting for $f_{m+1}$; It means that for $f_{m+1}$, it need to fit a dataset:

$$\left\{ i = 1,2,\ldots,N \middle| \left( x_i, -\frac{\partial L\left(\left(x_i,y_i\right),\hat{f}_m\left(x_i\right)\right)}{\partial \hat{f}_m\left(x_i\right)} \right) \right\} \tag{8}$$

What it is described in equation 7 is the gradient of loss function $L$. This means that in each iteration the target of fitting is the gradient of loss function, that's why it is called as "gradient boost".

The decision trees actually divide the space into multiple areas and assign predictions on each of area. Let $R_m$ as the areas divided by $f_m$, the prediction is $\gamma_m$, therefore:

$$f_m(x) = h_m\left(x; R_m, \gamma_m\right) \tag{9}$$

$$\hat{f}(x) = \sum_{m=1}^{M} f_m(x) = \sum_{m=1}^{M} h_m\left(x; R_m, \gamma_m\right) \tag{10}$$

According to that, each iteration of MART turns to solve the optimization problem:

$$h_m\left(x; R_m, \gamma_m\right) = \arg\min_{R,\gamma} \sum_{i=1}^{N} \left(-g_{im} - F\left(x_i; R, \gamma\right)\right)^2 \tag{11}$$

Here, $F$ represents a regression decision tree. Now, we introduce a very slight positive number $\eta$ called "learning rate", to adjust the target from $-g_{im}$ to $-\eta * g_{im}$. Therefore, every iteration of fitting is one small step towards the right direction. This kinds of shrinkage is used for preventing over-fitting.

The most important idea of MART is that the gradient of loss function is the target of each iteration of fitting. What should be noticed is that almost any derivable loss function is applicable. This means we can embed any derivable function into MART. LambdaMART uses $\lambda$ value to replace the gradient of loss function.

**Lambda** LambdaRank inherits from RankNet, the innovation of it is that it transforms the Ranking problems which are not suitable for gradient descending into the optimization problems for loss function of the cross entropy to possibility. The ultimate target for RankNet is to get a scoring function:

$$s = f(x; w) \tag{12}$$

According to this scoring function we can get $s_i$ and $s_j$ for item $x_i$ and $x_j$, and then their partial order probability can be calculated as below:

$$s_i = f\left(x_i; w\right) \quad s_j = f\left(x_j; w\right) \tag{13}$$

$$P_{ij} = P\left(x_i \supset x_j\right) = \frac{\exp\left(\sigma \cdot \left(s_i - s_j\right)\right)}{1 + \exp\left(\sigma \cdot \left(s_i - s_j\right)\right)} = \frac{1}{1 + \exp\left(-\sigma \cdot \left(s_i - s_j\right)\right)} \tag{14}$$

After we get partial order probability, we set cross entropy as loss function, and perform gradient descending:

$$\begin{aligned} L_{ij} &= -\overline{P}_{ij} \log P_{ij} - \left(1 - \overline{P}_{ij}\right) \log\left(1 - P_{ij}\right) \\ &= \frac{1}{2}\left(1 - S_{ij}\right) \sigma \cdot \left(s_i - s_j\right) + \log\left\{1 + \exp\left(-\sigma \cdot \left(s_i - s_j\right)\right)\right\} \end{aligned} \tag{15}$$

$$w_k \to w_k - \eta \frac{\partial L}{\partial w_k} \tag{16}$$

To combine the metrics that are not derivable, like NDCG, and RankNet. In LambdaRank the $\lambda$ is defined as below:

$$\lambda_{ij} \overset{\text{def}}{=} \frac{\partial L_{ij}}{\partial s_i} = -\frac{\partial L_{ij}}{\partial s_j} \tag{17}$$

$$\lambda_{ij} \overset{\text{def}}{=} -\frac{\sigma}{1 + \exp\left(\sigma \cdot \left(s_i - s_j\right)\right)} \tag{18}$$

When considering metrics $Z$(like NDCG), it is defined as

$$\lambda_{ij} \overset{\text{def}}{=} -\frac{\sigma}{1 + \exp\left(\sigma \cdot \left(s_i - s_j\right)\right)} \cdot \left|\Delta Z_{ij}\right| \tag{19}$$

For each specified item $x_i$

$$\lambda_i = \sum_{(i,j) \in P} \lambda_{ij} - \sum_{(j,i) \in P} \lambda_{ij} \tag{20}$$

This indicates that the direction and trend of the movement of each item are determined by all items and itself. Therefore the loss function of LambdaRank can be deduced:

$$L_{ij} = \log\left\{1 + \exp\left(-\sigma \cdot \left(s_i - s_j\right)\right)\right\} \cdot \left|\Delta Z_{ij}\right| \tag{21}$$

Here, MART is a boosting "framework" can be embedded by loss function to perform gradient boost; LambdaRank defines a gradient that transforms some underivable metrics into a form of derivable. LambdaMART is the combination of two of them. It has many advantages: It makes ranking problem solved directly instead of using classification and regression methods; The IR indexes like NDCG those not derivable can be transformed into derivable loss function, which has practical meaning;As it is an additive model, it can continues training from existed models; it is not sensitive to the ratio of positive and negative samples.

## 4 Experiment and Evaluation

### 4.1 Metrics

According to the document of this competition, the final score is based on Normalized Discounted Cumulative Gain which is a measure of ranking quality. The benefit of NDCG compared to DCG is that it reduces the effect of the length of query. More forward the relevant documents are pushed, higher the NDCG is. When we were training the LambdaMART model, we set NDCG as objective function, and monitored the performance according to the NDCG score on validation set.

### 4.2 Benchmark

To validate the improvement of our works, we set multiple benchmarks.

**Random Forest+Raw Data** The basic benchmark is applying Random Forest model on raw data. Here, our task is to set up a classifier to predict the items might be clicked, and assign each item a probability to be clicked as the relevance score. The output is ordered by their probabilities. In this case, without feature engineering, the NDCG score on test set is 0.31.

**Random Forest+Feature Engineering** The second benchmark it to apply Random Forest model on the processed data. It is the benchmark for modeling as the models afterward are all applied on processed data. We tried multiple parameter settings and the final score on test set is 0.33. It shows that the feature engineering is helpful for improving the accuracy of the model.

**Other models+Feature Engineering** We applied multiple models on processed data including linear regression, gradient boost classification and regression. However, the scores show that all these model are not better than Randomforest on this task.

### 4.3 Experiment and Parameter Tuning

Our implementation of LambdaMART is provided by xgboost, when we set ranking as the objective and NDCG as objective function. There are multiple parameters that affect the performance of model. We split a part of the train set as validation set to monitor the performance of during the training process and evaluate the parameter settings. There are multiple parameters that have significant effect on the performance of model:

- Learning Rate: the modulus $\eta$ that shrinks the step of gradient descending
- Max Depth: the maximum depth of decision tree
- N Estimators: the number of trees

The results of parameter tuning is shown on Table 2. Here we set fixed number of estimators. As LambdaMART is an additive model, we collected the best score during the training and the score of last iteration. In most case, if last score is the best score, it means that the training is not sufficient, there is still a room for improving. According to the table, we set 0.3 as learning rate, 8 as max depth and 100 estimators. The score on test set is 0.37240, compare to our benchmark random forest with and without feature engineering, there are 12.7% and 20% improvement respectively.

**Table 2.** Parameter tuning for LambdaMART

| Learning Rate | Max Depth | N_estimators | Best score | Last score |
|---|---|---|---|---|
| 0.1 | 8 | 100 | 0.564769 | 0.564769 |
| 0.1 | 9 | 100 | 0.566881 | 0.566881 |
| 0.1 | 10 | 100 | 0.568038 | 0.567995 |
| 0.1 | 12 | 100 | 0.567647 | 0.56754 |
| 0.1 | 15 | 100 | 0.561388 | 0.560896 |
| 0.3 | 8 | 100 | **0.572048** | **0.571936** |
| 0.3 | 9 | 100 | 0.568456 | 0.568165 |
| 0.3 | 10 | 100 | 0.569043 | 0.568728 |
| 0.3 | 12 | 100 | 0.564187 | 0.563967 |
| 0.3 | 15 | 100 | 0.557903 | 0.557151 |
| 0.5 | 8 | 100 | **0.572016** | **0.571857** |
| 0.5 | 9 | 100 | 0.568644 | 0.568512 |
| 0.5 | 10 | 100 | 0.567273 | 0.566584 |
| 0.5 | 12 | 100 | 0.562741 | 0.562009 |
| 0.5 | 15 | 100 | 0.54864 | 0.547982 |

## 5 Discussion and Conclusion

According to the results of parameter tuning, learning rate and number of estimators have effect on the performance. Too small learning rate will led model unable to reach the best point within fixed number of iteration(equal to number of estimators), however we can set a large number of estimators and let it stop if there is no improvement in few iterations. Larger learning rate will accelerate convergence, but it would probably miss the best point. However, we are not sure whether it reaches the local optimal solution or global optimal solution and in the same time the learning rate is fixed. We tested whether random seed has effect, the results shows that there is no difference with different seeds.

The maximum depth is very important, it should be neither too small nor too large: too small will let the model to conservative; too large will lead to overfitting which is harmful to the generalization. According to our parameter tuning, when set maximum depth as 8, we can achieve highest score among validation set.

When we evaluate the model on validation set, we can observe that the NDCG score is much higher than the score on test set. The reason we guess is that on test set, it does not have part of information. For example, there are two very important attributes click_prob and book_prob which reflect their popularity in history. However, in test set, there are some hotels that never appear in training set, therefore we can not assign a estimation on the quality of one hotel that does not show on training set. There will be a problem that if we assign 0 to them, it will make error to those with high quality. And if we ignore it and let it be null or NaN, the problem is that in training set, there is no NaN situation, the model is not able to learn how to handle NaN values. We proposed an idea that we can use K-NN to estimate that value, however, we did not do that due to the time limited.

In conclusion, in comparison between regression or classification and direct ranking models, ranking model is much more effective. Feature engineering is very important as well, it determines the ceiling of the performance.

## References

1. Burges, C.J.: From ranknet to lambdarank to lambdamart: An overview. Learning **11**(23-581), 81 (2010)
2. Liu, X., Xu, B., Zhang, Y., Yan, Q., Pang, L., Li, Q., Sun, H., Wang, B.: Combination of diverse ranking models for personalized expedia hotel searches. arXiv preprint arXiv:1311.7679 (2013)
3. Wang, J., Kalousis, A.: Personalize expedia hotel searches–2nd place. URL: https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013?preview =4_jun_wang.pdf, accessed December 8, 2013
4. Zhang, O., Woznica, A.: Personalize expedia hotel searches–1nd place. URL: https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013?preview =3_owen.pdf, accessed December 8, 2013