Rank hotels on Expedia.com to maximize purchases

Nishith Khantal, Valentina Kroshilina, Deepak Maini

December 14, 2013

1 Introduction

For an online travel agency (OTA), matching users to hotel inventory is very important. As such, having the best ranking (or sort order) of hotels for specific users with the best integration of price competitiveness gives an online travel agency the best chance of winning the sale. Imagine a hotel ranking system that predicts a permutation of relevant hotels to a user such that a hotel at a higher position always has higher or equal probability of purchase than a hotel at a lower position. Such an order will ensure that a user sees the most relevant hotels first and increase the likelihood that the online travel agency wins the transaction.

Ranking algorithms are typically used in document retrieval, expert search, definition search, collaborative filtering, question answering, keyphrase extraction, document summarization, and machine translation applications [5].

Even though ranking algorithms are used extensively in many fields, their application in the online travel booking industry is new. Data collected on travel booking sites are unique to the industry and present an interesting opportunity for applying machine learning algorithms to understand what features are important and which not so. Furthermore, there is very little literature on which algorithm is best suited to obtain optimal results. In this paper, you will find our analysis of some of the most popular algorithms and their relative performance.

2 Approach

Learning to rank is a supervised learning task and thus has training and testing phases. In our case, training data consists of search history of different users, including data on (1) hotel characteristics, (2) location attractiveness of hotels, (3) user's aggregate purchase history, and (4) competitive OTA information. Each search is represented by a 51-dimensional vector. The target vector for each search consists of two class variables (click and book) and a continuous variable (gross purchase amount).

Suppose that $Q_i = \{q_{i_1}, q_{i_2}, q_{i_3}, \ldots, q_{i_m}\}$ is a set of hotels returned for a search query i and suppose that y_{i_j} is a relevance label for a hotel q_{i_j}. Let $y_{i_j} \in \{0, 1\}$ where "1" means that the hotel was booked and "0" means the hotel wasn't booked. Therefore, label "1" is of higher relevance than label "0". We aim to train a ranking model that can assign a score to a given hotel-search pair. Each hotel q_{i_j} in search i have a set of features, some of which are hotel specific, others search, user, and competitive pricing and inventory specific.

Ranking is finding the permutation π_i of $\{i_1, i_2, i_3, \ldots, i_m\}$ for a given search i using the scores given by the ranking model. After training the ranking model we evaluate its performance on a held-out test data. The trained model predicts rank of hotels in the test data set and the predicted order is compared against the observed relevance labels. Evaluation measures, such as NDCG (Normalized Discounted Cumulative Gain), DCG (Discounted Cumulative Gain), MAP (Mean Average Precision), are widely used for ranking algorithms. In this paper, we use NDCG. As defined in [2]

$$NDCG(n) = Z_n \sum_{j=1}^{n} \frac{2^{R(j)} - 1}{\log(1+j)}$$

where n denotes position, $R(j)$ is the label at position j, Z_n is a normalizing factor such that a perfect ranking's NDCG at position n is 1. We report NDCG averaged for all queries for max position n.

3 Experiments

We initially used Logistic Regression and Naive Bayes to rank instances based on their predicted class. The challenge that we faced was presence of categorical features with large number of levels. Using categorical features without transformation produced poor results. We were able to obtain a performance gain by converting number of categorical features to booking probabilities. We were able to further increase NDCG by selecting a fewer number of features based on importance scores. Additionally, we wanted to investigate how random selection of features improves performance, therefore, we implemented Random Forest model. We also implemented Ranking SVM to see if pairwise approach that it utilized fits our data better.

3.1 Data

The data consists of records capturing Expedia users search history. We used a subset of the data available at Kaggle.com. Brief overview of the data used to evaluate proposed methods is given in Table 1:

num. of examples	1,000,000
num. of queries	40,173
num. of bookings	27,738
num. of continuous features	43
num. of features	51
num. of target attributes	3
percent of empty input values	43%
average number of returned items per query	24

Table 1: Characteristics of the data set

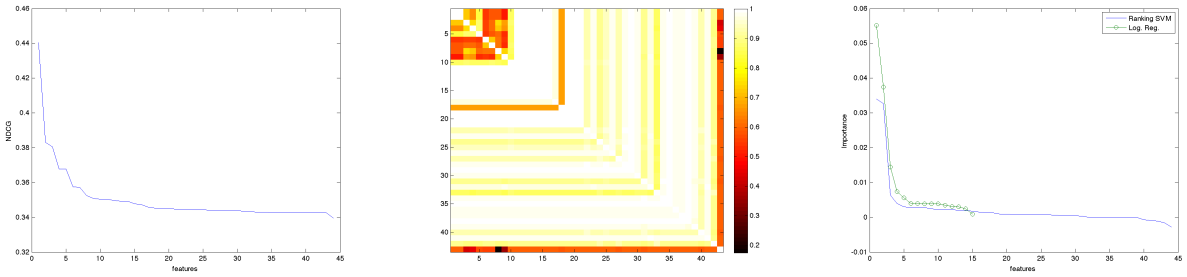
3.2 Features

Since redundancy of the features can reduce effectiveness of ranking algorithms, we used methodology proposed in [2] to investigate similarity between features. In [2] features are considered as undirected weighted graph where each feature is node. Each node is assigned an importance score and edge between any two nodes is assigned a weight equal to similarity score. Importance of a feature is defined as NDCG score if we rank based on feature values. Similarity is defined as a proportion of identically ranked items between two different features. Importance of the features is shown on Fig. 1(a). Fig. 1(b) reflects pairwise similarity. The level of lightness represents the strength of similarity between two features. Thus, there are two blocks of similar features, and much less number of attributes with more diverse effect on ranking.

For this exercise we used a data sample of $\sim$75K rows and all continuous features, as well as two additional fields:

1. delta between price of a property and average price of user's purchases ($DerivedFeature1$)

2. delta between rating of a property and average star rating of properties previously booked by user



(a) (b) (c)

Figure 1: (a): Importance scores (b): Similarity scores (c): Experimental importance scores

In our later experiments we defined an importance score for feature $F_1 \in S$ as following:

$$Importance(F_1) = NDCG(S) - NDCG(S \backslash F_1) \tag{1}$$

where S is a fixed set of features, $NDCG(S)$ is NDCG value obtained by training on set of features S. Fig. 1(c) shows importance scores of two sets of features used in different experiments described in the next section.

3.3 Models

3.3.1 First experiments

Naive Bayes For Naive Bayes, we hand-selected a few features, mostly property features, such as price, location scores, etc., to get a basic machine learning algorithm running. We then learned the model on 70% of 20k, 100k, and 500k examples and tested the resulting model on the remaining 30% examples and calculated NDCG values. We then added new features based on their importance scores. We calculated target class probabilities for the categorical features, and we normalized continuous variables based on target label values. For each attribute in the test set, we converted its value to booking probability based on the the observed probabilities in the training set, and finally we ranked instances using log of the sum of probabilities over all attributes. We obtained the best NDCG of 0.3818.

Logistic Regression Our initial attempts at Logistic Regression didn't predict any positive values due to sparsity of the data and abundance of negative examples. We took several corrective measures: subsampling, basic pruning of features, normalization of continuous features. We also experimented with derived features (Purchase rate for each property, purchase rate for each property and destination, average values of various features for each property etc.) to improve the performance. We ranked features based on importance scores and trained using 14 most relevant features. We see a potential future work focused on further exploration of effects of using probabilities as features on performance.

3.3.2 Second phase of experiments

Random Forest Because linear models didn't provide us with satisfactory results, we decided to a try non-linear classification model. We picked Random Forest because of its effectiveness and ease of implementation. Using the standard Random Forest implementation in R, we trained classification models on several subsets of the data. For training, we used all continuous variables and all categorical variables with fewer than or equal to

32 levels (a restriction imposed in the standard implementation of Random Forest). We used default settings of 500 trees in the model with 5 variables tried at each split in each tree. Furthermore, for computational speed, we downsampled number of negative examples in the training set so as to maintain a ratio of 5:1 between negative and positive examples. We believe that the benefit of increased computational speed outweighed any loss of model performance from downsampling. Using the trained model, we predicted booking probabilities for properties within each search query in the test set and then ordered the properties from the highest to lowest value of booking probability. We then used the predicted order to calculate NDCG for the model.

Ranking SVM Ranking SVM transforms ranking into pairwise SVM classification, where instances from different groups (searches in our case) are incompatible. To train we used SVM-Rank library based on [3]. We used a linear version of SVM, with $C = 20$. For training we've used all continuous features and two additional fields described in section 2. We explored several ways of normalizing the features and only normalizing results belonging to one query slightly increased NDCG. Since SVM uses dot product of vectors we set missing values to 0 to exclude their effect on minimization problem. Ranking SVM outperformed both Logistic Regression and Naive Bayes models in terms of NDCG. Our initial findings show that Ranking SVM performance on derived features described in section 3.3.1 drops but is comparable to Logistic Regression performance.

3.4 Comparison

Table 2 lists top 5 features[1] in terms of importance score as defined in Equation 1 in section 3.2. We can see that Random Forest and SVM have almost identical set of top features, while Logistic Regression list is diversified by $comp8\_rate$ and $comp5\_rate$. Both of these features belong to the biggest block of similar features that we see in Fig. 1(b) in section 3.2. Thus, not enough diversity in the features might explain lower performance of Logistic Regression compared to Random Forest and Ranking SVM. Top features for Naive Bayes are not listed due to complexity of calculation, but we know that after adding $price\_usd$ and $location\_score\_2$ features performance increased the most. Thus, it's also consistent with what we see for other models.

importance rank	Logistic Regression	Random Forest	Ranking SVM
1	$prop\_location\_score$	$prop\_location\_score2$	$prop\_location\_score2$
2	$price\_usd$	$price\_usd$	$price\_usd$
3	$prop\_star\_rating$	$DerivedFeature1(see\ 3.2)$	$prop\_review\_score$
4	$comp8\_rate$	$prop\_location\_score$	$DerivedFeature1(see\ 3.2)$
5	$comp5\_rate$	$prop\_log\_historical\_price$	$comp1\_inv$

Table 2: Top 5 features across models

We also compared NDCG values as a function of input size. The highest NDCG value observed for Naive Bayes was 0.3818. Therefore, Naive Bayes is outperformed by other models as we can see from Fig. 2. Naive Bayes still performs better than random ordering which has $NDCG = 0.34$.

Also, we can see on Fig. 2 that NDCG for Logistic Regression is declining with increase of data size. We think that such behavior can be explained by the way we normalize columns. In particular, with the increase of input size derived fields that represent mean of different attributes based on target label eventually converge to the global mean of the whole data set. Therefore, our strategy to average based on target label becomes less effective.

Ranking SVM outperforms Logistic Regression but is stagnant for the last two data sets. We believe that the optimal w to separate target labels won't be changing, because the data is even and it seems that for the last two data sets Ranking SVM converged to global optimum.

[1]Detailed description of features is available at http://www.kaggle.com/c/expedia-personalized-sort/data

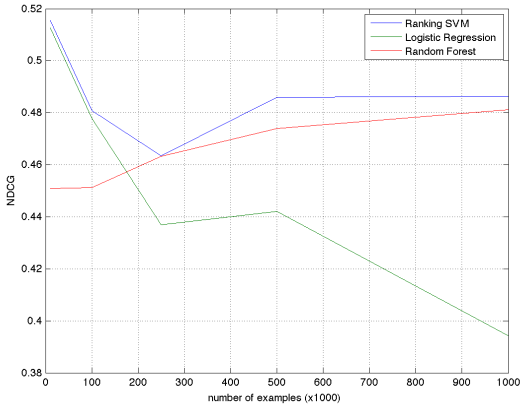


Figure 2: NDCG dependence on size of the input data set.

Unlike other models NDCG for Random Forest keeps growing with the increase of input size. Therefore, we think that Random Forest is the most promising approach. Additional advantage of Random Forest is that it can more effectively account for non-linear dependencies in the data than other models we've used. Thus, we believe that Random Forest will eventually outperform Ranking SVM, though the max NDCG value that we observed for 100mln examples belongs to Ranking SVM ($NDCG = 0.4863$).

4 Conclusion

We applied most popular machine learning algorithms to solve a ranking problem with an intricate set of features and reviewed the relative performance of each algorithm. Based on our findings, we made a recommendation for the best algorithm to solve the problem. We also proposed a list of top 5 most important features to demonstrate what drives purchase behaviour on online travel booking websites. One possible direction for our future work is to explore optimal parameters for the used algorithms, as well as perform further research on feature properties of the data set.

References

[1] MariaFlorina Balcan, Nikhil Bansal, Alina Beygelzimer, Don Coppersmith, John Langford, and Gregory B. Sorkin. Robust reductions from ranking to classification. *Machine Learning*, 72:139153, 2008.

[2] Xiubo Geng, Tie-Yan Liu, Tao Qin, and Hang Li. Feature selection for ranking. 2007.

[3] T. Joachims. Training linear svms in linear time. 2006.

[4] Hang Li. A short introduction to learning to rank. *IEICE Transactions on Information and Systems*, E94D(10), 2011.

[5] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: a view from the trenches. 2013.