Report on Expedia Hotel Ranking Competition

Dongqi Pu[2687343], Wouter Korteling[2526876], and Edeline Contempre[2636499]

Vrije Universiteit Amsterdam, DMT Group 84

Abstract. Expedia has provided a dataset that includes the information of user choices, user attributes, hotel attributes and competitors' attributes. The data is organized as an ordered list of hotels that the user sees after they search for a hotel on the Expedia website. The task is to explore the potential relationship between users search data and hotel recommendation list. Propose a reasonable hotel ranking recommendation list for a specific search query. Data processing includes feature generation, missing value processing, normalization and feature extraction. In the experimental process, we set XGBoost as basic model, and superimpose the models by using stacking method. The performance score of the model is 0.36008.

Keywords: Data Mining, Recommendation Systems, XGBoost, Stacking algorithm

1 Introduction

The recommendation algorithm is widely used in the Internet industry. Amazon, Spotify, Netflix, and other applications all have personalized recommendation functions. In the abstract, the recommendation algorithm is a fitting function for content satisfaction, which involves user characteristics and content characteristics. As the two major sources of the required dimensions for model training, click rate, page stay time, purchase, etc. can all be used as a quantified corresponding variable value, so that feature engineering can be carried out. A dataset is constructed, and then suitable supervised learning algorithms can be selected to train the models in order to recommend the preferred content for customers.

Expedia[1] is the world's largest online travel agency (OTA), providing search results to millions of travel shoppers every day. In this highly competitive market, matching users with hotel inventory is very important because users can easily jump from one website to another. Therefore, having the best hotel ranking ("sort") for a specific user and having the best price competitiveness integration will provide the best opportunity for OTAs to win sales. In this project, the main task is about how to provide a reasonable sorted list of hotel recommendations when a user performs a hotel search query.

2 Business understanding

Hotel recommendation competitions have taken place since many years. Expedia's platform created its own competition four years ago [1] where competitors

were asked to predict the likelihood of a user selecting a hotel among 100 different hotel groups.

3 Data understanding

In this part, we use the training set as an object to understand the characteristics of the data. Note that we cannot use the test set to understand its distribution, because this is a way of cheating. Because the attributes of the original data set can be divided into visitor attributes, property attributes, and search attributes, in the data set exploration part, we split the three different types of attributes in order to get a deeper insight. Specifically we selected *visitor_location_country_id*, *visitor_hist_starrating*, and *visitor_hist_adr_usd* as visitor attributes, then *prop_starrating*, *prop_review_score* and *price_usd* as property attributes, finally, *srch_length_of_stay*, *srch_booking_window*, and *srch_room_count* as search attributes.

3.1 Exploration

Dataset statistics: Expedia's datasets for the Kaggle competition consists of approximately 5 million records and 54 features. Among them, 4 attributes ('position', 'booking bool', 'click bool', and 'goss booking usd') are only available in the training set. Since our task is to predict which hotels are the most likely candidates to be booked, the book bool and click bool in the dataset will be used as corresponding variables.

In particular, the dataset contains relevant information about the user's search for hotels, such as the properties of the hotel and whether the user clicked and / or booked the hotel. Interestingly, about 95% of the search records have neither booked a hotel nor clicked on the details of the hotel. 31% of searches (an "srch id" indicates a search) did not book any hotels, and in case of users who book hotels successfully just need to click once (94% of the time), the probability of viewing only one hotel information. In other words, The data set is very unbalanced, and because it is a classification task, it may cause the model to not learn any knowledge, that is, the prediction is neither more likely to be booked nor clicked.

Data distributions: We draw pairplots to visually explore the relationship between pairs of the data features. The purpose is to explore whether there is a special relationship between data variables to provide some reference for our future component models.

Fig. 1, Fig. 2 and Fig. 3 show the data distribution and potential correlation of visitor attributes, property attributes, and search attributes, respectively. Overall, the distribution curve on the diagonal allows us to see the distribution of a single variable, while the scatter plot on the upper and lower triangles shows the relationship between the two variable pairs.

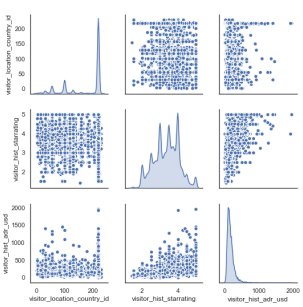

Fig. 1. visitor attributes distributions

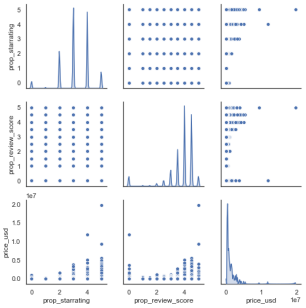

Fig. 2. property attributes distributions

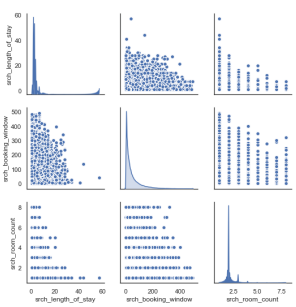

Fig. 3. search attributes distributions

It is worth noting that in Fig. 1, user history ratings for hotels show a normal distribution trend and there is a positive correlation between the average star rating of the hotel the customer previously purchased and the average nightly price of the hotel the customer previously purchased. In Fig. 3, There is a little positive correlation between booking window and length of stay.

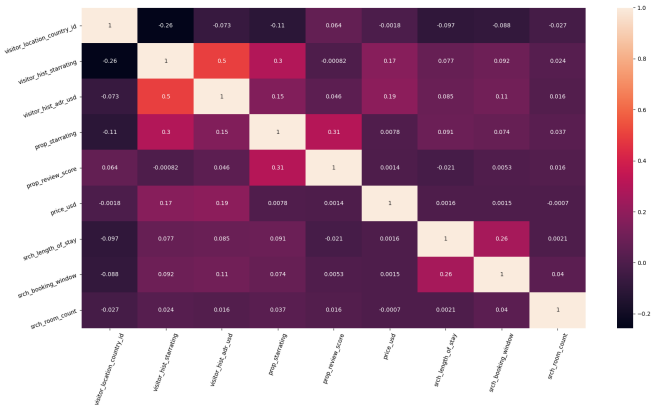

Fig. 4. Correlation coefficient heatmap

In the correlation coefficient heatmap diagram (Fig.4), it is also confirmed the previous judgment. The new finding is that there is a certain positive correlation between the star level of the hotel and the average star rating of consumers. In other words, better service will bring higher review.

3.2 Missing values

Fig. 5 shows that 31 attributes contain missing values. For this reason, different treatment schemes will need to set for the missing data for each attribute in the future. It should be noted that the missing values in $comp6\_rate$, $comp7\_rate\_percent\_diff$, $gross\_bookings\_usd$, $comp4\_rate\_percent\_diff$, $comp1\_inv$, $comp1\_rate$, $comp6\_rate\_percent\_diff$, $comp1\_rate\_percent\_diff$ are all greater than 95% and there are 17 attributes with missing values greater than 90%. For the data with a large proportion of missing values, it is difficult to provide effective information. We will consider deleting them or using other methods to deal with them.

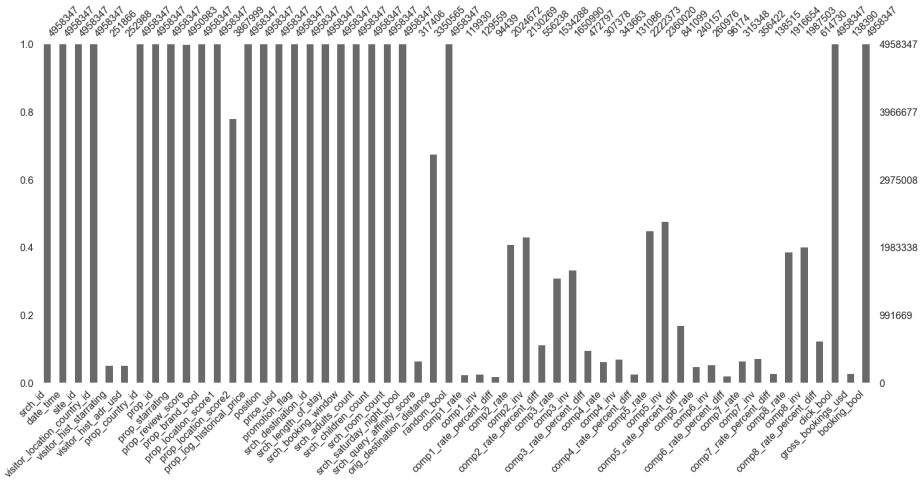

Fig. 5. Percentage of missing value for every attrubute

Fig. 6 gives us a deeper insight, it shows the correlation of missing values between every two attribute pairs. A value near -1 means if one variable appears then the other variable is very likely to be missing. A value near 0 means there is no dependence between the occurrence of missing values of two variables. A value near 1 means if one variable appears then the other variable is very likely to be present.

4 Data preparation

In the part, we introduce how to deal with missing values for different attributes, how to generate new attributes and how to select these attributes. Specifically, we use zero number, median number, and the 10% or 20% lowest numbers, XGBoost method to fill missing values.

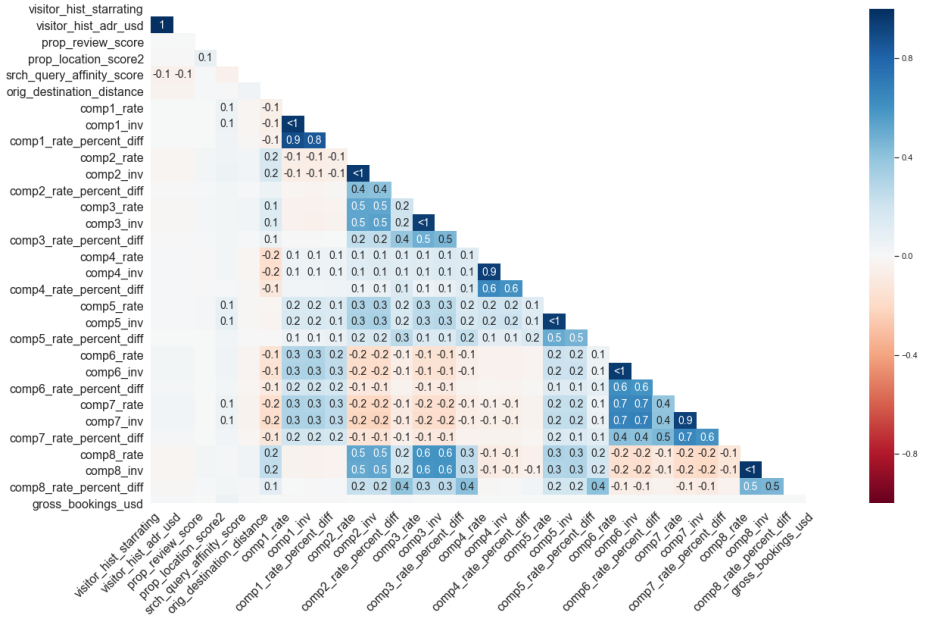

Fig. 6. Correlation coefficient heatmap of missing value

4.1 Handing missing data

Table 1. Missing value processing table

Features	Approach
comp_rate_{1-8}	Fill 0
comp_inv_{1-8}	Fill 0
$\text{comp\_rate\_perc}_{1-8}$	Fill 0
srch_query_affinity_score	The mean value of the lowest 10%
prop_review_score	The mean value of the lowest 20%
prop_location_score2	The mean value of the lowest 20%
orig_destination_distance	Mean by matching visitor_location_country_id
Others	XGBoost method

Table 1 presents how we handle the missing data. We assign the missing values of comp_rate_1 to comp_rate_8, comp_inv_1 to comp_inv_8 and comp_rate_perc_1 to comp_rate_perc_8 to 0. For srch_query_affinity_score, prop_review_score, prop_location_score2, we use the mean of the quantile to fill in the missing values. For orig_destination_distance, we fill in the mean of the distance by maching the same visitor_location_country_id.

The paper[2] introduces the handling of missing values of XGBoost. XGBoost treats missing values as sparse matrices, and the missing values are not considered when the node splits. The missing value will be divided into the left

sub-tree and the right sub-tree to calculate the layer loss, and choose the better one. If there is not missing data during training, but have missing data during testing, it is classified into the right sub-tree by default. So, for other attributes, we use XGBoost to deal with missing values.

4.2 Feature Engineering

Table 2. Feature Engineering

New Features	Approach
comp_rates	sum of comp_rate
comp_invs	sum of sum_inv
propclick	sum of click_bool / sum of prop_id
price_per_adult	price_usd / srch_adults_count
price_per_room	price_usd / srch_room_count
ranked_price	sorting based on price_usd
prop_score	prop_review_score + prop_starrating
price_per_star	price_per_adult / prop_starrating
price_per_night	price_per_adult / srch_length_of_stay
ranked_price_per_night	sorting based on price_per_night
price_for_one_person	price_usd / (srch_adults_count + srch_children_count) * srch_length_of_stay
bool_same_country	1: if visitor_location_country_id == prop_country_id, else 0
ratio_star_review	prop_review_score / prop_review_score
visitor_USD_diff	visitor_hist_adr_usd - price_per_night
visitor_star_diff	visitor_hist_starrating - prop_starrating
visitor_review_diff	visitor_hist_starrating - prop_review_score

To improve the quality and predictability of the model, new features are often created from existing variables. We can create some interactions (for example, multiplication or division) between each pair of variables, hoping to find an interesting new feature. However, this is a long process and requires a lot of coding. In feature engineering, we have created a total of 50 new features. The table 2 above shows how some features are generated.

4.3 Feature Selection

5 Modeling and Evaluation

5.1 Modeling

XGBoost algorithm XGBoost[3] is an integrated learning algorithm, which belongs to the category of Gradient Boosting Decision Tree (GBDT) model. The basic idea of GBDT is to let the new base model (GBDT uses the CART classification regression tree as the base model) to fit the deviation of the previous model, so the deviation of the addition model is reduced. Compared with the classic GBDT, XGBoost has made some improvements, so that the effect and performance have been significantly improved.

The general idea of XGBoost is to serially generate a series of CART regression trees, each regression tree is to fit the residual of the last tree and the target value, and this is repeated until the preset number of trees or convergence is reached.

Given a data set, n samples and m dimensional features, $D = \{(x_i, y_i)\}(|D| = n, x_i \in \mathrm{R}^m$. The output of the model is $\tilde{y} = \phi(x_i) = \sum_{k-1}^{K} f_k(x_i), f_k \in \mathrm{F}$

F represents the set of all decision trees, f_k is the k tree, $f_k(x_i)$ represents the k tree 's prediction result for the sample x_i.

Our goal is to optimize the following loss function:

$$L = \sum_{i=1}^{n} l(\tilde{y}_i, y_i) + \sum_{k} \Omega(f_k)$$

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda ||\omega||^2$$

γ,λ are hyperparameters; T is the number of leaf nodes in the decision tree, and ω is the value vector of the leaf node. Increasing γT is to prevent too many leaf nodes, which leads to over fitting of the model; Increasing $\frac{1}{2}\lambda||\omega||^2$ is to prevent the extreme value of the leaf node (to prevent a tree from learning too much, over fitting), l is a second-order derivative function, used to evaluate the difference between the predicted value and the true value.

The learning strategy of XGBoost is to learn the current optimal model. In this algorithm, learn the current optimal decision tree. That is to find the minimum value of the loss function of the following form (fine the best first $t-1$ round of the decision tree, and find the t optimal decision tree)

$$L(f_t) \simeq \sum_{i=1}^{n}[l(\tilde{y_i}^{(t-1)}, y_i) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \lambda||\omega||^2$$

$$= \sum_{i=1}^{n}[l(\tilde{y_i}^{(t-1)}, y_i) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \lambda||\omega||^2$$

$$= \sum_{j=1}^{T}[\sum_{x_i \in I_j}(g_i\omega_j + \frac{1}{2}h_i\omega_j)] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}\omega_j^2 + C$$

$$= \sum_{j=1}^{T}[G_j\omega_j + \frac{1}{2}(H_j + \lambda)\omega_j^2] + \gamma T + C$$

The second step uses Taylor's second-order expansion, $g_i = [\frac{\partial l(f, y_i)}{\partial f}], h_i = [\frac{\partial^2 l(f, y_i)}{\partial^2 f}], G_j = \sum_{x_i \in I_j} g_i, H_j = \sum_{x_i \in I_j} h_i$. If the structure of the tree has been determined, the value of the loss function is only relevant to ω_j. Set the derivative to 0 to minimize the loss function.

How to find the optimal decision tree? In constructing a decision tree, the most important thing is to find the optimal split point at each node. The selection

method of the optimal split point is different from the traditional CART selection method. The main idea is briefly that choose the split point where the loss function decreases the most after splitting. The cut point is the optimal cut point. Splitting in this way until the termination condition is reached, and the loss function of the resulting tree is minimal.

However, obtaining the optimal result is an NP-hard problem that cannot be solved in polynomial time, so we adopt a greedy algorithm to solve: from the root node, traverse all possible (feature, split), find the one that maximizes loss reduction.

Parameters of the algorithm used The parameters eta (learning_rate), min_child_weight (minimum weight of leaf nodes), max_depth (maximum depth of tree), max_leaf_nodes (maximum number of leaf nodes) and gamma (split gain threshold) can help control the generation of a single tree to prevent over fit or under fit. It is worth noting that adjust the scale_pos_weight parameter in XGBoost can help convergence problems caused by imbalanced data. This parameter needs to be adjusted because there are more negative examples than positive examples. The ratio is set to 19, and a typical calculation method is:

$$\frac{negative - examples}{positive - examples} = \frac{19}{5}$$

Stacking method In the process of building a machine learning model, Stacked Generalization[4] can effectively combines the results of multiple predictors, and then trains another model, as if it were a "stacking" model on the original model. The intuition is that different models can extract different information from the data. Due to data noise, different models tend to perform well on different characteristics of the data, but they also have poorer performance. Stacking can extract the parts of each model that have better features and discard the parts that are not performing well, which helps to effectively optimize the prediction results and improve the final prediction score.

Stacking integration is shown in Fig. 7, it consists of three XGBoost classifier. The top two XGBoost models select the first 30 attributes and the remaining attributes of the training set to prevent over-fitting. And their output is know as the meta features, at the same time, we also use these models to predict on test data to obtain the meta features for test dataset. Finally, XGBoost trains these meta features to obtain a model from meta features to ground-truth; then input the meta features of the test dataset into these models to obtain the final prediction results.

Parameters of the algorithm used Because the basic XGBoost model is an ideal model, we only send different parts of the dataset, as mentioned before we use mutually exclusive attributes, into the model so that the model can learn different aspects of the training data. The parameters of the base XGBoost model are consistent. For the final XGBoost model, the risk of overfitting is higher, so

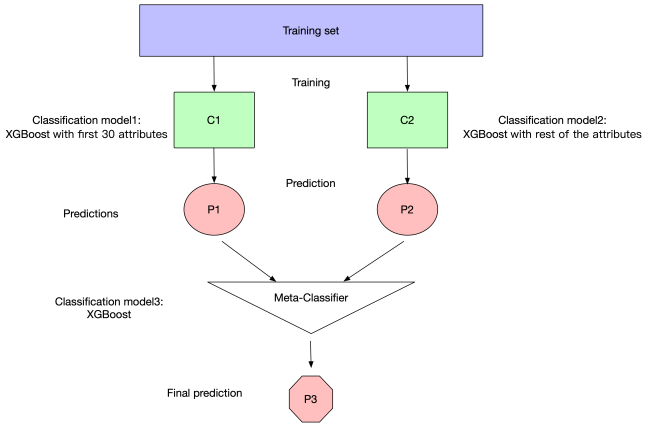

Fig. 7. The structure of the model

we reduce the maximum depth of the tree and increase the value of the learning rate.

5.2 Evaluation

Metrics According to the description of the competition, the calculation of the score is based on Normalized Discounted Cumulative Gain (NDCG)[5]. Compared with Discounted Cumulative Gain(DCG), NDCG can conduct a horizontal evaluation between different recommendation lists, because we cannot evaluate a recommendation system by only using one user 's recommendation list, but the results of the users in the entire test set.

The normalized discounted cumulative gain(NDCG) is derived as follows:

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2(i+1)}$$

Numerator: The greater the rel_i, the greater the correlation of the recommendation result i, the better the recommendation effect, and the greater the DCG.

Denominator: i indicates the position of the recommendation result. The larger i means the lower recommendation ranking in the recommendation list, the worse the recommendation effect, and the smaller the DCG.

Because the search results vary with the search ID, the number returned is inconsistent, and DCG is an accumulated value, it is cannot to compare two different search results, so it needs to be normalized.

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

IDCG is ideal DCG, refers to the list of best recommendation results returned by the recommendation system for a certain user. That is, assuming that the

returned results are sorted according to correlation, and the most relevant results are placed first. Therefore, the value of DCG is between $(0, IDCG]$, so the value of NDCG is between $(0, 1]$.

Cross validation We use the cross-validation method[6] to calculate the MSE[7] and NDCG[5] of the model in the validation set. In each iteration $i(i = 2)$, the training set is divided into k (k = 2) folds. The multi-fold results are averaged to obtain the corresponding evaluation index for each iteration to test the validation set.

6 Discussion and Conclusion

Data quality and data features determine the upper limit of Machine Learning, and models or algorithms just only try to approach this upper limit. To this end, we have done a lot of research on the features of the data, the purpose is to maximize the extraction of features from the original data for use by algorithms and models. In this part, we have discussed in detail in the feature extraction part. We rank features importance to select candidate features by using xgboost model. Generally speaking, importance scores measure the value of features in the model's promotion of decision tree construction. The more attributes are used to build a decision tree, the higher its importance. However, there are many ways to measure performance, and the choice of the number of candidate features is subjective. We have made a lot of efforts to select suitable parameters for this.

In this project, the response variable (Y) is very unbalanced, which means that the vast majority of search records do not book hotels. The XGBoost official document defines the parameter scale_pos_weight is to control the balance of positive and negative sample weights which can be used to process unbalanced sample classes, and help to solve the convergence problem caused by the imbalance of training data. In addition, although the sample of the data set is large, many attributes have missing values, and the proportion of missing values is very high, which reflecting the data quality is not so good. To this end, we use many methods to deal with missing values (as discussed before). In any case, the imbalance of the data and the handling of missing values are challenges that cannot be ignored on the performance of the model.

Time series is the key to improving the performance of the model. Since the XGBoost algorithm cannot handle the time series problem, this has become the main challenge to improve the competition score. Therefore, we should consider incorporating SLTM or RNN into the stacking method to improve the performance of our final model in the future.

References

1. "Personalize expedia hotel searches," https://www.kaggle.com/c/expedia-personalized-sort/overview.
2. D. Nielsen, "Tree boosting with xgboost-why does xgboost win" every" machine learning competition?" Master's thesis, NTNU, 2016.
3. T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.
4. D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.
5. A. Al-Maskari, M. Sanderson, and P. Clough, "The relationship between ir effectiveness measures and user satisfaction," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 2007, pp. 773–774.
6. M. W. Browne, "Cross-validation methods," *Journal of mathematical psychology*, vol. 44, no. 1, pp. 108–132, 2000.
7. D. Guo, S. Shamai, and S. Verdú, "Mutual information and minimum mean-square error in gaussian channels," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1261–1282, 2005.
8. "Expedia hotel recommendations — kaggle," https://www.kaggle.com/c/expedia-hotel-recommendations, (Accessed on 05/07/2020).
9. "2nd assignment dmt — kaggle," https://www.kaggle.com/c/2nd-assignment-dmt-2020/data, (Accessed on 05/07/2020).