Expedia Hotel Ranking Competition Scientific Report

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

Vrije Universiteit Amsterdam, DMT Group 84

Abstract. Expedia has provided a dataset that includes information about online user queries, user attributes, hotel attributes, attributes of competitors and finally user choices about which hotels were clicked on and booked after a query. The current report investigates the possibility of developing a model that can predict a user's behaviour such as likelihood of clicking on a hotel, booking a hotel using a specific query. The model therefore creates a reasonable ranked hotel recommendation list for its user. Data was initially preprocessed and new features were engineered to facilitate the algorithm's learning. In the experimental process, extreme gradient boosting classifier was used as the basic predictive model and was superimposed by using a stacking method. The resulting performance score (NDCG) of the final model on the test set was 0.36269.

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

1 Introduction

Expedia [1] is one of the world's largest online travel agencies and provides travel suggestions to millions of people every day. In a highly competitive online market, where users can jump from one website to another, it is crucial to instantaneously provide useful travel recommendations to users. Looking at hotel recommendations, a proper personalized hotel recommendation system that matches user queries to available hotels will provide a good opportunity for Expedia to increase sales. Many of the modern online recommendation systems, such as those used by Spotify, Amazon and Netflix are build on machine learning algorithms that autonomously learn to find associations between users, products and user choices. In this context, Expedia has provided a dataset with thousands of user queries and associated information related to user attributes, hotel attributes, attributes of competitors and user choices about clicked and booked hotels. Using this dataset, we want to investigate whether a self-learning algorithm can be used to predict hotel bookings for user queries. Specifically, we want to see if we can implement a supervised-learning algorithm that can learn to generate a list of hotel suggestions that is ordered based on their probability of being booked. In order to accomplish this, we will first explore other people's strategies in developing such predictive systems for the same (or similar) problems. We will then dive into Expedia's dataset and explore its characteristics. This will

result in the preprocessing of the data and the design of new features. Finally, we will implement, tune and test an XGBoost classification algorithm that aims at generating the sorted list of hotel suggestions for specific user queries.

2 Related work

In order to develop useful ideas for our solution-strategy, we will briefly review some relevant cases in which people have attempted to tackle similar prediction problems. Luckily, we found that some of the winners of the initial Kaggle competition about the Expedia dataset have shared a global outline of their strategy during the International Conference on Data Mining in 2013.[2]

Zhang[3], who was the competition's winner, used Gradient Boosting Machines (GBM) classification models[4] to predict the likelihood of the hotels being booked. Missing value imputation with negative values and bounding of the numerical variables where two of the important preprocessing steps that he applied. Also, in order to deal with the imbalance of the dataset, where the number of hotels that were not booked ("negatives") is far greater than the number of hotels being booked ("positives"), he downsampled negative instances. This possibly prevents the algorithm from always predicting a negative outcome. Further, he engineered about one hundred new features out of the original features and used different feature assets for a total of four different GBM models. Since every single GBM model has its own bias and variance, he combined the GBM ensemble by averaging over the different models and create the final predictions. The features that increased predictive performance the most where the expected hotel position (in the search results), the price and the desirability of the location.

Wang and Kalousis[5], who reached the second place, first point out that the ranking problem is non-linear and used LambdaMART as a non-linear ranking algorithm. Missing values were estimated based on other features, which possibly provides a better estimate than a naive imputation of, for example, the median value. This contrasts to the winning approach where missing values where imputed with negative values, which allows the algorithm to discriminate based on the missingness itself. Also, they tried to create features that could efficiently enable the tree-based algorithm to discriminate between positives and negatives by designing them in such a way that they monotonically relate to the target variables of clicking and booking. One important feature was the likelihood of the hotel being booked, which was defined by dividing the number of times it was booked by a user by the number of times it was shown to a user.

3 Data understanding

3.1 Data Statistics

Expedia's datasets for the Kaggle competition consist of approximately five million records with 54 features. The attributes 'position', 'booking bool', 'click

bool', and 'gross booking usd' are only available in the training set. The dataset is ordered in such a way that every user query has a 'search id' and contains all associated hotels that were displayed to the user as entries. About 31% of all searches did not result in any booking and in 94% of the searches that led to a booking, the user only clicked on the hotel that was booked. This suggests clicking and booking are strongly associated. Importantly, about 95% of the search results (i.e. entries) were neither clicked nor booked, This indicates a considerably imbalanced dataset which may challenge classification algorithms that aim to learn to discriminate between positives (booking) and negatives (no booking) [6].

31 attributes in the training set contain missing values (see Fig. 1). The proportion of missing values in 'comp6_rate', 'comp7_rate_percent_diff', 'comp1_inv', 'gross_bookings_usd', 'comp4_rate_percent_diff', 'comp6_rate_percent_diff', also 'comp1_rate' and 'comp1_rate_percent_diff' is greater than 95%. Moreover, 17 attributes in total have greater than 90% missing values. Due to this variance, several strategies may be appropriate to deal with missing values in different attributes during data preprocessing. For example, it may be hard to accurately impute values for attributes in which the proportion of missing data is substantive (e.g.>90%). Imputing missing values in these severe cases may introduce noise and thus obscure meaningful relationships within the data.

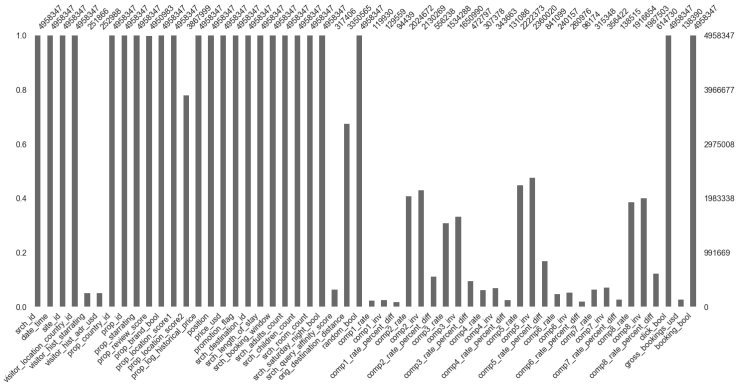

Fig. 1. Proportion of non-missing values for every attrubute.

Fig. 2 reveals the correlations between every two attribute pairs that contain missing data. A positive relationship indicates that the likelihood of a missing value in the one variable increases when the value in the other is missing. A negative correlation signifies the opposite relationship. We observe that missing information about hotel prices and availability of the hotel within a single competitor is strongly correlated, which makes sense. Also, missing information in the mean price and star-rating of previously booked hotels by the user are strongly correlated. This makes sense as this information is always present when

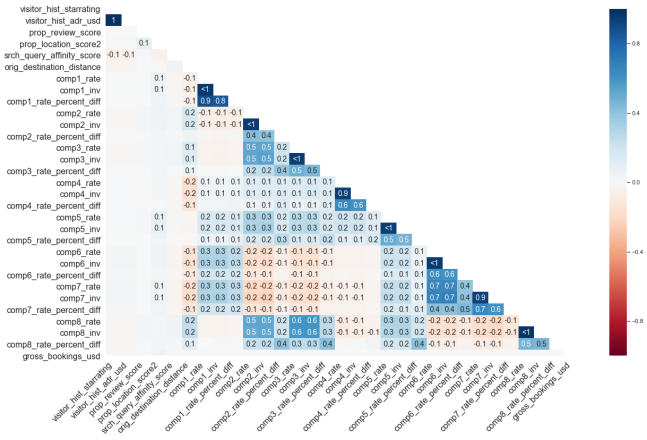

Fig. 2. Correlation coefficient heatmap of missing value

a user has previous bookings, and never present when he/she has no such booking history.

3.2 Data Distributions

Besides the obvious correlation between 'clicking' and 'booking' (Pearson's R = 0.78), the only other variables that show some correlation to the target variable 'booking' are the 'position' of the hotel in the search results (Pearson's R = -0.15) and the 'random_bool' that indicates whether the hotels were displayed in the regular sorted way or in random order (Pearson's R = -0.09). Naturally, hotels displayed on top of the page were more likely to be clicked and booked (see Fig. 3). Unfortunately, this variable is not available in the test set. We will see later if we can possibly estimate the hotel position in some way. Fig. 4 shows that when hotels are displayed in a random order, the search is less likely to result in any click or booking.

Looking at the other features, the fact that they do not show any linear correspondence to the target variable suggests that a linearly based model may not be adequate for the current problem. There may be, however, more complex interactions at stake. The features can roughly be divided into three categories: user characteristics, search characteristics and hotel characteristics, and it makes sense to keep these categories in mind when exploring their relationships. Fig. 5 shows pairplots of 'visitor_location_country_id', 'visitor_hist_starrating', and 'visitor_hist_adr_usd' as user features. The plots on the diagonal allow us to see the distribution of a single variable, while the scatter plots on the upper and lower triangles show the relationships with the other variables. Apparently, most users are from in a single country. Again, there is a positive relationship between the mean price and star-rating of the hotels users previously booked, hinting that these variables describe similar characteristics of the user. Fig. 6 shows pairplots

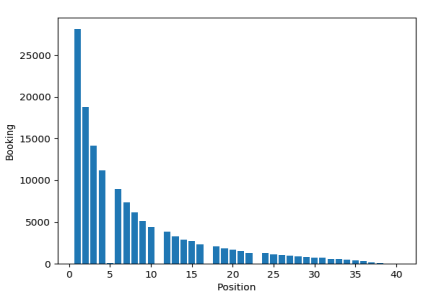

Fig. 3. Position on the search page.

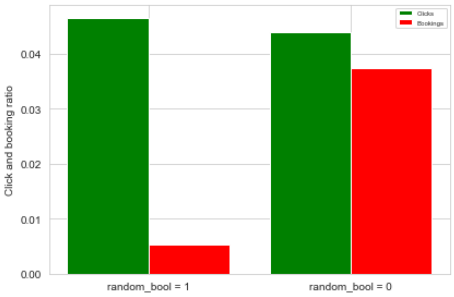

Fig. 4. Sorting.

between 'prop_starrating', 'prop_review_score' and 'price_usd' as hotel attributes. The distribution of the price of the hotels is skewed to the right and shows some extremely 'expensive' outliers with which we may have to deal with later on. Also, expensive hotels tend to have a higher rating. Fig. 7 shows 'srch_length_of_stay', 'srch_booking_window', and 'srch_room_count' as search attributes. Interestingly, room count is negatively associated with length of stay. There is a small negative correlation between booking window and length of stay. Looking at all variables, there does not seem to be any one-to-one relationships existent in the data, so it is quite possible that all features may eventually have a unique contribution to solving the classification problem.

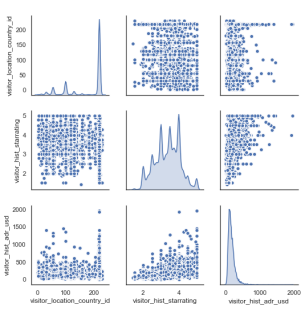

Fig. 5. visitor attributes distributions

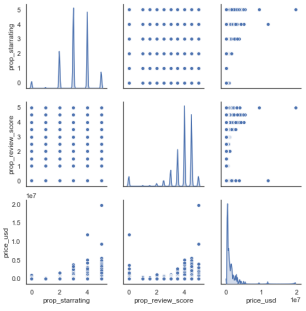

Fig. 6. property attributes distributions

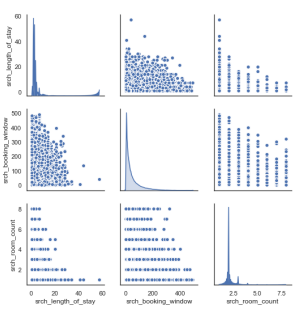

Fig. 7. search attributes distributions

4 Data preparation

4.1 Handling outliers and missing data

As discussed previously, there are a few numerical features with huge outliers, such as observed in the 'srch_length_of_stay'. Also, we observed quite some out-

Table 1. Missing value processing table

Features	Approach
$comp\_rate_{1-8}$	Fill 0
$comp\_inv_{1-8}$	Fill 0
$comp\_rate\_perc_{1-8}$	Fill 0
gross_bookings_usd	Remove
srch_query_affinity_score	The mean value of the lowest 20%
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
visitor_hist_starrating	XGBoost method
visitor_hist_adr_usd	XGBoost method

liers using Tukey's range criterium. However, in all cases we concluded that these outliers contained valuable information and existed in both train and test sets, so we left them intact.

When handling missing values, we do not just want to drop instances that contain missing values, as this would reduce the information in the dataset tremendously. Also, missingness itself may be a valuable asset for the algorithm when solving the classification problem. Table 1 presents how we handle all of the missing data. We fill the missing values within all the competitor variables ('rate', 'inv' and 'rate_percent_diff') of all competitors (1 to 8) with 0, which might be considered the "neutral" option (i.e. no differences between Expedia and competitor). Imputing 0 here also allows us to aggregate these features easier, which will be discussed later on. For 'srch_query_affinity_score', 'prop_review_score', 'prop_location_score2', we impute the mean of the first quartile. The reason to use this imputation method here is that we assume that missingness in these features is unattractive to the user. For 'orig_destination_distance', we fill in the mean of the distance based on the 'visitor_location_country_id', since we observed a correspondence between these variables.

XGBoost has an in-build method to deal with missing values.[7] It treats missing values as sparse matrices, and the missing values are not considered when the node is being split. The missing value will be divided into the left subtree and right sub-tree to calculate the layer loss, and the algorithm then chooses the optimal one based on this loss. We cannot think of any better way to deal with missingness in the 'vis_hist_rating' and 'vis_hist_adr_usd' variables, so we choose to use the XGBoost method here. Finally, 'gross_bookings_usd' contains more than 95% missingness, is not present in the test set and we would not know how we could put this variable to use, so we remove it from the database.

4.2 Feature Engineering

To help the model in finding interesting patterns in the data, we transform and design new features. One method we use here is designing "interaction fea-

tures", where features are in some way combined. For example, we calculate the probability of being clicked for every hotel by dividing the number of times it was shown by the number of times it was clicked. Another class of features that we engineered were "aggregates" of existing features. For example, for every entry we took the sum of the 'comp_rates' and 'comp_invs' features of all competitors to provide a feature that reflects the overall competitive pressure. The original competitor features were deleted to reduce complexity. Also, since we observed the hotel's position is important, we derived four features from the hotel's position, namely the min, max, mean and std of the positions of each unique property. In total, we created 53 new features, resulting in 89 features in total. Table 2 provides more detail on how some of them were generated. Since tree-based models are indifferent towards scaling, we did not perform standardization of the features. Finally, given that we want to provide a ranked list of hotels that is evaluated based on NDCG with both clicks and bookings having relevance (see Section 5), we merge clicks and books into one target variable: {book: 1, click: 1, no click or book: 0}.

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

4.3 Feature Selection

In order to prevent overfitting, we implement XGBoost's feature importance principle for feature selection. We use a 10-fold cross validation set up to get robust importance measures. The *import_type* parameter we use is *gain*, which is a measurement of the decrease in information entropy after the feature is used for division. To get more insight into feature importance, we also plot the F score of the top 20 features, which is a measure of divisive importance independent

from other features (see Fig. 8). The *gain* (averaged over the folds) revealed a similar order of importance and its distribution is shown in Fig. 9. Clearly, the probability of clicking is the most important feature. Features with *averagegain* >25 were selected to form a new dataset with 73 features in total.

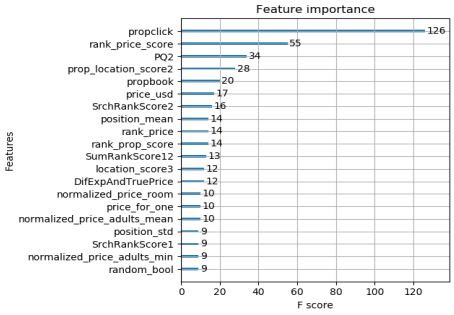

Fig. 8. Top 20 important features

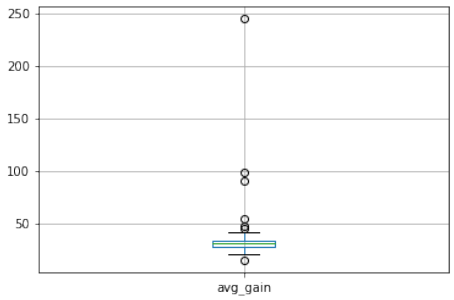

Fig. 9. Average gain for each feature

4.4 Balancing the dataset

As noted earlier, the dataset is considerably imbalanced, with the majority of the entries (95.5%) not containing any clicking or booking. This can provide problems to the classification algorithm, as the signal of the click/book class is relatively weak. In order to measure the effect of increasing the signal of clicks and books in the dataset, we create an additional dataset in which we perform random downsampling of the majority class (i.e. no clicks or books) to reach a 1:5 ratio. By comparing both set-ups, we can observe whether downsampling increases algorithm performance.

5 Modeling and Evaluation

5.1 Modeling

XGBoost algorithm XGBoost[4] belongs to the category of Gradient Boosting Decision Tree (GBDT) models that rely on the integration of multiple "weak" regression trees to boost a single overarching model. The basic idea of GBDT is to iteratively let a CART classification regression tree (as base model) fit the prediction residual of the previous model, which results in reduced error. This is repeated until the pre-set number of trees (or convergence) is reached. Compared with the classic GBDT, XGBoost has the advantage of parallel processing and in-build regularization, which makes it faster and more robust.[4] Besides these advantages, XGBoost has a weighing function to deal with imbalanced data, which matches our current problem given that we still have a binary imbalance

of 1:5. Also, the algorithm has proved to be useful for the current problem, as evinced by several of the winners of the initial Kaggle competition having used this model (e.g. see Zhang[3]).

The working of XGBoost is formalized as follows. Given a data set with n samples and m features, $D = \{(x_i, y_i)\}(|D| = n, x_i \in \mathrm{R}^m$. The output of the model is $\tilde{y} = \phi(x_i) = \sum_{k-1}^{K} f_k(x_i)$. Here, $f_k \in F$, where F represents the set of all decision trees and f_k the kth tree. $f_k(x_i)$ is the kth tree's prediction for sample x_i.

By finding optimal features to split within each branch and the optimal split point within each feature, the goal is to minimize the following loss function:

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

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

Here, l is a second-order derivative function, used to evaluate the difference between the predicted value and the true value; γ and λ are regularization parameters; T is the number of leaf nodes in the decision tree, and ω is the value vector of the leaf node. Increasing γ punishes the creation of extra leave nodes and increasing λ punishes extreme values in the leaf node. Therefore, both parameters can be used to prevent over fitting of the model.[4] Some of the other XGBoost parameters that are often used to regulate model complexity are *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). After we have outlined our stacking setup, we will explain how all these parameters are set.

Stacking method Stacking (or Stacked Generalization)[8] is the process of using multiple prediction models that all have a unique contribution to the final solution. Each individual model learns a portion of the entire problem space and feeds its prediction to the end model that integrates these sub-predictions to provide a final prediction. The idea is that different models can be optimized to extract different information from the data, and hence solve a portion of the total problem. The final model can integrate this information and learn the weaknesses and strengths of the predictions of the different models by comparing them to ground-truth. Fig. 10 shows our stacking set-up. We let a first XGBoost model learn to solve the categorization problem based on the first 30 features and a second model based on the remaining features. Their predictions on both train and test sets are then provided as meta features to the final model that learns based on the total set of features. In order to measure the effect of stacking, we will also use a regular setup in which we train a single XGBoost model on the entire feature set.

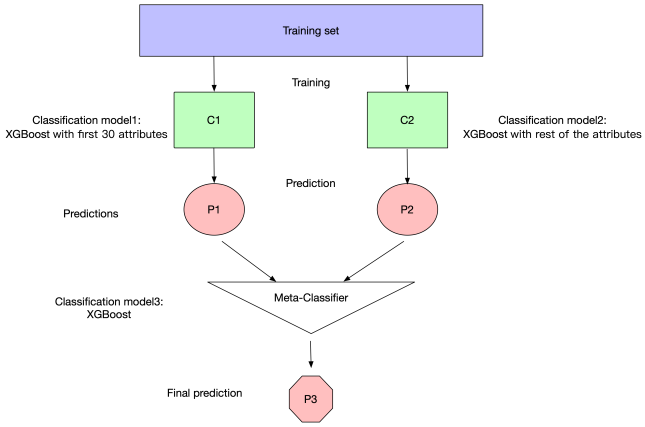

Fig. 10. Our stacking setup

Parameter settings We implement XGBoost using xgboost open source library. As mentioned before, there are several XGBoost parameters that determine model complexity and can therefore be used to obtain a balance between fitting and overfittig. Grid search over parameter space takes too much time and we will therefore manually adjust these parameters. Our stacking setup contains two base models and a final model. For the two base models we use $gamma = 0$, $eta = 0.3$, and $max\_depth = 6$. On the one hand, the final model receives input from the other models, which means that some of the problem has already been solved. On the other hand, parameter space is bigger and there is 'more to learn'. Therefore, to prevent overfitting we increase regularization using $gamma = 2$, but allow for complexity by relaxing constraints using $max\_depth = 8$ and $eta = 0.2$. The other parameters were left to default settings for all models. However, given that we still have imbalanced data (1:5), we use XGBoost's weighing function with $scale\_pos\_weight = 5$.

5.2 Evaluation

Metrics According to the description of the competition, the calculation of the score is based on Normalized Discounted Cumulative Gain (NDCG)[9]. Given that our predictions are being tested by this score, we want to use it to evaluate and tune our algorithm set up. We will briefly explain how NDCG is derived.

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

Numerator: rel_i is the actual relevance of the entry and i is its rank given by the algorithm. The better these match (i.e. entries with high relevance are ranked high by the algorithm) the greater the total DCG. NDCG is a variant of DCG that normalizes DCG by deviding it by the Ideal DCG, which is the score that would be obtained when the algorithm's ranking is perfect.

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

For the current problem, bookings have $rel = 5$, clicks (but not booked) $rel = 1$ and no clicks or bookings $rel = 0$. Therefore, we want to rank booked hotels highest, followed by clicked but not booked hotels, and others lowest.

Cross validation We use a k-folds cross validation setup[10] with $k = 5$, to calculate the MSE[11] and NDCG[9] of the model with respect to the validation set. This method allows for estimating generalization performance of the models and ensures that all datapoints are included in the test set once. The multi-fold results are averaged to obtain the final scores.

Results As discussed previously, we have developed different setups that can be compared to see which setup works best for the current problem. Cross validation revealed that our stacking method in combination with downsampling of the majority class, but without feature selection, provided the best results on both MSE and NDCG criteria. Especially downsampling proved to increase algorithm performance significantly. Using downsampling in combination with stacking resulted in a mean NDCG of 0.631 compared to 0.601 without downsampling. Downsampling without stacking decreased the mean NDCG to 0.572, indicating that the stacking setup increases model performance as well. Table 3 summarizes the MSE and NDCG scores of the final model.

Table 3. Results of Downsampling + Stacking setup

	Mean score	Best score	Worst score
MSE	0.045	0.045	0.045
NDCG	0.631	0.631	0.63

6 Discussion and Conclusion

Data quality and data features determine the upper limit of any machine learning algorithm. To this end, we have done a lot of research on the features of the data, such that we could prepare the dataset optimally for the XGBoost models. We have applied a variety of methods to deal with missing values in the dataset and created various features by combining or aggregating the original ones. Also, we have attempted to create a better balance in the dataset by downsampling the overrepresented class (no clicks or books), which proved to provide a significant contribution to model performance. Our stacking method slightly improved model performance as well. Interestingly, feature selection did not improve model performance, which possibly suggests that all features had a positive contribution to the predictive performance of the model.

There is definitely space left to improve our predictions. First, we could engineer more features. For example, normalizing more of the hotel features on the id of the search might be rewarding, since this reflects the relative characteristics of the hotels within the search. Principal Component Analysis (PCA) on the different feature categories might reveal relevant underlying factors in the dataset as well. Also, there may be dependencies between adjacent searches (i.e. time effects). These could exist due to the fact that the same user might search multiple times in a row. By, for example, defining a Recurrent Neural Network and providing its predictions as features, this time-dependency may be utilized. Second, using more advanced techniques to create a balanced dataset might be beneficial. An oversampling technique, such as Synthetic Minority Over-sampling (SMOTE)[12], would be a suitable option. By applying oversampling in combination with downsampling, instead of downsampling only, we could retain more entries of the overrepresented class and still reach a similar end-ratio. Hence, we would have to remove less valuable information. Third, we used a simple (naive) heuristic for assigning features to the base models in the stacking setup, namely the first 30 features versus the last features. Stacking is most effective when the different models learn based on the most distinctive feature sets, since this allows for learning unique properties of the dataset. By categorizing the features based on their meaning (e.g. user features vs hotel features), our stacking setup would have been more effective.

7 What have we learned?

The current project was a nice opportunity for us to get our hands dirty with a considerably huge and challenging dataset. When dealing with a small dataset with few features, mere trial and error suffices to get a near optimal result. With the current dataset, however, the number of options to be made increased vastly while computation and memory constraints required a prior consideration about what to do. For example, grid-search over parameters space turned out to take too much time. Instead, we elaborated on how the different parameters affected the model and based on that decided how to tune them optimally.

Another learning experience was the generation of new features. We feel that we have developed a much better intuition about how new features can be generated out of the original ones by, for example, combining them in clever ways or aggregating them.

Also, as we stated in our discussion, the quality of the dataset itself determines the upper limit of any model's performance. We noticed this firsthand when performance increased substantially in some cases after we had added just a few more features to the dataset. We observed the same when we downsampled the over-represented class to counter-act the imbalance.

Finally, we learned that thinking about our strategy as a team really makes a difference in such a time-constrained project. We noted that the division of tasks such that we can work in parallel instead of sequentially, and still provide quality, is crucial.

References

1. "Personalize expedia hotel searches," https://www.kaggle.com/c/expedia-personalized-sort/overview.
2. A. Woznica, J. Krasnodebski, and D. Friedman, "Personalized expedia hotel searches," *International Conference on Data Mining*, 2013.
3. O. Zhang, "Personalized expedia hotel searches, 1st place," *International Conference on Data Mining*, 2013.
4. 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.
5. J. Wang and A. Kalousis, "Personalized expedia hotel searches, 2nd place," *International Conference on Data Mining*, 2013.
6. B. Krawczyk, "Learning from imbalanced data: open challenges and future directions." *Progession in Artificial Intelligence*, vol. 5, no. 2, pp. 221–232, 2016.
7. D. Nielsen, "Tree boosting with xgboost-why does xgboost win" every" machine learning competition?" Master's thesis, NTNU, 2016.
8. D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.
9. 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.
10. M. W. Browne, "Cross-validation methods," *Journal of mathematical psychology*, vol. 44, no. 1, pp. 108–132, 2000.
11. 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.
12. N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "Smote: Synthetic minority over-sampling technique." *Progession in Artificial Intelligence*, vol. 5, no. 2, pp. 221–232, 2016.
13. "Expedia hotel recommendations winning approaches— kaggle," https://www.kaggle.com/c/expedia-personalized-sort/discussion/6203latest-283647, 2013, (Accessed on 28/05/2020).
14. X. Liu, B. Xu, and Y. Zhang, "Personalized expedia hotel searches, 4th place," *International Conference on Data Mining*, 2013.
15. "Expedia hotel recommendations — kaggle," https://www.kaggle.com/c/expedia-hotel-recommendations, (Accessed on 05/07/2020).
16. "2nd assignment dmt — kaggle," https://www.kaggle.com/c/2nd-assignment-dmt-2020/data, (Accessed on 05/07/2020).
17. Á. García-Crespo, J. L. López-Cuadrado, R. Colomo-Palacios, I. González-Carrasco, and B. Ruiz-Mezcua, "Sem-fit: A semantic based expert system to provide recommendations in the tourism domain," *Expert systems with applications*, vol. 38, no. 10, pp. 13 310–13 319, 2011.
18. S.-Y. Hwang, C.-Y. Lai, J.-J. Jiang, and S. Chang, "The identification of noteworthy hotel reviews for hotel management," *Pacific Asia Journal of the Association for Information Systems*, vol. 6, no. 4, p. 1, 2014.
19. K.-P. Lin, C.-Y. Lai, P.-C. Chen, and S.-Y. Hwang, "Personalized hotel recommendation using text mining and mobile browsing tracking," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2015, pp. 191–196.

20. N. Rianthong, A. Dumrongsiri, and Y. Kohda, "Improving the multidimensional sequencing of hotel rooms on an online travel agency web site," *Electronic Commerce Research and Applications*, vol. 17, pp. 74–86, 2016.
21. Y. Sharma, J. Bhatt, and R. Magon, "A multi-criteria review-based hotel recommendation system," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. IEEE, 2015, pp. 687–691.
22. Z. Chang, M. S. Arefin, and Y. Morimoto, "Hotel recommendation based on surrounding environments," in *2013 Second IIAI International Conference on Advanced Applied Informatics*. IEEE, 2013, pp. 330–336.
23. D. Jannach, F. Gedikli, Z. Karakaya, O. Juwig *et al.*, "Recommending hotels based on multi-dimensional customer ratings," in *ENTER*, 2012, pp. 320–331.