Data Mining 2nd Assignment

Ivo van Miert[2634569], Esteban Olivero Lopera[2614283], and Carol Rameder[2747982]

Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081HV Amsterdam, The Netherlands

Abstract. In this report we discuss the performance of Random Forest Classifier (Sklearn) and XGBoost models on a Kaggle competition for the course Data Mining Techniques. The competition is based on the 2013 Kaggle competition from Expedia on hotel recommendations. We found that XGBoost performs much better than Random Forest in terms of efficacy and efficiency, as it produced better scores and required less resources when training the model.

Keywords: Data analysis · Feature engineering · Random Forest · Random Search · XGBoost

1 Introduction

A massive amount of data is generated every day, which makes the possession of data increasingly precious. Actually, data became one of the most important resources nowadays, as a massive amount of people get access to the internet almost everywhere around the globe. Data mining applies Machine Learning techniques to exploit this resource by finding meaning and observing new patterns in it. As a result, big companies saw the opportunity to use increase their productivity, efficiency and customer experience by adopting data mining techniques in their daily routine. [3]

In our case, Expedia, one of the world's largest online travel companies, wants to increase their sales by creating a recommender system. This improves the user experience and the likelihood of buying by sorting the results of a search according to the probability of the property being booked. To achieve this goal, the company created a public competition with prizes for those who find the most accurate method to sort the properties, based on previous search metadata. Kaggle is the online platform open to a large public of coders and data scientists, that organized the competition by centralising, evaluating and ranking the solutions submitted by the teams.

In this paper, we describe our process step-by-step to create such a sorting method. Firstly, in Chapter 2, the analysis of the dataset is made by creating statistical graphics and visualisation methods in order to get new insights about the values and to check assumptions and hypotheses, that will indicate the right way of making further decisions. Chapter 3 presents the feature engineering process, in which raw input data is transformed into features, that can be used by the supervised machine learning (ML) model. It consists of creating and selecting new features and modifying data to make the learning process more efficient. Chapters 4 and 5 show implementation details of two ML models, that we chose to compare: Random Forest and eXtreme Gradient Boosting (XGBoost). Finally, Chapter 6 offers the conclusion of comparing and choosing one of the models.

1.1 Related work

The competition held originally place in 2013, and had a prize pool of 25.000 dollars, which resulted in a lot of data scientists enrolling into the competition and try to write the code that would get the highest score. Also after the competition closed, a lot of people have tried the challenge and an impressive amount of work on it can be found online. To get a clear idea of what different kind of approaches were possible to implement and which features were most important, we studied reports and code of various old participants of the challenge.

For example, we looked at how Liu et al. (2013) [6] set up their research. They explained the steps they took during the pre-processing, for example, the creation of composite features, which made us realise what possibilities there were for creating our own composite features, which will be described in Section 3 Feature Engineering. In addition, we read in their report which features were most relevant, so we knew which features were useful to use when creating

composite features. Liu et al. also described the different types of models they used. They used logistic regression, pairwise logistic regression, random forest and gradient boosting machine, among others.

Ma and Yao's study [2] showed us that one of the first problems we would encounter would be the presence of the many NA values in the dataset. They also gave us three different ways to solve for this NA value, depending on which column it was in. They did this by either assign a 0, assign the minimum value of the column or assign the average value of that column to that cell.

We also saw in several studies on the challenge that it was wise to use only part of the dataset by random sampling and not the whole dataset, to save time during the training of the model.

2 Exploratory data analysis

Firstly, we noticed that there is a high correlation between people who clicked on a property and our independent variable ('booking_bool') (Fig. 3). Furthermore, a correlation analysis showed that 55 of the variables had a significant Pearson correlation score ($alpha = 0.05$) with our outcome variable ('booking_bool") (Fig 1). For this reason, we decided to keep most of the variables for our model and instead of selecting a few, we transformed some of the variables into data that could be used to fit our models. These transformations are explained in the feature engineering section of the report. We ran some exploratory analysis on the variables 'season', 'month', and 'visitor_historic_rating' in order to understand if these variables had any effect on the booking likelihood of a person. Overall, we noticed that whether the person has a historical rating on Expedia or not, does not influence much if the person will book a property (Fig 3). However, we did notice that people with historical data have higher booking rates than people with missing values. For the time variables (month and season), we noticed that separating the data by season was more influential than separating it by month (Figs 4 to 7). However, it was also interesting to notice that we only have data for the first 6 months of the year, as well as November and December. Finally, we excluded the variables 'booking_bool', 'click_bool', 'gross_bookings_usd', and 'position' from the training dataset as none of these variables are present in the test dataset. We still decided to use the variable 'click_bool' for our EDA as it can be used as a validation variable to confirm the results from the comparisons to our independent variable.

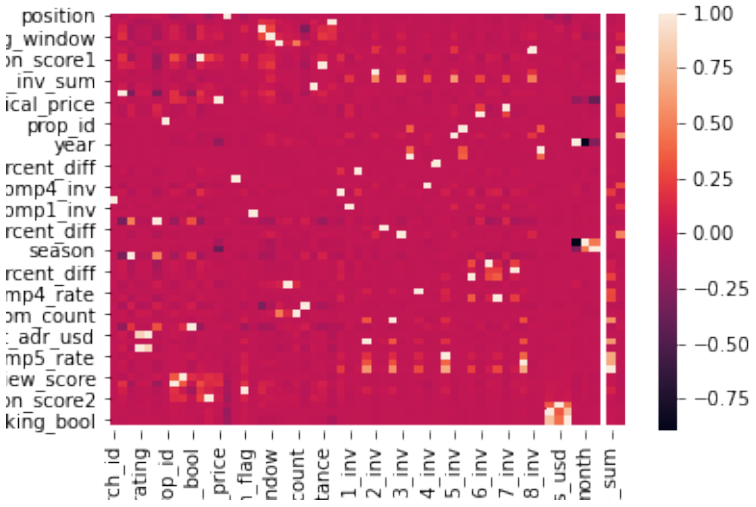

Fig. 1. Correlation of variables heatmap

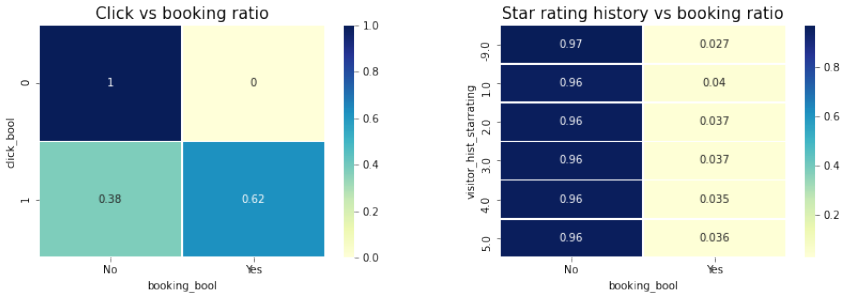

Fig. 2. Click vs booking heatmap

Fig. 3. Star rating history vs booking

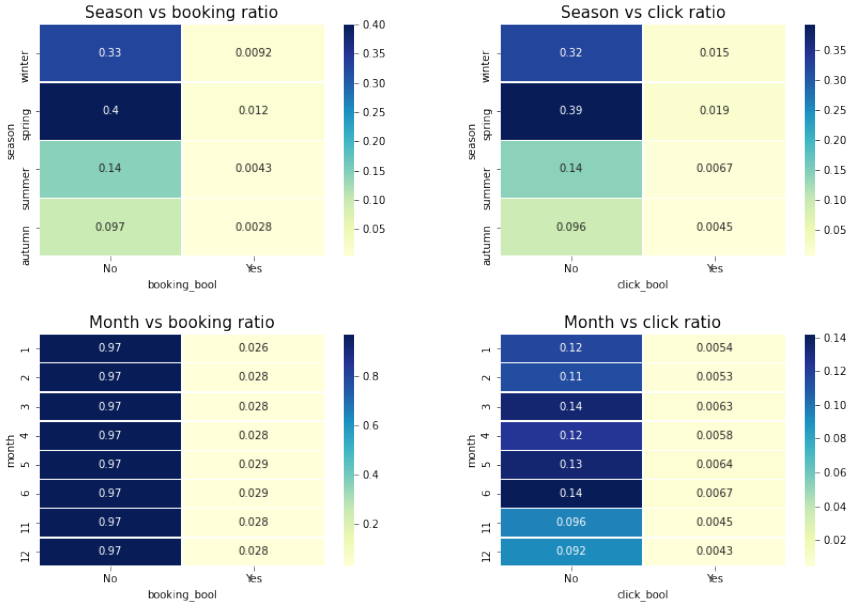

Fig. 4. Season vs booking heatmap

Fig. 5. Season vs click

Fig. 6. Month vs booking

Fig. 7. Month vs click

3 Feature Engineering

Dataset pre-processing: report a replicable process of querying and/or feature engineering, and provide a rationale for the choice of such (in)dependent variables: analysis / rational

We tried adding the "count_better" feature to speed up the training process, by directly indicating to the model the number of competitors, which have better offers for the same property, in the same timespan. However, this addition did not bring any significant improvement to the efficiency. The most logical reasoning for that is the low number of non-zero values added in this column, with a rate around 0.1

Before modifying certain variables, we firstly filled some of the missing values in order to continue with the feature engineering. Firstly, we aggregated the competitor's scores into a single variable called comp_inv_sum. For this transformation, we filled all the missing values with a 0 as indicated by the description of the data in the original K competition. Furthermore we proceeded to aggregate these values in order to have a single numerical value that represented the overall competitors scores. The second transformation we did was to divide the date_time variable into sub-components. We made this decision as neither Random Forest classifier nor XGBoost could handle data in a date time format. From this variable, we split it into the variables 'year' and 'month' and then proceeded to also add a new variable 'season' calculated in the month and location in which the purchase was made. This new variables has 4 levels (1,2,3,4) representing the 4 seasons of the year.

For the remaining data we filled the missing values with either a 0, a mean score or an arbitrary number. For the variables 'prop_review_score' we filled the missing data with the median value (3). For the 'prop_location_score' and the 'visitor_hist_adr_usd' we filled missing values with a 0. Finally, for 'prop-srch_query_affinity_score', we filled the missing values with the mean affinity score for each search ID. For the remaining variables, we filled the missing values with a score of -9 in order to assign a value and not miss too much data. We decided to not drop the missing values as many participants had missing values in a rather substantial part of their data. When testing the models, training on data where the missing values were dropped reduced the accuracy of the models on the testing data.

4 Random Forest

The Random Forrest Classifier is a supervised Machine Learning algorithm, that we used to predict if a search result received after a query made by the user will either be booked or not. This method leverages the power of multiple, less complex decision trees classifiers, which are independent to each other. In this case, each tree consists of a series of sequential decisions, where each node represents a numerical attribute of the search that ultimately lead to the label choice located in leaf nodes. The trees are top-down generated by selecting a subset of attribute, from which one is randomly chosen and selected for model. When deciding the attributes and the internal splits, the dataset is filtered for each branch, and only data instances with values respecting the criteria above are kept. Each line entry of the data is top-down fed through the decision rules of the tree and a label is resulted for each one of them.

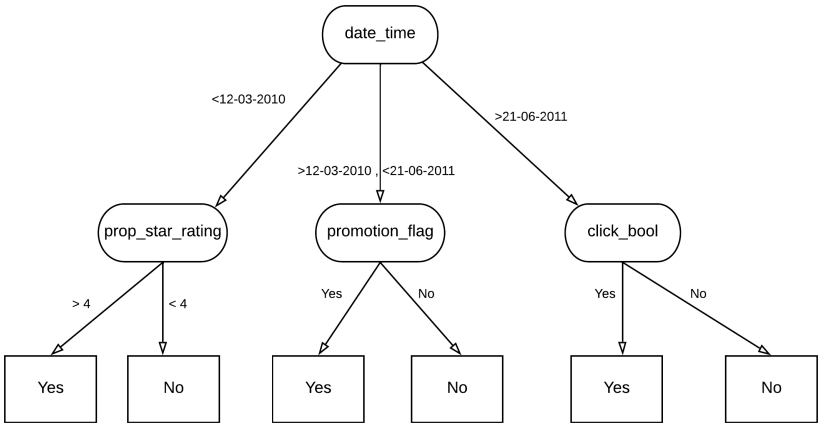

Fig. 8. Decision tree structure

To determine the label of the whole set of trees generated by the Random Forest model, each tree casts a unit vote for its outcome class and the majority wins, becoming the overall output.

4.1 Random Search for Hyper-parameter optimisation

The Hyperparameters of the Random Forest are set before the training process and their configuration directly impacts the performance and the structure of the model. We are looking for the best configuration to optimize the accuracy with the respect to testing data, in a reasonable short computation time. More precisely, these are the possible values for each one of them:

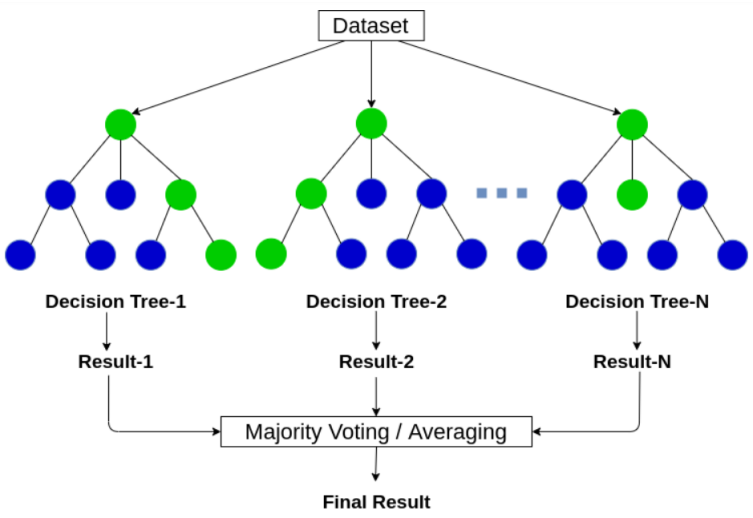

Fig. 9. Random Forest[4]

- n_estimators - The number of trees generated for the forest, a numerical value between 90 and 250
- max_features - the maximum number of features taken into consideration when randomly choosing a new node. The options are *Auto*, for the number of all features in the dataset, *sqrt*.
- max_depth - the maximum size allowed for the path from the root to any of the leaves - a numerical value between 10 and 110
- min_samples_split - The minimum number of data entries required to split an internal node - [1,2,4]
- min_samples_leaf - The minimum number of data entries required to be at a leaf node - [2,5,10]
- bootstrap - Whether bootstrap samples are used when building trees - *True* or *False*

When gathering more data and feature engineering do not bring any improvement to the results or become not feasible anymore, hyperparamter optimisation is the method, that still can improve the efficiency of the model. Random Search is a trial-and-error based engineering procedure, which exhaustively tries all the hyperparamter combinations defined. It compares the outcome of the models, each one generated with a different configuration of hyperparameters, and outputs the best values. It is an computational expensive method, that relies more on experimental results and tries to leverage learning time and performance. [5]

We used scikit-learn library to work with this model. The final sort of the properties is made by getting the probability of each one being booked (having label=1), with a predict_proba function from the same library.

5 XGBoost

The second model used to predict if a search result would lead to a booking is the XGBoost algorithm, which implements the gradient boosting decision tree algorithm. XGBoost, short for Extreme Gradient Boosting, belongs to a family of boosting algorithms and uses the gradient boosting framework as its core. XGBoost is a software library that is mainly focused on computational speed and model performance.

Boosting is a sequential technique which works on the principle of an ensemble. It combines a set of weak learners and delivers improved prediction accuracy. The basic idea behind boosting algorithms is building a weak model, making conclusions about the various feature importance and parameters, and then using those conclusions to build a new, stronger model in which the misclassification error of the previous model is reduced. Gradient boosting is a certain way of boosting and is one of the most powerful techniques for building predictive models. In gradient boosting, the loss function is minimized by adding weak learners, in an iterative and sequential process, using a gradient descent optimization algorithm.

5.1 XGBoost's hyperparameters

As with the random forest algorithm, the hyperparameters of the XGBoost algorithm are set before the training process and their configuration impacts the performance and structure of the model. The hyperparameters that were needed to be set are:

- objective - determines the loss function to be used like reg:linear for regression problems, reg:logistic for classification problems with only decision, binary:logistic for classification problems with probability. Since our problem is a classification problem with probability, we chose the latter.
- colsample bytree - which is the percentage of samples used per tree. A low value can lead to underfitting, a high value to overfitting. In our algorithm, we used 0,3, as this was recommended in their article by Chen et al (2016)[1]
- learning rate - step size shrinkage used to prevent overfitting. In our algorithm we chose a learning rate of 0.1
- max depth - determines how deeply each tree is allowed to grow during any boosting round. In our algorithm we chose for a max depth of 5.
- n estimators - The number of trees you want to build. We chose n = 150 for our final model. We also tried n = 100, but got a slight worse prediction score from this change

6 Conclusions

After repeating several trials of XGBoost and Random forest Classifier with similar parameters, we noticed that XGBoost always performed better when testing the score on the Kaggle competition. For both models, we trained the models with the entirety of the training data set. For our final submission with the XGBoost, we obtained a score of 0.36882, whereas with the Random Forest Classifier we got a score of 0.33567. Furthermore, we also noticed that altering the hyper-parameters on XGBoost had a smaller inpact on the score we could obtain. Training the model with the same parameters but changing the number of estimators, lowered the score to 0.36209. Making the same change on the Random Forest model, the score decreased to 0.30787. These results indicate that XGBoost is more likely to yield better recommendations with similar parameters. Furthermore, the resilience to changes that we observed also makes us believe that it would be possible to obtain better results with XGBoost while using less computing power. This would be the case as we can obtain a significant improvement on the Kaggle results by using less estimators than with the Random Forest classifier.

6.1 What we learned ?

Participating in this Kaggle competition was a hands-on experience with the steps of the general process in Data Mining, as we gained both practical and theoretical knowledge on this topic. During the Exploratory Data Analysis part, we learned how to get familiar with data that we have to work with, and how to select the right statistical tools to get as much relevant information. We discovered the correlation between features, the distribution of values and the proportion of missing data. In the feature engineering step, we got knowledge on how to optimise the learning process by adding new features and how to determine the parameters that are kept for the ML models mainly with trial and error. Moreover, by searching for ML models and making them work with our features we got insights into how the most common ML models work and for which task they are generally more suitable and also the role of tuning the hyperparameters. An unexpected outcome that we got on this step was that Random Search does not bring a significant improvement when the run time available is not large enough or the training data set is too big.

Finally, it is important to mention the practical side of this task which acquires knowledge of the notebook style of coding in Python and the suitable libraries and functions to work with data frames and ML models.

References

[1] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: (Aug. 2016), pp. 785–794. DOI: 10.1145/2939672.2939785.

[2] $cs249_Parker_Proj1$. https://github.com/shawnhero/ICDM2013-Expedia-Recommendation-System. Accessed: 2022-05-05.

[3] Thomas H Davenport and Jill Dyché. "Big data in big companies". In: *International Institute for Analytics* 3.1-31 (2013).

[4] *Decision Tree vs. Random Forest – Which Algorithm Should you Use?* https://bit.ly/3MMrgy1. Accessed: 2022-05-25.

[5] *Hyperparameter Tuning the Random Forest in Python.* https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74. Accessed: 2022-05-20.

[6] Xudong Liu et al. "Combination of Diverse Ranking Models for Personalized Expedia Hotel Searches". In: (Nov. 2013).