Hotel Booking Cancellation Prediction Using Machine Learning Classification Models

by

Ramin Ziaei

Final Report

Under the Supervision of Dr. Cheng

Department of Mathematics

University of Nebraska - Omaha

Omaha, Nebraska

December 17^{th}, 2020

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Significance

Hotel booking cancellation has significant impact on the demand-management decision in the hospitality industry. Although hotels usually put strict policies regarding cancellation to mitigate their negative effects, the same policies can bring a bad reputation for the hotel as well as a negative impact on the revenue. According to a study conducted by D-Edge Hospitality solutions in 2019 [1], more that 40% of the booked hotel reservations got canceled before arrival in Europe. The company analyzed the online distribution performance of more than 200 different channels for 680 properties in Europe between 2014 and 2018. As table 1 shows, the rate of cancellation rose from 32.9% in 2014 to 39.6% in 2018 with a high of 41.3% in 2017. They reported that due to free cancellation policies that many hotels offer, people have got used to canceling their bookings more often. This particular behavior will eventually cause a loss in gross revenue due to the fact that hotels managers cannot accurately forecast their need and supply anymore. Kimes and Wirtz (2003) defined the revenue management as "the application of information systems and pricing strategies to allocate the right capacity to the right customer at the right price at the right time"[2]. In the field of hospitality, this definition can be defined as "making the right room available for the right guest and the right price at the right time via the right distribution channel"[3]. That is why in today's hospitality industry, machine learning models are used to predict which reservations are "likely to cancel" to reduce the negative impact of possible cancellations.

Table 1: Yearly average percentage of cancelled bookings in Europe[1]

	2014	2015	2016	2017	2018	Change
Booking Group	43.4%	43.8%	48.2%	50.9%	49.8%	6.4%
Expedia Group	20.0%	25.0%	25.8%	24.7%	26.1%	6.1%
Hotelbeds Group	33.2%	37.8%	40.3%	38.3%	37.6%	4.4%
HRS Group	58.5%	51.7%	55.2%	59.4%	66.0%	7.5%
Other OTAs	13.7%	15.2%	27.0%	24.4%	24.3%	10.6%
Other Wholesalers	31.2%	30.3%	34.6%	33.8%	32.8%	1.6%
Website Direct	15.4%	17.7%	18.0%	18.4%	18.2%	2.8%
Average	**32.5%**	**34.8%**	**39.6%**	**41.3%**	**39.6%**	**7.1%**

1.2 Research questions

In this work, I will try to make different classification models to predict the chance of a reservation getting canceled. This is the method that many hotels around the world have started to adopt. There are 3 main goals that I try to achieve by the end of this project:

- Identifying the features that contribute the most to a possible booking cancellation.
- Building a predictive model capable of classifying guests with high cancellation probability.
- Understanding if one single model can be applied to different hotels.

2 Data

2.1 Data collection

The data was acquired from a weekly data project in github called TidyTuesday which comes from a journal paper [4]. This dataset contains information of 2 different hotels, one being a resort hotel (H1) with 40,060 observations and the other one being a city hotel (H2) with 79,330. Both data sets share the same structure with 31 predictors. From these predictors, 13 are characters, 1 is date and the rest are numeric.

2.2 Data overview and variable introduction

Table 2 shows all the predictors, their classes and what each one represents. According to this table, there are 32 columns in this dataset (31 predictors and 1 response which is bolded). So, in the original dataset, there are 13 characters and 19 numerics. The data cleaning process starts with converting all the class character columns to factors. Also, column "is_canceled" which is the response along with columns "arrival_date_year" and "is_repeated_guest" were converted to factors. Finally, the column "reservation_status_date" was changed to date. As table 2 represents, predictors "country", "agent" and "company" are all factors with 178, 334 and 353 levels, respectively. So, to make things more manageable, only the top 30 most frequent levels of each variable were kept, and the rest was merged to a new level called "Other". The top 30 most frequent levels cover 97%, 86% and 98% of the 3 mentioned columns, in order. At the end, I added a new column called "rowNo" which is just an indicator to keep track of any changes. Therefore, the converted data has 16 factors, 16 integers and one date. Finally, "adr" stands for average daily rate, and the value must be non-negative. There is 1 observation with a negative value for "adr" while the reservation was not cancelled. So, this row was removed. Finally, the cleaned dataset has 119,389 rows and 33 columns.

Table 2: Data descriptions

	Variable	Class	Description
1	hotel	character	Hotel (H1 = Resort Hotel or H2 = City Hotel)
2	**is_canceled**	**double**	**Value indicating if the booking was canceled or not**
3	lead_time	double	No. of days between the booking entering date and the arrival date
4	arrival_date_year	double	Year of arrival date
5	arrival_date_month	character	Month of arrival date
6	arrival_date_week_number	double	Week number of year for arrival date
7	arrival_date_day_of_month	double	Day of arrival date
8	stays_in_weekend_nights	double	No. of weekend nights the guest stayed or booked to stay at the hotel
9	stays_in_week_nights	double	No. of week nights the guest stayed or booked to stay at the hotel
10	adults	double	No. of adults
11	children	double	Number of children
12	babies	double	Number of babies
13	meal	character	Type of meal booked
14	country	character	Country of origin
15	market_segment	character	Market segment designation
16	distribution_channel	character	Booking distribution channel
17	is_repeated_guest	double	if the booking name was from a repeated guest or not
18	previous_cancellations	double	No. of previous bookings that were cancelled by the customer
19	previous_bookings_not_canceled	double	No. of previous bookings not cancelled by the customer
20	reserved_room_type	character	Code of room type reserved
21	assigned_room_type	character	Code for the type of room assigned to the booking
22	booking_changes	double	No. of changes/amendments made to the booking
23	deposit_type	character	If the customer made a deposit to guarantee the booking
24	agent	character	ID of the travel agency that made the booking
25	company	character	ID of the company that made the booking
26	days_in_waiting_list	double	No. of days the booking was in the waiting list before it was confirmed
27	customer_type	character	Type of booking
28	adr	double	Average Daily Rate
29	required_car_parking_spaces	double	No. of car parking spaces required by the customer
30	total_of_special_requests	double	No. of special requests made by the customer
31	reservation_status	character	Reservation last status
32	reservation_status_date	double	Date at which the last status was set

Table 2 and Figure 1 show that the response is imbalanced, especially for the Resort hotel, and this is something that we will consider when modeling the data.

Table 3: Response class imbalancy

Name	Canceled?	%
City Hotel	no	58.27304
City Hotel	yes	41.72696
Resort Hotel	no	72.23595
Resort Hotel	yes	27.76405

Figure 1: Response class imbalancy

Figure 2 shows the correlation among numeric values of the dataset to see if there are any pairs that are highly correlated. The highest correlation ratio comes from "stays_in_week_nights" and "stays_in_weekend_nights" which makes sense. The correlation ratio is 50% which does not seem to be problematic.

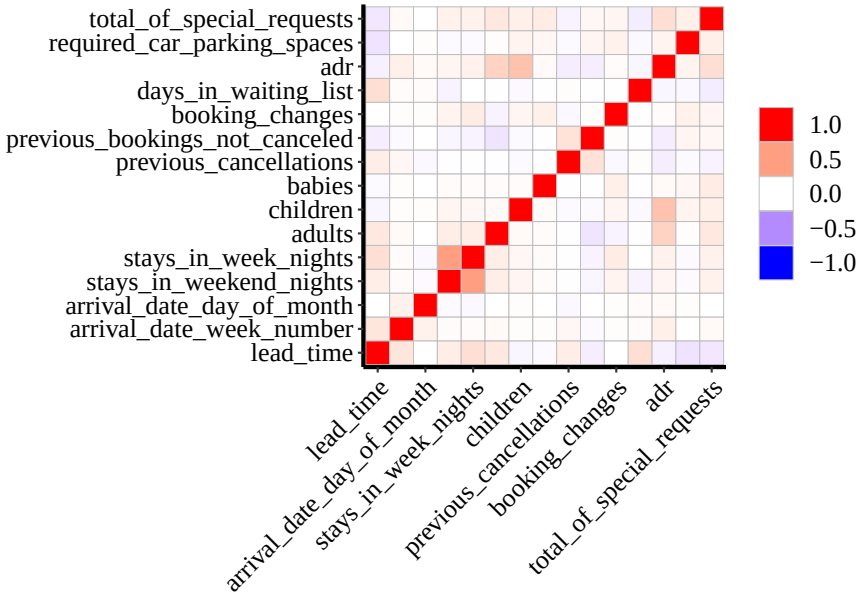

Figure 2: Correlation plot

2.3 Missing data and data imputation

There are only 4 missing values in the "children" predictor column. However, after digging into all the predictors, variables "meal", "market_segment" and "distribution_channel" have a level named "unidentified". Also, predictors "country", "agent" and "company" have a level called "Null". Both of these levels show missing data that also need to be taken care of. The "company" column has 112,593 "Null" or missing values which consists of more than 94% of the column. So, it is better to remove this column entirely. Since all other missing values, except for the 4 rows in the "children" column, happen in categorical variables, the mode of each column is chosen to fill the missing values. For the "children" column, the median is used for imputation.

2.4 Data re-structuring and New variable creation

Data does not need re-structuring and the current format is acceptable for now. Depending on the method that will be used for modeling later, for instance standardization and normalization for neural networks and one-hot encoding for gradient boosting, data will be adjusted accordingly. Also, the variables that the raw data has seem to be enough for now. Again, adding more data or removing some of it greatly depend on the performance of the model that will be constructed. There is always room for adding more predictors; however, the more variable a model has, the more difficult it is to interpret the results.

3 Methods

3.1 RandomForest

Random Forests (RF) are an ensemble learning method for classification and regression by constructing many individual decision trees on the training data set and outputting the mode of classes or mean prediction for classification and regression problems, respectively [5]. One of the problems that decision trees face is overfitting which is overcome using RF since there are many of them and the input for each tree is drawn randomly with replacement (bootstrap aggregating or bagging). This approach will significantly reduce the variance of the results. In decision trees, trees are grown very deep in order to fit very irregular patterns. While this will lead the tree to have low bias, its variance can be very high [5]. RFs are a way to average multiple deep trees that are trained on different parts of the same training set to balance this variance. Since sampling is done with replacement, the performance of the constructed model on the training set is evaluated on the part of the training set that was not selected in the bootstrap which is called *Out-Of-Bag* (OOB) [6]. Another important feature of RFs is that they do not use all the predictors at each split point. A random subset of the predictors is selected for each split which makes the variance to be balanced even more. The reason for that is there might be a few features that are strong predictors, and their high correlation could lower the bias and increase the variance of the model. At the end, the variable importance matrix of the features can be plotted to see which features are most important based on the criteria we define for the model to follow for splitting. There are so many arguments that can be tuned in a RF model some of which are: the number of trees to grow which should not be a small number to make sure that every row gets selected in bootstraps at least a few times, number of variables randomly sampled as candidates at each split, cutoff (for classification only) the 'winning' class for an observation is the one with the maximum ratio of proportion of votes to cutoff. The advantage that RFs have over many other machine learning approaches is that they do not need much data cleaning and manipulation. They can handle both numeric and nominal predictors which means one-hot encoding is not needed. Also, unlike Neural Networks (NNs), standardization is not necessary. That is why they are very popular among people who work on machine learning models on a daily basis.

3.1.1 Data split

There are several steps to create a useful model, including parameter estimation, model selection and tuning, and performance assessment. Throughout this work, I will use the *Tidymodels* package for all steps and explain what each step and function does [7]. The tidymodels framework is a collection of packages (like caret and mlr3) for modeling and machine learning using tidyverse principles. The first step is to split the data into train and test sets, and build the model only on the train set while the test set is untouched. Usually,

when the number of observations is large enough, a split of 80-20 for train-test sets, respectively, is considered good. Also, in this step, the data is stratified by the response column to make sure that there are equal ratios of "canceled" and "not canceled" in both train and test sets. For this part, the *rsample* package is used.

```
set.seed(2020)
hotel_split <- initial_split(hotels, prop = 0.8, strata = is_canceled)
hotel_training <- training(hotel_split)
hotel_test <- testing(hotel_split)
```

3.1.2 Recipe

Feature engineering encompasses activities that reformat predictor values to make them easier for a model to use effectively [7]. This includes transformations and encodings of the data to best represent their important characteristics. This is what can be done in a *recipe*. Not only can a recipe handle data manipulation and feature engineering, it can also do missing data imputation, standardization, one-hot encoding, to name but a few [7]. In a recipe, first, we need to define a formula and a data set. When the data is passed to the recipe, all character columns will automatically convert to factors. *step_medianimpute* is used to impute missing values with the median of that column. *step_other* is a step to pool infrequently occurring values into an "other" category. Other functions like *step_mutate* and *step_rm* are extensions of the *dplyr* package. The last step *step_corr* is added to remove variables that have large absolute correlations with other variables (if there is any) [7].

```
hotel_recipe <- recipe(formula = is_canceled ~ ., data = hotel_training) %>%
  step_filter(adr >= 0) %>%
  step_medianimpute(children) %>%
  step_mutate(meal = fct_recode(meal, "BB" = "Undefined")) %>%
  step_mutate(country = fct_recode(country, "PRT" = "NULL")) %>%
  step_mutate(market_segment = fct_recode(market_segment, "Online TA" = "Undefined")) %>%
  step_mutate(distribution_channel = fct_recode(distribution_channel,
                                                "TA/TO" = "Undefined")) %>%
  step_mutate(agent = fct_recode(agent, "9" = "NULL")) %>%
  step_mutate_at(c("is_canceled", "is_repeated_guest", "arrival_date_year"),
                 fn = as.factor) %>%
  step_mutate(is_canceled = fct_recode(is_canceled, no = "0", yes = "1")) %>%
  step_mutate(is_repeated_guest = fct_recode(is_repeated_guest, no = "0", yes = "1")) %>%
  step_other(country, threshold = 167) %>%
  step_other(agent, threshold = 351) %>%
  step_rm(company, reservation_status) %>%
  step_corr(all_numeric(), threshold = 0.7)
```

It should be noted that recipe does not execute any of the steps; it is just a specification of what should be done. The second step in a recipe is *prep*() where any quantity in each step will be estimated. For example, *step_medianimpute* will calculate the median of "children" column, or *step_other* will calculate what levels have a frequency lower than the specified threshold to merge them all into a new "other" level. The last phase of a recipe is to apply the aforementioned pre-processing operations to the train set. This will be done through a function called *bake*() [7].

```
hotel_recipe %>%
  prep() %>%
  bake(new_data = NULL)
```

For the train set the argument "new_data" in *bake*() is set to NULL. However, for the test set, "new_data" will be the test set. The output of a baked recipe is a regular data frame.

3.1.3 Model

For this section, we start with a random forest model. The *parsnip* package provides a fluent and standardized interface for a variety of different models. For each model available in *tidymodels*, there is a general process to follow which is defining the model with its parameters (here random forest), setting a mode (classification for this dataset) and setting the engine to use (here ranger is used) [7].

```
rf_fit <- rand_forest(
  mtry = tune(),
  trees = 1000,
  min_n = tune()
) %>%
  set_mode("classification") %>%
  set_engine("ranger",
             num.threads = 4)
```

This is also where the parameters that are going to be tuned are defined. In this example, mtry (the number of predictors to sample at each split) and min_n (the number of observations needed to keep splitting nodes). These are hyper-parameters that can't be learned from data when training the model. For the tuning part, the *tune* package is used [7].

3.1.4 Workflow

The next step in a tidy modeling approach is to define a workflow. The purpose of this object is to encapsulate the major pieces of the modeling process that have already been explained. The *workflows* package allows the user to bind modeling and pre-processing objects together [7]. It can be seen that the previously defined recipe (hotel_recipe) and model (rf_fit) are passed to the workflow to be combined. Note that when we use a workflow, we do not need to separately *prep*() or *bake*() a recipe. It will be done automatically inside the workflow.

```
rf_workflow <-
  workflow() %>%
  add_recipe(hotel_recipe) %>%
  add_model(rf_fit)
```

3.1.5 Resampling

Resampling methods are empirical simulation systems that emulate the process of using some data for modeling and different data for evaluation [7]. Most resampling methods are iterative, meaning that this process is repeated multiple times. For this example, a 10-fold cross validation is used to assess the performance of the trained model. Also, notice that "strata" is set to the response in order to make sure that all folds have the same ratio of canceled bookings. *rsample* is the package to be used for resampling [7].

```
set.seed(2020)
hotel_folds <- vfold_cv(hotel_training, strata = is_canceled, v = 10)
```

Next step will be defining a grid that contains different combinations of the parameters we chose. For this grid, I set the grid to consider 20 combinations of the mtry and min_n. After a tuning is done, we can see the best combination of the tuned parameters based on the performance we like (i.e. accuracy or roc_auc). For this example, accuracy is chosen as the measure of performance.

```
set.seed(2020)
tune_res <- tune_grid(
  rf_workflow,
  resamples = hotel_folds,
  grid = 20
)
```

```
show_best(tune_res, "accuracy")
best_acc <- select_best(tune_res, "accuracy")
```

After finding the best parameters, we need to finalize the model and consequently the workflow (or we could have finalized the workflow directly using $finalize\_workflow$). Lastly, there is a function called $last\_fit$. This function fit the model on the whole train set and evaluates it on the test set automatically. Notice that the argument passed to the $last\_fit$ is "hotel_split" which includes both train and test sets [7].

```
final_rf <- finalize_model(
  rf_fit,
  parameters = best_auc
)

final_wf <- workflow() %>%
  add_recipe(hotel_recipe) %>%
  add_model(final_rf)

final_res <- final_wf %>%
  last_fit(hotel_split)
```

3.1.6 Performance Evaluation

At the end, the *yardstick* package helps to collect and show all possible forms of performance evaluations (i.e confusion matrices, roc_auc, mae) [7]. Figure 3 displays variables importance of this RF model. It seems that the reservation date is the most important parameter as a split point.

```
final_res %>%
  collect_metrics()
```

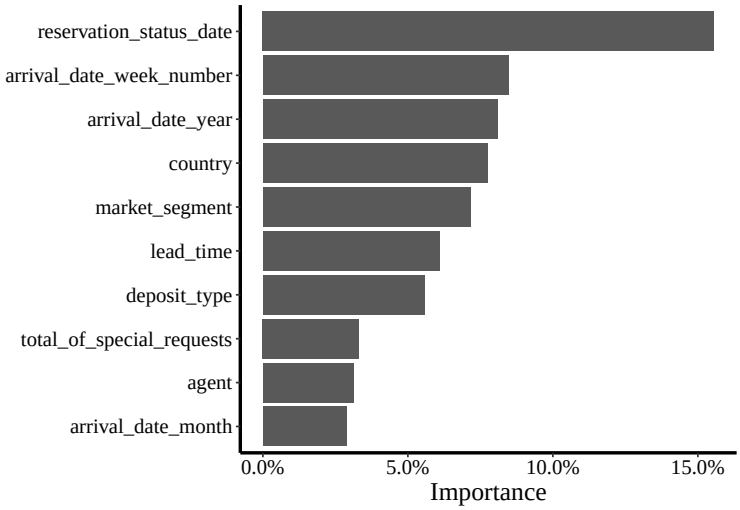

Figure 3: Variables importance for the RF model

Figure 4 shows the parameters used in the tuning process. Based on this figure, min_n = 8 and mtry = 14 are the optimum combination that gives the highest accuracy.

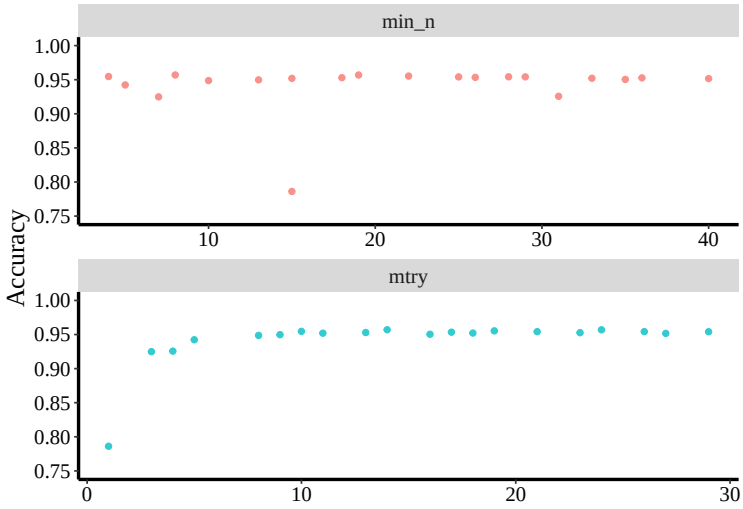

Figure 4: RF parameter tuning

Table 4 shows the confusion matrix of this model, and table 5 shows accuracy, precision and recall for this matrix.

Table 4: Confusion matrix for the RF model

		Truth	
		No	Yes
Prediction	No	14838	781
	Yes	194	8063

Table 5: Accuracy, precision and recall for the RF model

Measure	Value
Accuracy	95.916
Precision	97.650
Recall	91.169

Finally, figure 5 depicts the ROC curve of this model.

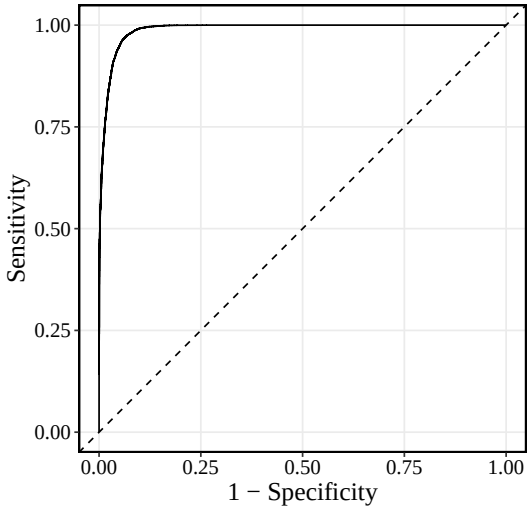

Figure 5: ROC curve for the RF model

3.2 Balanced and Weighted Random Forest

As mentioned in section 3.1, the performance (the majority vote for classification problems) is calculated over the out-of-bag observations. In learning extremely imbalanced data, there is a significant probability that a bootstrap sample contains few or even none of the minority class, resulting in a tree with poor performance for predicting the minority class [8]. In the RF algorithm, there is an argument called $cutoff$ which is a vector of length equal to the number of classes. The 'winning' class for an observation is the one with the maximum ratio of proportion of votes to cutoff. The default is 1/k where k is the number of classes (i.e., majority vote wins) [8]. In a Balanced Random Forest (BRF), this cutoff point is changed in order to make the observations more balanced. In imbalanced data sets, not only is accuracy important, there are two more criteria to consider namely *Precision* and *Recall*. They are defined as follows:

$$Precision = \frac{\#\ of\ correctly\ predicted\ as\ positive}{\#\ of\ examples\ predicted\ as\ positive}$$

$$Recall = \frac{\#\ of\ correctly\ predicted\ as\ positive}{\#\ of\ positive\ examples\ in\ test\ set}$$

Another approach to make RF more suitable for learning from extremely imbalanced data follows the idea of cost sensitive learning. Since the RF classifier tends to be biased towards the majority class, a heavier penalty shall be placed on mis-classifying the minority class [8]. A weight is assigned to each class, with the minority class given larger weight (i.e., higher mis-classification cost). The class weights are incorporated into the RF algorithm in two places. In the tree induction procedure, class weights are used to weight the "Gini" criterion for finding splits. In the terminal nodes of each tree, class weights are again taken into consideration. The class prediction of each terminal node is determined by "weighted majority vote"; i.e., the weighted vote of a class is the weight for that class times the number of cases for that class at the terminal node. The final class prediction for RF is then determined by aggregating the weighted vote from each individual tree, where the weights are average weights in the terminal nodes. This approach is called Weighted Random Forest (WRF).

All the steps are similar to regular RF that was explained in 3.1. The only difference is the cutoff point and class weights that are tuned in these models to address class imbalancy (response contains 37% of the minority class and 63% of the majority). Figures 6 and 7 display the three measures (accuracy, precision and recall) for BRF and WRF, in order. Figure 7 shows that unlike BRF (figure 6), WRF does not change any of the three measures very much. It is seen that the highest accuracy comes from a cutoff point equal to 0.5 which is the same as the regular RF. So, in this case, although the data is imbalanced, BRF does not help to increase the accuracy. However, the highest accuracy of WRF (95.391%) is slightly better than that of BRF (95.946%).

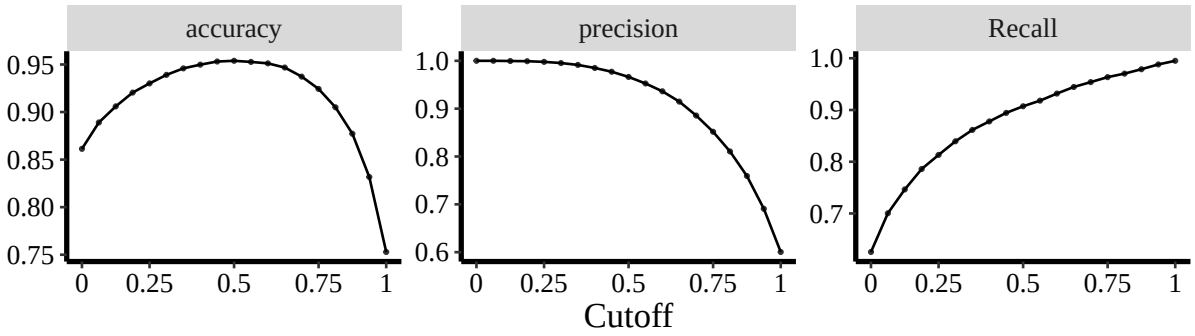

Figure 6: BRF accuracy, precision and recall

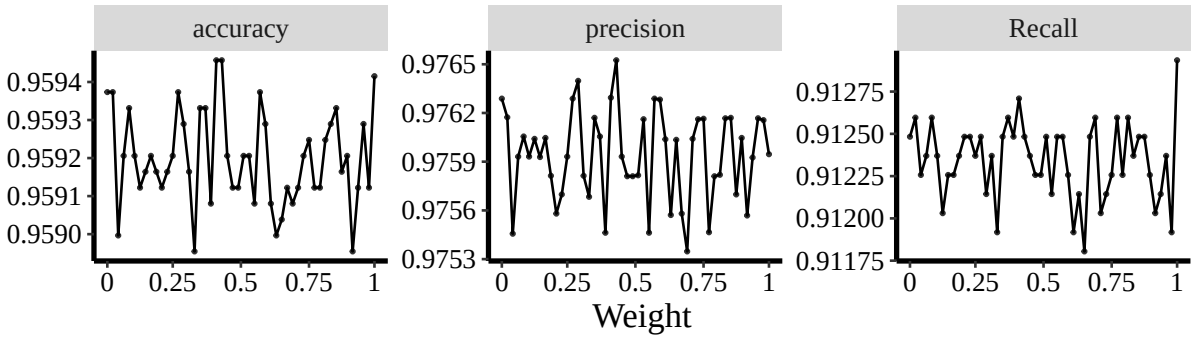

Figure 7: WRF accuracy, precision and recall

3.3 XGBoost

Gradient boosting is a machine learning technique used for classification and regression problems. In this approach the final prediction is an ensemble of weak prediction model, in this case, decision trees. Like all other boosting methods, it combines weak learners into a single strong learner in iterations [9]. The Extreme Gradient Boosting (XGBoost) is a scalable tree boosting system that can be used for both regression and classification. XGBoost uses an ensemble of K classification and regression trees (CARTs) [10]. For each leaf, all possible cutting points are considered, and the one that gives the highest gain score, is used to as the split point. As we go deeper, the number of residuals in each leaf gets smaller, until it becomes smaller than the minimum number of residuals required by default by XGBoost. Each tree is built on the residuals from the previous tree, and this process continues until reaching the maximum number of trees or the residuals are very small, smaller than the default threshold in XGBoost [11]. There are so many parameters in an XGBoost model that can be tuned. Some of the most important ones are: learning rate (eta, η) which scales the contribution of each tree by a factor between 0 and 1. Low η value means model more robust to overfitting but slower to compute; minimum loss reduction (Gamma, γ) required to make a further partition on a leaf node of the tree; maximum depth of each tree; subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees, and this will prevent overfitting; subsample ratio of columns when constructing each tree; L_1 and L_2 parameters for linear regularization.

Since half the features (columns) are factors, one-hot encoding needs to done before sending the data into the XGBoost algorithm. Considering different levels of all numeric columns, after one-hot encoding, both train and test sets will have 171 columns. A very small grid search of 54 combinations of 4 parameters (eta, max_depth, col_sample and sub_sample) was conducted to find the optimum value of their combination. The result is shown in table 6. For this model, 10-fold cross validation was used to assess the performance of

the model. Figure 8 displays the importance matrix of the model (top 10). The accuracy of this model is 88.04%.

Table 6: XGBoost tunned parameters

Parameter	Value
eta	0.1
max_depth	3.0
col_sample	0.8
sub_sample	0.8

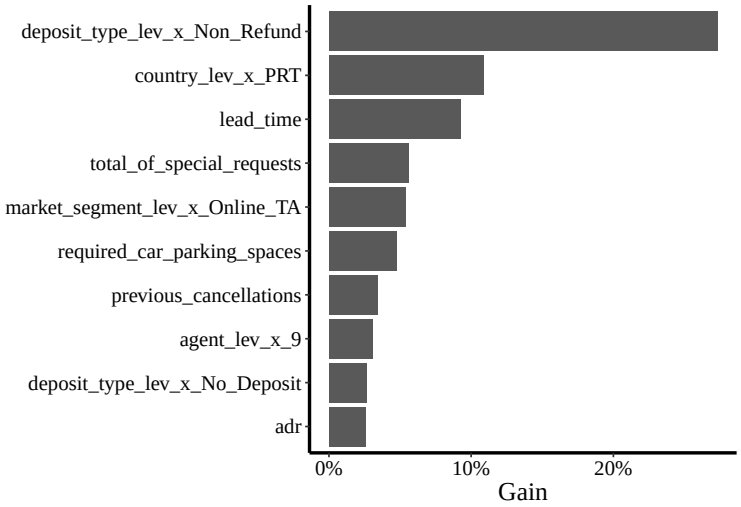

Figure 8: Variables importance for the RF model

3.4 ConvXGB

Classification problems are a very important part of the supervised learning, and over the course of the past few decades, many researchers have come up with different algorithms to classify an object. These algorithms can range from the simplest ones like logistic regression to more complicated ones like Random Forest (RF), Support Vector Machine (SVM) and Extreme Gradient Boosting (XGBoost). Among all of the mentioned algorithms, XGBoost has gained much reputation due to its speed and accuracy, especially in classification problems. In recent years, there has been a massive work and research on Artificial Neural Networks (ANN) that can be used in a variety of fields from regression and classification to image and text recognition. Convolutional Neural Network (CNN) is a part of the neural network family that is mostly used for image recognition problems. A common step in all machine learning algorithms is to find the most important features that will contribute the most towards the final classification. This is something that is usually done manually by people who run algorithms like RF, SVM and XGBoost, to name but three. On the other hand, one of the beauties of the CNN is its automatic feature learning capability which is a huge advantage over all other algorithms. Since the capabilities of a single model may not be sufficient to be used for different problems, and each model has its own advantages and disadvantages, it can be a good idea to combine models and take advantage of their strengths. ConvXGB is a new deep learning model which is a combination of CNN and XGBoost [12]. ConvXGB consists of a net of several stacked convolutional layers and with a XGBoost at the end of the network to do the classification part. The main difference beetween this novel model and the conventional CNN model is that ConvXGB does not have either a pooling layer or a fully connected layer which was done to add simplicity to the model [12]. ConvXGB can be used for both image and general classification problems and according to the literature has shown promising results. Figure

9 shows the architecture of the ConvXGB model. Figure 10 displays how the output from the convolutional layers are reshaped and fed into the XGBoost part of the model.

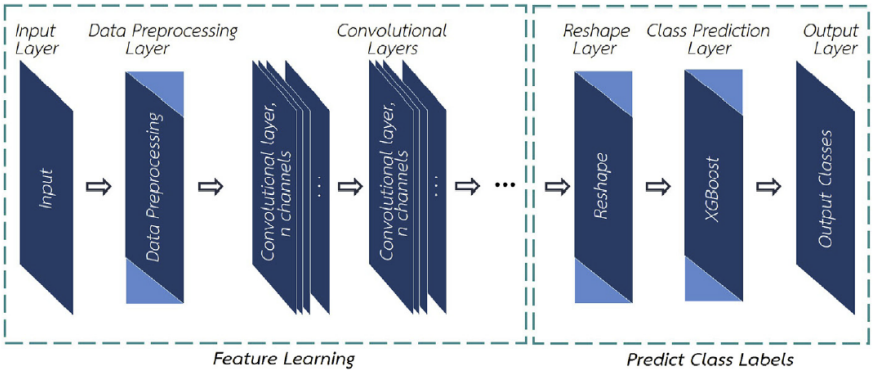

Figure 9: Arcitecture of the ConvXGB model [12]

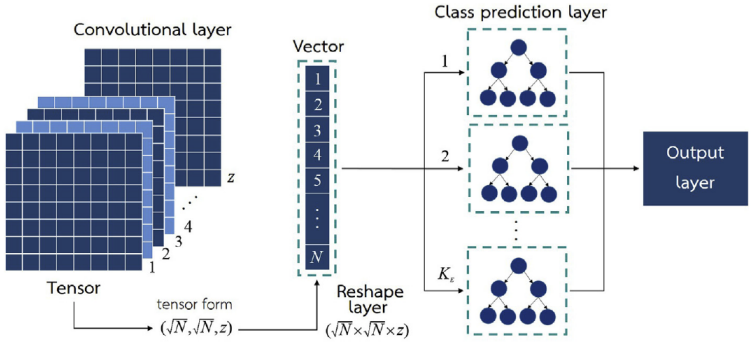

Figure 10: The process of operation in part of predict the class labels consists of reshape layer, class prediction layer and output layer [12]

A vital part of the ConvXGB model is the preprocessing part where the input data is reshaped to 2-D square matrix. If the number of features for each observation does not match a square number, zero elements will be added to the features vector until a square number is reached. Then, it will be reshaped to square matrix [12]. This process which is called zero-padding is shown in figure 11.

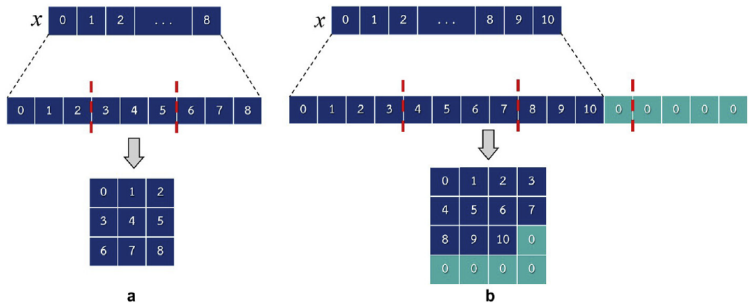

Figure 11: Example of converting data to the standard criterion: (a) Describes of a 'square' feature vector which can be simply copied whereas, in (b), zeroes are added so that the feature vector is square [12]

In summary, the following is the steps need to be taken for this approach [12].

1. Initialize the training data set
2. If necessary, pad the N elements of each training data item, so that new data item can be formed into a square matrix of dimensions, $\sqrt{N} \times \sqrt{N}$.
3. Convert the tensor format to $(\sqrt{N} \times \sqrt{N} \times z)$, where z is the size of filters.
4. Set the parameters of the convolutions for learning features

- (a) number of convolutional layers
- (b) convolutional layer output depth
- (c) for each layer, set the filter sizes
- (d) filter strides

5. For each layer, calculate the convolutions (left side of figure 10)
6. Reshape the output of the previous layer of a vector of length $(\sqrt{N} \times \sqrt{N} \times z)$
7. Initialize a new training data set for class prediction layer (which is the output of the previous layer)
8. Initialize parameters for the prediction step, set

- (a) total number of trees, KK
- (b) regularization parameters, g and l,
- (c) column subsampling parameter,
- (d) maximum tree depth and
- (e) learning rate

9. Determine the class labels for output

After one-hot encoding, the number of columns will be 171. We know the first step in this approach is zero-padding if necessary. Since 171 is not a square number, 25 zero columns were added to the matrix to raise the number of columns to 196. Then, the 2-D matrix was reshaped to a 4-D array of size train-test $\times$ $14 \times 14 \times 1$. Three convolutional layers were used each of which had 32, 64 and 128 filters successively with the kernel size of 3×3. The ReLU was used as the activation function for all layers, the "padding" option was set to "same" in order to keep the size of the matrix constant through all the layers. However, in this case, since the number of columns would be 25088, it had to be reduced so the system could handle it. Then, the last layer was flattened and connected to a dense layer with 256 neurons. The information of this layer was extracted and fed into the XGBoost model (a model with 256 columns). The same parameters as the pure XGBoost model was used here in order to be coherent. Also, a 10-fold cross validation was used to assess the performance of the model. Figures 12 and 13 show the model as well as the network. In the CNN part of the model, there can be thousands or even millions of parameters that the network learns throughout its process. All parameters come from filters and kernels. For example, let us consider the network in figure 13. The last column of this figure shows the number of parameters in each layer. For the 2^{nd} convolutional layer, we can see the number of parameters is 18496. This number comes from the filters in the first and second layers, the kernel size (here 3×3) and the bias for the second layer(for each filter there is a single bias). Therefore we have $(32 \times 64 \times 9) + 64 = 18496$ parameters. The accuracy of this model was 88.26%.

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", padding = "same",
                input_shape = c(14, 14, 1), name = "Conv_01") %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu", padding = "same", name = "Conv_02") %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu", padding = "same", name = "Conv_03")
  layer_flatten(name = "Flat_01") %>%
  layer_dense(units = 256, activation = "relu", name = "Dense_01")
```

Figure 12: CNN model generated in keras for ConvXGB

```
Model
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv_01 (Conv2D)             (None, 14, 14, 32)        320
_________________________________________________________________
Conv_02 (Conv2D)             (None, 14, 14, 64)        18496
_________________________________________________________________
Conv_03 (Conv2D)             (None, 14, 14, 128)       73856
_________________________________________________________________
Flat_01 (Flatten)            (None, 25088)             0
_________________________________________________________________
Dense_01 (Dense)             (None, 256)               6422784
=================================================================
Total params: 6,515,456
Trainable params: 6,515,456
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: CCN network used in ConvXGB

For this model the Keras [13] and xgboost packages were used which is a high-level API to build and train deep learning models. In Keras, *layers* are assembled to build *models*. For this model, the most common type of models which is stack of layers was used through *keras_model_sequential()*, and each convolutional layer was made of *layer_conv_2d()* [13]. Then, to get the output of the last layer of the network, *keras_model()* function was used. Finally, this output was reshaped and fed into the xgboost model which comes from the xgboost package.

3.5 Logistic Regression

Logistic models are used to model the probability of a certain class such as win/lose and pass/fail. Logistic regression is a statistical model that uses logistic functions (equation 1) to predict a binary or multi-level dependent variables [14]. In fact, logistic regression is very much like a linear regression but instead of modeling the response directly, it models the *probability* that the response belongs a specific class. So, regardless of the magnitude of the predictor, the probability is always between 0 and 1.

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{1}$$

The beauty of the *tidymodels* package is the only change needed to convert random forest in section 3.1 to a logistic regression model is the following.

```
Logit_fit <- logistic_reg(
  penalty = tune(),
  mixture = tune()
) %>%
  set_mode("classification") %>%
  set_engine("glmnet")
```

Everything else remains the same. For this model, penalty which is the same as "lambda" in the glmnet package (the total amount of regularization in the model), and mixture which is "alpha" in the glmnet package (a number between zero and one (inclusive) that is the proportion of L_1 regularization) are set to be tuned. A grid search of size 20 is used to find the optimal combination of these two parameters. Figure 14 shows the parameter tuning combinations, and figure 15 displays the ROC curve of this logistic model. The best combination of these two parameters is when penalty $= 2.34e^{-5}$ and mixture $= 0.874$.

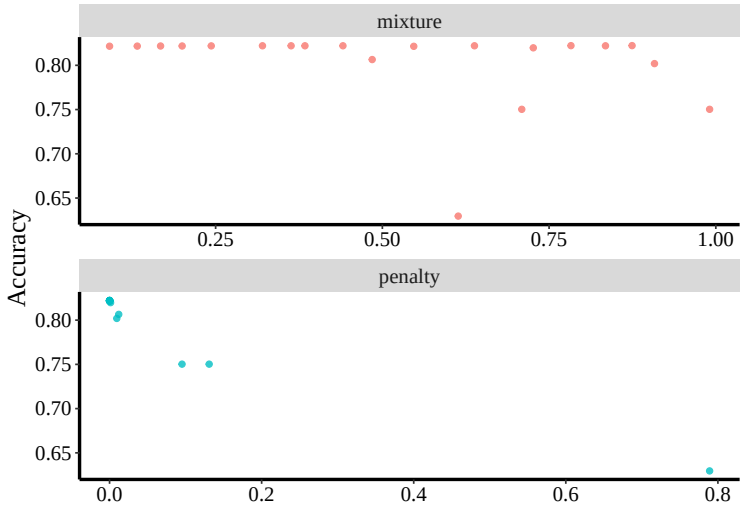

Figure 14: Logistic regression parameter tuning

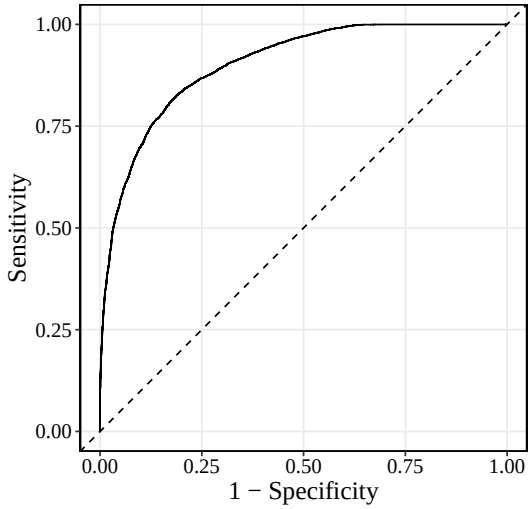

Figure 15: ROC curve for the logistic regression model

Finally, table 7 shows the highest accuracy and roc_auc of this model. A 10-fold cross validation was used to assess its performance.

Table 7: Logistic regression tunned results

Accuracy	roc_auc
0.822	0.903

4 Results

In this work, the goal is to classify if a specific hotel booking was canceled or not based on many predictors, including but not limited to, the country of origin, number of adults and children, if the person is a repeated guest, number of nights they stayed and what meals they booked. The data set is imbalanced with its response containing 63% of the majority class and 37% of the minority. So, it is a good idea to address this imbalancy. In this work, 6 models are used including: Regular Random Forest (RF), Balanced Random Forest (BRF), Weighted Random Forest (BRF), XGBoost, ConvXGB and Logistic Regression. Table 8 shows the the comparison among all these models in terms of accuracy.

Table 8: Accuracy comparison

Model	Accuracy (%)
Weighted Random Forest	95.946
Balanced Random Forest	95.391
Random Forest	95.916
ConvXGB	88.260
XGBoost	88.040
Logistic Regression	82.207

Based on this table, the best performance comes from the family of RF models with WRF at top followed by BRF and RRF. They are very close to each other, but it seems that addressing the class imbalancy has an effect on the accuracy of the model since WRF and BRF have higher accuracies compared to RF. After, ConvXGB which is a novel model to combine both CNN and XGBoost stands. However, its accuracy is not even nearly as good as RF family. After that and very close to ConvXGB, there is XGBoost. Finally, and at the end of the list, Logistic Regression stands with almost 82% accuracy. This preliminary analysis shows that tree-based models, especially RF family, outperform others significantly. Most of the time, people care most about the accuracy, but in significantly imbalanced data sets, accuracy may not be as important as precision/recall are. So, based on the type of data we deal with and the purpose of that work, a suitable measure needs to be chosen. Also, this analysis shows that although one may think that XGBoost is always better than RF, it is not the case all the time. In fact, without checking all possible models, one cannot decide what model to use. This is because every data set is unique and should be approached differently. Finally, the *tidymodels* package is an amazing toolbox that can be used for many different purposes. Although it is a very young and newly introduced package, it has so much capability. As was explained in section 3.5, when we have a working model in this package, making new models is so simple (as simple as changing a few arguments!) to get a totally new model. This way, models can be compared so easily and effectively. Also, since the idea and people behind this package are the same people who have been working on *tidyverse* and similar packages, everything is tidy and consistent with *tidyr*.

5 Summary

5.1 Summary of the project

Hotel booking cancellation has significant impact on the demand-management decision in the hospitality industry. Although hotels usually put strict policies regarding cancellation to mitigate their negative effects, the same policies can bring a bad reputation for the hotel as well as a negative impact on the revenue. That is why in today's hospitality industry, machine learning models are used to predict which reservations are "likely to cancel" to reduce the negative impact of possible cancellations. Therefore, in this work, the goal was to classify and predict if a booking would get canceled or not. As it was shown in figures 3 and 8, the country of origin, the deposit type, the reservation status date and the week that people would arrive are top features that can affect their decision to cancel their bookings. Results show that the accuracy of such predictions can go up to almost 96% which is can be a great achievement for this industry to minimize their

money loss due to unexpected cancellations.

5.2 Interesting findings

Before starting this project and only based on my previous experiences, I was expecting XGBoost to outperform other models, but seeing Random Forest on top of the table 8 with almost 8% advantage over XGBoost was kind of a surprise for me. It simply shows that all machine learning problems should not be treated the same, and while some models might work very well for a specific problem, other may work better for other problems. This is where the *tidymodels* package becomes handy. Using this package and only with a few changes, we can get a totally new model. Also, there are some types of behaviors that can be expected from the people who are more/less likely to cancel a booking. For instance, according to figures 3 and 8, the total number of special requests that was made by the guest at the time of reserving the room is an important factor that can lead to booking cancelation. Although these information may seem unnecesary for someone who is not familiar with data analysis or machine learning, it is pure gold for hotel managers to modify their hotels' systems. Also, I really enjoyed using the *tidymodels* package, and comparing to other ML tool boxes like *caret* and *mlr3*, I think it will shortly become the top package that people would use for any kind of machine learning problems. Not only has it been developed by great people like Max Kuhn and Hadley Wickham, it is also very easy to use and intuitive. As the name of the package suggests, one of the goals of the tidymodels team was to make a machine learning package that was *tidy*, and I think they nailed it!

5.3 Discussion on the unsatisfying models or results

As mentioned in section 5.2, XGBoost did not perform as well as one would expect. The logistic regression model was the one that performed worst among all. When considering different models, it is not just the accuracy that matters; Interpretability is also another very important factor that needs to be considered. If a model gives higher accuracy compared to others, it does not always mean that we should pick that one. Sometimes, interpretability is of greater importance although the model might be slightly less accurate. For instance, in this work, the accuracy of the ConvXGB model is slightly higher than that of the XGBoost model. However, when it comes to interpretability, XGBoost is much more understandable than the ConvXGB. So, we need to know the audience to whom we want to present the results.

5.4 Future work

Something I would like to see is how such a machine learning model will work in reality. Based on this project, they can be expected to be almost 96% efficient, but is it how they would really perform? Data needs to be collected from different hotels around the world to compare their results with the predicted values from the model. Also, in this work, ConvXGB was the combination of two models that were considered. Considering that Random Forest models have the highest accuracy among all, a combination of CNN and Random Forest can also be considered. Tidymodels team, very recently, released a new package called *stacks*. Model stacking is an ensembling method that takes the outputs of many models and combines them to generate a new model. This is something that can be added to the model since it can boost the accuracy.

References

1. https://www.hotelmanagement.net/tech/study-cancelation-rate-at-40-as-otas-push-free-change-policy
2. Kimes, S. E., & Wirtz, J. (2003). Has revenue management become acceptable? Findings from an International study on the perceived fairness of rate fences. Journal of Service Research, 6(2), 125–135.
3. Mehrotra, R., & Ruttley, J. (2006). Revenue management (second ed.). Washington, DC, USA: American Hotel & Lodging Association (AHLA).
4. Antonio, N., de Almeida, A., & Nunes, L. (2019). Hotel booking demand datasets. Data in brief, 22, 41-49.
5. https://en.wikipedia.org/wiki/Random_forest
6. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: springer.
7. https://www.tmwr.org/
8. Chen, C., Liaw, A., & Breiman, L. (2004). Using random forest to learn imbalanced data. University of California, Berkeley, 110(1-12), 24.
9. https://en.wikipedia.org/wiki/Gradient_boosting
10. https://en.wikipedia.org/wiki/XGBoost
11. https://www.youtube.com/watch?v=8b1JEDvenQU&t=820s
12. Thongsuwan, S., Jaiyen, S., Padcharoen, A., & Agarwal, P. (2020). ConvXGB: A new deep learning model for classification problems based on CNN and XGBoost. Nuclear Engineering and Technology.
13. https://keras.rstudio.com/articles/guide_keras.html
14. https://en.wikipedia.org/wiki/Logistic_regression