Machine Learning Final Project - Predict the quantized revenue of hotel

Chi-luen Feng, Ching-wei Yang, Yu-chieh Huang

1.Task definition and introduction

In this task, what we need to predict is the daily profit after Quantized. However, training data is not based on one data per day, and there isn't a "revenue" column in the training data. Therefore, our first task is to find out the correlation between Quantized profit and training data.

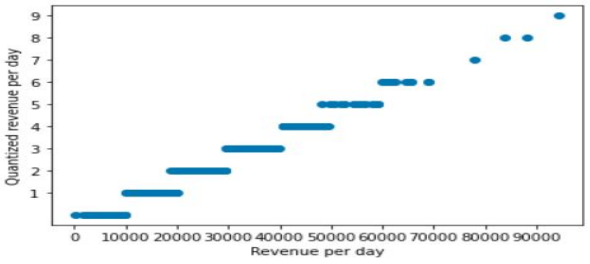
By reading the description of each Column and our understanding of profit, we have done a total of three steps:

1.Disassemble the expected profit of a single data into：

Expected profit = (adr * total days of stay (Week days + Weekend days) * is_canceled)

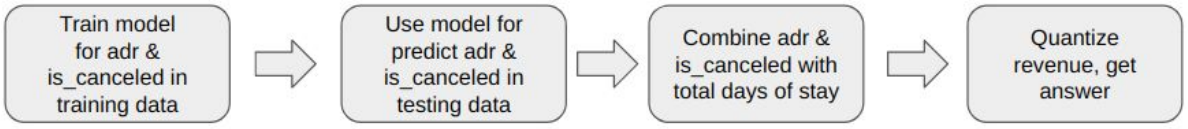
2. Divide the data by arrival_date_year/month/day_of_moth, group each data by time (in the unit of day), and add up the profit of each day to get the total profit of a single day

3. Compared the total profit of a single day with the corresponding Quantized profit and found that the unit of Quantized profit is based on 10,000, and the total profit is divided into 10 blocks. As shown below:



Therefore, through such observations, we only need to predict the expected profit of each row of data, and add up based on time to find the corresponding quantized label.

On this basis, since the testing data already has the total days of stay (Week days + Weekend days), and there is no "adr" and "is_canceled" column in testing data. Therefore, we only need to predict these two variables to achieve the goal. The picture below shows our workflow.



2. Data analysis & Data Preprocessing

2.1 Data analysis

In this section, we will explore the data more carefully and analyze them with some visualizations. After analyzing them, in the next part, we will do some preprocess on them.

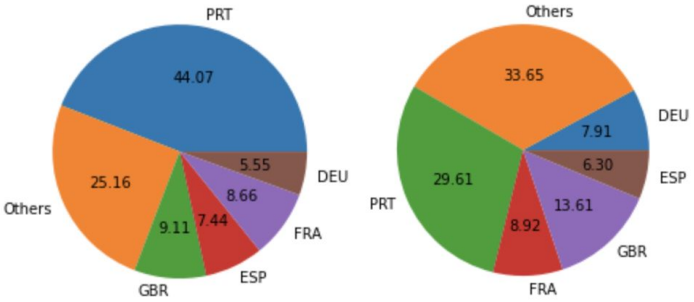
(1) NaN values

The table below are the numbers of the missing values for each feature in training and testing data. Since most machine learning algorithms can't deal with NaN, we have to deal with them.

	children	country	agent	company
train_d	4	468	13217	85917
test_d	0	20	3123	26676

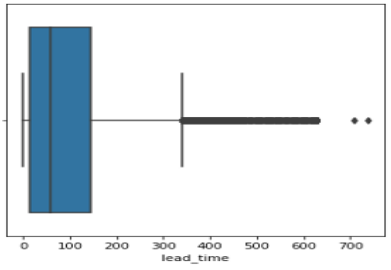
(2) Countries

The picture below is the percentage of each country according to the training and testing data. We can simply tell that some countries' proportions are dominating the entire data.

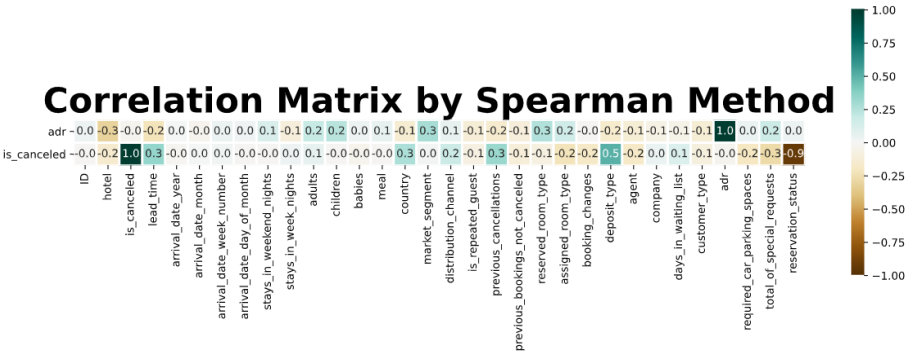


(3) Outliers

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population, which is probably caused by some human error. We must deal with them, since it might affect the performance of our prediction. From the box plot below, we can tell that there are several features with outliers, so in the next step, we try to deal with them.



(4) Correlation between each feature adr and is_cancelled



After analysing this matrix, we found some features having high correlation with "adr" or "is_canceled". For example, the feature "deposit_type" is important for the prediction of

"is_canceled". Specifically speaking, if the customer made a deposit to guarantee the booking, the customer would unlikely to cancel it.

2.2 Data Preprocessing

In this section, based on the analysis from the previous section, we will do some simple preprocessing to the data.

(1) Handling NaN values

We decided to drop two features, 'agent' and 'company', because the missing values are too much. And for the small amount of missing value for features 'children' and 'country', we fill in with the mode.

(2) Countries

We pick the top five countries in training data and testing data which is 'PRT', 'GBR', 'FRA', 'ESP', 'DEU' out and set all the others as the label 'Others'.

(3) Outliers

We observe the description of each feature, and from the max value, min value, mean, std, percentiles, we try to make those outliers make more sense. For example, we make the ones in the feature 'lead_time' not larger than 500.

train_d.loc[train_d.lead_time > 500, 'lead_time'] = 500

(4) Correlation between adr and is_cancelled

From the Correlation matrix above, we drop some features which are less correlated with "adr " and "is_cancelled". We finally dropped the features including "arrival_date_year", "arrival_date_month", "arrival_date_day_of_month", "ID".

*** One-Hot Encoding ***

Eventually, we are almost done with the preprocessing part, the last step is to do the one-hot encoding. It's for categorical variables where no such ordinal relationship exists, the integer encoding isn't enough. In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value. The picture below are all the features we did with this technique.

train_d = train_d.join[pd.get_dummies(train_d.meal, prefix='meal'))

3. Model selection and discussion

In this part, we will first introduce the three prediction models we use, and then we will use these models to predict adr and is_canceled respectively, and analyze results below

Model introduction:

XGBoost:

XGBoost is very similar to GBDT(Gradient boosting Decision Tree) which have been mentioned in the class, the main difference is that XGBoost calculates loss residual by Taylor Expansion. Moreover, XGBoost adds some techniques like parallel computing, shrinkage, etc.

Random Forest:

Random Forest uses numerous decision trees, then voting between these decision trees to get the final answer.

Linear Regression:

Linear Regression use the training data to find a linear equation closest to each point

Model Experiment:

The following shows the experiment with different models. There are some common setting:

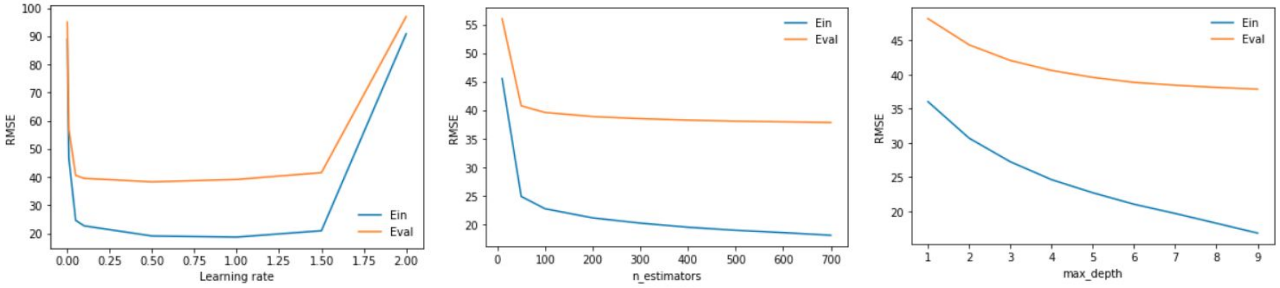
1. Use preprocessing data mentioned in Section 2.
2. Split training data into two part by train_test_split(train, target, random_state=1126)
3. Measure Ein, Eval for training data(after split), validation data
4. When experimenting with one variable, we fix the model for other variables, which use the XGBoost with default settings.(There are only two variable we need to predict: "adr" and "is_canceled")

Experiment for "adr" column:

XGBoost:

We test three parameters for XGBoost, they are "learning_rate", "n_estimators" and "max_depth". The outcome is shown below(We fixed another parameters in default when test with one parameter):

*** Ein/ Eval: RMSE*



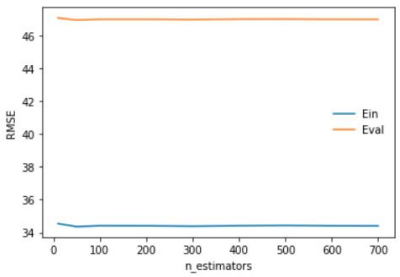
From the picture, we can find that the **learning rate needs to be adjusted to between 0.15 and 1.5** to make the model perform better. If the learning rate is too small, the change of parameters is too small, so that the optimal solution cannot be found; If the learning rate is too large, it may also make the gradient jump too large, making the parameters unable to converge on the optimal point.

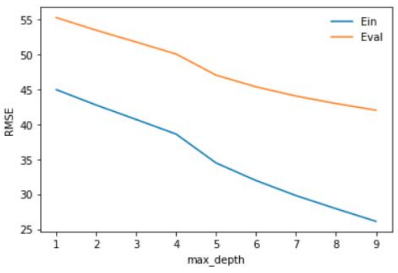
In contrast, n_estimators and max_depth show the larger the better, the reason is maybe we didn't find the overfitting point. In addition, **we found that if n_estimators is below 50, the error rise rapidly.** Although the cost of increasing n_estimators is that the training speed will increase rapidly, **it is still recommended to adjust n_estimators to 50 or more, and it is recommended to adjust to 200-400**, because the overall error decline has not changed much, but the training speed has increased significantly.

Random Forest:

We test two parameters for Random Forest, they are "n_estimators" and "max_depth". The outcome is shown below(We fixed another parameters in default when test with one parameter):

*** Ein/ Eval: RMSE*





From the picture, we find that on n_estimators, no matter how many settings are set, the errors that come out are similar. This is contrary to our expectations. We expect that the more n_estimators, the lower the error. The possible reason is that **this task only needs less n_estimator to achieve convergence on the random forest**.

The max_depth performance is the same as XGBoost. When max_depth is larger, the error is smaller, but the overall performance is not as good as XGBoost.

Linear Regression:

Because Linear Regression doesn't have many parameters to adjust. we analyze coefficients to find out which column is most important. Below are the columns corresponding to top 5 coefficients.

meal_BB	meal_HB	meal_SC	reserved_roomtype_A	reserved_roomtype_D

We can find that the most weighted items are concentrated in two items, meal and reserved_roomtype, which means that in this task, food and room type are most likely to affect adr. This result is in line with our usual intuition of booking a hotel room.

Overall comparison:

Compare the public score and private score of the three models. The comparison results are as follows (the best performance results of these three methods are selected below):

XGBoost		Random Forest		Linear regression	
Public	Private	Public	Private	Public	Private
0.3157	**0.3896**	0.4868	0.5454	0.6973	0.8181

It can be seen that XGBoost gets the highest performance, and the training time of this XGBoost model is less than 1 minute, but good results are obtained. Therefore, XGBoost is recommended for predicting adr.

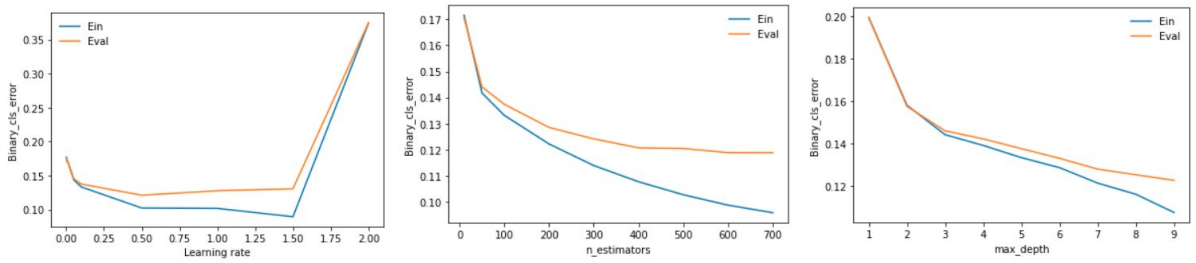
Experiment for "is_canceled" column:

We only test the XGBoost performance for the "is_canceled" variable.

XGBoost:

We test three parameters for XGBoost, they are "learning_rate", "n_estimators" and "max_depth". The outcome is shown below(We fixed another parameters in default when test with one parameter):

*** Ein/ Eval: RMSE*



We can find from the results that the learning rate is similar to the outcome when predicting adr, learning rate cannot be too high or too low, otherwise the error will rise considerably. For the part of n_estimators and max_depth, in terms of trend, the more the better, but in n_estimators, it can be seen that when n_estimators=700, Eval didn't decrease a lot. So when n_estimators exceed 700, it may cause overfitting.

In general, we suggest that when predicting is_canceled, learning rate can be adjusted to 1.5, n_estimators can be adjusted to 700, and max_depth can be set to 9.

4. Conclusion and future work

In this project, we improved our performance step by step. To have better performance, we found model selection and data analysis were quite important, following are the three main changes we made to get the higher score.

First, we used all features to predict “revenue” (not adr) and “is_canceled” and then directly multiplied “revenue” with “is_canceled” to get expected profit. ***Public score: 0.37***

Secondly, we drop some features which are less correlated with “adr” and “is_canceled”, and also change to predict “adr” (not revenue) and “is_canceled”. ***Public score: 0.37 -> 0.34***

Finally, we cleaned the outlier in the dataset. ***Public score: 0.34 -> 0.31***

However, we found our private score is a little bit lower than what we expected. In fact, the reason would come from overfitting. In the future, we also want to try blending to improve our performance, meanwhile, we have to pay more attention to the overfitting problem as well.

Recommendation model: XGBoost for "adr", XGBoost for "is_canceled"

	Rank	Score
Public scoreboard	19 / 148	0.315789
Private scoreboard	34 / 148	0.389610

XGBoost	
Pros	Cons
*Can get higher performance *The speed is fast	*When n_estimator too high, often get overfitting result

Work division:

Chi-luen Feng (b05701113)	Code for predict revenue / Part 1, 3 for report
Ching-wei Yang (r09942160)	Data preprocessing / Part 2 for report
Yu-chieh Huang (r09942076)	Code for predict revenue / Part 4 for report