Machine Learning Engineer Nanodegree

Capstone Project Report

Xirui Zhong

05/29/2021

I.Definition

Project Overview

Have you ever book a hotel room but canceled it later? Did you hear the complaint from a hotel manager when he/she has prepared everything but the visitor was absent? Is there a way to predict whether the customer is going to cancel the reservation? Let's use machine learning to help the manager!

This research is going to use machine learning to solve predict whether a customer may cancel his/her room reservation. The data set was generated from https://www.kaggle.com/jessemostipak/hotel- booking-demand. It is originally from the article Hotel Booking Demand Datasets, written by Nuno Antonio, Ana Almeida, and Luis Nunes for Data in Brief, Volume 22, February 2019.

Problem Statement.

I will use the collected data set which contains booking information for a city hotel and a resort hotel, and includes information such as when the booking was made, length of stay, the number of adults, children, and babies, among other things to train a

model. The goal is to predict whether the customer may cancel the reservation by using some relevant variables with machine learning.

Metrics

The very first metric is the total accuracy. Due to the previous attempt, I will pay more attention to the true-positive value of the prediction result, which is the number of canceled room that are predicted as canceled. What is more, since the data classes are imbalanced (there are much more not canceled bookings than that were canceled), I will also take precision, recall, and F1 score into consideration.

II. AnalysisData Exploration

Data Exploration

The original dataset looks like below:

	hotel	is_canceled	lead_time	Unnamed: 3	Unnamed: 4	arrival_date_week_number	Unnamed: 6	stays_in_weekend_nights	stays_in_week_nights	adults	children
0	Resort Hotel	0	68	NaN	NaN	27	NaN	0	4	2	0
1	Resort Hotel	0	14	NaN	NaN	27	NaN	0	2	2	0
2	Resort Hotel	0	10	NaN	NaN	27	NaN	0	2	2	2
3	Resort Hotel	0	9	NaN	NaN	27	NaN	0	1	2	0
4	Resort Hotel	0	51	NaN	NaN	28	NaN	1	3	2	0

babies	meal	Unnamed: 13	reserved_room_type	customer_type	adr	total_of_special_requests
0	BB	NaN	D	Transient	97.00	3
0	BB	NaN	A	Transient	98.00	1
0	BB	NaN	G	Transient	153.00	0
0	BB	NaN	C	Transient	94.71	0
0	BB	NaN	G	Transient	117.81	2

There are in sum 13 meaningful features: *hotel, lead_time, arrival_date_week_number, stays_in_weekend_nights, stays_in_week_nights, adults,*

children, babies, meal, reserved_room_type, customer_type, adr, total_of_special_requests.

The target variable is *is_canceled*.

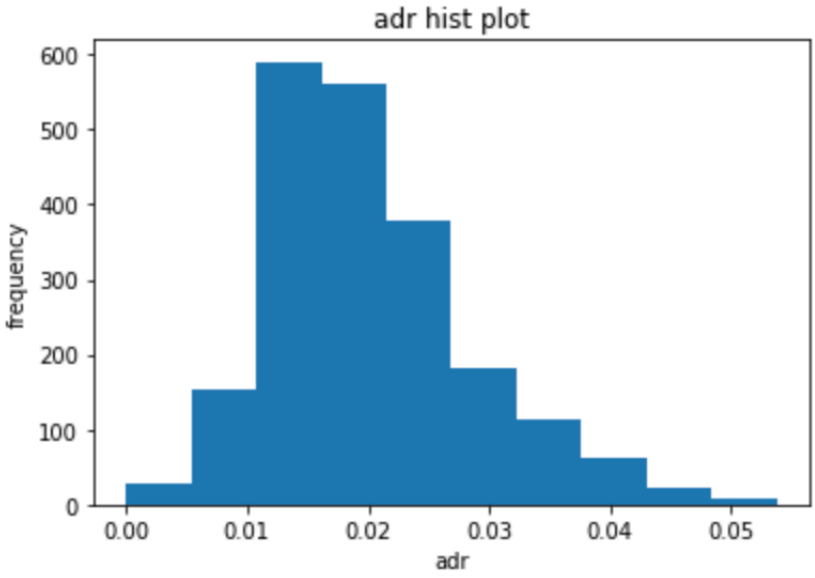
There are 2097 instances in total, with 501 of them canceled and 1596 not canceled.

Within all the features, *hotel, meal, reserved_room_type, customer_type* are categorical, and the others are numerical.

There are several missing values (NAN) but no outliers.

Exploratory Visualization

One important feature is 'adr', Average Daily Rate. Obviously, extremely high room rates may highly effect whether a customer will cancel the booking. Those observations may be outliers. So I normalize 'adr' and plot it to see if there are any outliers:



We can see in the figure that the data is quite average and there are no outliers.

Algorithms and Techniques

I will first transform categorical variables to numerical. Then, check if there are any highly correlated features by looking at the correlation matrix.

I will use XGBoost machine learning models to train and test the data set with all features as factors. I will try several kinds of models and change the hyperparameters until I find the best model to fit.

Next, I will do some transformation with several features, to see what may happen to the prediction. I will add two features: *stays_in_night = stays_in_weekend_nights + stays_in_week_nights, total = adults + children + babies*. I will use these 2 features instead of the original 5 (*stays_in_weekend_nights*, *stays_in_week_nights, adults, children, babies*) and train the model again. See if there is any difference.

Finally, I will reduce the number of features and only keep the ones that are most related to the target variable. See if the prediction result will be better.

In the end, I will select the best model (with features), and crossing compare it with the benchmark result.

Benchmark

I before made a naive Bayes classifier assuming multivariate normal distributions. I set the prior probability of sampling a canceled booking as pi1= 0/25, and the prior probability of sampling a not canceled booking as pi0 = 0.75, since the rooms that were canceled in the data set is approximately 1/4 of total. The prediction result is like:

```
##        is_canceled
## nbayes    0    1
##       0 1507  414
##       1   89   87
```

In this model, accuracy = 0.76, precision = 0.49, recall = 0.17, F1 = 0.26.

III. Methodology

Data Preprocessing

First, I delete all useless columns, and then change all categorical variables to numerical and delete rows with missing values. The data look like this now:

	hotel	is_canceled	lead_time	arrival_date_week_number	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	meal	reserved_room_type	customer_type	adr	total_of_special_requests
0	0	0	68	27	0	4	2	0	0	0	3	2	97.00	3
1	0	0	14	27	0	2	2	0	0	0	0	2	98.00	1
2	0	0	10	27	0	2	2	2	0	0	6	2	153.00	0
3	0	0	9	27	0	1	2	0	0	0	2	2	94.71	0
4	0	0	51	28	1	3	2	0	0	0	6	2	117.81	2

Next, I check the correlation matrix of the data to see if there are any highly related features. The answer is no.

```
In [17]: corr>0.5

Out[17]: array([[ True, False, False, False, False, False, False, False, False,
                  False, False, False, False, False],
                [False,  True, False, False, False, False, False, False, False,
                  False, False, False, False, False],
                [False, False,  True, False, False, False, False, False, False,
                  False, False, False, False, False],
                [False, False, False,  True, False, False, False, False, False,
                  False, False, False, False, False],
                [False, False, False, False,  True, False, False, False, False,
                  False, False, False, False, False],
                [False, False, False, False, False,  True, False, False, False,
                  False, False, False, False, False],
                [False, False, False, False, False, False,  True, False, False,
                  False, False, False, False, False],
                [False, False, False, False, False, False, False,  True, False,
                  False, False, False, False, False],
                [False, False, False, False, False, False, False, False,  True,
                  False, False, False, False, False],
                [False, False, False, False, False, False, False, False, False,
                   True, False, False, False, False],
                [False, False, False, False, False, False, False, False, False,
                  False,  True, False,  True, False],
                [False, False, False, False, False, False, False, False, False,
                  False, False,  True, False, False],
                [False, False, False, False, False, False, False, False, False,
                  False,  True, False,  True, False],
                [False, False, False, False, False, False, False, False, False,
                  False, False, False, False,  True]])
```

Finally, I myself add 2 else variables: *stays_in_night*, *total*. They represent the total night customers plan to stay and the total number of customers of a booking, respectively.

stays_in_night = stays_in_weekend_nights + stays_in_week_nights, total = adults + children + babies

The data is good now.

Implementation

The implementation of training and testing is quite simple.
Read the preprocessed data. Split them into training set, test set, and validation set. Split X (factors) and Y (target). Upload the data to S3. Now, we can start to build the model using XGBoost.
One thing important is that, in order to find out the best kind of model to fit, I firstly use all the original 13 features to train. So in this attempt, I drop the two added columns, *stays_in_night, total*. In the next attempt, I keep *stays_in_night & total* and drop *stays_in_weekend_nights*, *stays_in_week_nights, adults, children & babies*. In the final attempt, I only keep '*hotel', 'is_canceled', 'lead_time', 'arrival_date_week_number', 'reserved_room_type', 'customer_type', 'adr', 'stays_in_night',* and *'total'*.

Then, create a xgb estimator and set the hyperparameters. Fit the XGBoost model, test it with the test set, and check the prediction result. I pay more attention on the true-positive value, since precision, recall, and F1 score are all related with it. And also check the total accuracy. I write a *result* function to achieve the values.

```
def results(predictions, test_Y):
    tp = fp = fn = tn = 0

    for i in range(len(test_Y)):
        # true positive
        if test_Y[i]==predictions[i]==1: tp = tp+1
        # true negative
        elif test_Y[i]==predictions[i]==0: tn = tn+1
        # false negative
        elif test_Y[i]==1: fn = fn+1
        # false positive
        else: fp = fp+1

    return tp, fp, fn, tn
```

Refinement

In the first attempt, I use the original 13 features to train. The initial model is:

```
xgb.set_hyperparameters(max_depth=5, # change these set_hyperparameters to see the influence
                        eta=0.2,
                        gamma=4,
                        min_child_weight=5,
                        subsample=0.8,
                        silent=0,
                        #num_class = 2,
                        objective='binary:logistic', # try different models
                        early_stopping_rounds=10,
                        num_round=100)
```

The initial parameters are chosen like this because there are only about 1500 instances for training, so the model may be quite small.

I change the hyperparameters in turn to find the model that performs the best.

The factor **objective** is the most important. I keep the other hyperparameters and try all suitable kinds of **objective**: 'reg:linear', 'reg:logistic', 'binary:logistic', 'count:poisson', 'multi:softmax', 'multi:softprob', 'rank:pairwise'.

None of them work well. The best one with the highest accuracy is 'multi:softmax'.

Then, I keep objective = 'multi:softmax', and make some refinement with other parameters: **max_depth, eta, min_child_weight, num_round**. Some of them do affect the result much, and I keep the one with the highest prediction accuracy and the smallest size in the end. It is with these hyperparameters:

```
(max_depth=4, # change thes
 eta=0.1,
 gamma=4,
 min_child_weight=6,
 subsample=0.8,
 silent=0,
 num_class = 2,
 objective='multi:softmax',
 early_stopping_rounds=10,
 num_round=80)
```

In the second attempt, just as mentioned before, I transform some of the features. I keep *stays_in_night & total* and drop *stays_in_weekend_nights*, *stays_in_week_nights*, *adults*, *children & babies*. All the hyperparameters are the same as before.

In the third attempt, I eliminate half of the features which are not significantly related with the target variable, and only keep '*hotel', 'is_canceled', 'lead_time', 'arrival_date_week_number', 'reserved_room_type', 'customer_type', 'adr', 'stays_in_night',* and *'total'*. Train the model again. Compare the results.

IV. Results

Model Evaluation and Validation

In the first step, I keep all other parameters and try all suitable **objective**. The predictions are not good for canceled rooms, since True-Positive is always 0. The results are as shown below:

objective = binary: logistic

```
# How many canceled rooms are predicted correctly?
tp, _, _, _ = results(predictions, test_Y)
tp
```

```
0
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
print('accuracy:', accuracy_score(test_Y, predictions))
```

```
accuracy: 0.6222222222222222
```

objective = multi:softprob

```
tp
```

```
0
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
print('accuracy:', accuracy_score(test_Y, predictions_1))
```

```
accuracy: 0.6126984126984127
```

objective = multi:softmax

```
# How many canceled rooms are predicted correctly?
tp, _, _, _ = results(predictions, test_Y)
tp
```

```
0
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
print('accuracy:', accuracy_score(test_Y, predictions))
```

```
accuracy: 0.6634920634920635
```

objective = reg:linear

```
# How many canceled rooms are predicted correctly?
tp, _, _, _ = results(predictions, test_Y)
tp
```

```
0
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
print('accuracy:', accuracy_score(test_Y, predictions))
```

```
accuracy: 0.6222222222222222
```

objective = reg:logistic

```
# How many canceled rooms are predicted correctly?
tp, _, _, _ = results(predictions, test_Y)
tp
```

```
0
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
print('accuracy:', accuracy_score(test_Y, predictions))
```

```
accuracy: 0.6126984126984127
```

objective = rank:pairwise

```
# How many canceled rooms are predicted correctly?
tp, _, _, _ = results(predictions, test_Y)
tp
```

```
0
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
print('accuracy:', accuracy_score(test_Y, predictions))
```

```
accuracy: 0.2571428571428571
```

objective = count:poisson

```
# How many canceled rooms are predicted correctly?
tp, _, _, _ = results(predictions, test_Y)
tp
```

```
0
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
print('accuracy:', accuracy_score(test_Y, predictions))
```

```
accuracy: 0.5428571428571428
```

And then, I make some refinement with other parameters while keeping objective = 'multi:softmax'. The test results are as shown:

```
s(max_depth=5, # change these
  eta=0.2,
  gamma=4,
  min_child_weight=5,
  subsample=0.8,
  silent=0,
  num_class = 2,
  objective='multi:softmax',
  early_stopping_rounds=10,
  num_round=100)
```

```
true-positive 0
accuracy: 0.6634920634920635
```

```
s(max_depth=6, # change these
  eta=0.2,
  gamma=4,
  min_child_weight=4,
  subsample=0.8,
  silent=0,
  num_class = 2,
  objective='multi:softmax', #
  early_stopping_rounds=10,
  num_round=150)
```

```
true-positive 0
accuracy: 0.4507936507936508
```

```
s(max_depth=4, # change these
 eta=0.15,
 gamma=4,
 min_child_weight=6,
 subsample=0.8,
 silent=0,
 num_class = 2,
 objective='multi:softmax',
 early_stopping_rounds=10,
 num_round=150)
```

```
true-positive: 0
accuracy: 0.7841269841269841
```

```
(max_depth=4, # change the
 eta=0.1,
 gamma=4,
 min_child_weight=5,
 subsample=0.8,
 silent=0,
 num_class = 2,
 objective='multi:softmax'
 early_stopping_rounds=10,
 num_round=100)
```

```
true-positive: 0
accuracy: 0.7841269841269841
```

```
(max_depth=3, # change thes
 eta=0.1,
 gamma=4,
 min_child_weight=6,
 subsample=0.8,
 silent=0,
 num_class = 2,
 objective='multi:softmax',
 early_stopping_rounds=10,
 num_round=100)
```

```
true-positive: 0
accuracy: 0.4158730158730159
```

```
(max_depth=4, # change thes
 eta=0.1,
 gamma=4,
 min_child_weight=6,
 subsample=0.8,
 silent=0,
 num_class = 2,
 objective='multi:softmax',
 early_stopping_rounds=10,
 num_round=80)
```

```
true-positive: 0
accuracy: 0.7841269841269841
```

Obviously, the hyperparameters should be like the last one.

Next, I succinct *stays_in_weekend_nights, stays_in_week_nights, adults, children, babies,* and substitute them with *stays_in_night, total.* The result is:

```
predictions = pd.read_csv(os.path.join(data_dir, 'test.csv.out'), header=None)
predictions = [round(num) for num in predictions.squeeze().values]
tp, _, _, _ = results(predictions, test_Y)
tp
```

```
0
```

```
print('accuracy:', accuracy_score(test_Y, predictions))
```

```
accuracy: 0.7841269841269841
```

There is no difference with the last one.

Finally, I delete half of the features and only keep the ones that are most related with *is_canceled.* The prediction result is as shown:

```
predictions = pd.read_csv(os.path.join(data_dir, 'test.csv.out'), header=None)
predictions = [round(num) for num in predictions.squeeze().values]
tp, _, _, _ = results(predictions, test_Y)
tp
```

```
0
```

```
print('accuracy:', accuracy_score(test_Y, predictions))
```

```
accuracy: 0.3873015873015873
```

Justification

We can see that the true-positive value is always 0. The results have shown that all the predictions have failed, at least for canceled rooms. The accuracy of predicting a not canceled room is good though. But in general the predicted results, especially for canceled rooms, cannot be trusted.

The best XGBoost model is the one using all 13 original features with these hyperparameters:

```
(max_depth=4, # change thes
 eta=0.1,
 gamma=4,
 min_child_weight=6,
 subsample=0.8,
 silent=0,
 num_class = 2,
 objective='multi:softmax',
 early_stopping_rounds=10,
 num_round=80)
```

The prediction using this model has accuracy 0.78, while its precision, recall, F1 score are all 0 since its true-positive value is 0.

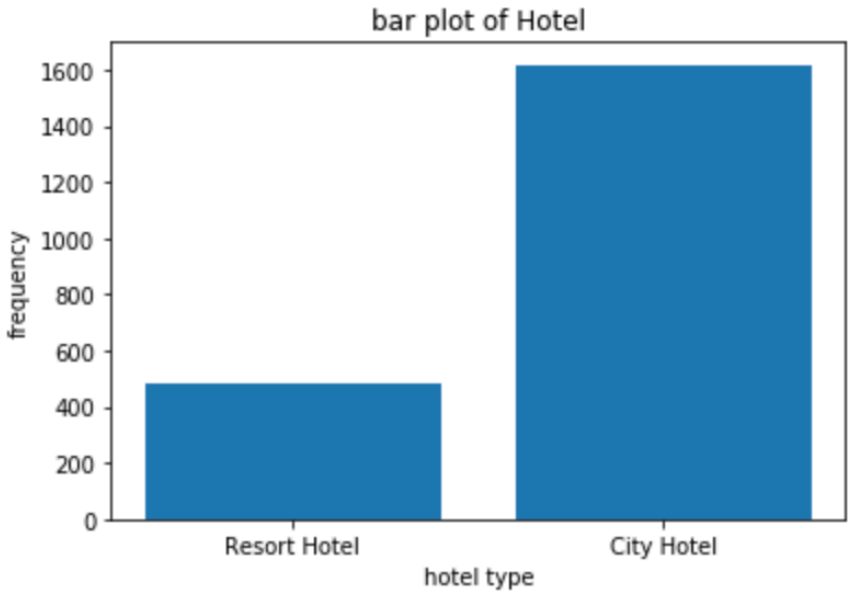
That means, although this model is not good for predicting canceled rooms, it is good for predicting not canceled rooms. If the prediction shows that a room will not be canceled, it is quite credible. However, if it shows that the room will be canceled, the hotel should not treat it as a canceled reservation immediately.

Comparing with the benchmark result, this prediction is not better. They neither can predict a canceled room well. However, the prediction created by my XGBoost model can predict a not canceled room better. In some ways, it is better than the benchmark result.

V. Conclusion

Free-Form Visualization

The prediction result is not satisfying. Actually we can foresee it when looking at the data. Let us look at the ‘hotel’ variable.



We can see that there are about 3 times as Resort Hotel reservations as City Hotel reservations. It is common sense that booking of a city hotel is much more difficult to predict since it is related with too many issues: local whether, plan changing, price comparing, etc. However, a booking of a resort hotel is seldom canceled since the holiday plan are usually carefully made and seldom changed.

Reflection

This is my first time to build a whole machine learning program. It is fun to try solving a realistic issue using what I learnt in courses.

The entire process I used for this project is quite simple. There are two main steps: data exploring and model building. Make the data suitable for training is very

important. The process of model selecting is kind of boring, since I have to train and wait for many times until I have tried all possible solutions. This is also the most difficult part of the program. I have spent above 3 days on training and selecting models. The most interesting part is comparing the results. Sometimes, a little change in parameters will make the prediction results differ a lot.

The final model and result actually do not fit my expectations for the problem. It cannot be used to predict canceled rooms. So it cannot be really used for hotels. It is a pity, but I have tried my best.

Improvement

The final result of this project is not satisfying for a hotel. I think there three main reasons: First, the data is too small. There are only about 2000 instances. We need much more data for training. Second, there are some other important features missing, such as local weather and hotel rankings. Third, the normal models are not suitable in this problem. In summary, we cannot find out a model to well predict whether a booking may be canceled with current data set and my current academic ability. The model I build should not be used. It may be a good new benchmark for another similar project.

I have searched for many papers and found that a model called “random forest tree” may fit this problem much better. However, I do not know how to do it now. Maybe I will try it someday in the future to have a better prediction way.