Kaggle InClass Competition Report

CompSci 671
Guanqi Zeng:linda_gqz

December 10 2021

1 Exploratory Analysis

As my purpose is to build models for best prediction, this section mainly deals with data cleaning and simple exploration of features' potentials for predictability.

1.1 Data Overview

There are 7471 observations and 21 features for prediction in the data set. After checking the data types, I changed *Price* from string to integer and turned *Month* into numbers instead of texts.

Among all these predictors, two of them are sticking out for having too many levels: *Bathrooms_text* and *Property_type*. Problems will arise if we use keep the original levels. My main concerns are 1)random inputs, 2)repeated information and 3)overfitting. Therefore, I decided to manually decrease the number of levels for these two features.

There are 23 levels in *Bathrooms_text*. It describes the number and condition (private or shared) of the bathroom of the place to be leased. Intuitively, I think this feature should bring us some insight for the airbnb availability, as I assume people would prefer to have private bathroom if they travel by themselves. I looked at the cross table of *Bathrooms_text* and *Decision*, the response variable indicating if the airbnb is still available or not. The result confirms my reasoning. There is a difference between the probability of airbnb availability for different numbers of bathrooms, but such difference is not very huge. It the airbnb even has no shared bathroom, it is not likely to be take. Below 5 bathrooms, the availability is rate does not change too much. When the number of bathrooms is above 5, however, the availability seems to be changing by random. If *Half-bath*, *Private half-bath*, *1 private bath* seem to have a higher probability to be taken. Therefore, I reduce the level of this feature into 4 categories: **5 and above**, **below 5**, **None**, and **Private**.

There are more than 50 levels for *Property_type*. This feature describes the type of housing of the airbnb. By looking at the entries, I don't think it is easy to combine them into larger categories. The inputs are too diverse, and sometimes even random. For example, I find entries like 'Treehouse' and 'Private room in bed and breakfast'. I look at the counts of each of these levels of this feature. It seems that a lot of the individual levels are below 2% of the entire data. I also look at the cross table and make a stacked bar chart to simultaneously visualize the proportion of availability of the airbnb in each level and the number of observations in each level. It seems that, although those levels do not seem to have too many individuals, they almost all have a high probability of being taken. For those major levels, for example ‘Entire house’ which has 1132 observations and ‘Entire guest suite’ which has 1118 observations, their probabilities of being taken are actually every close, and lower than those small levels. Therefore, I combined those smaller levels with numbers of observations below 2% individually into one category, **Special**. The second stacked bar chart indicates that this combination does cause loss of information. In addition, the variable importance of random forest models trained with and without this modification also show the same level of importance.

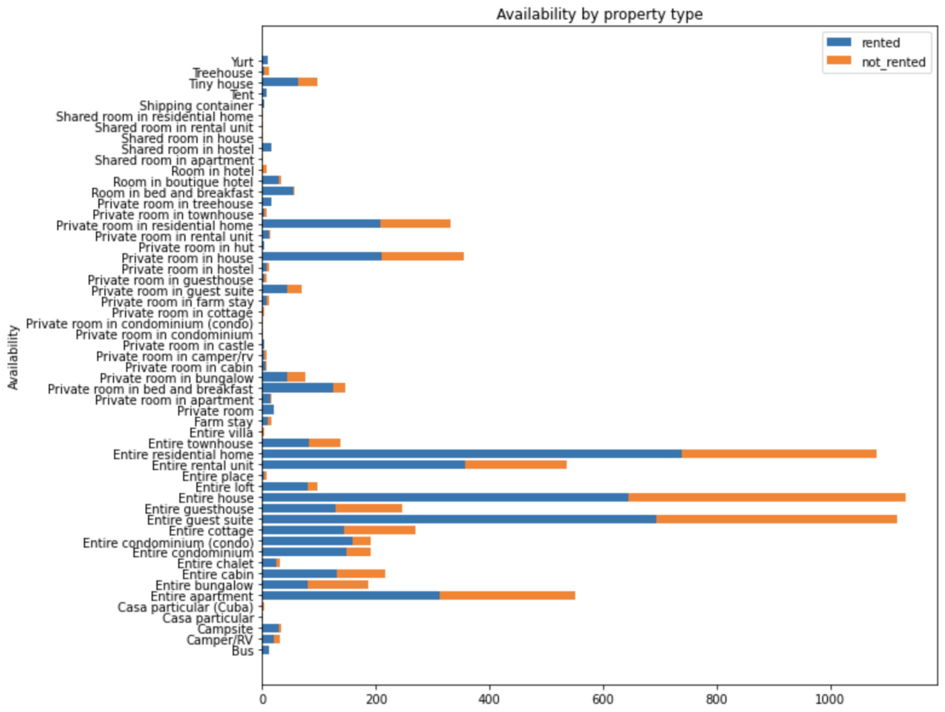

Figure 1: Stacked Bar Chart of Property Type and Availability

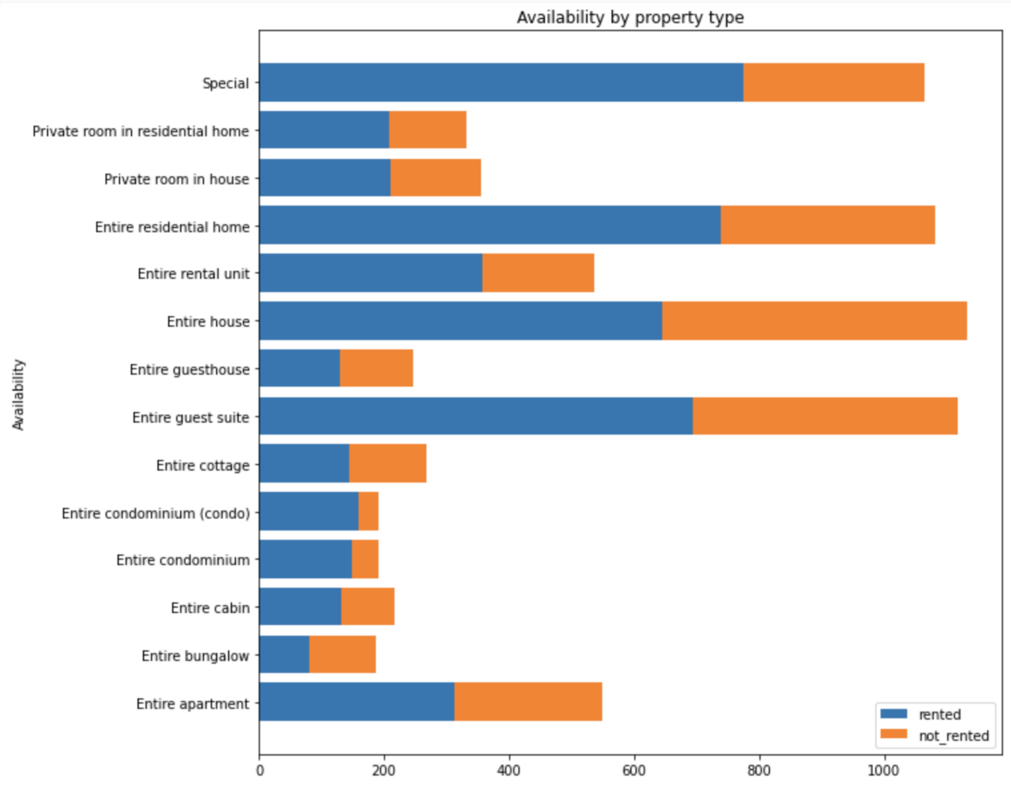

Figure 2: Stacked Bar Chart of Modified Property Type and Availability

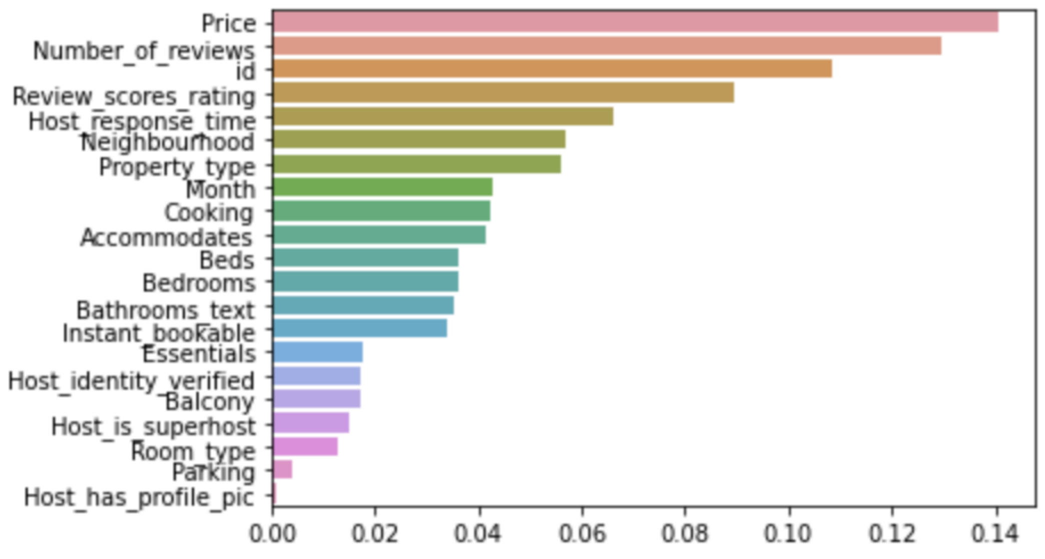

Figure 3: Variable Importance

1.2 Missing Data

There are 1605 observations that contain missing values in this data set, consisting of slightly more than 20% of the entire data. Deleting them is definitely going to impact our model training. Therefore, we need to deal with them to keep the most information.

The feature that contains the most missing data is *Host_response_time*. The cross table indicates that, different level of this feature does have different probability distribution of availability. The sooner the host response the guest, the more likely the airbnb is going to be taken. The feature that contains the second-most missing data is *Bedrooms*. The probability distribution of availability for each level is relatively stable, except that for some airbnb with high number of bedrooms. For *Review_scores_rating*, which has the third most missing values, the probability distribution for availability seems to be random.

Special case are *Host_is_superhost*, *Host_has_profile* and *Host_identity_verified*. All of them have the same number of missing entries. This is reasonable because if the any of them is missing, the other are likely to be missing.

We want to impute the missing values to make use of the most information. A mean or median imputation is definitely not going to work and, more importantly, at risk of changing the relationship between variables. Therefore, I choose to impute the data with regression model between the features to be imputed and the rest of the features, excluding *Decision*. I used 'IterativeImputer' from 'sklearn.impute'.

Before imputing the data, I encode them by 'category_encode', which I learned from a kaggle tutorial, to encode all the text data into ordinal data. Although this encoding might add random information in the data, for example, creating the level of extend on *Property_type*, that no ordinal meaning is embedded, this is only an issue when fitting regression model.

1.3 Ending remark

For the rest of the features, I choose to keep them as the models created later will 'self-decide' which features contribute little to prediction.

2 Methods

With 7471 observations, I randomly draw 6000 samples to be the training set, and the remaining are validation set. There is definitely a trade-off between splitting the data into train and validation and using the entire data, as the first option reduces the sample size and makes the data less representative, while the second option omits any opportunity to test the algorithms' performance on a new data set.

2.1 Models

As the data contains many categorical features, regression model may not be a good choice. The first two algorithms I use are Random Forest and AdaBoost. They are very easy to implement and efficient to train, and produce good performance. The last algorithm I use is a feed forward neural network. I choose it because it seems to produce good prediction results, although I've only seen situation in image classification (e.g.MNIST). I want to see how well this algorithm can be used for binary classification.

2.1.1 Random Forest

Random forest is an ensemble algorithms of trees and bagging. We're going to grow T trees and output the average result as our prediction. For a new observation, we use the majority vote of the trees on this observation as the prediction.

The algorithm is as follows. We draw T number of boostrap samples with sample size the same as the training set and with replacement. For each boostrap sample, we grow a tree with some splitting and stopping criteria. For each split, we choose m features at random out of the total p features. For each of the features, we evaluate the splitting criteria, and split on the best feature. If the node has less than some number, we stop. The model performs well as it averages many different yet accurate models to reduce the variance. Also, there is no pruning. This allows each tree to fit tightly and reduces bias.

Although it is a blackbox, we can check the variable importance to make some interpretation from variable importance.

2.1.2 AdaBoost

Adaboost is an ensemble of algorithms that make predictions by combining a bunch of weak learners. With a weak learning algorithm, a bunch of weak classifiers are produced. For each round, Adaboost constructs a discrete probability distribution the sample indices, 1 to n. We run this weak algorithm on this distribution, and produce the prediction. Then, we calculate the error. For each round, we increase the weight on the examples that are harder to classify. Therefore, Adaboost tends to correct itself for each round.

2.1.3 Feed Forward Neural Network

A Feed Forward Neural Network has many levels. For each level, a weight is put on the output of the last level. For binary classification, the final output will be passed into the sigmoid function to be turned to probability. Then, we convert the probability into predictions (1 or 0).

2.2 Training

The training process for Random Forest classifier and Adaboost classifier are embedded in the distributions of the algorithm. For Random Forest, the splitting criteria is the information gain or gini index. I choose gini index. We want to start by splitting the feature with the highest information gain. According to my final models for these two algorithms, the estimated run time for training Random Forest classifier is 8.34 μ s (wall time); the estimated run time for training Adaboost classifier is 6.68 μ s (wall time).

For Neural Network, the training process is a little complicated. We first need to decide the loss function. As this is a binary classification problem, our loss function is 'BCEWith-LogistcLoss', which represents binary cross entropy. Then, I use the Adam optimizer for optimizing this loss function, with learning rate 0.01 In the training loop, I set 50 epochs to go over the whole data set 50 times. For each time, I take 500 samples from my training samples one at a time, calculate the loss, and optimize it. The estimated run time for training my neural network is 6.2 μs.

2.3 Hyper-parameter Selection

For Random Forest, I tuned two parameters: number of trees grown and the maximum depth of the trees. I provide a grid search as choices. The values of the grid search is based several previous tries and the resulting accuracy. I also use 10-fold cross validation to take the parameters with the highest 10-fold cross validation accuracy.

For Adaboost, I tried three 'weak learning algorithms' and tune the parameters from the best one. I tried support vector classifier, logistic regression, and decision tree. It turned out that, the performance of svc is pretty bad. Also, it takes several minutes for training with only 2 svc. The logistic regression turned out to be too weak in turns of accuracy. In the end, I choose Decision tree. For hyperparameter tuning, I tuned the max depth of the decision tree using 10-fold cross validation. The best model has max depth of 10. Tuning the number of decision trees grown is also of interest.

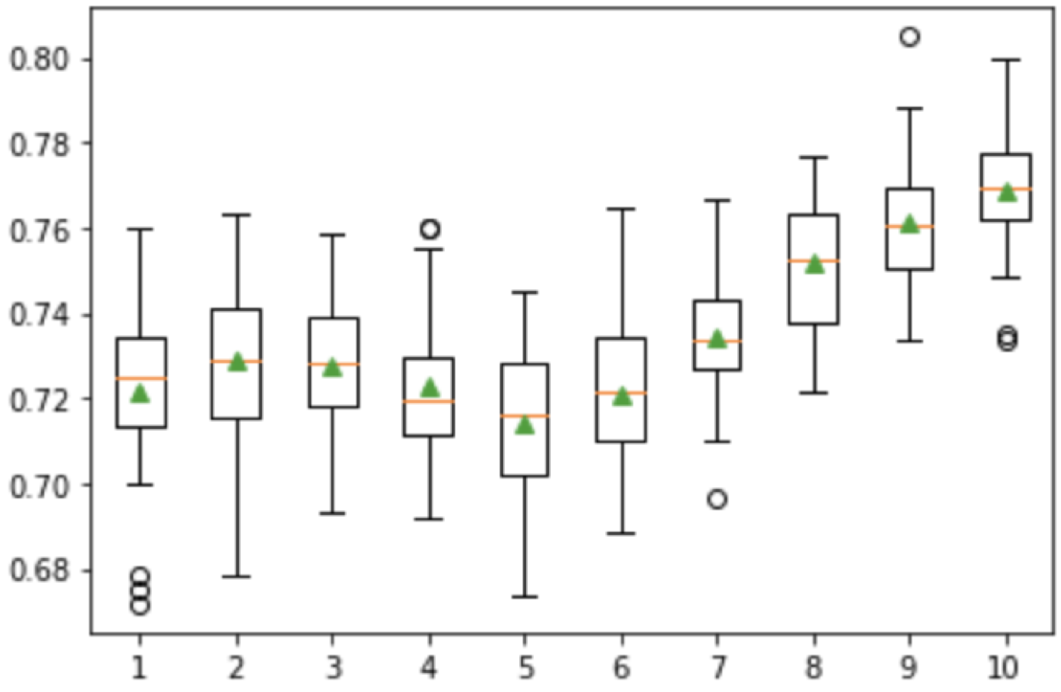

Figure 4: Adaboost: n estimators vs accuracy

For my feed forward neural network, aside from increasing the number of epochs to reach convergence and the learning rate to move faster towards convergence, I also tried modified the architecture of the network. Unfortunately, no matter how I modified the network, the validation accuracy is still low.

3 Results

In this section, I'll briefly present the performance of the models and describe the difficulties I've encountered during this project.

3.1 Prediction

My best model is the RandomForest model with max depth 50 and number of trees 500. This is the best parameters chosen by the grid search. The training accuracy is 100% and the validation accuracy is 80.5%. However, the best score from kaggle is the output of one of the classifiers picked from the grid search parameters, with max depth 100 and 100 trees. The training accuracy is 100% and the validation accuracy is 80.2%.

The performance of Adaboost is not as good as the Random Forest. The train accuracy is 100%, but the validation accuracy is 78.9%. The result seems to be pretty close to the tuned Random Forest classifier. The kaggle score for this classifier is also slightly lower.

The performance of the neural network is pretty bad. The training accuracy reaches 78.8% after 50 epochs, but the validation accuracy is below 60%.

3.2 Fixing Mistakes

The hardest part for this competition is to figure out why the results of the validation set is not always consistent. As I described above, this happens to my Random Forest classifiers. The second hardest part is to train the neural network. It takes me so much time but the result is very unsatisfactory.

There are two things I need to comment on the issue of inconsistency between validation accuracy and kaggle scores: the method I chose for imputation and the train-validation set split. As I mentioned before, there is a trade-off between using a validation set and using the whole data set. I highly doubt that, although the imputations based on regression model are reasonable based on the relationship in the data set, it actually strengthens some assumption we make on these relationship solely based on training set. This might cause larger different between the distribution of the training set and the test set. When we use the same imputer for the test set, the distribution of the test set has been changed further, and thus the prediction on kaggle is not consistant with the validation. For the data splitting, particularly, we loose some diversity of the data. I admit that, for easier training loops for the neural network, I choose X000 numbers of samples as the training. The resulting validation set is still large.

Citations

Data set: insideairbnb.com/get-the-data.html
Reading for RandomForest, Adaboost, and Neural Network:
usepackage_/dashboard/assignments/MFFCKDEQ2HdyYpb72
RandomForest tutorial (by Prashant Banerjee): https://www.kaggle.com/prashant111/random-forest-classifier-tutorial
Use GridSearch for tuning RandomForest (article written by Will Koehrsen):

https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
Code for tuning Decision Tree max depths in Adaboost (sample code written by Jason Brownlee): https://machinelearningmastery.com/adaboost-ensemble-in-python/
Code for binary classification using neural network: https://towardsdatascience.com/pytorch-tabular-binary-classification-a0368da5bb89

Code

See below.