Kaggle InClass Competition Report

CompSci 671
Quan Wang quanwang99

December 10th

Abstract

In the first part, an exploratory analysis is presented, which contains how I preprocess data and search for the correlation between target variable and features. I also present results on the feature importance and feature selection. In the second part, I introduce the evaluated models and the motivations of selecting them. One novel part I want to highlight is that I employ a powerful hyperparameter auto-tuner and provide a reusable and easy-to-config implementation. Finally, I report the prediction results. Code is available on my public git repository t.ly/52ij. (Please download the original document if you cannot click links.)

1 Exploratory Analysis

In this section, I will describe raw data processing procedure, feature selection by Recursive Feature Elimination [1], and report the feature importance.

1.1 Data Processing

1.1.1 Data Normalization

The target variable is "Decision", which is a binary variable. Among the 20 features, there are 10 categorical variables and 10 quantitative variables. For each categorical variable, I ordered groups by sorting the values contained in them. Next I assign labels (0, 1, 2, 3, ..., n) to n groups accordingly. Since the dataset has varying scales, I normalize the data to use a common scale, without distorting differences in the ranges of values or losing information.

1.1.2 Missing Data

Among 7471 samples, around 30% of them contain missing data, which we can't simply ignore. I propose two solutions. First, I use average values to replace the missing values. Second, I use simple linear regression to fit the missing values. Later analysis shows that the first method has a better performance. I also utilize models like CatBoost and XGBoost that can handle the missing values well.

1.1.3 Training and Testing Data split

To test whether the training overfits on the data, I need to split the training and testing data. In all the model I tested, all the training and testing data ratio is 8:2. When consider the k-fold cross validation, I set the $k = 5$ for all the evaluations.

1.1.4 Correlation between Target Variable and Features

After preprocessing, I draw the scatter plots of the target variable vs. each exploratory variable. We can't find a strong correlation between the target variable and each individual exploratory variable. Due to the page limit, the figure is omitted here. Instead, I reported the correlation matrix (see Figure 1). The value on each small box represents the Pearson correlation between two corresponding variables. The lighter the color, the larger the correlation. Host_response_time has the largest correlation with the target variable. Moreover, there are strong multicollinearity between several pairs of variables, that is, Room_type and Property_type, pairwise combination between Bathroom_text, Bedrooms, and Beds. The findings are intuitive since these features are similar to each other with our domain knowledge. We may consider drop or combine some features (like generate new variable beds/bedrooms = beds_per_room) to remedy the multicollinearity issue if these features were selected simultaneously in later procedure.

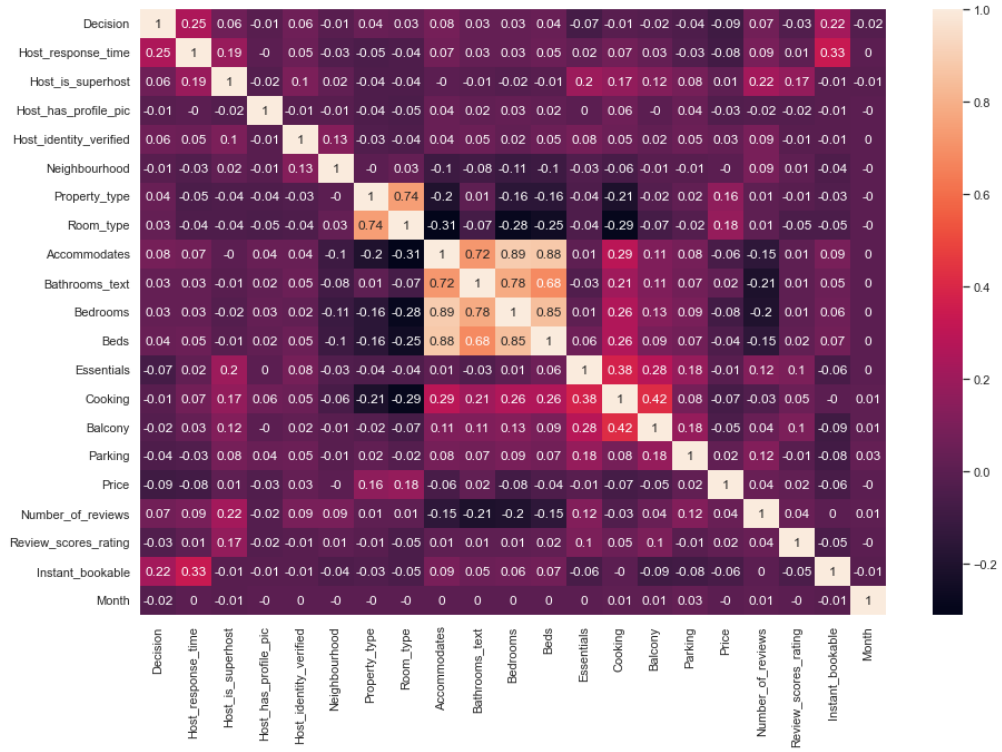

Figure 1: Correlation Matrix

1.2 Feature Selection

The dataset contains 20 features, which is a large number. Fewer features can allow machine learning algorithms to run more efficiently with less space or time complexity. Also, some machine learning algorithms may be misled by irrelevant input features, resulting in worse predictive performance. Thus, I will first use Recursive Feature Elimination (RFE) to select some important features. RFE is effective at selecting features in a training dataset that are more or most relevant in predicting the target variable. This is achieved by fitting the given machine learning algorithm used in the core of the model, ranking features by importance, discarding the least important features, and re-fitting the model. This process is repeated until a specified number of features remains. There are two important configuration options when using RFE. First, I use the Random Forest algorithm to choose the features. Second, I compare the accuracy of the training data when changing the number of features (from 2 to 20) to select. According to Figure 1, choosing more features yields a higher training accuracy. Since selecting 14 to 19 features yields similar classification accuracy, further analysis needed to determine which features are more important.

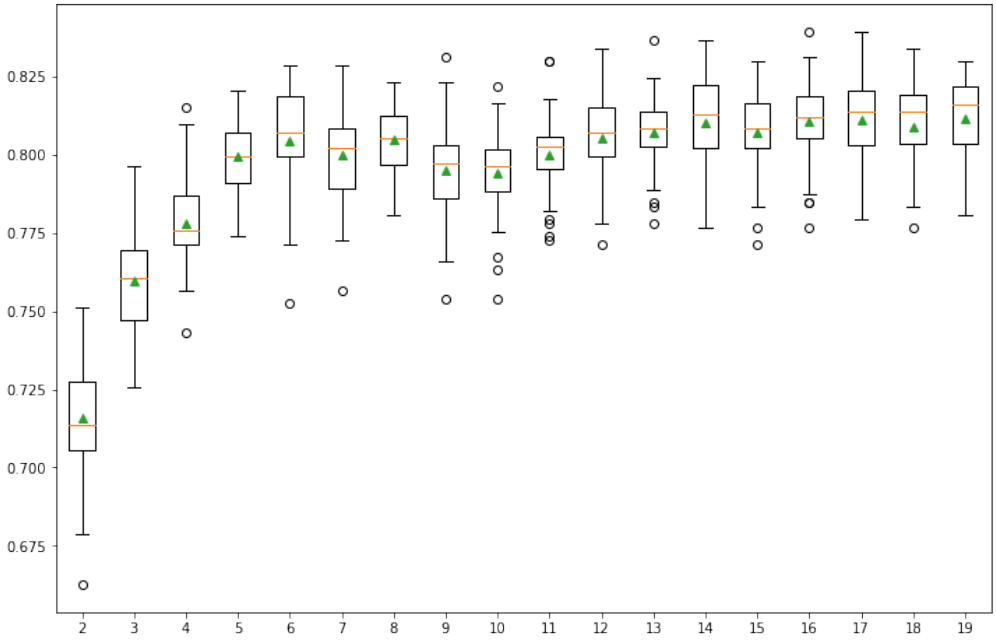

Figure 2: Box Plot of RFE Number of Selected Features vs. Classification Accuracy

1.3 Feature Importance

Feature importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable. From Figure 2, I observed that features with index 2, 6, and 14 (Host_has_profile_pic, Room_type, and Parking) have siginificantly lower score. The score for feature with index 2 (Host_has_profile_pic) is nearly 0. This is also intuitive since this is the easiest requirement, just upload a photo. I believe most hosts have profile picture, thus it can't influence the Decision much. The feature importance provides

some guidelines for me to drop several features when run the model later. I'll try to drop different features and then compare the accuracy.

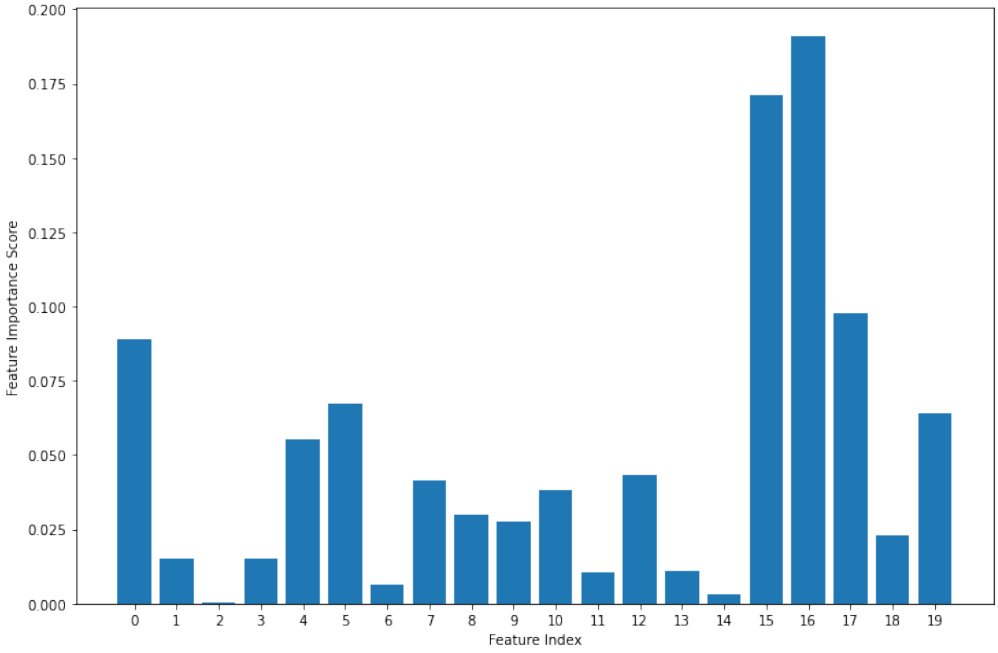

Figure 3: Bar Chart of DecisionTreeRegressor Feature Importance Scores

I remove irrelevant features according to their feature importance score - the lower-score feature will be removed first. Surprisingly, I found if I did not remove any feature, the average performance of all of my models performs best. The removing feature operation can be easily done in one-step with the config file provided in my github repository.

2 Methods

In this section, I introduce five implemented models, the training process, and hyperparameter selection with novel auto hyperparameter tuning framework.

2.1 Models

In the early stage, I tried two simple classifiers, that is, linear regression and decision tree. The main purpose is to check the preprocessed data works as expected. Another benefit is that they are highly interpretable, which helps me capture some intuition about the feature importance and correlation. Since we are working on a relatively small dataset, strong classifier like deep neural network and SVM may easily overfit. I believe the ensemble weak classifiers are the good choices for this dataset . Hence, I tried 4 different ensemble models, namely, AdaBoost, Random Forest, CatBoost, and XGBoost.

2.2 Simple Classifier

The simple models are for sanity-check on my data preprocess pipeline and help me grasp some intuition about the data.

Logistic Regression Simple logistic regression is easy to implement and almost does not require any hyperparameter tuning. Interpretability ia a benefit it provides. The weight of each feature dimension reflects the importance and the positive / negative correlation between features and outputs. I compared the interpretability results got from the logistic regression and the results got in Sec. 1, and found that there are some features like Host_Respone_Time is commonly marked as very important features.

2.3 Ensemble Models

Ensemble models empirically perform better on the small dataset. I evaluated four different ensemble models. This section introduces the motivation to select them.

AdaBoost Since the dataset contains 20 features and relatively small sample size, overfitting might be an issue. Adaboost is less prone to overfitting as the input parameters are not jointly optimized. Other advantages include low generalization error, easy to implement, works with a wide range of classifiers, and no parameters to adjust.

Random Forest Random Forest is suitable for situations when interpretability is not a major concern. It doesn't require normalization and can handle missing data. As a popular ML algorithm, there are many high-quality libraries and resources available. Also, versatility is another biggest advantage. It is useful in predicting both regression and classification tasks.

CatBoost CatBoost is a recently open-sourced machine learning algorithm from Yandex. It can work well with multiple categories of data to help solve a wide range of problems. It is especially powerful in two ways: It yields state-of-the-art results without extensive data training typically required by other machine learning methods, which is efficient.

It provides powerful out-of-the-box support for the more descriptive data formats that accompany many business problems. Since the Airbnb dataset contains a mixture of categorical and quantitative data, this model may provide good results.

XGBoost XGBoost has an in-built capability to handle missing values. It provides various intuitive features, such as parallelisation, distributed computing, cache optimisation, and more. It works well in small to medium dataset.

2.4 Training

In this section, I describe the training procedure for each algorithm. In the Table 1, I report the estimates of run time per round (total run time / rounds) required to train the model.

2.4.1 Training Procedure

Logistic Regression
1) Calculate Prediction: Assigning 0.0 to each coefficient and calculating the probability of the first training instance that belongs to class 0. Then plug in numbers and calculate a prediction.
2) Calculate New Coefficients: Calculate the new coefficient values using a simple update equation.
3) Repeat the Process

AdaBoost
1) Splitting the dataset into training and test samples.
2) Classifying the predictors and target.
3) Initializing Adaboost classifier and fitting the training data.
4) Predicting the classes for test set.
5) Attaching the predictions to test set for comparing.

Random Forest
1) Randomly pick N data points from the data set.
2) Based on these N data points, build a decision tree.
3) Choose parameter max samples if bootstrap = True (default), and repeat steps 1 and 2. Otherwise, the whole data set will be used to build each decision tree.
4) In classification problem, each decision tree in forest will predict that which category this new data point belongs to. And the final category is the one that wins the majority vote.

CatBoost The main idea of catboost is to sequentially combine many weak models (a model performing slightly better than random chance) and thus through greedy search create a strong competitive predictive model.

XGBoost The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction.

2.4.2 Training Time

Table 1: Estimates of training time for models. I run each models 100 times with different hyperparameters and computed the average time for per round. The sample range of hyperparameters are introduced in Sec. 2.5.

Model	Training Wall Time per Round (seconds)
Logistic Regression	2.6262 ± 0.4231
AdaBoost	3.0106 ± 0.2314
Random Forest	2.5594 ± 0.3661
CatBoost	3.2995 ± 0.1266
XGBoost	2.5920 ± 0.4311

2.5 Hyperparameter Selection

Inspired by the recent progress on AutoML [2], I implemented a hyperparameter autotuner with the Ray-tune framework [3]. This autotuner frees me from tedious hyperparameter tuning process with limited setup on a customized config file.

The hyperparameter tuning process can be modeled as a blackbox optimization problem,

$$\mathcal{H}^* = \arg\max_{\mathcal{H}} f(\mathcal{H}) \tag{1}$$

f here is the objective function (e.g., accuracy, F1 score) we need to evaluate on. We want to search for the best hyperparameter set $\mathcal{H}^*$ such that maximized the objective function f. However, the objective function is unknown (i.e., blackbox) to us. Thus, the blackbox optimization techniques like hill-climbing, Bayesian optimization can be applied. In my implementation, I adapt the HyperBand early stopping algorithm for BOHB.

The range of hyperparameters for all the 5 models are also provided in the config file. I provided the XGBoost config demo to interpretable the field of the hyperparameter configuration.

```
xgb_search_space = {
    "algo_wrapper_cls": XGBoostWrapper,
    "objective": "binary:logistic",
    "eval_metric": ["logloss", "error"],
    "max_depth": tune.randint(1, 20),
    "min_child_weight": tune.choice([1, 2, 3, 4, 5]),
    "subsample": tune.uniform(0.1, 1.0),
    "eta": tune.loguniform(1e-4, 1e-1),
    "irrelevant_features": default_irrelevant_features
}
```

Source Code 1: Hyperparameter Code Snippet for XGBooster

This code Snippet shows how I config the hyperparameters range of an algorithm. The algo_wrapper_cls defines the training model, and the irrelevant_features specifies the features that are not used as the input of the model. The other key-value pairs are the hyperparameter and their ranges. The meaning of tune model's functions can be found in the official document.

3 Results

3.1 Prediction

In this section, I report the best model accuracy on local test set and Kaggle test set.

Table 2 summarized the best experiment results I got on all the 5 models. The local test set is generated with 20% of training data and I get the Kaggle test set result by uploading the predicted results to Kaggle.

Table 2: Best experiment results on all the 5 models

Model	local test set	Kaggle test set
Logistic Regression	0.7313	0.690
Random Forest	0.8102	0.708
AdaBoost	0.7299	0.702
CatBoost	0.7687	0.702
XGBoost	0.7874	0.704

Table 3: Optimal hyperparameter for random forest (sklearn API)

Hyperparameter	n_estimator	criterion	max_depth	min_samples_split	min_sample_leaf	max_feature
Value	60	entropy	22	2	3	sqrt

The random forest model performs best on both local test set and Kaggle test set. The optimal hyperparameters are in Table 3.

To replicate the similar results, you only need to change the line 7 in run.py.

3.2 Miscellaneous Issues

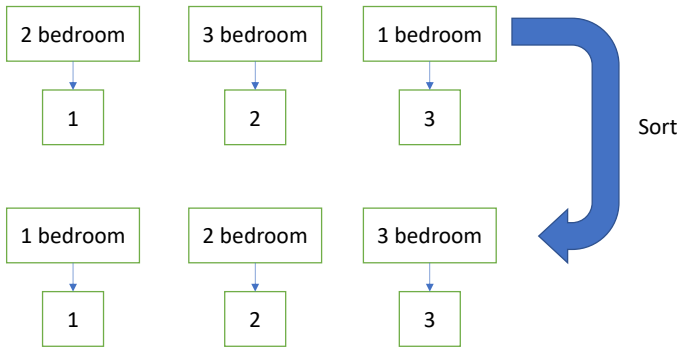

Figure 4: Handle the possible information loss when casting string to int.

3.2.1 Information Loss Caused by Casting

When we normalize the date, we need to cast all the string data to the numerical data. However, such cast may cause the information loss. For example, if we cast 1 bedroom, 2 bedroom, and 3 bedroom to 3, 1, 2, respectively. We will lose the quantitative relationship contained in the string. Thus, the order to map string to number is important. The simplest way to handle this small issue is sorting the string before the mapping. A demonstration is presented in Figure 4. However, this is just a workaround. For the case like one bedroom, two bedroom, three bedroom, simply sorting will not help.

3.2.2 "Smart" Missing Data Completion

In my final version of code, I used the average data to complete these missing data. In addition to average data imputation, I also tried KNN data imputation, linear regression data imputation. The idea of KNN imputation is fill the missing data with the average of k nearest neighbors' average on the missing dimension. The linear regression data imputation predicts the missing data field with all the data in other feature dimensions. However, none of these "smart" completion works better than the average completion. All of these "smart" imputers screwed up the final prediction results on both local test set and the test set on Kaggle.

References

[1] Xue-wen Chen and Jong Cheol Jeong. Enhanced recursive feature elimination. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 429–435. IEEE, 2007.

[2] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[3] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.