Airbnb Price Prediction Using Machine Learning

Linlin Li

November 16th
Link

1 Background

Airbnb is an online marketplace for vacation rentals that provides arrangements for lodging, primarily homestays, or tourism experiences. Homeowners can put their property online so that guests can pay to stay in them. The platform does not own any of the properties and does not host events. It acts as a broker and collects commissions from each booking. Although Airbnb and other websites may provide some general guidance, the price of each property is determined by its host. Appropriate prices are needed, because too high a price will result in a low booking rate, and too low a price will result in a loss of potential revenue. When determining the price, many factors should be considered, such as location, capacity, room type, etc.

In this project, I am going to look at Airbnb listings in Buenos Aires and trying to provide some exploratory analysis around predicting listing prices. First, I will use Exploratory Data Analysis (EDA) to get to know the data. This will help me get an initial sense of which variables are associated with price, and which variables to include in the model. Using the important variables identified in EDA, I will try several classification models attempting to predict the price. Finally, I will evaluate the performance of the model.

2 Exploratory Analysis

To begin with the EDA, first I got a summary of the variables using `pandas_profiling` [1] . For the full details, feel free to check the Appendix **??**. There are 9681 listings in the training set, which contains 24 features. The following are my findings:

- There are no missing values in the training set.
- `is_business_travel_ready` is the same for all observations, so it cannot provide any information for the price. I choose to remove it.
- There are many binary variables. I will convert them into dummy variables after EDA.
- `host_since` is a datetime column and should be converted into metric that measures the number of days the host has been on Airbnb. I used November 5th, 2020 to calculate this metric and named it `host_days_active`. Similarly, `last_review` is converted to `time_since_last_review`.
- `require_guest_phone_verification` and `require_guest_profile_picture` are highly correlated.

- `number_of_reviews` and `reviews_per_month` are highly correlated. This is reasonable because the latter is the "average value" of the former.
- `bedrooms`, `beds` and `bathrooms` are highly correlated. The number of beds has traditionally been a more high priority search parameter on Airbnb, as it is more relevant for the number of people accommodated than the number of bedrooms (and is still the second highest priority parameter when searching on the site. In addition, `guests_included` is correlated with these three variables.

2.1 Investigating Binary Variables

Next, I noted that a large amount of the variables are binary. I investigated the association with price via stacked bar plots.

Figure 1: Stacked bar plots between price and some binary variables

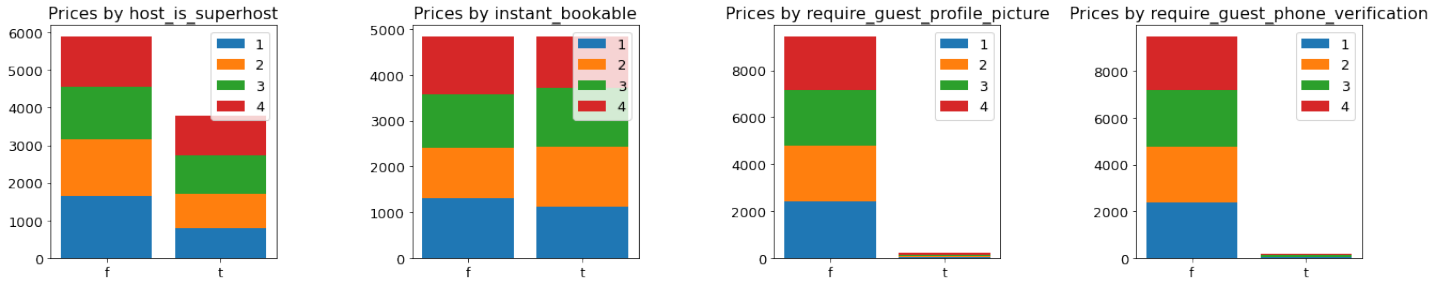


Figure 1 provide a great visualization on how a given binary variable is associated with changes in price. I saw the following findings from the graph:

- `host_is_superhost` seems to be a potential predictor because the distribution of price is different between the two categories of `host_is_superhost`.
- The difference in the distribution of price in the two categories of `instant_bookable` seems to be trivial, maybe `instant_bookable` cannot provide much information about price. I will drop it after EDA.
- The distribution of price for `require_guest_phone_verification` and `require_guest_profile_picture` are very similar. Since these two variables are highly correlated, we can simply drop one of them. I choose to drop `require_guest_profile_picture` after EDA because I think phone verification is more strict.

2.2 Investigating Categorical Variables

Several variables have multiple categories (`room_type`, `bed_type` and `cancellation_policy`). I can also visualize these with stacked bar plots, there will just be more bars than a binary variable.

Figure 2: Stacked bar plots between price and some categorical variables

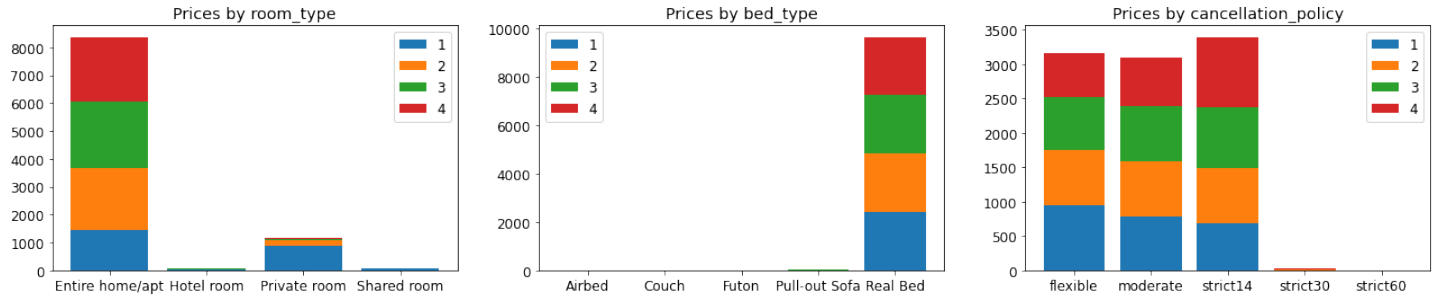


Figure 2 provide a great visualization on how a given categorical variable is associated with changes in price.

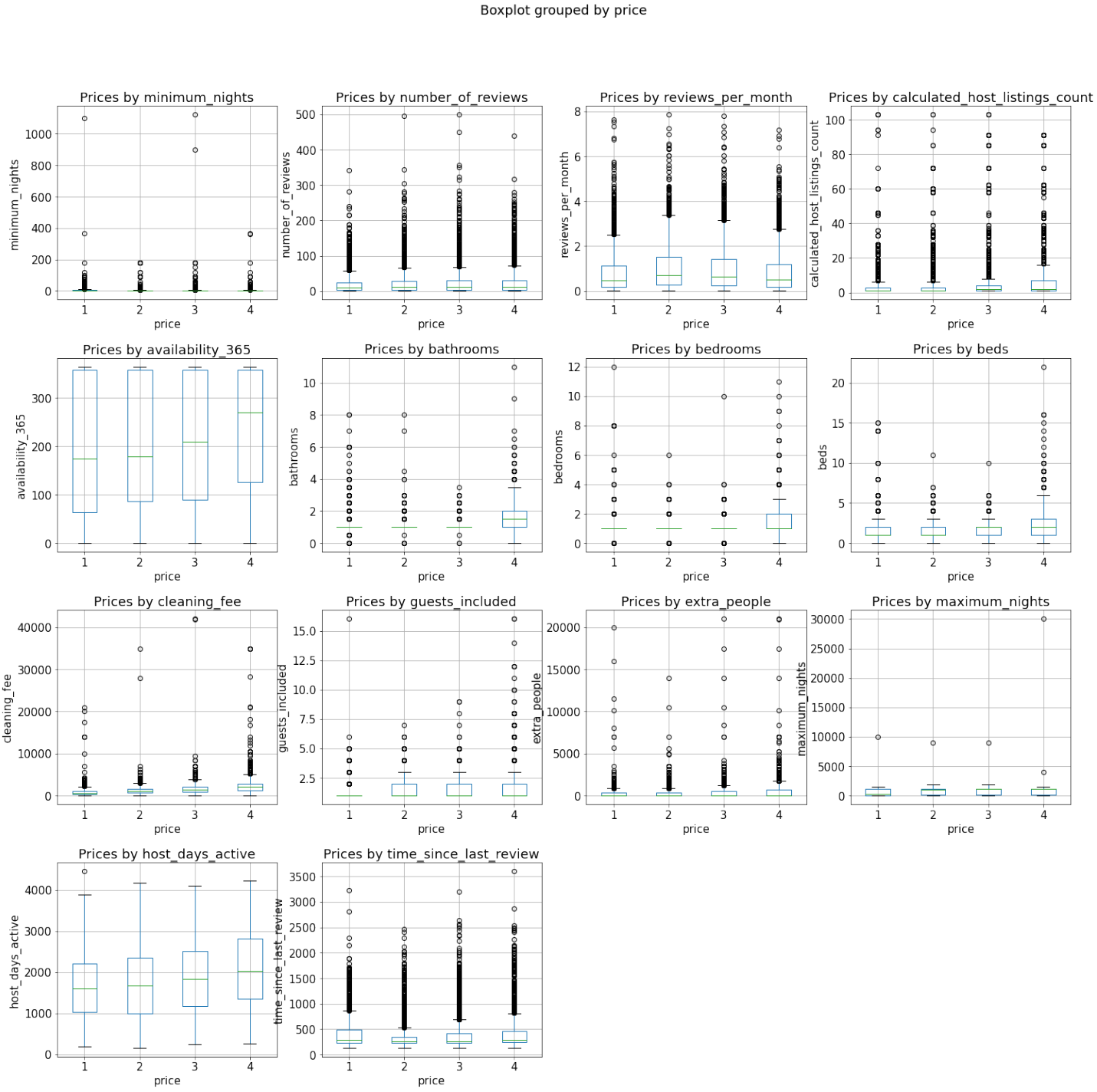
- Several of the categories for variables have very small sample sizes (`bed_type` and `cancellation_policy`).
  - For `cancellation_policy`, 'super_strict_30' and 'super_strict_60' only appear in 26 and 3 listings respectively, which are very small in the training set. Since the super strict options are only available to long-term Airbnb hosts and is invitation only, it is clear that the super strict options are stricter than 'strict_14_with_grace_period'. As the number of 'super strict' is too small, I will combine it with 'strict_14_with_grace_period' after EDA.
  - Similarly, for `bed_type`, I will combine 'Couch' and 'Airbed' into 'other' after EDA.
- It would make sense that `room_type` could be an important factor; perhaps certain rooms types are more private and therefore more expensive.
- The association between `cancellation_policy` and price does not seem so close, which is a bit counterintuitive, because customers tend to pay higher prices when choosing a more flexible policy.
- For `neighbourhood`, I also combine those classes with particularly small sample sizes (< 7) with 'other'.

2.3 Investigating Numerical Variables

Finally, certain variables are continuous in nature. We can investigate the relationship with price via boxplots. Figure 3 provide a great visualization on how a given numerical variable is associated with changes in price.

- `minimum_nights` and `maximum_nights` seem to provide very little information about price.
- It seems that `bathrooms`, `bedrooms`, `cleaning_fee`, `guests_included`, `availability_365`, `cleaning_fee`, and `calculated_host_listings_count` are influencing factors.

Figure 3: Boxplots between price and some numerical variables



3 Data Splits

After performing EDA, I divided the training set into two parts: about 70% of the training set (6776 records) was used for training, and the remaining 30% of the training set (2905 records) was reserved for testing the out-of-sample performance.

In the hyperparameter selection part, I used 10-fold cross-validation on my training set (6776 records) and divided it into 10 folds. Each time I set aside 1 fold for testing and trained the classifier on the other 9 folds and evaluated the categorization accuracy on the 1 fold. After adjusting the hyperparameters, I computed the out-of-sample accuracy of each of my models on my test set (2905 records).

4 Models

After cleaning and dropping columns, the available features in the model are:

- Room type
- Minimum and maximum nights stay
- Total number of reviews
- Average number of reviews left by guest each month
- How many listings the host is responsible for in total
- Number of days available to book in the next 365 days
- Whether or not a host is a superhost, has their identity verified (e.g. by verifying a phone number)
- The number of bathrooms, bedrooms, and beds
- Type of bed
- Cleaning fee and extra person fee
- The number of guests included in the booking fee
- The type of cancellation policy
- How many days the host has been listing on Airbnb
- Amount of time since the most recent reviews
- Neighborhood

The continuous variables were standardized using `scikit-learn`'s `StandardScaler()`. Categorical features were encoded into indicator variables using `pandas.get_dummies()`.

In order to have a basic sense of which algorithms to go. I first tried many baseline classification models using `compare_models()` from `PyCaret` [2], which is an end-to-end machine learning and model management tool that automates machine learning workflows and accelerate the experiment cycle exponentially. Due to limited time and computating resources, I used the default setting of `compare_models()`, where some models that require a long running time are prevented for comparison. Table 1 displays the results of models with default hyperparameters using 10-fold cross-validation on my training set (6776 records).

Random forest and some boosting algorithms seem to perform better than others. Thanks to the high-quality libraries available online, these algorithms are easy to train and use for prediction. In the next section, I will try these algorithms, and tune their hyperparameters.

	Model	Accuracy	AUC	Recall	Prec.	F1	TT (Sec)
rf	Random Forest Classifier	0.5403	0.7933	0.5420	0.5485	0.5432	0.2170
lightgbm	Light Gradient Boosting Machine	0.5331	0.7919	0.5346	0.5410	0.5359	0.1160
catboost	CatBoost Classifier	0.5319	0.7894	0.5337	0.5366	0.5334	3.9600
xgboost	Extreme Gradient Boosting	0.5298	0.7888	0.5310	0.5381	0.5328	1.4750
gbc	Gradient Boosting Classifier	0.5283	0.7844	0.5301	0.5396	0.5323	0.6360
et	Extra Trees Classifier	0.5252	0.7753	0.5270	0.5309	0.5271	0.1610
ada	Ada Boost Classifier	0.4993	0.7371	0.5012	0.5045	0.5009	0.0630
lr	Logistic Regression	0.4907	0.7564	0.4929	0.5011	0.4942	0.1970
lda	Linear Discriminant Analysis	0.4824	0.7474	0.4836	0.5232	0.4921	0.0200
ridge	Ridge Classifier	0.4777	0.0000	0.4805	0.4832	0.4780	0.0750
knn	K Neighbors Classifier	0.4604	0.7026	0.4616	0.4720	0.4618	0.2630
dt	Decision Tree Classifier	0.4451	0.6294	0.4463	0.4476	0.4452	0.0840
svm	SVM - Linear Kernel	0.4418	0.0000	0.4438	0.4474	0.4184	0.1200

Table 1: Baseline classification models. Results are based on 10-fold cross-validation on my training set (6776 records). Table is sorted by accuracy. Certain models are prevented for comparison because of their longer run-time.

5 Training

5.1 Random Forest Classifier

In the random forest classifier, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set [3]. When splitting each node during the construction of a tree, the best split is found from a random subset of variables [3]. The purpose of these two sources of randomness is to decrease the variance of the forest estimator [3]. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias [3]. The `scikit-learn` implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class [4].

It took about 6.58 seconds (wall time) to train a random forest classifier on the whole training set (9681 records).

5.2 Gradient Boosting Classifier

In boosting, the individual models are not built on completely random subsets of data and features but sequentially by putting more weight on instances with wrong predictions and high errors [5]. In each round of training, the weak learner is built and its predictions are compared to the correct outcome that we expect [5]. The distance between prediction and truth represents the error rate of our model, which can be used to calculate the gradient [5]. The gradient can be used to find the direction in which to change the model parameters in order to (maximally) reduce the error in the next round of training by "descending the gradient" [5].

In Gradient Boosting, we are combining the predictions of multiple models, we are not optimizing the model parameters directly but the boosted model predictions [5].

It took about 26.6 seconds (wall time) to train a gradient boosting classifier on the whole training set (9681 records).

5.3 XGBoost Classifier

XGBoost, also called Extreme Gradient Boosting, is a specific implementation of the Gradient Boosting method which uses more accurate approximations to find the best tree model [6]. While regular gradient boosting uses the loss function of the base model (e.g. decision tree) as a proxy for minimizing the error of the overall model, XGBoost uses the second-order derivative as an approximation, which tends to provide more information about the gradients [6].

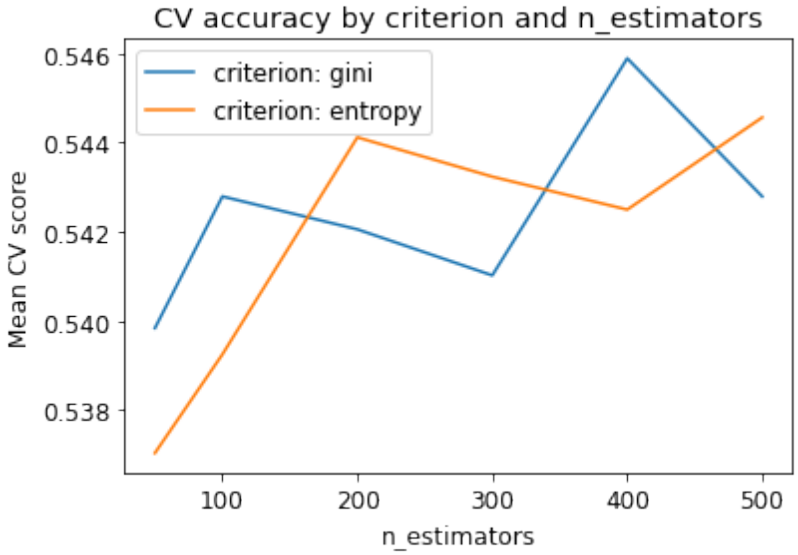
It took about 11.3 seconds (wall time) to train a XGBoost classifier on the whole training set (9681 records).

6 Hyperparameter Selection

6.1 Random Forest Classifier

In the random forest model, I tried to tune "criterion" and "n_estimators". I used grid search to get the accuracy using 10-fold cross-validation on my training set (6776 records). Figure 4 shows the cross-validation accuracy on my training set and these two hyperparameters.

Figure 4: Cross-validation accuracy using Random Forest Classifier on my training set (6776 records).

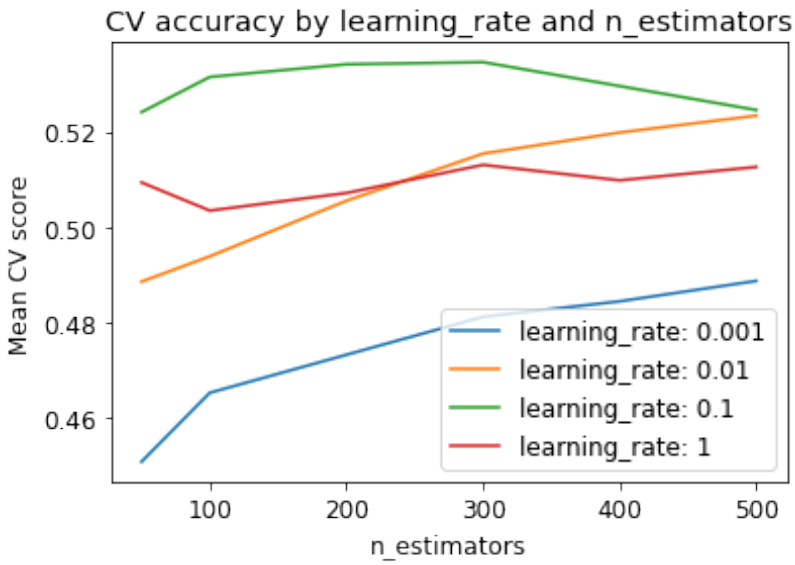


As can be seen in Figure 4, the best combinations of these two hyperparameters is "criterion = 'gini' " and "n_estimators=400".

6.2 Gradient Boosting Classifier

In the gradient boosting model, I tried to tune "learning_rate" and "n_estimators". I used grid search to get the accuracy using 10-fold cross-validation on my training set (6776 records). Figure 5 shows the cross-validation accuracy on my training set and these two hyperparameters.

Figure 5: Cross-validation accuracy using Gradient Boosting Classifier on my training set (6776 records).

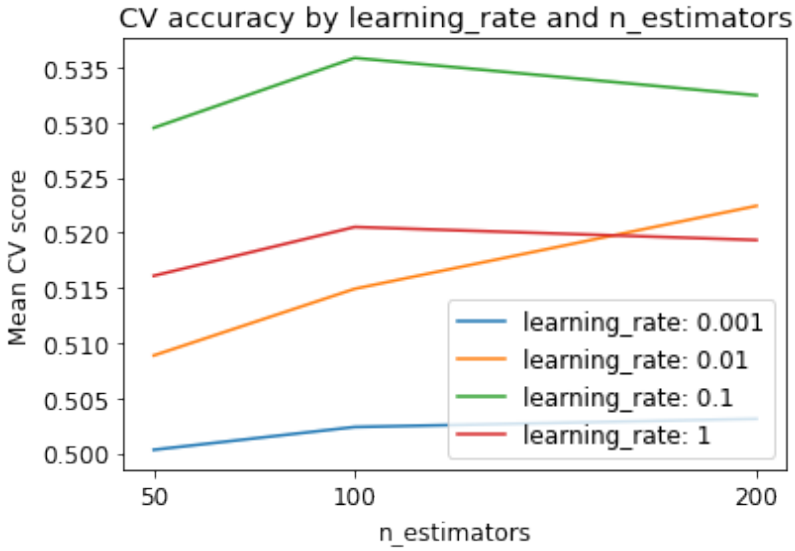


As can be seen in Figure 5, the best combinations of these two hyperparameters is "learning_rate = 0.1 " and "n_estimators=300".

6.3 XGBoost Classifier

In the XGBoost model, I tried to tune "learning_rate" and "n_estimators". However, due to limited computing resources, I cannot directly use grid search to get the results using 10-fold cross-validation on my training set (6776 records). In this case, I tried to manually calculated the 10-fold cross-validation accuracy for each combinations of hyperparameters. Figure 6 shows the cross-validation accuracy on my training set and these two hyperparameters.

Figure 6: Cross-validation accuracy using XGBoost Classifier on my training set (6776 records).



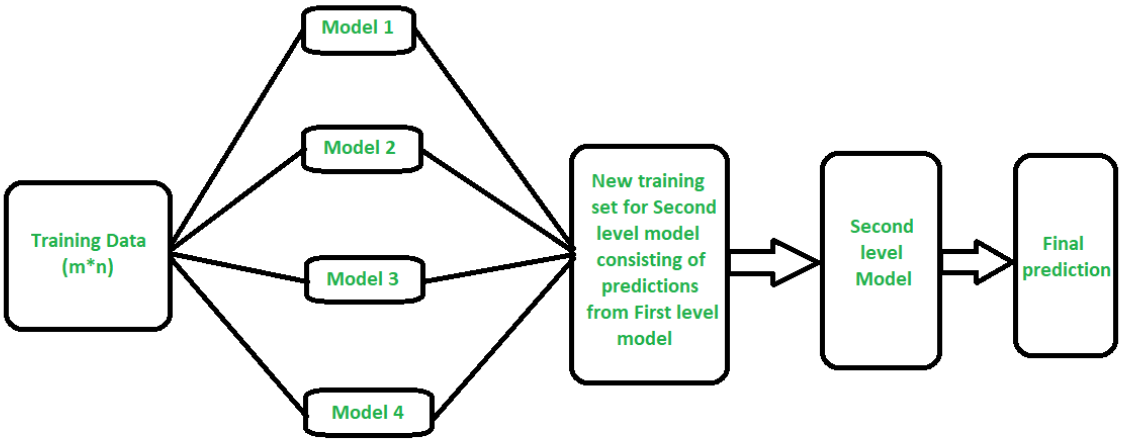
As can be seen in Figure 6, the best combinations of these two hyperparameters is "learning_rate = 0.1 " and "n_estimators=100".

7 Stacked Models

One possible way to improve the predictive accuracy is to combine different models. Stacking (sometimes called "stacked generalization") involves training a new learning algorithm to combine the predictions of several base learners [7]. First, the base learners are trained using the available training data, then a combiner or meta algorithm, called the super learner, is trained to make a final prediction based on the predictions of the base learners [7]. Figure 7 shows the struc Such stacked ensemble often outperforms any of the individual base learners (e.g., a single random forest) and has been shown to represent an asymptotically optimal system for learning [8].

I tried to stack the predictions from the top 7 classifiers in Table 1 up to get a final prediction using a linear model as the combiner. All of these 7 classifiers form the base layer of the stack, and their predictions are used as input to the meta model. It is very convenient to do this by `stack_models()` in `PyCaret`. Similarly, the output of this function is the result of the stacked models using 10-fold cross-validation.

Figure 7: Stacked Models



8 Errors and Mistakes

This competition is not that easy as it seems to be. For me, feature selection is the hardest part. In the original dataset, there are 24 features (except the response), including DateTime features, categorical features, and numeric features. Additionally, some features are highly correlated, which increases the demand for feature selection and increases the difficulty for modeling. If I delete too many features, I will lose too much information about prices; if I keep too many unnecessary features, the predictive accuracy may not be satisfactory.

9 Predictive Accuracy

My Kaggle username is Linlin Li. Table 2 displays the performance of my models.

	Models	CV Accuracy	Out-of-sample Accuracy	Test Accuracy*
rf	Random Forest Classifier	0.5494	0.5491	0.5643
gbc	Gradient Boosting Classifier	0.5331	0.547	0.5402
xgboost	Extreme Gradient Boosting	0.5342	0.5497	0.545
stacked5	Stacked model of 5 base learners	0.5502	0.559	0.5619
stacked7	Stacked model of 7 base learners	0.55	0.5608	0.5788
stacked10	Stacked model of 10 base learners	0.5561	0.5597	0.5691

Table 2: Categorization accuracy of models on the training set and the test set. Out-of-sample accuracy was obtained on my test set (2905 records). Note that test accuracy was obtained from Kaggle (based on 30% of test set). For each model, I've submitted several versions to Kaggle and the results here in test accuracy are the best among each model.

From Table 2, "stacked7" is probably my best model, which performs slightly better than other models on both my test set and the test set on Kaggle. We can visualize its performance on the out-of-sample data (2905 records) in Figure 8 and 9.

From the confusion matrix (Figure 8) of the model, we can see that the overall performance of the model seems to be good. And it performs better on class 1 and 4 than on class 2 and 3.

In a multi-class model, we can plot K number of ROC Curves for K number classes using One vs ALL methodology. For Example, if you have three classes named X, Y, and Z, you will have one ROC for X classified against Y and Z, another ROC for Y classified against X and Z, and the third one of Z classified against Y and X. As shown in Figure 9, AUC for each class is at least 0.73, which is much greater than 0.5, which means that there is at least 73% chance that the model will be able to distinguish each class from the other classes.

Figure 8: Confusion matrix of the stacked model of 7 base learners on my test data (2905 records).

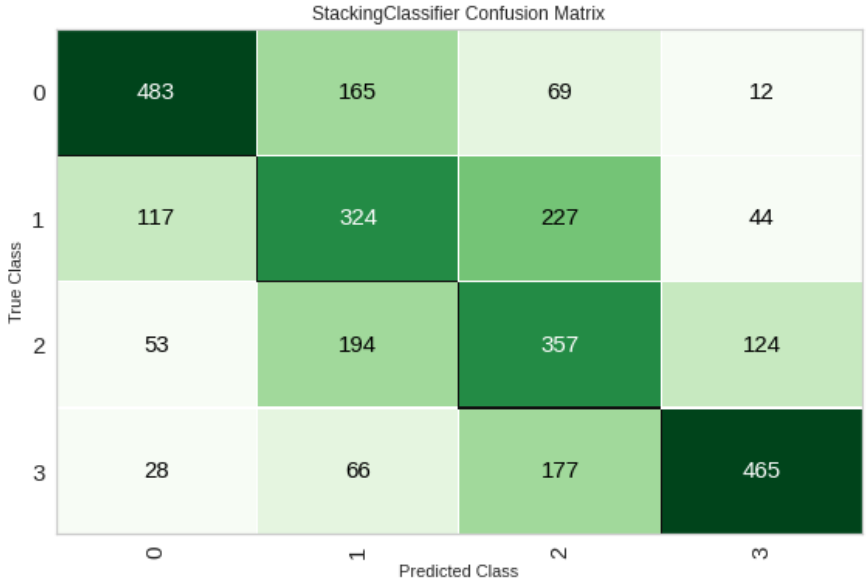
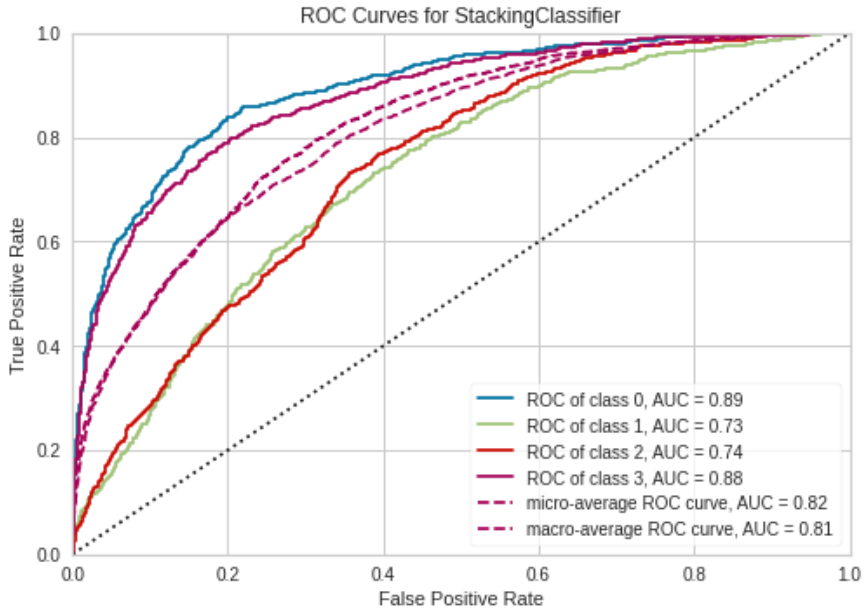


Figure 9: AUC curve for the stacked model of 7 base learners on my test data (2905 records).



References

[1] Simon Brugman. pandas-profiling: Exploratory Data Analysis for Python. https://github.com/pandas-profiling/pandas-profiling, 2019. Version: 2.X, Accessed: November 5, 2020.

[2] Moez Ali. *PyCaret: An open source, low-code machine learning library in Python*, July 2020. PyCaret version 2.1.

[3] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[6] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, and Yuan Tang. Xgboost: extreme gradient boosting. *R package version 0.4-2*, pages 1–4, 2015.

[7] Brad Boehmke and Brandon M Greenwell. *Hands-on machine learning with R*. CRC Press, 2019.

[8] Mark J Van der Laan, Eric C Polley, and Alan E Hubbard. Super learner. *Statistical applications in genetics and molecular biology*, 6(1), 2007.