Ensemble Project: Airbnb Price Prediction

Mengyu LIANG, Nhat Mai NGUYEN, Jinji SHEN and Vanshika SHARMA

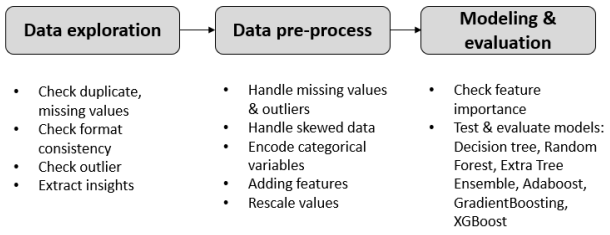
1. INTRODUCTION

Predicting Airbnb prices in New York City was a fascinating project because it gave us an excellent opportunity to study and understand the factors that impact Airbnb prices in one of the most vibrant and diverse cities in the world. The project is relevant to real-world applications, and the insights obtained can have a practical impact in the hospitality industry. For example, the hosts and guests can make more informed decisions, leading to better experiences and more effective use of resources.

In the context of this project, we explored the relationships between different features such as location, amenities, and host characteristics, and how they affect the price of a listing. Additionally, we investigated how different ensemble methods like Random Forest, Gradient Boosting, etc. improve the accuracy and robustness of the predictions.

2. METHODOLOGY

Our approach to the problem is illustrated as followed:



2.1 Data exploration & pre-process

From our check, there are no duplicates or format inconsistency but there is about 10k null data in last_review and reviews_per_month. Since these imply the airbnb does not have reviews, we impute these missing values by 0.

From Fig 1, the distribution of price is highly skewed and fat tailed from the right side. This is usually the case for most of the products' distribution as there are more medium priced products available in the market as compared to high priced products. Thus, one can conclude from this distribution that as per the data, there are more low and medium priced Airbnbs available in the market as compared to high priced Airbnbs (but they are still there).

By taking logarithm, the distribution is closer to normal distribution. Given that many statistical models and machine learning algorithms assume that the target variable follows a normal distribution, we use log(price+1) as target to ensure the performance of models since division by zero is a problem. We still test out the models using price as target variable for comparison purpose.

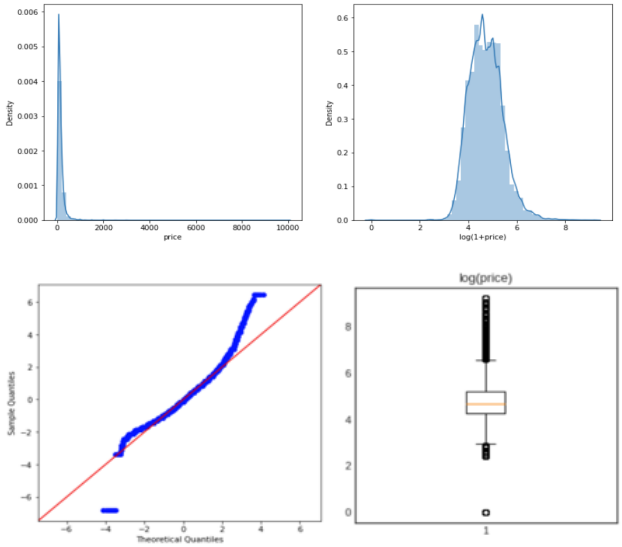


Fig 1: Distribution of price and log_price

Normally, any observations that are more than 1.5 IQR below Q1 or more than 1.5 IQR above Q3 are considered outliers. In this case, the amount of values outside the mentioned range is too large. For example, 22% data have the log(1+price) more than 1.5 IQR below Q1 or more than 1.5 IQR above Q3. Certain Airbnbs that are very well-furnished at perfect location may have extreme prices and the data distribution simply reflects such fact. So we decide to narrow down the outlier range and create 2 scenarios to test out:

- Exclude 5% of the data that have extreme prices
- Not remove any data

Outlier removal step will be done after the train and test data split to make sure that our test data is close to the truth (containing extreme values).

From Fig 2, we can tell the price highly depends on the location of the apartment. We can see that Manhattan is the most expensive because it attracts the most tourists. Brooklyn is also becoming more gentrified, which is the reason why it is slightly more expensive than the other boroughs. As Manhattan & Brooklyn seem to have more of the higher priced properties, the graphs of them tend to be more dispersed. In contrast, Bronx, Staten Island, and Queens have much more reasonable prices compared to Brooklyn and Manhattan, which can be attributed to being relatively far away from New York.

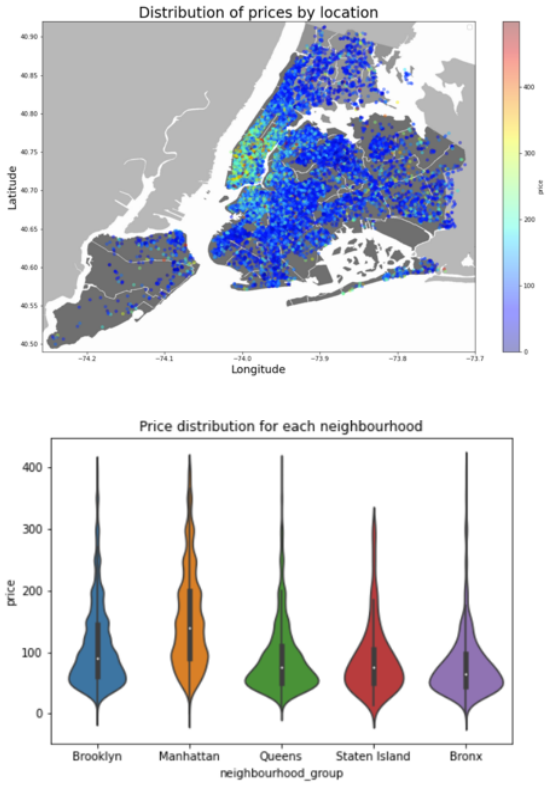


Fig 2: Price by location & neighborhood

From Fig 4, as expected, shared rooms have the lowest mean price while entire homes have the highest. All room types seem to have a similar spread, however private rooms and shared rooms seemed to be more centered around their meanings. There is more disparity of price in entire homes.

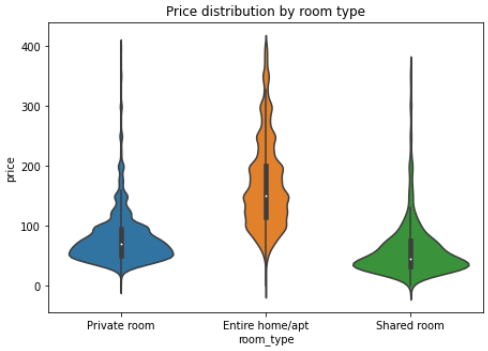


Fig 4: Price by room type

In Fig 3, except for location features, distributions of other numerical variables are highly skewed. Though taking logarithm does not help normalize these variables, we keep them as potential features. Besides, since the variables are highly concentrated in certain values, we think other yes/no features such as no_review, fully_available, low_available, etc. can provide some information for predicting price, so we added them to the feature pool.

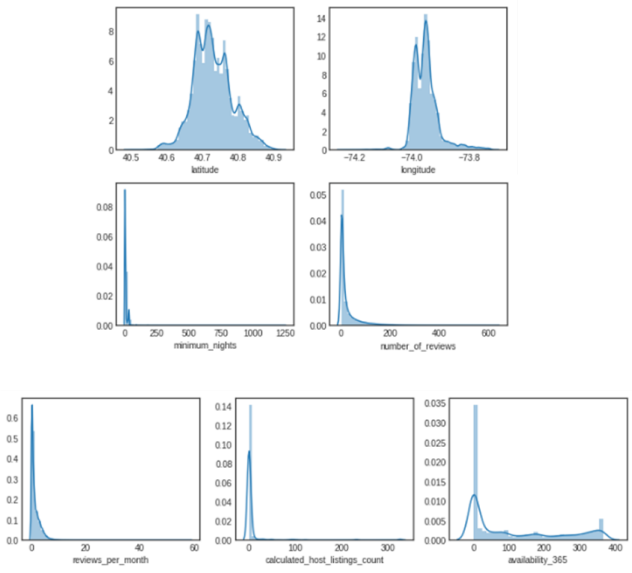


Fig 3: Distribution of numerical variables

For correlation analysis, we applied the chi-square test to test between categorical variables, calculated correlation coefficient between numerical variables and applied anova test to test between categorical and numerical variables. All categorical variables seem to be correlated somehow and most numerical variables are not (except new features extracting from existing features). During anova test, we noticed that the location (latitude) seems to be correlated somehow with the availability & room type. Also so, 1

minimum night requirement (yes/no) is correlated to availability as well.

Finally, we encode categorical data and rescale values to avoid the dominance of large-value features. Also, we measure the feature importance using impurity reduction via the default random forest model. Location info and room type info seem to have high predicting power. Using such insight, we proceed with the modeling phase.

2.2 Modeling & evaluation

2.2.1 Models

The 7 models used are:

- Decision tree: a supervised learning algorithm that creates a tree-like structure to classify or regress data based on features
- Random forest: an ensemble learning method that combines multiple decision trees to improve accuracy and handle high-dimensional data
- XGBoost: an ensemble learning method that combines weak learners to improve accuracy and handle missing data, outliers, and non-linear relationships
- CatBoost: a gradient boosting algorithm that can handle categorical features
- Bagging: combines independent models to improve accuracy, stability, and interpretability
- Gradient Boosting: a powerful technique that produces accurate results but can be prone to overfitting and computational expense

2.2.2 Hyperparameters tuning

To improve the performance, GridSearchCV is used for all models to optimize the hyperparameters. GridSearchCV automates the process of searching through different combinations of hyperparameters to find the best set of hyperparameters for a given model. It does this by creating a grid of hyperparameter values and systematically testing each combination of hyperparameters using cross-validation.

For instance, for Random Forest, we define a parameter grid that includes the max_features of *['auto', 'log2', 'sqrt']*, the number of trees in the forest, the maximum depth of the trees and the minimum number of samples required to split a node, all with range from 1 to 10 , also, we specify the scoring metric as *'neg_mean_absolute_error'* to evaluate the performance of the model, and it ends up giving us the best combination with the *'best_params_'* attribute of the GridSearchCV object (*max_depth=9, max_features='auto', min_samples_leaf=6, n_estimators=9*). We then use these hyperparameters to train a new Random Forest model on the entire training dataset, and get the final R-square of 0.57 on the test set in scenario 3.

2.2.3 Evaluation metrics

Since this is a regression problem, we used 3 following performance metrics:

- Mean Absolute Error: average absolute difference between the predicted and actual values
- Root Mean Squared Error: square root of the average of squared differences between the predicted and actual values
- R squared: the proportion of the variance in the dependent variable that is explained by the independent variables. It ranges from 0 to 1, with a value of 1 indicating that all the variance in the dependent variable is explained by the independent variables

3. RESULT

3.1 Scenario-1, use original price as dependent variable without removing outliers

– Model with all features

	RMSE	MAE	R-squared
Decision tree	236.4	71.0	0.09
Random forest	228.9	68.6	0.147
Adaboost	242.1	76.6	0.05
Gradient Boosting	231.4	70.4	0.13
Bagging	225.3	69.5	0.17
XGBoost	230.1	69.1	0.14
Catboost	**221.9**	**66.6**	**0.20**

– Model with important features selected with Random Forest

	RMSE	MAE	R-squared
Decision tree	236.4	72.0	0.09
Random forest	**227.9**	**70.3**	**0.154**
Adaboost	241.2	76.6	0.05
Gradient Boosting	235.1	71.5	0.10
Bagging	230.0	71.1	0.14

XGBoost	233.2	70.1	0.11
CatBoost	231.8	70.1	0.13

Based on the results, we observe that all models perform really badly when choosing the real price as the targeted variable to predict considering the high RMSE and MAE and low R-squared value. Despite this, training model using all features generally outperforms using only important features, except Random Forest which performs slightly better with selected features. To sum up, CatBoost, Bagging and Random Forest are the top 3 models who perform the best in scenario 1.

3.2 Scenario-2, use original price as dependent variable by removing outliers

– Model with all features

	RMSE	MAE	R-squared
Decision tree	237.4	62.6	0.082
Random forest	237.1	61.8	0.085
Adaboost	239.4	87.9	-0.016
Gradient Boosting	236.6	61.5	0.089
Bagging	236.5	61.3	0.089
XGBoost	236.2	60.6	0.092
Catboost	**236.0**	**60.4**	**0.093**

– Model with important features selected with Random Forest

	RMSE	MAE	R-squared
Decision tree	250.0	87.5	-0.012
Random forest	250.2	87.7	-0.019
Adaboost	249.8	87.8	-0.016
Gradient Boosting	250.1	87.6	-0.018
Bagging	250.3	88.7	-0.02
XGBoost	250.2	87.6	-0.02
Catboost	250.1	87.7	-0.018

For the real price prediction, removing outliers from the training set significantly hurts the performance of all models, especially when we only use important features, the R-squared values all decrease to negative numbers, this means that these models cannot explain any of the variation in the dependent variable. Even if we use all features, these models still do not perform well, based on the generally high MSE and RME, and also low R-squared value. Among all the models, CatBoost performs relatively better, followed by XGBoost and Bagging.

3.3 Scenario-3, use log_price as dependent variable without removing the outliers

– Model with all features

	RMSE	MAE	R-squared
Decision tree	0.477	0.336	0.54
Random forest	0.46	0.323	0.572
Adaboost	0.512	0.369	0.465
Gradient Boosting	0.452	0.318	0.587
Bagging	0.446	0.313	0.598
XGBoost	0.445	0.312	0.6
Catboost	**0.44**	**0.308**	**0.609**

– Model with important features selected with Random Forest

	RMSE	MAE	R-squared
Decision tree	0.499	0.353	0.498
Random forest	0.488	0.344	0.518
Adaboost	0.515	0.369	0.465
Gradient Boosting	0.49	0.345	0.516
Bagging	0.481	0.338	0.533
XGBoost	0.483	0.341	0.529
Catboost	**0.48**	**0.339**	**0.534**

All models perform much better on log_price prediction than real price prediction. Based on MAE, RMSE and R-squared, CatBoost gives the best performance. On the second position, we have the XGBoost and then, Bagging Model.

3.4 Scenario-4, use log_price as dependent variable by removing outliers

– Model with all features

	RMSE	MAE	R-squared
Decision tree	0.487	0.334	0.521
Random forest	0.480	0.326	0.535
Adaboost	0.521	0.363	0.452
Gradient Boosting	0.472	0.321	0.550

Bagging	0.469	0.317	0.555
XGBoost	0.467	0.317	0.559
Catboost	**0.464**	**0.314**	**0.564**

– Model with important features selected with Random Forest

	RMSE	**MAE**	**R-squared**
Decision tree	0.508	0.351	0.478
Random forest	0.503	0.346	0.490
Adaboost	0.523	0.365	0.447
Gradient Boosting	0.503	0.347	0.490
Bagging	0.469	0.317	0.555
XGBoost	0.499	0.343	0.498
Catboost	**0.496**	**0.341**	**0.503**

Based on MAE, RMSE and R-squared, CatBoost gives the best performance. On the second position, we have the XGBoost and then, Bagging Model. The performance is very similar to scenario 3, but the scores in Scenario 3 are better than in this scenario. This is happening because by removing the 5% outliers, we are reducing the ability of our model to predict the price of high-end/expensive/premium Airbnbs.

We can also see here that scores are better when we use all the features instead of just the important features.

4. CONCLUSION

The best 3 models in all the scenarios are more or less the same. Based on the performance indicators of the regression model (i.e., mean absolute error, root mean squared error, and R-squared), the top 5 models across all the scenarios are mentioned below:

1. CatBoost Model with all features (Scenario 3)
2. XGBoost Model with all features (Scenario 3)
3. Bagging Model with all features (Scenario 3)
4. Bagging Model with important features (Scenario 3)
5. Gradient Boosting Model with all features (Scenario 3)

From this one can conclude that predicting on log_price and using all the data (i.e., not removing any outlier) is the best way. This is because by not removing any outlier, we ensure that our model is able to make predictions not only for lower or medium priced Airbnbs but also for more expensive (high priced) Airbnbs.

Also, from the results of all the scenarios, we can clearly see that using all the features to make the predictions yield better results rather than using just the important features.

It is also important to note that we can change the prediction from log price to price by simply applying the exponential function.

REFERENCES

[1] https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data

[2] https://www.tutorialspoint.com/correlation-between-categorical-and-continuous-variables

Decision Tree Implementation in Python

—----------

1. INTRODUCTION

In this project, we implemented a decision tree from scratch in Python. Our implementation allows for both classification and regression tasks and uses the Gini impurity and mean squared error as splitting criteria, respectively.

2. DATASET

For the classification task, we used the load_wine() dataset provided by scikit-learn. This dataset contains 13 features and 3 classes of wine. We loaded the dataset using the load_wine() function and split it into training and testing sets using the train_test_split() function.

For the regression task, we generated synthetic data using the make_regression() function provided by scikit-learn. We generated 2000 samples with 5 informative features and 1 target variable. We also added some noise to the data to make it more realistic. We split this dataset into training and testing sets using the train_test_split() function as well.

3. DECISION TREE ALGORITHMS

Our decision tree algorithm works by recursively splitting the data based on the feature that maximizes the information gain (or minimizes the impurity) at each node [1]. For classification, we use the Gini impurity as the splitting criterion, and for regression, we use the mean squared error (MSE) as the splitting criterion [2].
We implemented the decision tree algorithm using the following step [3]s:

1. Determine the splitting criterion (Gini impurity or MSE) for the current node.
2. Compute the impurity or MSE for each possible split of the data based on each feature.
3. Choose the feature that minimizes the impurity or MSE and split the data based on its threshold value.
4. Recursively build the tree by repeating steps 1-3 for each child node until a stopping criterion is met.

We used the following stopping criteria for our implementation [3]:

- max_depth: the maximum depth of the tree.
- min_samples_split: the minimum number of samples required to split an internal node.
- min_samples_leaf: the minimum number of samples required to be at a leaf node.

4. RESULTS

We evaluated the performance of our decision tree algorithm on the classification and regression tasks using the testing sets.

For the classification task, our implementation achieved an accuracy of 91.67% on the testing set, while the result of the decision tree classifier from the scikit-learn library is 94.44%. For the regression task, our implementation achieved a RMSE of 27.65 on the test set, while 18.24 of MSE for the decision tree regressor from the scikit-learn library.

5. CONCLUSION

In this project, we implemented a decision tree from scratch in Python that can be used for both classification and regression tasks. Our implementation is flexible and allows for customization of the stopping criteria and splitting criteria. We demonstrated the effectiveness of our implementation on both a classification and regression task using real and synthetic data. Our implementation can serve as a starting point for further exploration and experimentation with decision trees.

REFERENCES

[1] Jijo, B. T., & Abdulazeez, A. M. (2021). Classification based on decision tree algorithm for machine learning. *evaluation*, *6*, 7.

[2] Loh, W. Y. (2011). Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, *1*(1), 14-23.

[3] Chauhan, H., & Chauhan, A. (2013). Implementation of decision tree algorithm c4. 5. *International Journal of Scientific and Research Publications*, *3*(10), 1-3.