
# CS5228 Knowledge Discovery and Data Mining

# Project Report – Kaggle Competition

Kaggle Team Name: ACCS


| Team Members |
| --- |
| E1101715 - Anubhaw Kuntal Xess |
| E0925491 - Chen Jianyu |
| E0950103 - Chen Yanrong |
| E0945871 - Samriddha Basu |

# Contents

# 1. Introduction

In this project, we explore the strength of various regression models on predicting house prices and the impact of location attributes on housing price. The task is to create a prediction model for the resale price of HDB flats in Singapore and compete in the Kaggle in-class competition. The data is sourced from data.gov and 99.co.

**Training dataset:** *train.csv* dataset containing features such as *month* of sale, *flat_type*, *flat_model*, *planning_area*, *region*, etc., along with the target variable *resale_price*

**Auxiliary datasets**: the file consists of demographic data and location data about key infrastructures such as train stations, hawkers, and shopping malls

**Test dataset:** *test.csv* dataset containing the same features as the training dataset but without the target variable *resale_price*

# 2. Methodology

## 2.1 Exploratory Data Analysis

Exploratory is an imperative step before data preprocessing and modelling. Through data visualization, researchers can summarize the main characteristics of datasets.

Some notable observations are as follows:

- There are 431732 rows in the training data and 107934 rows in test data.
- The target variable 'resale_price' representing the resale prices of HDB flats.
- We have 6 numerical columns and 11 categorical columns. The density distribution of numerical features is given in Fig 1.
- Upon plotting the target variable *'resale_price'*, we observe that it is not normally distributed and is instead skewed to the right (positive skew) as shown in Fig 1.

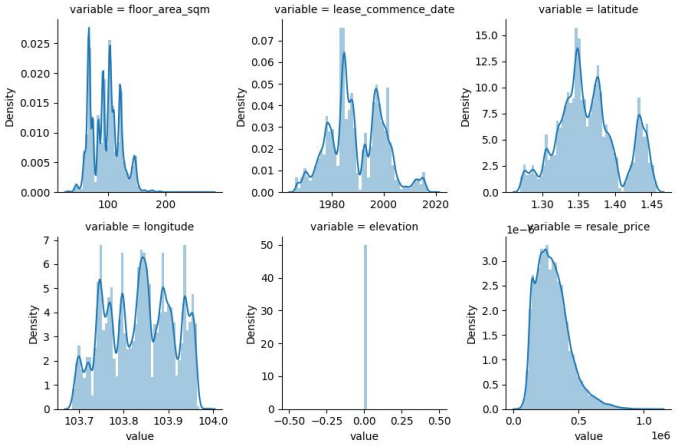


*Fig. 1 Density distribution plots for numerical features*

## 2.2 Data Preprocessing

The results of exploratory data analysis provide valuable indications for data preprocessing. We first remove the redundant features since those features provide little or no additional information to the model.

Four features are identified as redundant features:

- *'elevation'* is a redundant feature as it has only one unique value and does not contribute to the prediction model.
- '*eco_category*' is redundant because all flats in the dataset are in the same category
- *'block'* is redundant since the block number of Singapore HDBs is not determined by the flats
- '*street_name'* is generally considered as an unimportant variable

As shown in Fig. 1, the distribution of *'resale_price'* is not normally distributed and right skewed. Since most regression algorithms perform better with normally distributed data, we apply logarithmic transformation to the target variable. According to Fig. 2 and Fig. 3, the target variable is more normally distributed after the transformation.

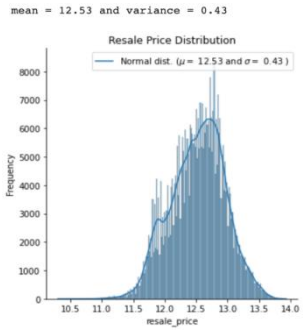


*Fig. 2 Resale Price Distribution*

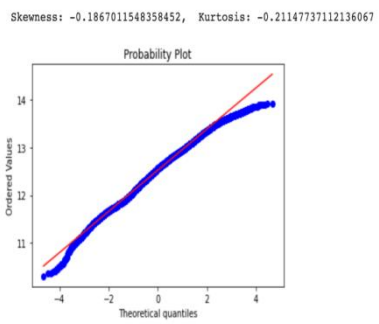


*Fig. 3 Probability Plot*

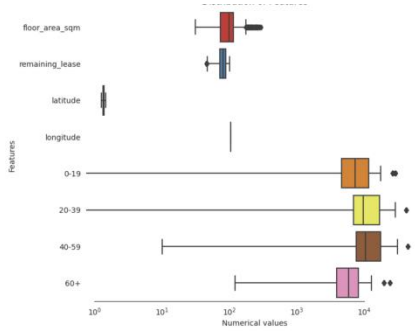


*Fig. 4 Boxplot of the numerical features*

Subsequently, we created eight new features representing location attributes with the auxiliary datasets.

- *'0-19', '20-39', '40-59', '60+'* represent the population of different age groups
- *'mrt'*, represents the number of MRT stations next to the flat
- '*sp_mall*' & *'commercial_center*', are numeric features indicating the number of shopping malls and commercial centers close to the flat, respectively
- *'hawker',* is the variable showing the number of hawkers near the flat
- *'secondary_school'* & *'primary_school'* represent the number of secondary and primary schools next to the flat, respectively
- *'remaining_lease'*, denotes the remaining lease (years) of the flat when it is traded

The next step is to examine missing values in the data. There are 200 missing values in *'0-19', '20-39', '40-59', '60+'*, respectively. The mean value of each feature is used to impute the missing data. We then inspect the distribution of the newly generated numerical features. As shown in Fig. 4, there is no highly skewed feature (absolute skewness greater than 0.5) in the dataset. In addition, we recognize that certain values of '*flat_type*' are ambiguous after checking the unique values of the categorical features. For example, '1-room' and '1 room' represent the same flat type.

Categorical features must be converted into numerical data so that machine learning algorithms can process them. Notably, some features like *'lease_commence_date'* should be treated as categorical variables and must be encoded appropriately.

Finally, we split the concatenated set back to the training and the validation sets. Before the dataset is ready for modelling, it is better to remove duplicated data in the training set. The final cleaned training set has 431665 rows and 33 columns.

*Table 1. List of Attributes*

| Attribute Name | Data Type | Description |
|---|---|---|
| month | object | Trade time with month and year (2000 - 2020) |
| town | object | Town in which the flat located |
| storey range | object | Storey range of the flat |
| floor area | float64 | Floor area of the flat in square meter |
| flat model | object | Flat model |
| lease commence date | int64 | Lease commence date of the flat |
| latitude | float64 | Latitude of the flat |
| longitude | float64 | Longitude of the flat |
| subzone | object | Subzone in which the flat located |
| region | object | Region in which the flat located |
| planning area | object | Planning area in which the flat located |
| 0 - 19 | int64 | Population of 0 - 19 |
| 20 - 39 | int64 | Population of 20 - 39 |
| 40 - 59 | int64 | Population of 40 - 59 |
| 60+ | int64 | Population of 60+ |
| MRT | int64 | Number of nearby MRT stations |

| | | |
|---|---|---|
| shopping mall | int64 | Number of nearby shopping malls |
| secondary school | int64 | Number of nearby secondary school |
| primary school | int64 | Number of nearby primary school |
| hawker | int64 | Number of nearby hawkers near the flat |
| commercial centre | int64 | Number of nearby commercial centers |
| remaining lease | int64 | Remaining years of 99-year lease |

## 2.3 Modelling and Results

Regression models are verified powerful tools for house prices prediction. In this paper, we focus on exploring tree algorithms such as random forest and gradient boosting. Optuna is employed for hyperparameter tuning using Root Mean Squared Error (RMSE) as evaluation metric.

### 2.3.1 Random Forest

Random forest is a type of ensemble learning method that combines multiple decision trees to make predictions and has been proven powerful in previous studies [1].

The key parameters we tuned are:

- *'n_estimators'* denotes the number of trees in the forest and its default value is 400
- 'max_depth' represents the maximum depth of the tree and it is normally between 3 and 15

Based on the results of *GridSearchCV*, it was found that the optimal value for *'n_estimator'* among 400, 800, 1000, and 1200 is 1000 and *'max_depth'* is 10. Other parameters such as *'random_state'* which controls the randomness of the bootstrapping of the samples used when building trees, was set to 42 to achieve reproducibility of results, *'n_jobs'* which specifies the number of jobs to run in parallel, was set to -1 to accelerate the training process by using all processors. Fig. 5 shows the model's prediction in X-axis, and the actual price in Y-axis for the training data.

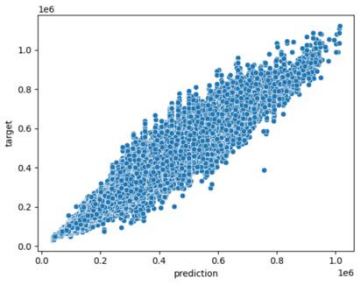


*Fig. 5 RandomForest's Predictions on Training Data*

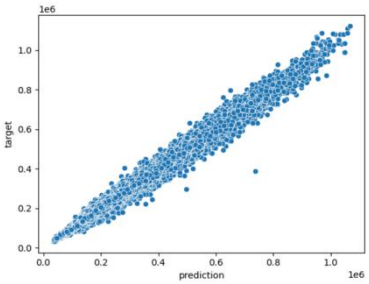


*Fig. 6 XGBoost's Predictions on Training Data*

### 2.3.2 Extreme Gradient Boosting (XGBoost)

XGBoost is one of the most popular implementations of gradient boosting algorithm. The framework has achieved great success and been widely recognized in the machine learning community [2].

To build a robust model with XGBoost, there are many essential hyperparameters that must be carefully tuned:

- *'n_estimators'* is the number of gradient boosted trees which is equivalent to number of boosting rounds. Increasing its value can cause overfitting. Searched from 400 to 3000
- *'learning_rate'* is the boosting learning rate. It must be considered together with *'n_estimator'* to handle underfitting and overfitting. Searched from 0.01 to 0.1
- *'max_depth'* is the maximum tree depth for base learners. Common choices are between 3 and 10.
- *'subsample', 'colsample_bytree'* and *'colsample_bylevel'* are respectively the subsample ratio from training set, the number of columns when constructing each tree and the number of columns for each level. Decreasing their values can reduce the impact of noise in the training set and prevent overfitting.
- *'reg_alpha'* and *'reg_lambda'* control L1 and L2 regularization applied to the model respectively. Increasing the two hyperparameters reduces the likelihood of overfitting.

After meticulous hyperparameter tuning, the final choices are n_estimators = *2800, learning_rate = 0.05, max_depth = 8, subsample = 0.8, colsample_bytree = 0.8, colsample_bylevel = 0.65, reg_alpha = 0.95, reg_lambda = 0.65.* Fig. 6 illustrates its performance.

### 2.3.3 Light Gradient Boosting Machine (LightGBM)

LightGBM is a gradient boosting framework that has faster training speed with lower memory usage compared to XGBoost [3].

For LightGBM, the following hyper-parameter values were chosen based on Optuna's output:

- Set *'n_estimators'* = 3000 and *'learning_rate'* = 0.05
- *'num_leaves* specify the maximum tree leaves for base learners and is calibrated to the value of 12

- *'subsample'* and *'colsample_bytree'* are set to 0.8 and 0.7 respectively to reduce the likelihood of overfitting.
- *'min_child_sample'* is the minimum amount of data needed in a child. We've set it to 10 to ensure at least 10 datapoints in every leaf node.

Fig. 7 displays the relationship between its prediction and the actual resale prices.

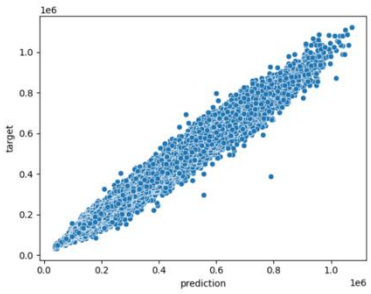


*Fig. 7 LightGBM's Predictions on Training Data*

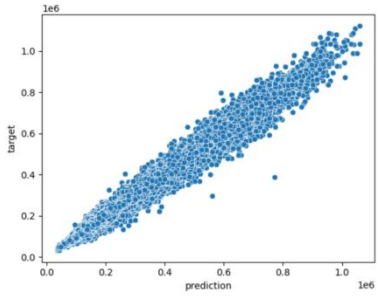


*Fig. 8 CatBoost's Predictions on Training Data*

### 2.3.4 Category Boosting (CatBoost)

Unlike other gradient boosting frameworks, CatBoost attempts to solve for Categorical features using a permutation driven alternative [4]. Therefore, CatBoost tends to outperform other frameworks on datasets with more categorical features than numerical features.

We set some of the important hyper-parameters in CatBoost as follows.

- *'learning_rate'=0.5* and *'interations'=3000*: these 2 hyper-parameters respectively control the speed and length of learning process. For *'learning_rate'*, we chose a commonly used value 0.05 to make training faster while guarantee accurate convergence. Considering the size of dataset, *'interations'* was set to 3000.
- *'depth'=10*, *'l2_leaf_reg'=8*, *'random_strength'=1* and *'max_ctr_complexity'=1*: the 4 hyper-parameters control the complexity of CatBoost. When setting their values, a trade-off between accuracy and reduction of overfitting must be considered. *'depth'* is the maximum depth of each decision tree, which we set to 10, considering there were only 23 predictor variables. *'l2_leaf_reg'* controls the strength of L2 penalty. *'random_strength'* controls the degree of randomness in the model by adding random weights to input features, which we wanted to be small and introduce a little randomness. *'max_ctr_complexity'* denotes the number of constructed interaction pairs between features, which we set to 1 as we hoped to decrease complexity.

- *'grow_policy'= 'Lossguide'*: it specifies how decision trees are constructed during training. Lossguide is a favorable strategy as it gives the optimal splitting by evaluating loss functions, leading to more accurate results and faster convergence.

The model's performance is shown in Fig. 8.

### 2.3.5 Stacked Model

Stacked generalization is an ensembling technique introduced by DH Wolpert in 1992 [5]. The method attempts to combine the predictions of lower-level models using a meta model to achieve higher accuracy.

We designed a two-layer stacked model with several strong regressors in the first layer and one meta regressor in the second layer. Fig. 9 provides an overview of the stacking architecture. The regressors are selected based on the 10-fold cross-validation scores as shown in Fig. 10.

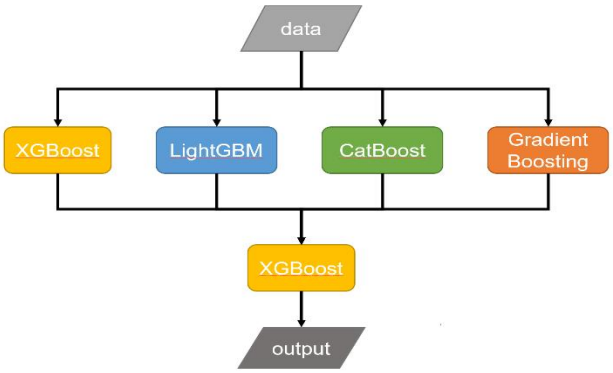


*Fig 9. Structure of Stacked Model*

| | Regressors | Scores | Std |
|---|---|---|---|
| 0 | XGB | 0.054904 | 0.000249 |
| 1 | LightGBM | 0.057053 | 0.000232 |
| 2 | Cat | 0.057054 | 0.000202 |
| 3 | KNN | 0.071036 | 0.000298 |
| 4 | ElasticNet | 0.184983 | 0.000347 |
| 5 | GBR | 0.056669 | 0.000128 |
| 6 | RF | 0.098902 | 0.000466 |

*Fig 10. RMSEs of Different Regressors*

The stacked model achieved exceptionally high accuracy on the training set with a RMSE at 0.052083. As illustrated in Fig. 11, it significantly outperformed the low-level models. However, such low error on training data is not a desirable sign since it indicates high likelihood of overfitting.

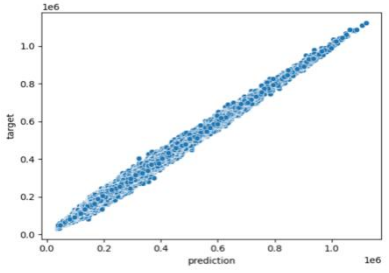


*Fig 11. Stacked Model's Predictions on Training Data*

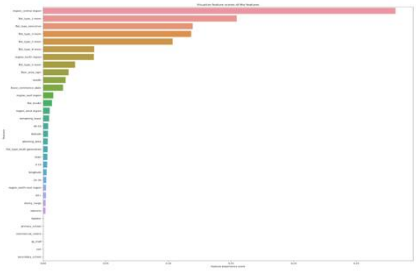

*Fig. 12 Feature Importance of XGBoost*

### 2.3.6 Cross-Validation

To analyze the generality of the models, we performed 10-fold cross-validation using *cross_val_score* in *sklearn.* The final output is displayed in Table 2.

*Table 2. Prediction Results*

| Model Name | RMSE | |
|---|---|---|
| | Training Set | Cross-validation Score |
| Random Forest | 0.097094 | 0.098902 |
| XGBoost | 0.053601 | 0.054017 |
| LightGBM | 0.055997 | 0.057053 |
| CatBoost | 0.055028 | 0.056883 |
| Stacked model | 0.052083 | 0.056514 |

According to the cross-validation scores, XGBoost is the best model with the lowest RMSE. The stacked model's performance in the cross-validation drops to the fourth place which implies that it overfitted on the training set. The final score of XGBoost on Kaggle public and private leaderboard are 10901.82178 and 11016.19030 respectively.

## 3. Discussion and conclusion

In this project, we investigated a variety of regression algorithms, especially gradient boosting algorithm, for housing price prediction. Even though most models have achieved satisfactory accuracy on both training and validation data, different models have their own pros and cons.

The random forest model has lower performance but is relatively easy to tune and faster to train. The gradient boosting models all achieved excellent accuracy but relies heavily on GPU to accelerate training. In addition, more tunable hyperparameters make those models unfriendly to beginners. Despite having the lowest error on the training data, stacked model is prone to overfitting and its special architecture makes it even more difficult to analyze and tune. Moreover, cross-validation score is more reliable than training error in model evaluation.

To analyze the impact of different features on the target variable, we use the feature importance graph to visualize the contribution of each feature. As shown in Fig. 12, the most important feature of the XGBoost model is *'region_central_region'* which represents whether the flat is in the central region. This serves as strong evidence supporting that location attributes are indeed crucial in determining the resale price of a flat.

# Reference

[1] Breiman L. Random Forests. SpringerLink. https://doi.org/10.1023/A:1010933404324 (accessed September 11, 2019)

[2] Chen T, Guestrin C. XGBoost. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 16 2016. doi:10.1145/2939672.2939785

[3] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems 30 2017:3146–54

[4] Dorogush, Anna Veronika, Vasily Ershov, and Andrey Gulin. "CatBoost: gradient boosting with categorical features support." *arXiv preprint arXiv:1810.11363* (2018).

[5] Wolpert DH. Stacked generalization. Neural Networks 1992;5:241–59. doi:10.1016/s0893-6080(05)80023-1