CS5228 Term Project Report

Mingyue Song
mingyue.song@outlook.com

Jiechen Ou
e0950449@u.nus.edu

Lianda Zhu
e0767629@u.nus.edu

Baoqi Jia
baoqi@u.nus.edu

***Abstract* - This is the term project report for NUS module CS5228 Knowledge Discovery and Data Mining. The entire project is divided into three subtasks. First two tasks are pre-defined while the last one is an open task. In this report, we first covered the EDA and preprocessing to prepare the dataset for all the following subtasks. Then we sequentially covered the three tasks: task 1 property resale price predator, task 2 property recommendation engine and task 3 MRT passenger volume analysis. Finally we give a conclusion to our overall project.**

I. EDA & Preprocess

Across the three subtasks in our project, the same treatment is applied to preprocess the input data. However, each subtask may only require a subset of the preprocessed data features to complete the data mining. Firstly, we will cover which columns in the input data will be dropped from our data mining tasks and explanation on such decisions. Secondly, preprocessing will be described for each data feature and we will present some visualization of it. Lastly, we will talk about how we utilize the auxiliary data.

A. Features to drop

The following columns will be dropped from our data mining tasks and we provide a brief explanation of our decisions.

- listing_id/property_details_url: they are both artificial identifiers or resource locators, which do not reveal any useful information about housing insights.
- title: let's see an example of the data, "4 bed condo for sale in meyerhouse". It is clearly shown that the data is overloaded and hard to extract and quantify. Furthermore, the same set of data like how many bedrooms or what type is the property can also be found in other columns.
- address: 1. data is inconsistent across the dataset, some addresses are block numbers with road names, and others are district codes; 2. the address is basically geographical locations, but there are simply too many different values and they are hard to encode. We will utilize subzone and planning_area to achieve this goal.
- property_name: there are too many different values for this column and we will utilize the tenure, built_year and geographical location information from other columns to substitute this column.
- floor_level: highly homogeneous, most values are NA.
- elevation: extremely homogeneous, all values are 0.
- available_unit_types: the information is quite overloaded as the values are all in list format, it is hard to encode such data.
- total_num_units: we believe it has very little contribution to the resale price, it only reveals how many units are in the community.
- furnishing: highly homogeneous, most values are unspecified.

B. Features in use

1) property_type

Text values are all normalized to avoid duplicates of the same type. Apart from that, there are some HDB types including the number of bedrooms. We generalize all those HDB types by removing the bedroom information as they will be reflected in num_beds features. The encoding of property_type will be discussed together with subzone.

2) tenure

There are only three common tenure types in Singapore, 99-year, 999-year and freehold. [1] Thus we map all the tenure values to the nearest common type and do ordinal encoding.

For NA values, we group the properties by their type. For each type, fill the NA tenure randomly with the probability proportional to the number of different tenures.

3) built_year

For NA values, we want to fill it the same way as tenure but there are too many year values. Thus, we discritize the built_year into 10-year intervals and fill it the same way as the tenure. Ordinal encoding is applied.

4) num_beds & num_baths

First, let's get an overview of the room numbers.

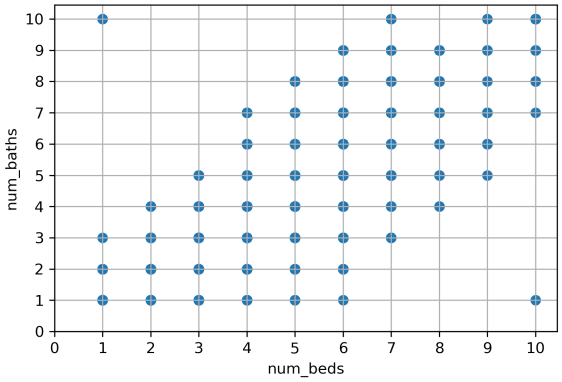


Fig. 1. *bedroom and bathroom number plot*

We can see majority of the combinations of bedroom number and bathroom number is fine.

For num_beds NA values: 1. some num_beds are assigned with 0 if they are studios; 2. But there are some studios with many bathrooms, it is actually the kind of project that allows the owner to configure the number of bedrooms themselves, so we just fill the number the same as the bathrooms; 3. For special properties, we manually lookup online and fill them.

For num_baths NA values: we group by num_beds and select the most common value in the num_beds group to assign.

The two outliers, (1 bedroom + 10 bathrooms) and (10 bedrooms + 1 bathroom), will be fixed according to the most common values as well.

5) Latitude & Longitude

We first plot the latitudes and longitudes with kernel density as colour coding.

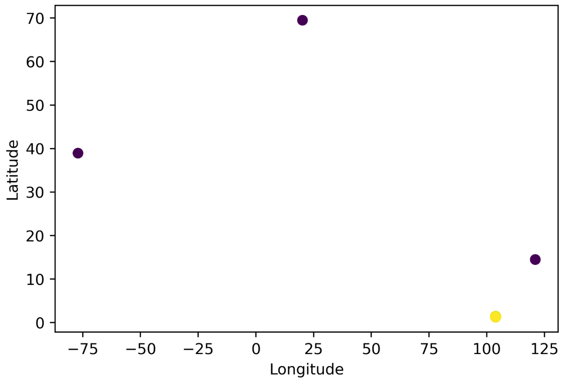


Fig. 2. *Geographical plot pre-processing*

It is clearly shown the bright yellow dot should be the correct coordinates for Singapore properties. The other dark ones should be outliers and we manually lookup from Google Maps to assign the correct values.

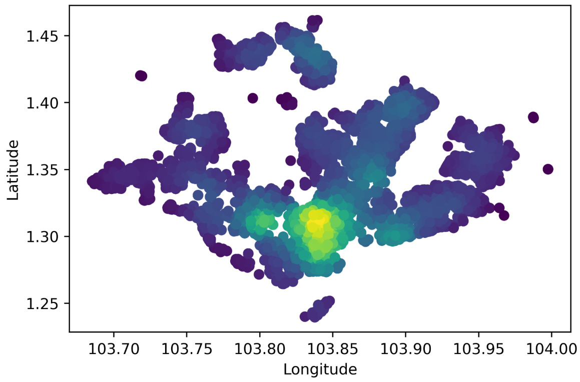


Fig. 3. *Geographical plot post-processing*

After the fixing, we can see that the coordinates form the shape of Singapore territory.

6) subzone & planning_area

The only thing that needs to be cared about is the NA values. For subzone, we implement a KNN classifier to take in lat and lng as the features to predict the missing subzone. This is intuitive as the property subzone should follow those in the vicinity. For palnning_area, it has the hierarchical order higher than subzone so we can just fill it by the subzone values.

Encoding-wise, subzone and property_type is joined and we apply target encoding. To be specific, the average price per sqft is used per subzone per property_type. For test data, we will assign the value from the training data. For property_type and planning_area, we apply an additional one-hot encoding to help the model training. The sparsity is not an issue here because we finally choose XGBoost as our model and it has a sparsity-awareness algorithm. [2]

7) size_sqft & price

The processing of these two columns are extremely challenging because there is a very wide variation. Some prices may be normal for a bungalow, but it would be an outlier for an HDB. Similarly, some sizes in sqft might be normal to a landed house, but it would be again an outlier for an HDB. Thus, the optimal way to analyze the data is to divide them by property types. We will iteratively plot the price per sqft to check whether it is bell-curve-like and fix the outliers.

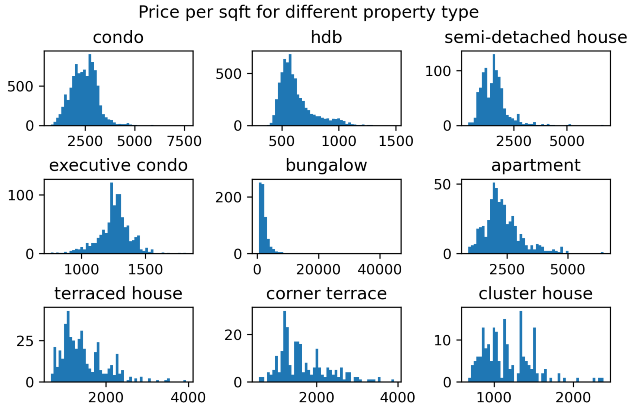


Fig. 4. *Price per sqft histogram for each property type*

After fixing all the outliers, we will plot the histogram to check whether it is a bell-curve-like shape for each property type. But we can only do it for those property types with adequate number of candidates. There is an exception in the plot for bungalows, but after we look into the data and research online, we find the data samples are actually credible. For test data, the same processing is applied for size_sqft only.

C. Auxiliary Data

We decide to utilize the auxiliary data by calculating the distance between the property and those commercial centres, schools, etc. For different kinds of targets (schools/MRT/shopping malls), we use the number of those targets within a certain range as a new feature for the property. Additionally, we add the distance to the nearest MRT station also. Specifically for school admissions, we found that priority will be given to those families living within 1km, then given to 1km to 2km. [3] So we count the school numbers within 1km and from 1km to 2km as the new feature.

II. Task 1 Prediction of Property price

A. Motivation

The reason for designing such a model is that for home buyers, they can predict the future trend of house prices and choose the best time to buy the property of their choice. Although our model may not be completely accurate in predicting house prices, it can still help determine the general house price trend.

For property investors, they can use our model to predict which sites, house types, etc are advantageous to buy, so that they can invest in the right property.

B. Model Selection

After analyzing the task, we selected several models for investigation and finally selected the XGboost model. In this panel, we will detail those models that we eliminated, which are linear model, lightGBM and Random forest.

In the model selection process, we choose to cut the train data file into two 8:2 parts, and use the piece with 80% as the new training set and the remaining 20% as the test set. This way, although a part of the training sample is missing, the advantage is that we can compare the predicted value with the real one and calculate the deviation. Through negative feedback adjustment, we can change each model parameter to make the model better.

1) Linear Model

There are multiple linear basis models, here we choose linear regression model, kernel ridge regression model, Here we choose linear regression model, kernel ridge regression model, polynomial kernel regression model, RBF kernel regression model and standardized RBF kernel regression model to train.

Taking the linear regression model as an example, the regression algorithm is a supervised learning algorithm used to establish the mapping relationship between the independent variable X and the observed variable Y. Since there are multiple data in our dataset, which are not monolinear, and if the different features are not independent of each other, then it may lead to covariance among the features, which in turn leads to inaccurate models.

Therefore, in addition to digitizing the data, we also need to remove some of the data. Data such as bedroom number, bath number, etc. are not linearly independent of area, so we only keep area. However, for other data, it is still not certain that it is linearly unrelated, and if too much is discarded, useful data will be lost again. Therefore, after several filtering runs, the linear model still does not work well.

Since we used the two-eight method to split the training dataset, for both real and predicted data, we plotted the scatter plots of predicted and actual values for each model separately for comparison, and the scatter plots are as follows.

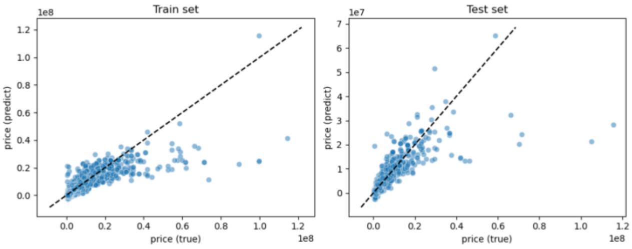


Fig. 5. *Prediction of linear regression model*

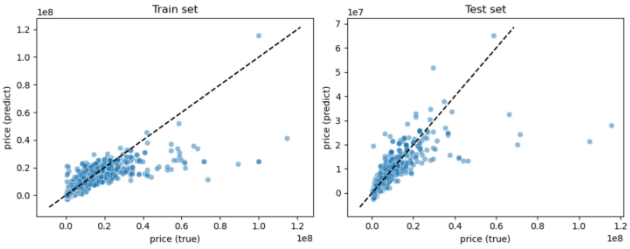


Fig. 6. *Prediction of kernel ridge regression model*

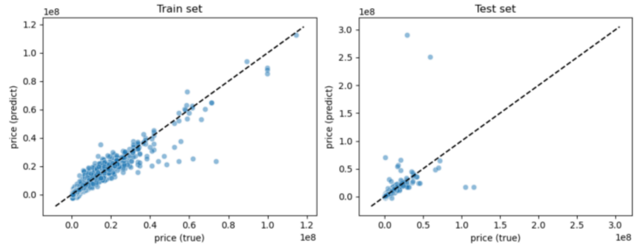


Fig. 7. *Prediction of polynomial kernel regression model*

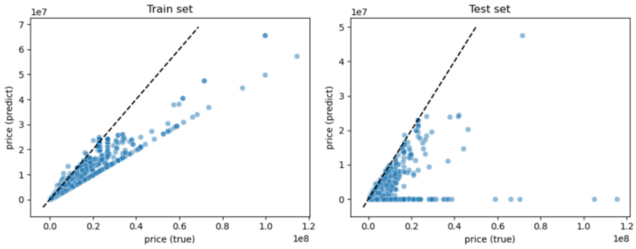


Fig. 8. *Prediction of RBF kernel regression model*

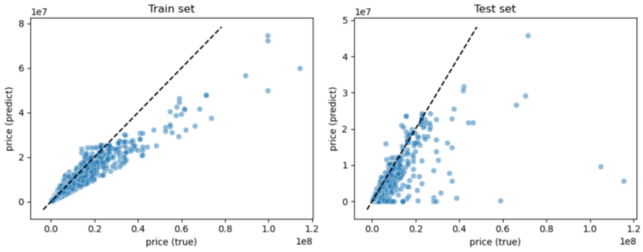


Fig. 9. *Prediction of standardized RBF kernel regression model*

2) Light Gradient Boosting Machine (LightGBM)

Light Gradient Boosting Machine (LightGBM) is a distributed gradient boosting (Gradient Boosting Decision Tree, GBDT) framework based on decision tree algorithm. In the data environment of large training samples and high-dimensional features, compared with GBDT, LightGBM has faster training speed and efficiency, lower memory usage, better accuracy, supports parallel learning and can process large-scale data.

LightGBM uses a histogram-based algorithm to discretize continuous eigenvalues into K integers, construct a histogram with a width of K, traverse the training data, and count the cumulative statistics of each discrete value in the histogram. When selecting the split points of a feature, it is only necessary to traverse the discrete values of the sorted histogram.

The use of the histogram algorithm reduces the computational cost of the algorithm. The pre-sort adopted by XGBoost needs to traverse each eigenvalue and calculate the split gain, while the histogram algorithm only needs to calculate K times, which improves the efficiency of finding split points; reduces the memory of the algorithm Consumption, it is not necessary to store the pre-sorted results, only the discretized values of the features need to be saved.

Because the decision tree itself is a weak learner, the use of the histogram to discretize the eigenvalues can have a regularization effect, and the discretized split points can improve the generalization ability of the algorithm.

Most decision tree learning algorithms use a level-wise strategy for tree generation. The difference is that LightGBM adopts a more efficient leaf-wise strategy.

The strategy finds a leaf node with the largest splitting gain from all the leaf nodes of the current decision tree each time, and then splits it, and so on. Such a mechanism reduces the split calculation of leaf nodes with lower gains and reduces a lot of unnecessary overhead.

Compared with the level-wise strategy, leaf-wise can reduce the error and get better accuracy with the same number of splits. The disadvantage of the leaf-wise algorithm is that it may generate deeper decision trees. Therefore, LightGBM adds parameters that limit the maximum depth to leaf-wise, which prevents overfitting while ensuring the efficiency of the algorithm.

During the initial screening process, we did not set parameters and looked at the general results. Although with great expectation, the results did not turn out as we had hoped, so we abandoned the model.

3) *Random Forest*

Random forests are integrated learning, and the core idea is to integrate multiple weak classifiers to achieve the effect of three stinkers. The random forest uses the idea of Bagging. First, n training samples are taken from the training set to form a new training set; second, the new training set is used to train M sub-models; finally, a simple averaging method is used to obtain the prediction values.

The generation of the tree is a recursive process. In general, as the partitioning process continues, we want the branch nodes of the decision tree to contain samples that belong to the same class as much as possible, i.e., the "purity" of the nodes becomes higher and higher. Pruning is used to increase the generalization ability of the model and to prevent overfitting.

The n_estimators, max_depth, min_samples_leaf, and min_samples_split parameters of the model are set more broadly, but the results of the run are still poor, so this model is not selected.

4) *Comparison*

We calculated the MAE, MSE, RMSE and other errors of different models separately, and put some of them in the table below. From the table, we can see that the RMSE of the train set is at the 1,000,000 level, and the RMSE of the test set is roughly at the 2,000,000 to 3,000,000 level.

TABLE I. the MAE, RMSE of different model

Model			MAE	RMSE
Linear	Linear regression model	Train	991702.4406	2414434.714
		Test	1051036.321	2991114.525
	Polynomial kernel regression model	Train	989244.604	2415968.577
		Test	1047415.793	2993177.175
	RBF kernel regression model	Train	542182.4181	1263227.202
		Test	784876.5506	5902451.517
	Standardized RBF kernel regression model	Train	427204.8418	1552123.229
		Test	748134.9185	3435188.624
LightGBM		Train	279262.6835	778454.1889
		Test	398522.8177	2337357.865
Random Forest		Train	223327.6356	1179220.151
		Test	351684.7969	2389510.878

C. *XGBoost*

1) *Description*

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners (CARTs generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements. it approaches the process of sequential tree building using parallelized implementation. to improve run time, the order of loops is interchanged using initialization through a global scan of all instances and sorting using parallel threads. The second improvement is tree pruning. The stopping criterion for tree splitting within GBM framework is greedy and depends on the negative loss criterion at the point of split. XGBoost uses hyperparameters as specified instead of criterion first, and starts pruning trees backwards, which improves computational performance significantly. These characteristics make XGBoost outperform normal GBMS in many cases. [2]

2) *Exploratory research*

Firstly, our group tested the performance of XGBoost with default parameters on the training data. The result shows that the root mean square error (rmse) of the validation set was lower than those of the models in the previous chapter. Simultaneously, the XGBoost algorithm refers to an Over-fitting Problem. In this case, the rmse of the validation set is almost six times bigger than that of the training data. Therefore, the following sections focus on hyperparameter tuning and feature variables selection to mitigate overfitting and try to make the optimal trade-off.

3) *Hyperparameters tuning*

Since hyperparameters are tunable and can directly affect how well a model performs, this section lists some popular parameters and the potential range of them to find out the best combination for the data.

Max depth: Deeper trees may improve performance, but also increase complexity and risk of overfitting.

Learning rate: Determines the step size at each iteration when a model optimizes toward its objective. A low learning rate makes computation slower and requires more rounds to achieve the same reduction in residual error as a model with a high learning rate, which may be beneficial to improve generalization.

Number of estimators: Describes the number of boosting iterations.

Columns sample ratio: Represents the fraction of columns to be randomly sampled for each tree. Tuning it iteratively plays a role in feature selection and it may prevent over-fitting.

Subsample ration: Represents the fraction of observations to be sampled for each tree. A lower value makes models robust but might lead to under-fitting.

Regulation coefficient lambda: L2 regularization on the weights, which might help to reduce overfitting.

4) *Cross Validation*

Considering that there are hundreds of potential values for hyper parameters, exhaustive tuning is impractical and time-consuming. A Random Search can give us the intuition on how to set them. It uses a large (possibly infinite) range of hyperparameters values, and randomly iterates a specified number of times over combinations of those values to obtain a local optimal set. After randomly searching, our group get a specific and small range of reasonable values and then utilize GridSearchCV to optimize hyperparameters setting.

Even though the deviation between rmse of training and validation data is alleviated, overfitting still exists. Therefore, we dive into feature Importance and redo grid search with the top 10 most significant features obtained above to clarify whether auxiliary features and one-hot encoding for planning area and property types have a negative impact on model generalization. The result shows that the model fitting with only top 10 features has a worst prediction performance than that using the entire dataset. Therefore, it's unnecessary to cover feature reduction in this task.

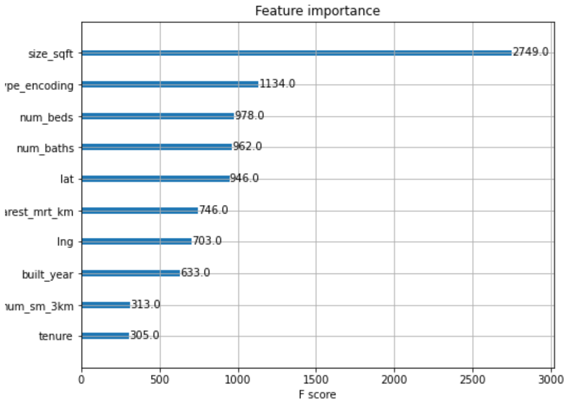


Fig. 10. *Feature Importance for XGBoost*

5) *Evaluation*

a) *Sensitive analysis*

Explore and visualize the performance of different hyperparameters suppressing over-fitting issues on this dataset. With other parameters fixed, our group tune parameters one by one iteratively to illustrate the actual effect of different parameters in dealing with overfitting problems. Figure 6 shows the deviation of rmse between training data and validation data with respect to reasonable values for 3 different parameters. Even with extreme values of subsample ratio, column sample ratio and coefficient lambda, the deviation is still significant, and we realize that the volatility of sale price may be the internal problem and hardly to be captured without considering time and housing policies. In this case, we specify parameters to the value whose rmse of validation is lowest to create a final model.

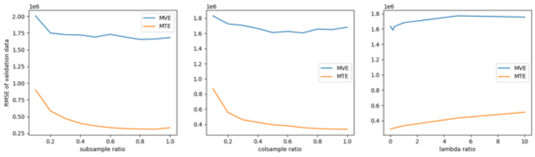


Fig. 11. *Performance of different hyper parameters against over-fitting*

b) *Model and outcome*

According to the results of grid search and sensitivity analysis, the final hyper parameter combination is as follows:

TABLE II. HYPER PARAMETERS OF THE FINAL MODEL

para	**Max depth**	**# Estimators**	**Learning rate**	**Sub sample**	**Col sample**	**lambda**
value	5	500	0.1	0.8	0.7	0.2

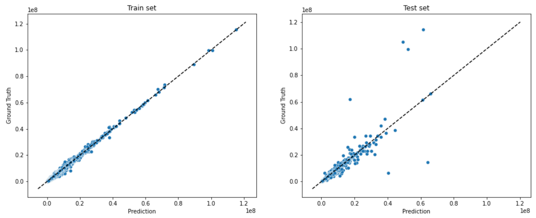


Fig. 12. *Prediction of XGBoost model*

Figure 12 illustrates the performance of XGBoost is superior to the algorithms discussed above, and the root mean squared error of the validation set decreases 10% and drops at 1980000 (random state = 20). With the final model, here explores the prediction error of the model on different housing characteristics such as size sqft, location, and property type due to overfitting.

As for size sqft, although there are fewer training samples for super-sized units (size sqft > 10000), the forecast error is smaller than that of units whose size sqft ranges from 0 to 7,500. Figure 13 illustrates that it is more difficult to predict the resale price of houses with intermediate sale sqft, especially that range from 0-2500. Too many transaction records with such sale sqft, feature diversification as well as time issue may all have negative influence on XGBoost regressor and contribute to a big deviation in predicting prices of these houses. On the other hand, the bias between training error and validation error (refers to the overfitting problem) is attributed to the prediction result whose scaled error is larger than one. Therefore, one potential way to increase model generalization is diving into these odd results to find the similarity of these records, and turning back to the preprocess part again to increase model's generalization.

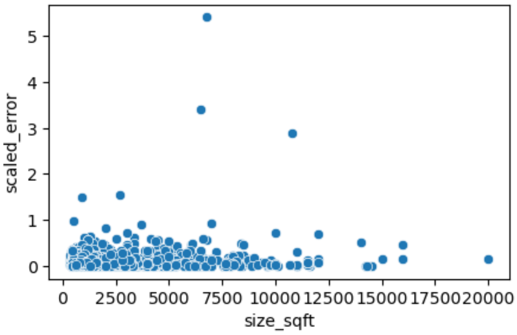


Fig. 13. *Prediction deviation for different sale_sqft*

In terms of location, the scaled error for the majority of areas is between 0.2 and 0.4, except some special regions, such as Marina Bay, Clementi and Bukit Batok. Figure 14 shows that the scaled error of these areas is much higher and reaches almost 0.8. Therefore, the price per square foot in these three places fluctuates so greatly that we need to assess the price with external factors or the help of real estate experts.

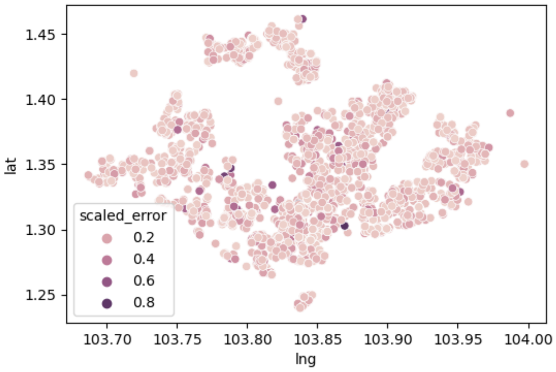


Fig. 14. *Scaled error for different location*

When considering property type, apartment houses have the lowest forecast accuracy overall, with an average percentage of forecast error as high as 0.15, while townhouse houses have the highest forecast accuracy of about 0.025. And the forecast errors for the rest of the housing types are basically between 0.05 and 0.10. To make an improvement to decrease bias of prices prediction within a property type, it's possible and valuable to split data and then train an XGBoost model for each particular property type.

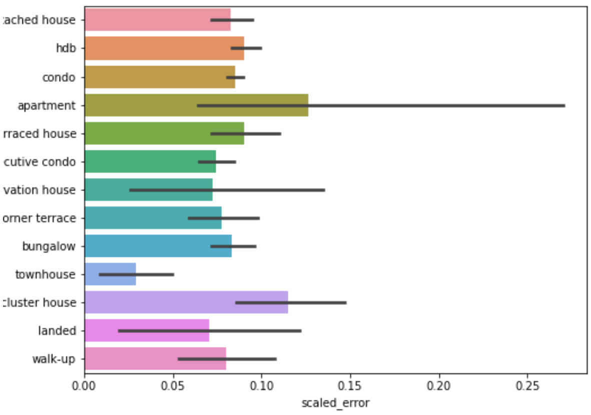


Fig. 15. *Scaled error for different property types*

6) *Prediction:*

We estimate the resale price of test.csv with the final model we discuss above and submit the result in Kaggle.

III. Task 2 Recommendation Engine

When a user browses properties on 99.co, after clicking into the detail page of a certain property, there will be a 'Similar listings' section on the bottom to recommend similar properties. In this task, we need to design and implement a model to give recommendations based on our task comprehension.

A. *Motivation*

The motivation for recommendation system in real estate website can be represented in two aspects, for website owners and for users.

- For website owners, they can recommend similar properties to users, and this may increase their property sales number. Besides, precise recommendation can improve users' experience.
- For users, recommendation system can help them find suitable property in a shorter time and better experience as they can always click on the recommended property links and view all similar properties, instead of continuously searching.

B. *Comprehension*

The goal of this task is to choose and design a suitable recommendation model as there are multiple recommendation methods and implement this model to make recommendations. Besides, we regard this recommendation problem as two subtasks [4], one is to collect multiple recommended properties for a certain property, and the other is to rank these recommended properties as there should be an order for webpage display.[5]

C. *Assumptions*

Referring to reality website browsing situations and task comprehension, we have following assumptions,

- For those users who do not sign up while browsing the website, users' information is lacked, and we cannot get their basic personal info nor their browsing preference.

- For those registered users, their browsing preference may change dramatically as their demand could change with age. For example, a person may prefer condo properties with smaller size for high-quality life in their twenties and turn to HDB with larger size for raising children in their thirties.

D. Different Approaches Discussion

There are three types of recommender systems and some naive recommendation approaches according to lectures and in the following we will discuss whether these systems or methods are suitable for this task in our consideration. After that, we will decide on which kind of method to use and how to design our model.

- Content-based system contains recommendations based on pairwise item-item similarity and user-item similarity. Both recommendation methods are based on known user preferences but with different calculation focus. According to our basic assumption that we lack user browsing preference, this recommendation system is not considered.
- Collaborative filtering is based on the idea that an item should be recommended to a user if other users with similar taste rated highly on this item or if this item is similar to another item which is highly rated by many other users. As for model-based CF, we import k latent representations to simulate user tastes on different items. As methods in this system are based on knowing multiple users' taste and which is contradictory to our assumption, we do not consider using this system.
- Hybrid techniques are combined from multiple other systems or methods and in this task, we do not consider this technique.
- Naive recommendation approaches are easiest to understand, and we can make recommendations based on item similarities without considering user preferences. So, in this task, we use naive approaches.

E. Model Decision and Design

From above discussions, we decide to use regression model, explicitly, LGBMRanker [6], to output final top recommendations with rank orders. As regression model need to have a training target and we cannot get rank data directly from 99.co, we have the following assumption,

- Top 10 ranking data are accessible for each listing property on 99.co. For example, webpage may rank the recommended properties by user clicking times on each property details page

and in this task, to fabricate ranking data, we use KD tree (which is based on KNN but could handle high dimensionality) to calculate top 10 similar properties and rank these 10 items regarding each property.

Overall, our final model design is to use KD tree to get ranking data and input this data into LGBMRanker model training as prediction target to predict top k recommendations with ranks.

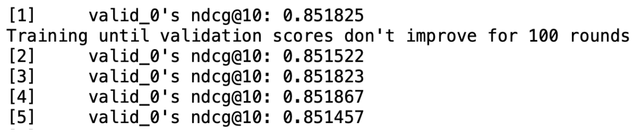
F. Model Construction

Main steps in model construction as follows:

- Build a KD tree model and train KD tree to get ranking data.
- Build a LGBMRanker model and train this model with ranking data from KD tree.[7]
- Get top k recommendations given a row id for split test data.

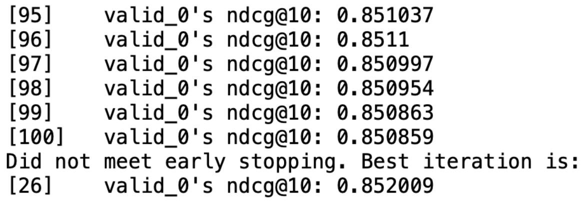
G. Model Performance

Train model for 100 rounds to get max NDCG metric and there is no obvious improvement according to figure (10) and figure (11).



```
[1]     valid_0's ndcg@10: 0.851825
Training until validation scores don't improve for 100 rounds
[2]     valid_0's ndcg@10: 0.851522
[3]     valid_0's ndcg@10: 0.851823
[4]     valid_0's ndcg@10: 0.851867
[5]     valid_0's ndcg@10: 0.851457
```

Fig. 16. Ranker model training performance (1)



```
[95]    valid_0's ndcg@10: 0.851037
[96]    valid_0's ndcg@10: 0.8511
[97]    valid_0's ndcg@10: 0.850997
[98]    valid_0's ndcg@10: 0.850954
[99]    valid_0's ndcg@10: 0.850863
[100]   valid_0's ndcg@10: 0.850859
Did not meet early stopping. Best iteration is:
[26]    valid_0's ndcg@10: 0.852009
```

Fig. 17. Ranker model training performance (2)

After model training, we plot feature importance and price is the most important feature while predicting property recommendation ranks, which suits intuition.

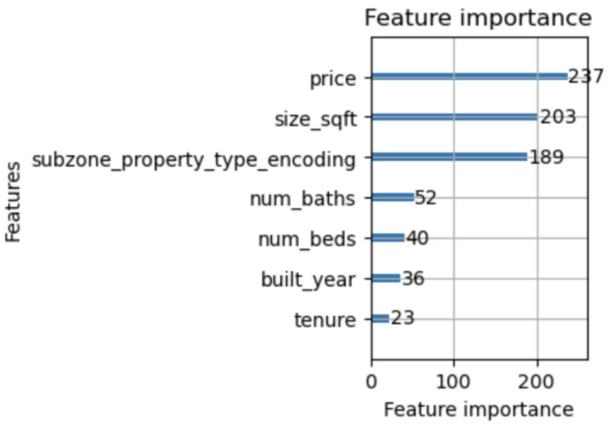


Fig. 18. Ranker model feature importance

IV. Task 3 MRT Passenger Volume Analysis

A. Motivation

MRT plays an important role in Singapore land transportation. Millions of people travel by MRT daily. As the Singapore government or SMRT corporation, protecting the everyday operation of MRT lines is crucial. Thus, we want to perform an MRT passenger volume analysis to give more insights and provide some reference on how to better organize MRT trains and manpower.

B. Methodology

Firstly, we want to identify the popular starting stations and destinations. We cannot simply assume all the MRT stations will have the same amount of passenger volume. A good estimate can be obtained from the data of properties, commercial centres and shopping malls. We want to calculate the distance between those targets mentioned above and the MRT stations. Based on the number of different targets within a certain range from the MRT station, we further categorize MRT stations into living, working and shopping ones. Top stations will be selected from each category and most passengers should travel among these stations. This is our simulation of the passenger flow.

Secondly, we realize the busy stations might be different at different times. For example, most people will travel between home and work places during weekdays. While at weekends, most people should travel between home and shopping malls. Thus we will group living stations and working stations to analyze for weekdays, then group living stations and shopping stations to analyze for weekends.

Lastly, we will borrow the idea of betweenness centrality to simulate the passenger flow. We will first find the shortest paths between the living station and the working/shopping stations. Then we aggregate the stations along those shortest paths and count the occurrence of those stations. The stations with the top occurrence number should experience heavy passenger flow.

C. *Results and Discussion*

We did the analysis for weekdays scenario and weekends scenario separately and select top ten stations. Below is our result:

Weekday: 1. little india; 2. dhoby ghaut; 3. newton; 4. chinatown; 5. botanic gardens; 6. bugis; 7. farrer road; 8. clarke quay; 9. stevens; 10. buona vista.

Weekend: 1. dhoby ghaut; 2. little india; 3. newton; 4. chinatown; 5. clarke quay; 6. bugis; 7. botanic gardens; 8. stevens; 9. promenade; 10. fort canning.

Below are our discussions on the result:

- The top four stations for both ranking are the same, just some minor sequence differences. It implies these four stations play a critical role in the MRT system and they should be given the highest priority in the daily operation.
- We can see the majority of the stations in the rankings are from the new downtown line or interchange including the downtown line. This is a proof that the newly constructed downtown line is a great success as it fulfils the public transportation need.

D. *Future Improvements*

Firstly, our passenger volume analysis only concentrates on the stations with the most passenger flow. One thing to be extend is to analyze the interchange stations in which most passengers will alight and change the lines. Thus, we can obtain our shortest path and trace the stations to see which interchanges the MRT line will change. Then we can get more insights on which stations need more manpower to maintain the order.

Secondly, we assume the people from different living stations have equal probability of going to all shopping malls. But in reality, that may not be the case. What we can do is to limit the shortest path finding to only link the living stations and the shopping stations within 5 stations, or the nearest shopping station. This will better fit the real-world scenarios. Furthermore, this can predict if the load for some MRT stations will drop if the government decides to build some new shopping malls near some MRT stations.

V. Conclusion

From this project, we utilize the Singapore housing dataset to perform a comprehensive EDA and meaningful preprocessing. After preprocessing the data, we complete three concrete subtasks. First one is resale price predictor, we can achieve a general percentage error of 20%-40% for the price, which is considerably good enough result for complicate task of price prediction. Second one is taking the common pain points of most property websites into account, we propose a feasible and concrete methodology to build the recommendation engine. With two previous tasks focusing on the home buyers and website companies, we perform an MRT passenger volume analysis in the end to provide more insights on how the government and SMRT corporation can better plan the MRT operation.

References

[1] Tenure comparison https://www.99.co/singapore/insider/freehold-vs-leasehold-condos/

[2] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939785

[3] How distance affects priority admission https://www.moe.gov.sg/primary/p1-registration/distance

[4] An analysis on a constructor of how Zhihu give out search recommendation and ranking and tells why ranking is improtant in recommendation problems: https://zhuanlan.zhihu.com/p/53518665

[5] Ranking learning in recommender system: https://cloud.tencent.com/developer/article/1710131

[6] Principles and usage of LGMBRanker, which is a ranking algorithm in LearningToRank (LTR): https://blog.csdn.net/wuzhongqiang/article/details/110521519

[7] Gradient Boosting Raning Algorithm: https://medium.com/@raghavbhutani41/gradient-boosting-ranking-algorithm-lightgbm-667050ddda