'I Need a Place to Settle Down!' CS5228 Final Project Report Team Maka-Baka

Zeng Jia
E0966278
A0256694M
e0966278@u.nus.edu

Li Guanzhen
E0966232
A0256648N
e0966232@u.nus.edu

Lian Ziyue
E0962963
A0255951X
e0962963@u.nus.edu

Fu Shishan
E0678901
A0229394U
e0678901@u.nus.edu

***Abstract*—This document is the final report for the project of CS5228: Knowledge Discovery and Data Mining, National University of Singapore, AY22/23 Semester 1. In this project, we completed 3 data mining tasks, including the Prediction of Property Prices, Property Recommendation, and Determination of User Portraits.**

I. Motivation

The housing market is always significant and interesting, especially in places such as Singapore, which is considered relatively crowded as compared to other countries/cities.

We are given a dataset of property details and prices in Singapore. The geographical locations of the properties are also available. Moreover, we have access to the locations of MRT stations, shopping malls, and schools.

Based on the datasets given, we have come to design the following 3 tasks to tackle:

A. Task 1: Prediction of Property Prices.

The first task is to determine the property prices based on the details of the properties. We plan to extract meaningful features from the basic dataset, use the geographical locations of important landmarks (MRT stations, shopping malls, and schools), and fit in the suitable data mining models for training. We then use the trained models to predict the unseen prices in the test set.

B. Task 2: Property Recommendation

The second task is to design and implement a recommendation engine. With one single input from the user or a list of properties browsing history of a user given, the recommendation engine can recommend alternative properties to the user so that the user can explore more properties that meet his/her interest.

C. Task 3: Find your next buyer!

In task 3, we aim to build user portraits for the housing trading platform to categorise their users and offer services that can meet their personal needs.

User portrait theory is a set of management tools that abstract and classifies the objects of products and services in the field of marketing.[3] The user portrait has been widely used in existing research, Dalia(2012) applies user portrait to analyse travel agencies' customers[5], Jing(2019) implements user portrait model to analyse their purchasing behaviours[6], Hongpeng(2020) researches the artwork market based on user portrait theory[8], etc. Besides, task 3 is based on an assumption that each sample in the training dataset represents a buyer's transaction record, and the dataset consists of many different users' buying histories.

We plan to cluster users into different groups with different preferences over price, transportation convenience, education, room size, etc. These results can help the trading platforms to abstract the diversified users, accurately locate target users and make recommendations with a higher transaction success rate, which might greatly grow the turnover.

II. Exploratory Data Analysis

A. Missing Values

1) Tenure: Firstly, we filled in '99-year-leasehold' for all HDB properties with missing tenure values. We filled in the same values as those samples with the same property_name and address. We filled samples that has no other sample with the same property_name and addressbased on the subzone and planning_area.

2) Built_year: For the rows with missing values of built_year, we filled in the same built_year value with the same property_name and address. For the rest of them, we filled in with the nearest record based on lat, lng, subzone, and planning_area.

3) Num_beds and Num_baths: We simply filled in 1 for all missing value rows with 'studio' property_type. We also added num_beds ad num_baths together as a new feature num_rooms. Then, we used linear regression on num_rooms and size to fill missing values in num_rooms.

B. Drop Values

1) Price: We deleted the rows with 0 as price.

2) Subzone and planning_area: we deleted the rows with missing value on subzone and planning_area.

TABLE I
PROPERTY_TYPE ENCODING

Labels	Property_type
0	hdb, hdb executive, walk-up, executive condo, shophouse
1	condo, apartment, landed, terraced house, cluster house
2	townhouse, corner terrace, good class bungalow, semi-detached house
3	bungalow, conservation house

TABLE II
TENURE ENCODING

Labels	Tenure
0	99-year leasehold, 110-year leasehold, 103-year leasehold, 102-year leasehold, 100-year leasehold;
1	999-year leasehold, 946-year leasehold, 956-year leasehold, 929-year leasehold, 947-year leasehold;
2	freehold;

3) Other columns: We deleted the columns listing_id, title, address, property_name, available_unit_types, total_num_units, property_details_url. Elevation and floor_level were also deleted due to the high portion of missing values.

C. Features' Encoding

1) Property_type: We firstly changed all property_type to lowercase , and aggregate some similar types to reduce the computation. For instance, we aggregate *hdb 2 rooms*, *hdb 3 rooms*, *hdb 4 rooms*, *hdb 5 rooms* as a single type called *hdb*. Property_type is a contributing factor to a property's price. According to different types' average prices(Figure 1), we decided to encode this feature as an ordinal attribute. More specifically, we divided all types into 4 classifications, and the labels are shown in Table I.

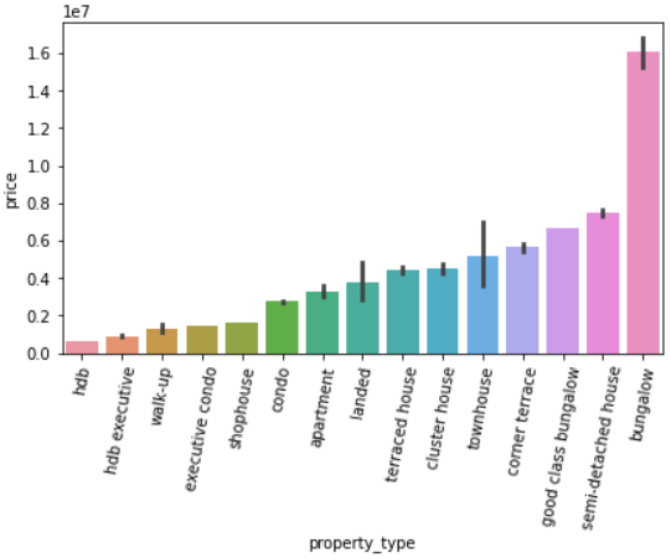

Fig. 1. Each property_type's average price

2) Tenure: The attribute tenure can be obviously split into three types, specific labels are illustrated as Table II.

3) Built_year: As we all know, a country's economy fluctuates within a certain cycle, and the prices of houses also fluctuate consequently (as Figure 2 shows). Hence, instead of directly using the exact year to train the models, we decided to split the built_year into 5 periods. More specifically, the period 1963-1970 was assigned the label 0, the period 1971-1990 was assigned the label 1, the period 1991-2005 was assigned the label 2, the period 2006-2020 was assigned the value 3, and the period beyond 2020 was assigned the label 4.

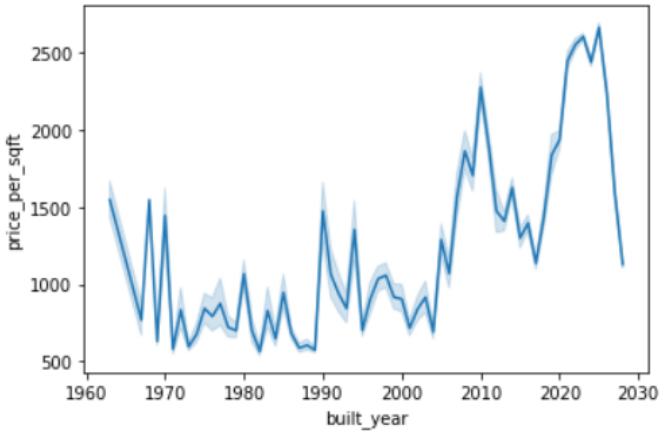

Fig. 2. Each year's average price

4) Auxiliary data: To make use of the auxiliary data, we used distance to generate new features. Take the MRT Stations' locations as an examples, we calculated the distance between each house and the nearest MRT Station, and used the distance as a new feature dist_mrt. We processed other five datasets in a similar way, and finally chose three features (dist_mrt, dist_pri, dist_mall) because of better performance in experiments.

5) Subzone: By plotting each subzone's average house price (Figure 3), we found that different subzones' average prices differ significantly. Hence, we used each subzone's average price to indicate the regional price level.

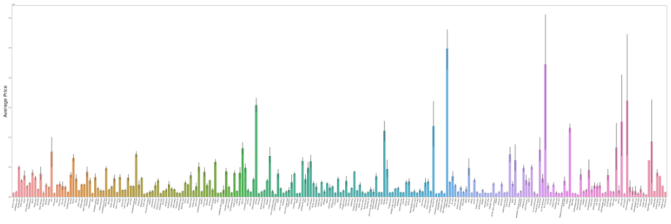

Fig. 3. Each subzone's average price

D. Outliers Detection

1) Unreasonable number of rooms: As some samples come with unreasonable number of rooms or unreasonable ratios (e.g. 1 bedroom with 10 bathrooms), we treat them as outliers and remove these data samples.

TABLE III
THE DESCRIPTION OF UNIT PRICE

Mean	12701.92
Std	125749.2
Min	1.05
Max	1.691484×10^8

There are also some samples with unreasonable ratios between the size and the number of rooms (e.g. 1000 sqft with 20 bedrooms). These samples are also regarded as outliers and need to be removed.

We firstly introduces two pairs of ratios (totally four ratios): beds_to_baths, baths_to_beds, sqft_to_rooms, and rooms_to_sqft for each sample. We then used DBSCAN from *sklearn* to identify outliers by each ratio respectively. The outliers detected in the four rounds of DBSCAN are removed.

2) Unreasonable unit price: As price is the variable to be predicted, it is important that we have removed all price outliers before analysing the data. Different houses have different sizes, so it was necessary to transform the attribute price into price per square feet to detect outliers.

According to the boxplot (Figure 4) and the description (Table III) of all data's price per square feet, there actually exists outliers.

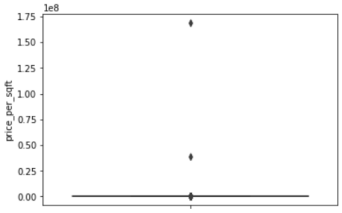

Fig. 4. Boxplot of Unit Price

we used the 3-sigma rule to detect outliers. Let μ, σ denote the mean value and standard error of data points respectively, the outlier removal process is as follows: In each iteration, we find out all data points locating out of the scope $[\mu - 3\sigma, \mu + 3\sigma]$, and remove them from the train dataset. We repeat the above steps until all data points are within the 3σ scope. After that, we get new train data with the following distribution (Figure 5 and 6).

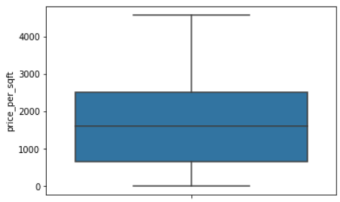

Fig. 5. Boxplot of Unit Price

We can notice that some prices are so cheap that they are obviously unreasonable, so we remove all the data with unit price less than 300. After Processing the data, we get the train dataset, with the following heat map (Figure 7).

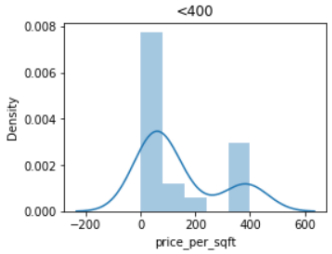

Fig. 6. Distribution of Unit Price which is less than 400

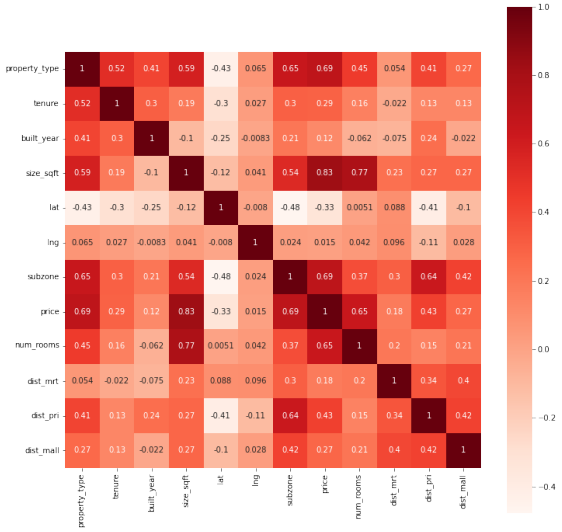

Fig. 7. The Heat Map for training data set

III. DATA MINING METHODS

In this section, we will illustrate all the data mining methods we used for all three tasks.

A. Task 1: Prediction of Property Prices

The goal of this task is to predict unknown property prices according to the provided features, including houses' basic attributions, geographical location, etc. In this section, based on the previous EDA results, we mainly tried Regression Models, Tree-based Models, Ensemble Learning Models, Deep Learning Models, and some other commonly used models to complete the prediction task.

1) Linear Regression Models: Linear Regression models are the simplest and the easiest methods to be implemented, which are always an entry point for a data mining task. In this task, we tried the standard Linear Regression model, as well as its two regularised variants: Lasso Regression and Ridge Regression.

Firstly, we tried the standard Linear Regression model by directly importing the model from *sklearn*. There is no hyperparameter available for tuning for Linear Regression.

Secondly, we implemented Lasso Regression from *sklearn*. Lasso is short for 'least absolute shrinkage and selection operator', which performs regularisation and variable selection for linear regression. Lasso regression penalises the variables that deviates from the mean largely. When some of the coefficients are set to zero, this is equivalent to 'removing' the corresponding variables. This reduces the complexity of the model and the multi-collinearity between variables. Lasso enhances the performance for a linear regression model by removing some less-related variables, and also by regularising the variables to prevent dominance of any variable.

$$L_{lasso}(\hat{\beta}) = \sum_{i=1}^{n}(y_i - x\prime_i\hat{\beta})^2 + \lambda\sum_{j=1}^{m}|\hat{\beta}_j|$$

There is one hyperparameter in Lasso Regression: Alpha. Alpha can be any number greater or equal to zero, and the larger the Alpha is, the larger the regularisation. When Alpha is zero, the model is equivalent to a standard Linear Regression model. When Alpha is high, most of the variables are discarded, and only few of them are used to construct the regression model.

We used Grid Search to tune the model by the value of Alpha, and we got the best result from Lasso Regression when Alpha = 442.

Lastly, we implemented Ridge Regression from *sklearn*, another regularised form of Linear Regression. Similar to Lasso, Ridge Regression uses Alpha as a penalty factor to shrink the sizes of the coefficients. What is different is that Ridge Regression penalises the coefficients on their squared values instead of the original value.

$$L_{ridge}(\hat{\beta}) = \sum_{i=1}^{n}(y_i - x\prime_i\hat{\beta})^2 + \lambda\sum_{j=1}^{m}w_j\hat{\beta}_j^2$$

We again used Grid Search to tune the model by the value of Alpha, and we got the best result from Ridge Regression when Alpha = 1.84.

2) Tree-based Models: In consideration that tree-based models always perform well with good interpretation of results, we tried this type of models and implemented them by directly importing from the *sklearn* package. We implemented three tree-based regressors here, Random Forest Regressor, Decision Tree Regressor, and Gradient Boosting Tree Regressor.

To begin with, Decision Tree Regressor has the lowest time cost. We used Grid Search to tune two parameters here: max_depth, and ,min_samples_split, which can help to avoid overfitting. After the Grid Search, the Decision Tree Regressor performs best when max_depth = 110 and min_samples_split = 6.

Random Forest Regressor uses both Bootstrap method and Bagging Method to avoid overfitting. When tuning the model, as the parameter min_samples_split = 2 always leads to the best performance, we mainly adjust n_estimators, and max_depth two parameters. The Random Forest Regressor performs the best when n_estimators = 100 and max_depth = 230.

Gradient Boosted Tree Regressor applies boosting method to use multiple weak regressors to reduce the error continuously, which also makes it have the highest time cost. During the training, we found that its training time increases significantly as the parameter n_estimators increases. But the performance of the model is also improved consequently, before the model converges well. Hence, we mainly tuned learning_rate, max_depth, n_estimators three parameters here. When learning_rate = 0.1, max_depth = 10, n_estimators = 100, the model performs best.

As all tree-based models have the common parameter max_depth, we also create a plot to record the process of tuning. Figure 8 illustrates the relationship between all models' MSE(Mean Squared Error) for each fold and their corresponding max_depth.

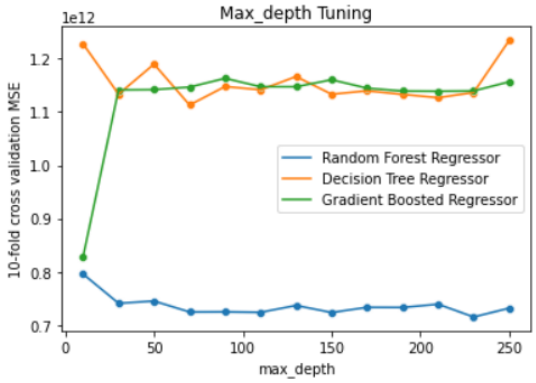

Fig. 8. Max_depth Tuning Process for Tree-Based Models

3) Ensemble Learning Models: As our tree-based models performed well, we explored on the ensemble models further. We tried AdaBoost, XGBoost and LightGBM respectively.

AdaBoost is short for 'Adaptive Boosting'. The algorithm continuously trains a model, calculates the error, and sets the weight for this tree as well as the coefficients of the variables. We used the AdaBoost implementation from *sklearn* with the classic Decision Tree that was tuned earlier on. We tuned the AdaBoost model by n_estimators and learning_rate, and got the best results with n_estimator = 50 and learning_rate = 1.

XGBoost is short for 'eXtreme Gradient Boosting'. It is an implementation of Gradient Boosting. In this project, we used XGBRegressor from *xgboost*. Similar to AdaBoost, the hyperparameters available for tuning are n_estimators and learning_rate. We used Grid Search to find out the best hyperparameters are learning_rate = 0.1 and n_estimators = 900.

LightGBM (Light Gradient Boosting Machine) is another Gradient Boosting framework. We made use of LGBMRegressor from *lightgbm*. We again tuned to model by n_estimators and learning_rate, and got the best results at n_estimators = 0.1 and learning_rate = 250.

As almost all ensemble learning models have the parameter n_estimators, we also record this parameter's tuning process of different models(Figure 9).

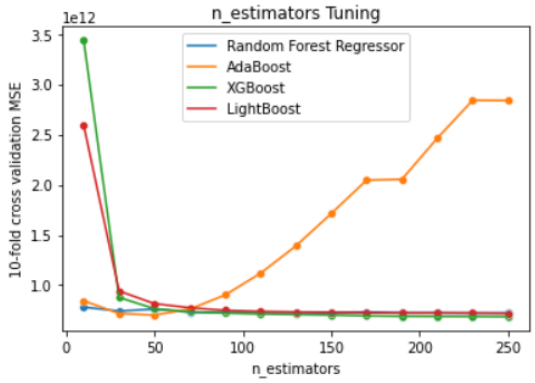

Fig. 9. n_estimators tuning

4) Deep Learning Models: As Deep Neural Networks (DNN) are commonly used for prediction tasks nowadays, and DNNs always have a strong capability to extract features, we also try implementing some deep learning models here. More specifically, we use Pytorch framework to implement the baseline Multilayer Perception (MLP), and integrate Attention Mechanism to improve the baseline MLP models. Besides, we also propose some Hybrid Models, which means using Deep Learning Method to combine multiple Machine Learning Regressors together, to improve the overall performance.

Besides, deep learning models always need lots of time to converge, we develop the experiments in the *Google Colab* environment with GPU resource to accelerate the training process, and we implement the DNNs based on the *Pytorch* framework. The details are as follows.

For all the deep learning models here, we set *MSELoss* as the loss function, and use *Adam* to optimize the models. More specifically, we set the optimizer's learning rate as 0.0001, β_1 as 0.9, β_2 as 0.999.

Firstly, we implement the Multilayer Perception(MLP), which is one of the most basic deep learning models. It consists of two fully-connected linear layers and an activated layer between them. Besides, the dropout layer is also introduced to alleviate the effect of overfitting.

Nowadays, a common method to optimize the baseline deep learning models is Attention Mechanism. Since the proposal of BERT (Bidirectional Encoder Representation from Transformers[2]) Model in the field of NLP (Natrual Language Processing) and Transformer Model[7] in the field of CV(Computer Vision), the Attention Mechanism has been more and more widely used. In this model, we try introducing the Self-Attention Mechanism to assign a weight for each feature while training. The architecture for the model is shown in Figure 10.

While training the above two models, we also record each model's loss value to illustrate their training process(Figure 11). The two loss curves also indicate that how the Self-Attention Mechanism improves the baseline Mlp Model.

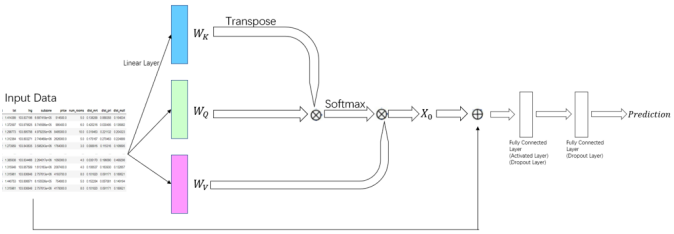

Fig. 10. The Architecture for Mlp+Attention

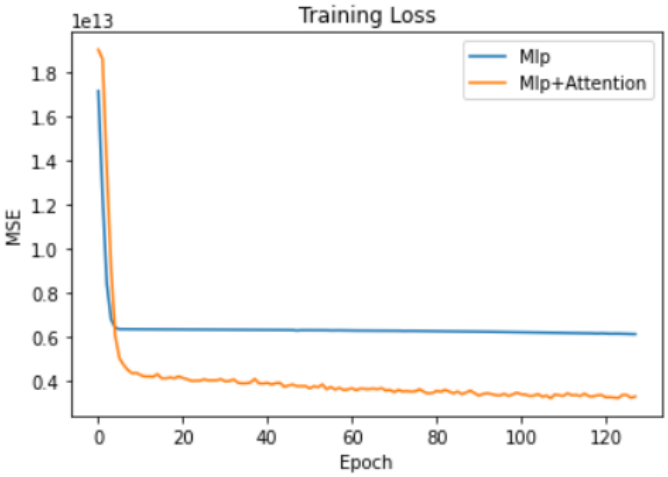

Fig. 11. Loss curve for Mlp and Mlp+Attention

To simultaneously get multiple state-of-art Machine Learning Models' advantages, we also design a deep learning based hybrid model. Figure 12 shows the architecture of the hybrid model. More specifically, we use a fully-connected linear layer(with activated layer and dropout layer) to process the results of Random Forest Regressor, Decision Tree Regressor and K-Neighbor Regressor. To avoid overfitting, we also implement the k-fold cross validation, and the MSE loss value is significantly reduced compared with any single well-performed algorithm, like Random Forest, XGBoost, etc. The specific results will be shown later in the evaluation section.

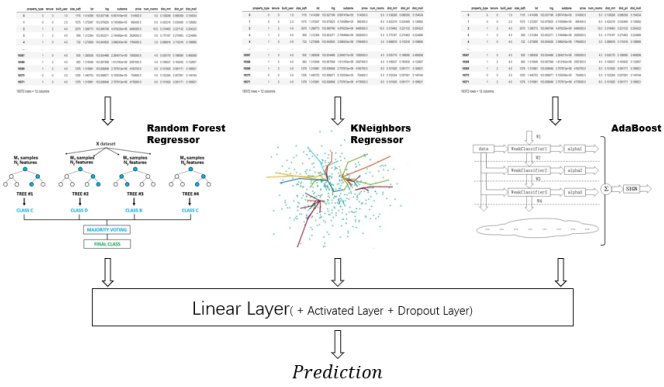

Fig. 12. Architecture for Hybrid Model - 1

The loss curve during the training process is as Figure 13 shows, and the 10-fold cross validation MSE results for the 10 models is shown in Figure 14.

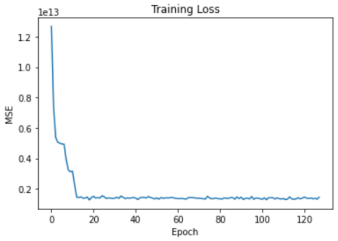

Fig. 13. Loss Curve for Hybrid Model

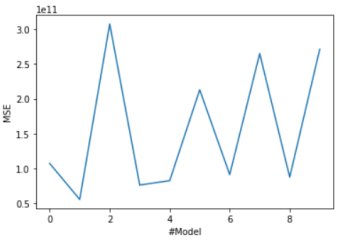

Fig. 14. 10-fold cross validation results for Hybrid Model

Besides, we also notice that in the training dataset, there are multiple transactions which are related to the houses from the same block (lat and lng) with the same size. Hence, some unknown data in the test set may be obtained by searching similar houses from the training set, which is a part of the KNN-Variant algorithm that we'll propose later. Based on the assumption that similar houses at the same block always have similar prices, we use the searching method to replace some of the results of hybrid model. We denote this method as hybrid model-2 in the following, and its architecture is shown in Figure 15.

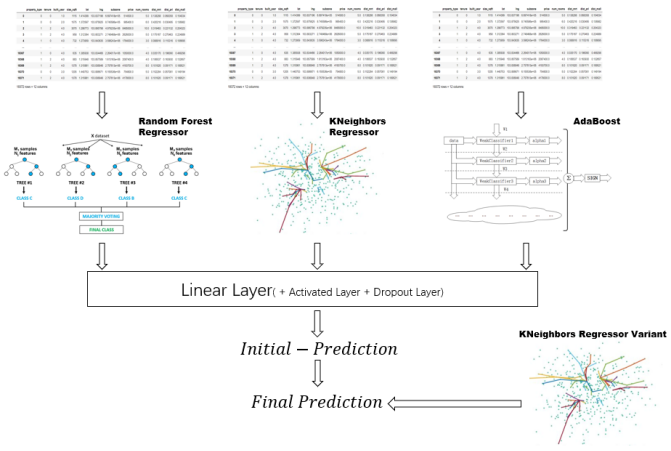

Fig. 15. Architecture for Hybrid Model - 2

5) Other Models: We also try implementing some methods that cannot be classified into a group, including K-Neighbors Regressor and SVR (Support Vector Regressor).

Firstly, we implement the K-Neighbors Regressor. Before training the model, we normalize the training set, to make each feature's weight equal. As the parameter n_neighbors affects the model's performance significantly, we mainly use Grid Search to tune this parameter, and we found that the model performs best when $n\_nrighbors = 2$.

Besides, as mentioned above, we also propose a KNN-Variant algorithm, which is based on the assumption that the similar houses located at the same block always have similar prices. Hence, when predicting the price of a house in the test set, if we can find the transaction records of houses locating at the same block(with the same lat & lng) in the training set, we can use the average price of these houses as the prediction. Or, we find the house nearest geographically and regard that it has the same unit price as the house we predict. And then, we can get all houses' prices predictions.

Next, we also implemented SVR.

SVR is short for 'Support Vector Regression'. It is the regression implementation for SVM (Support Vector Machine). The goal of SVR is to find decision boundaries for all the samples. We used LinearSVR from *sklearn.svm*. The tuning involves the regularisation factor C. We tuned the model by Grid Search, and got the best results when C = 9795918.

B. Task 2: Property Recommendation

For this sub-task, the main goal is to build a recommendation engine to recommend properties to a certain user based on the data given by that user. Since we only have data of one user, we choose to use a content-based recommender system among all different recommenders, for example collaborative filtering recommender and model based recommender, which requires multiple user rating histories. In this section, we also explored two types of content-based recommender systems:

1) Pairwise Item Similarity:

- Assumptions: The recommendations are given based on one single property which was searched by the user. The important features are predefined and expressed as weights in the calculation step.
- Preparation: All the features of each property record in the dataset is normalized for later calculation.
- Steps: We randomly choose a row from the dataset as the input property. This property is used as a baseline and we iteratively calculate the cosine similarity between its features with all the other properties in the dataset. The result is simply sorted in the descending order of similarity scores. And the input record is removed from the recommendation list. Users can define the number of items in the list by passing it as a parameter to the recommender systems.

2) User-Item Similarity:

- Binary Utility Matrix: User profile generated by mean of browsing history.
  - Assumptions: A set of user property viewing history is given. As long as the user has a viewing history of this property, we assume that user has shown

a certain level of interest. Also, the same feature weights as Pairwise Item Similarity is applied to the item similarity calculation.

- Preparation: In real life, a potential buyer usually views a big portion of similar properties that meet their requirements, and a small portion of resale houses that are very different from their target house. To make the browsing history relevant, the 50 viewing records are generated in the following way: 1. 45 property records are generated by using the above Pairwise Item Similarity recommender system. 2. 5 property records are selected from the rest of the dataset randomly.
- Steps: 1. User profile is simply calculated based on average of different features from user browsing history. 2. User profile and dataset are normalized, then we calculate cosine similarity between user profile and other properties. 3. Sort the properties based on the similarity score and return top properties based on user request.

- Real-Valued Utility Matrix: Add a property viewing time as the user ratings for user profile.
  - Assumptions: Different from Binary Utility Matrix, besides a set of user property viewing history, the viewing time of each property is also given. We assume that the longer this user stays on this page, the more interested is this user. Therefore, this viewing time is acting as user ratings in this assumption.
  - Preparation: Same as above, the viewing history also contains 50 viewing records. It contains 25 user interested property records, which have longer viewing duration which is 120 sec - 180 sec and 25 of user uninterested properties, which have viewing time between 30 sec to 90 sec. The interested properties are still chosen by using the above Pairwise Item Similarity recommender system. For each record, a random number between 120 to 180 is given as the 'stay time' feature. Likewise, the user's uninterested properties are randomly selected from the rest of the dataset and the 'stay time' value within the range of 30 and 90 is randomly generated as a new column. These two lists are concatenated as the property viewing history for the user.
  - Steps: 1. To reveal the user's preference, Real-Valued Utility Matrix is used to normalize user ratings. Therefore we can have both positive and negative weights for each record feature. 2. User profile is generated based on these normalized rating weights. 3. Same as the above method, cosine similarity is used as the measurement, and recommendations are returned in order

C. Task 3: Find your next buyer!

Firstly, we learned about the distribution of properties in Singapore by plotting heatmaps, clustering properties only by their positions, and visualizing them.

Then, we applied the K-means clustering algorithm to gain the local distribution. The knees of the curve appear both at around 4.

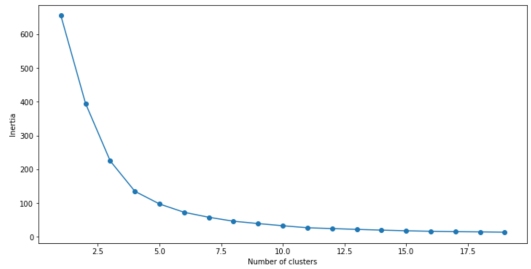

Fig. 16. K-Means HDB

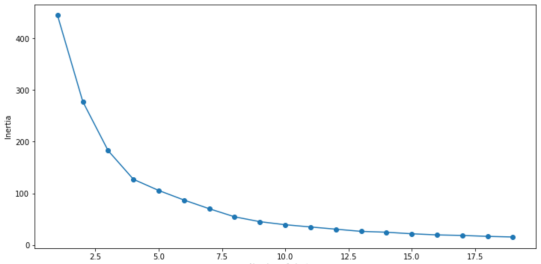

Fig. 17. K-Means Condo

After that, we start to build the user portrait based on the knowledge. The following features are considered important factors in the user portrait:

- Property price
- Price per square feet
- Size
- Number of rooms (bathroom + bedroom)
- Built year
- The average price in the sub-zone
- Distance to MRT Station
- Distance to Primary School
- Distance to Secondary School
- Distance to Shopping Mall
- Distance to Commercial center

Then we apply K-Means on the dataset and find the knee of the curve around 5.

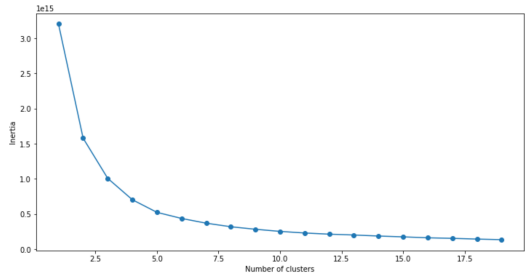

Fig. 18. K-Means User

So we combined many features from the datasets given and clustered the users into 5 groups, each of which has a different

preference over price, transportation convenience, education, room size, etc. These results can help the trading platforms to accurately locate target users and make recommendations with a higher transaction success rate, which might greatly grow the turnover.

IV. Evaluation and Interpretation

A. Task 1: Prediction of Property Prices

To evaluate and compare the performance of different models, we use two indicators: MSE and the score on Kaggle Leaderboard. The specific equation for MSE is as follows:

$$MSE = \frac{\sum_i^n (\hat{y}_i - y_i)^2}{n}$$

Besides, while evaluating the models, we use 10-fold cross validation to calculate each fold's MSE. The smaller the MSE value is, the better the model performs. The specific MSE, Kaggle score and time cost are shown in Table IV.

From the result, among baseline models, Tree-based Models and Ensemble Learning Models perform significantly better, while regression models, KNN and SVR lead to a relatively high MSE. Besides, among all models, the two Hybrid Models that we proposed perform significantly better than all the other models, especially when they're evaluated by the MSE for each fold. The Hybrid Model's MSE is almost 4 times smaller than four Ensmeble Learning Models.

As for the time cost, boosting and bagging methods generally have a higher time cost compared with the other models. However, we also notice that LightBoost performs well in terms of the time cost. Besides, although we use GPU to train the deep learning models, it still needs a siginificantly longer time to converge.

Generally speaking, the Hybrid Model-2 performs best among all models, no matter evaluated by 10-fold cross validation MSE or Kaggle Leaderboard Score.

B. Task 2: Property Recommendation

1) Result evaluation: The top 3 recommendations for the Pairwise Item Similarity recommender system is shown in TABLE V. User input is row 10 of the dataset as shown on the top row.

The top 3 recommendations for the User-Item Similarity Binary Utility Matrix and Real-Valued Utility Matrix recommender system can be found in the task 2 code file. The property viewing history for 45 user-focused properties is generated based on the Pairwise Item Similarity recommender system using row 10 as the user input.

1) Intra-list Similarity [1]
   We can calculate the average cosine similarity of the recommendation list. For the above recommendation list, the results are below:

TABLE VI
Evaluation of Recommendations

Recommender System	Intra-list Similarity
Pairwise Item Similarity	8.21×10^{-11}
User-Item Similarity Binary	7.10×10^{-12}
User-Item Similarity Real-value	3.93×10^{-8}

   As shown in the table, User-Item Similarity Binary generates the list with the slightest cosine difference between each element. However there are two potential problems with this result:
   1. The result is highly sensitive to the input data, with a different viewing time or different browsing record, this list can vary greatly.
   2. This value does not reflect the user preference, which the below measurement can solve.
2) Precision [4]
   In real life, we can measure the relevance of each recommendation item by tracking user clicking/browsing behavior later. Based on this relevant measurement, we can calculate the precision which indicates the number of relevant items among all the recommended ones to evaluate the recommendation system and improve it further.

2) Future improvements:

1) Weight adjusting
   - User-defined feature weight: In this project, both recommend systems use the predefined weight for similarity calculation. However, in reality, users' preferences for important property features may differ greatly. It will be better to provide a way for the user to set their own strong requirements for the property.
   - Auto adjust feature weight based on user property browsing data. Use a logistic regression model to study user preference and predict the weight for each feature value.
2) Right training data (for User-Item Similarity system)
   - Remove noisy history: If a type of property only appears once or twice in the history dataset, removing it will make the user profile more accurate and thus lead to better recommendation result.
   - Enough history data Small dataset has a huge bias with different page viewing duration and causes the user profile unreliable. In our project, we have to generate the same amount of user interested and uninterested data to ensure the relevant recommendation results. However in real scenarios, with increasing comprehensive user data, the user profile will become more and more trustworthy.

C. Task 3: Find your next buyer!

1) Properties Distribution: To understand the properties distribution in Singapore, we selected two main properties: hdb and condo, as our main objects of study.

TABLE IV
RESULTS FOR TASK 1

Model Type	Model	MSE for each fold	Kaggle Score on Leaderboard	Training time for each fold
Regression Models	Linear Regressor	2.58×10^{12}	410066607	0.00670
	Lasso Regressor	2.59×10^{12}	412058108	0.13933
	Ridge Regressor	2.59×10^{12}	409175856	0.00494
Tree based Model	Decision Tree Regressor	1.21×10^{12}	3735444	0.05899
	Gradient Boosted Tree	8.25×10^{11}	3163986	4.26496
	Random Forest Regressor	7.29×10^{11}	2767824	4.08448
Ensemble Learning Model	AdaBoost	7.16×10^{11}	2699032	0.95888
	XGBoost	6.71×10^{11}	3033595	2.52984
	LightBoost	7.16×10^{11}	3140131	0.22099
	Random Forest Regressor	7.29×10^{11}	2767824	4.08448
Deep Learning Model	MLP	6.10×10^{12}	4577368	98.51681
	MLP+ATTN	1.70×10^{12}	8313767	162.23587
	Hybrid Model-1	1.56×10^{11}	2925859	85.63697
	Hybrid Model-2	-	2754431	92.30815
Other Models	SVR	2.79×10^{12}	4269670	0.26039
	K-Neighbors Regressor	1.17×10^{12}	4657856	0.04064
	KNN-Variant	-	4269670	6.61176

TABLE V
TASK 2 PAIRWISE ITEM SIMILARITY TOP 3 RECOMMENDATIONS

	property_type	tenure	built_year	num_beds	num_baths	size_sqft	lat	lng	price
10	2	2	2023	2	1	646	1.329703	103.905683	1365000
13440	2	2	2023	2	1	646	1.329703	103.905683	1361400
5742	2	2	2023	2	1	646	1.329703	103.905683	1360800
3772	2	2	2023	2	1	646	1.329703	103.905683	1374900

We tried to learn the distribution from a global perspective through heat maps.

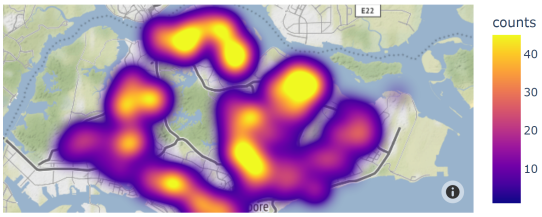

Fig. 19. HDB Heatmap

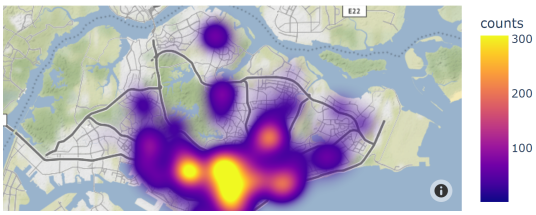

Fig. 20. Condo Heatmap

These heat maps are based on the density of houses according to latitude and longitude. From these figures, we can conclude that the condo properties are dense at the center of Singapore and relatively sparse in other places. In contrast, the distribution of hdb properties is more balanced, probably because of the nature of the house.

Then we focus on the local perspective of the property distributions. The local distribution of HDB and Condo are as follow:

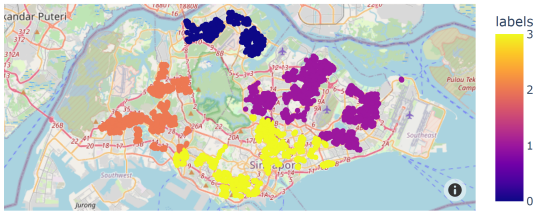

Fig. 21. Cluster HDB

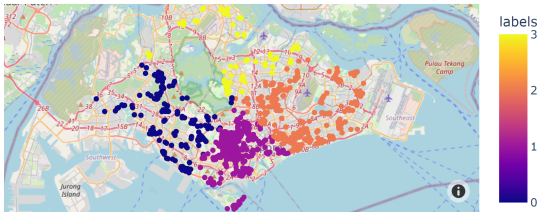

Fig. 22. Cluster Condo

From the local perspective, HDB and Condo are both divided into four parts by their locations. However, HDBs are denser than Condo in each cluster. The sparsity may be due to the small overall size of the condo building group compared with the HDB group.

2) User Portrait: We choose the radar chart to visualize the result of user-clustering.

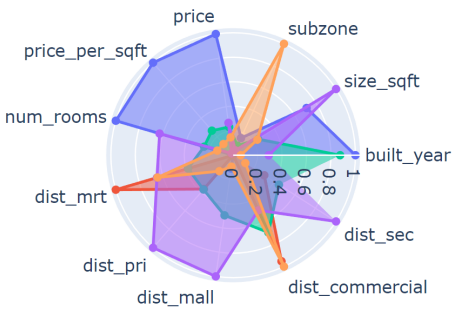

Fig. 23. Radar Chart

The 5 main types of users are:

1) Users who are willing to buy the newest, newest, largest house at a high price, without too many requirements for the location. Example: millionaire.
2) Users who attach great importance to the location, need the house to be close to the subway station, commercial center, and close to primary school and secondary school. The price can not be too high, and there are no other requirements. Example: normal families with school-age children.
3) Users who are willing to buy a new house and can afford the median price. In addition, the proximity to the commercial center can be a bonus. Example: Working middle class.
4) Users who like the big house best. Location is also important, need to be close to shopping malls and schools. The very old house is acceptable, and the price cannot be too high. Example: Retired seniors with stable income, possibly with school-age children.
5) Users who attach great importance to the distance to the commercial center and M.R.T Station. They don't care about the built year and their ability to pay is limited, which means they can only buy small and old houses. Example: young man starting a job.

Trading platform consultants can make recommendations with user portraits to increase the chance to make a deal.

References

[1] Claire Longo. *Evaluation Metrics for Recommender Systems*. https://towardsdatascience.com/evaluation-metrics-for-recommender-systems-df56c6611093. 2018.

[2] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[3] Haiqian Gu et al. "Modeling of user portrait through social media". In: *2018 IEEE international conference on multimedia and expo (ICME)*. IEEE. 2018, pp. 1–6.

[4] Muffaddal Qutbuddin. *An Exhaustive List of Methods to Evaluate Recommender Systems*. https://towardsdatascience.com/an-exhaustive-list-of-methods-to-evaluate-recommender-systems-a70c05e121de. 2020.

[5] Tomáš Pitner, Dalia Kriksciuniene, and Virgilijus Sakalauskas. "Tracking customer portrait by unsupervised classification techniques". In: *Transformations in Business & Economics. Kaunas Faculty of Humanitie* 3 (2012).

[6] Jing Sun et al. "Purchasing Behavior Analysis Based on Customer's Data Portrait Model". In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2019, pp. 352–357.

[7] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[8] Hongpeng Yang et al. "Customer Group Description of Chinese Artwork Market Based on User Portrait Theory". In: *International Journal of Frontiers in Sociology* 2.9 (2020).