CS5228 Team 46 Final Report

Ma Yuan   
A0228536Y  
e0674520@u.nus.edu

Huang LifuA0254314M  
e0954715@u.nus.edu

Jefferson Sie  
A0166887E  
e0175320@u.nus.edu

Joel HuangA0225233R  
e0575867@u.nus.edu

*Abstract*—This electronic document is a “live” template and already defines the components of your paper [title, text, heads, etc.] in its style sheet. *\*CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract*. (*Abstract*)

Keywords—component, formatting, style, styling, insert (key words)

# Introduction

Our primary responsibilities for task 1 are those related to data preprocessing, such as handling missing data, categorical data transformation, and feature extraction. In the meantime, we concentrate on feature extraction (data mining) on the auxiliary dataset to gather additional data that could enhance the model's prediction performance. A few of our data preprocessing strategies will assume that we will use tree-base models for this task. In task 2, we intend to run recommendation models and evaluate the outcomes of various solutions using the previously extracted and transformed variables from task 1. In task 3, we decided to research on out-of-distribution detection, something that has become increasingly important to consider in machine learning models due to its prevalence in intolerant domains like medicine and autonomous driving. We then discussed an application of out-of-distribution detection in recommendation systems and evaluated how such an addition could be beneficial to our model in task 2.

Figure 1: raw data information

# Task I:

## Exploratory Data Analysis & Preprocessing

### Price

Figure 2.1 shows that there are two unique price records that are outside of the price distribution. For building the model, we were concerned that the results of these 2 samples would be skewed, and we assumed that such irrational property prices were unlikely to be forecast for actual use. So, we made the decision to remove these two outstanding records. The price distribution of the reset records is presented in Figure 2.2.

Graphical user interface, application, Teams

Description automatically generated

Figure 2: 1. raw data price distribution(left) 2. processed price distribution(right)

### Title

The 'title' attribute always contains the following details: “{n bed} {property type} for sale in {location}”. Even though this information may overlap with those of other variables or be irrelevant for modelling, they can still be used to impute missing values or conduct sanity check for crucial attributes like the number of beds and property type. Figure 3 compares the features taken from the title to the current fields.

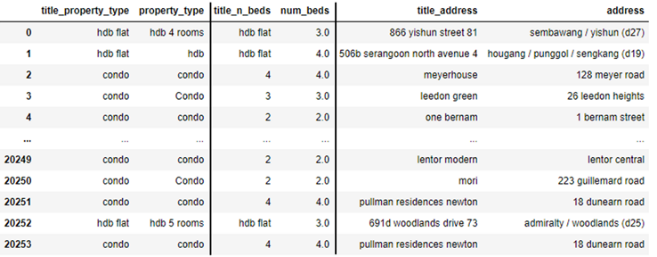


Figure 3: Features extracted from Title

### Latitude & Longitude

We primarily concentrated on the incorrect records for latitude and longitude features. Given that the goal is to forecast Singapore house property prices, the latitude and longitude ranges should be close by (1.290270, 103.851959). However, a few records are located outside the Singapore region, and further research has revealed that those records (by title, address) belong to Singapore. Since there aren't many incorrect records (Figure 4.1), we decide to use Google to find the correct location (Figure 4.2) and manually amend the entries. The results before and after the correction are shown in Figure 4.3 and Figure 4.4.

Graphical user interface, application

Description automatically generated

Figure 4: 1. unique properties with missing location (top left); 2. correct locations for imputation (bottom left) 3. before location correction (top right); 4. after location correction (bottom right)

### Property type

Transforming this property into ordinal categorical data with potential ranks is the goal of handling it. Prior to the change, we must address the issues listed below that existed in this column:

1. Uniform lower-case letters
2. Different types of HDB types/description: “hdb”, “hdb {n} rooms”, “hdb Executive”

Regarding the second issue, we chose to remove the extra "n rooms" information from the property attribute because it is already included in the num beds and num baths information. Figure 6.1 shows fewer noise property types as compared to Figure 5.

A picture containing scatter chart

Description automatically generated

Figure 5: property types and price before processing

Before converting them to a numerical representation, we decided to rank each category by its average price in consideration of improved separation and understanding by the tree-base models (Figure 6.2).

Graphical user interface, chart, application

Description automatically generated

Figure 6: 1. property types and price before processing (left); 2. average price trend given different property types (right)

### Tenure

Three main activities must be completed in order to pre-process this feature: cleaning up noisy records, handling of missing data, and categorical data conversion.

Graphical user interface, application

Description automatically generated

Figure 7: 1. conventional Singapore tenures (left); 2. tenure-price distribution before processed (right)

There are a few unrecognized tenure types in these categories when compared to the Singapore standard types of tenures (Figure 7.1) shown in Figure 7.2, which presented the raw data tenure categories. Our solution was to merge them with the neighbouring tenure categories in terms of the length of the lease.

There are 1595 missing values in this column overall, so we just placed them to a new category called "others" to handle the missing data. Additionally, we think that records falling under the "others" category can be handled by other features when utilizing a tree-based model. And the Figure 8 shows the result distribution of each category.

Since there aren't many categories in this column, we chose to apply one-hot encoding to complete the final task, categorical data conversion.

A screenshot of a computer

Description automatically generated with low confidence

Figure 8: tenure-price distribution after processed

### Built year

We only consider how to impute the missing values from this column for the "built year" attribute. With the help of other features like property type, location (latitude and longitude), and block number (extracted from title address), we used a strategy to impute the missing records as much as we could. We found a common group that might contain the unique built year and used it to impute the missing records that belong to the same group.

After the imputation, we were able to reduce the number of records missing the built year from 922 to 494. Regression algorithms like XGboost and LightBoost, which can handle the missing value by default. In the case of other regressors, we simply discard these incomplete records. The graphs in Figure 9 below demonstrate the relationship between built year and price for the top 4 types of properties by population

Graphical user interface, application

Description automatically generated

Figure 9: top 4 property types built-year-price distribution

### Num beds

The 80 missing values for the number of beds were imputed using an approach that mainly relied on two sources: the number of beds data we obtained from the title field as indicated in earlier sections and the unique value or median value of grouping by the property size in sqft. We can handle every missing value from this field using these 2 sources. Figure 10 shows that the price increases in tandem with the number of bedrooms.

Chart, box and whisker chart

Description automatically generated

Figure 10: num\_beds-price distribution

### Num baths

The "num baths" column includes 432 missing values; thus, we used a similar procedure to "num beds" to estimate the data. In order to complete this task, we decided to group data by "property type," "size sqft," and "num beds" and utilize either the unique or median value for imputation. Only 1 record remained NaN after imputation, therefore we decided to remove it. Figure 11 shows that the relationship between the number of bathrooms and the price follows a similar trend as the bedrooms.

Chart

Description automatically generated

Figure 11: num\_baths-price distribution

### Size sqft

There are multiple incorrect records under the feature "size sqft," as we discovered. To start, we use the z score to eliminate excessively big values. Two records that included "size sqft" values more than one million have been removed from Figure 12.1. In addition, some records include exceptionally small size, and we assume this is because the incorrect unit (sqm) was used. As a result, we also convert them using the ratio 10.76391 to convert from square meters to square feet (Figure 12.2). It is clear from Figure 12.3 that the size of the property has a positive relationship with the cost of the property.

Graphical user interface, chart, scatter chart

Description automatically generated

Figure 12: 1. comparison between before and after all processes (top left); 2. comparison between before and after unit correction (bottom left); 3. crrorelation bewwten size and price(right)

### Floor level

Figure 13 shows the quantity of blank values and distinct values in the "floor level" field. Since there are too many missing records, we decided to group them together under the heading "nan" for handling missing data. It's worth noting that this method is similar to how the XGboost algorithms handle Nan values by default. Additionally, we choose to separate the total level for some records from the "floor level," which will divide all values into categories in ['ground', 'low', 'mid', 'high', 'top', 'penthouse', 'nan', ‘no\_level’]. And we defined the properties such as bungalow, townhouse, and others as "no level." Then the total level specified earlier will likewise be regarded as an independent variable Figure 14.1 illustrates how the floor level affects the price of a house for the majority of properties, and Figure 14.2 demonstrates how the price of a high-floor house varies depending on the total number of building levels on the property. We choose one hot encoder for this field since it has a small number of features for category encoding.

Text

Description automatically generated

Figure 13: numbers of missing values in floor level and unique floor levels

Graphical user interface, chart, application, Teams

Description automatically generated

Figure 14: 1. overall floor\_level-price distribution (left); 2. total levels-price distribution given high floor (right)

### Furnishing

In raw data, there are total 5 categories under furnishing feature: ['unspecified' 'partial' 'unfurnished' 'fully' 'na']. After encoding, we combine the categories "unspecified" and "na" to minimize the dimensionality. Due to the small number of features after encoding and the fact that there is no obvious correlation (Figure 15) between furnishing and price, we decided to use one hot encoder for this task.

Chart, box and whisker chart

Description automatically generated

Figure 15: top 4 property types furnishing-price distribution

### available\_unit\_types

The following 4 features can be extracted from "available unit types" (Figure 16):

1. min\_br\_available: the building's smallest number of bedrooms available
2. max\_br\_available: the building's largest number of bedrooms available
3. number\_of\_types\_available: how many distinct types of units are offered by the building
4. has\_studio: Indicates whether a building has a studio-type property

Chart, box and whisker chart

Description automatically generated

Figure 16: correlation bewteen price and 4 fratures extracted from available\_unit\_types(min\_br\_available, max\_br\_available, number\_of\_types\_available, has\_studio)

### Planning area

There are 113 records in Figure 17.1 that missing planning areas, however they are all aligned with the incorrect data listed in the "Latitude & Longitude" section. As a result, we used data from earlier studies to impute the missing information, as shown in Figure 17.2.

Graphical user interface, text, application

Description automatically generated

Figure 17: 1. numbers of missing values in planning areas and unique planning areas (left); 2. Planning area correction information (right)

Chart, scatter chart

Description automatically generated

Figure 18: plainng\_area-price distribution

### Auxiliary Datasets

There are a total of 5 auxiliary datasets: shopping mall, primary school, secondary school, mrt station, and commercial center. We often employ the same technique to extract the following data from these datasets.

1. Closest interested location:

When determining the closest point of interest, we divide the area into four categories: CR(), IEBP(), BN(), and IHL(). The features for the remaining 3 datasets are, respectively, mrt station name, primary school name, secondary school name, and shopping mall name

1. Distance to the closest interested location:

The distance from the properties to the previously indicated nearest location of interest

1. Number of interested locations given radius range:

Counting the number of sites inside the circle with the hyperparameter radius R is a method similar to the DB scan method

### Others (address, subszone, Property name, property\_details\_url, elevation)

Fields like subzone and property\_details\_url are simply removed from the model building in the case of the curse of dimensionality. Additionally, we remove elevation from the training dataset because it only contains one unique value and makes no contribution to the model at all.

### Others (address, subszone, Property name, property\_details\_url, elevation)

Fields like subzone and property\_details\_url are simply removed from the model building in the case of the curse of dimensionality. Additionally, we remove elevation from the training dataset because it only contains one unique value and makes no contribution to the model at all.

## Modeling and Hyperparmeters tuning

Task 1 house price prediction was accomplished using eXtreme Gradient Boosting (XGBoost). The motivation of using XGBoost is due to the following key algorithm implementation features [1]:

1. Implementation of sparse awareness with default handling of missing data values.
2. Different hyperparameters offer a Regularized Learning Objective for dealing with the overfitting issue.
3. Merge and prune operations are performed on quantile summaries over the data via the weighted quantile sketch.
4. Parallel learning with high efficiency

To determine the best hyperparameter settings, we utilized the package Hyperopt with cross validation (Figure 19).

Graphical user interface, text, application, email

Description automatically generated

Figure 19: Hyperparameter tuning and cross validation

## Results

As for now, our team current best result is with root mean square score of 1894611(Figure 20).

Graphical user interface, text

Description automatically generated

Figure 20: Kaggle score

# Task II: Property Recommendation

In this task, we recommend similar listings from a given query listing. Such as task can be useful to users looking to purchase houses and have difficulties searching for listings that satisfy most, if not all their preferences. It would be useful if they could select certain features that are more important to them. Therefore, we aim to develop a recommendation system that suggests the most similar listings to the user’s query while providing the option of selecting certain features.

Before we dive into the implementation of our recommendation system, we first discuss the basic idea of a recommendation engine and the considerations that resulted in our implementation.

## Recommender Systems

A recommendation engine can be built in many ways. The two broad categories are content-based recommendation systems and collaborative-filtering. In content-based recommendation systems, items are recommended based on similarities with other items in the dataset. These similarities are calculated based on some concept of distance, such as cosine similarity or Manhattan distance. The limitation here is that a reference item is required. On the other hand, in collaborative filtering, recommendations for a user are made based on similarities in preferences with other users. Since only one user is of interest in this task, we use content-based filtering rather than collaborative filtering.

## Considerations

Since we are only given listings of houses in the property market and not any user information, we will work with these listings as our items. As mentioned earlier, a limitation of content-based filtering is that a reference item is required. This is not an issue for this task as a reference item will be selected from the dataset as the query item. We choose cosine similarity as the metric to calculate the similarities between the query listing and all the other listings in the dataset as it provides simplicity and accuracy [1]. The formula for cosine similarity is given by:



## Dataset

The dataset we use for this task is the pre-processed dataset with auxiliary data produced from Task 1. This dataset is appropriate for this task for several reasons. Firstly, the auxiliary data is important as data such as proximities to primary and secondary schools are very relevant considerations in Singapore’s context. Living within a 1km radius from a primary school would provide a child with higher priority in entering a primary school; and living close to secondary schools would allow for ease of commute. Secondly, the data had already converted columns with characters to ordinal encoding or one-hot encoding, which fits the requirements of this task as cosine similarity cannot be applied to characters. The only further processing done to this dataset was to fill all ‘NA’ values with -1 as null values cannot be computed in cosine similarity.

## Implementation

When generating the cosine similarity, the most naïve method would be to concatenate all the features of the query listing and calculate the cosine similarities of this vector with the concatenated features of each listing. However, doing so would lead to extremely skewed cosine similarity scores as the features with larger values would dominate the calculation. The result would be every listing having cosine similarities of 1 or 0.9999 with the query listing. Therefore, there is a need to normalize each feature. To accomplish this, we use the MinMaxScaler function from sklearn’s preprocessing library.

Once we normalize the data, the next step is to create a feature dataframe containing all the concatenated values of each row. We then use sklearn’s cosine\_similarity function to calculate the cosine\_similarities of each feature vector with the query listing’s feature vector, obtaining a column with all the cosine similarity scores.

As mentioned in III., a useful feature of this recommendation engine for users is the option to select certain features of the listing that suit their preference. For example, a user may like everything about their current query listing except for the fact that it was built in 1971. Their ideal listing would have all the same features as their query listing (i.e., same number of bedrooms, same proximity to primary schools) except that the built year of the house is later than year 2000. To implement this feature, we sieved through all the features in the dataset to come up with a selection of features that makes the most practical sense in the context of Singapore. Structuring it this way also avoids having an overly complicated argument list that confuses users. For example, it makes less sense for users to search for a house that was built before a given year since the values of older houses are significantly lower. Thus, we make it such that when the user includes ‘built\_year’ in the argument, it means that the output will contain only listings built after the given ‘built\_year’ input value. The full list of options available for selection and their input types are explained below:

* **built\_year (int):** Filters listings built during or after the input year as aforementioned.
* **max\_price (int):** Filters listing with price at most the given input. Homebuyers often have a budget in mind when searching for a house.
* **num\_beds (int):** Filters listings with at least the input number of bedrooms. If a user is looking for a house with at most 2 bedrooms, they can filter by max\_price instead since houses with fewer bedrooms tend to be cheaper.
* **pri\_sch (bool):** Default value is False. If True, filters listings with primary schools present within a radius of 1km. Parents looking to stay near primary schools are likely applying under the scheme where priority is given if they live within a 1km radius of the school.
* **sec\_sch (bool):** Default value is False. If True, filters listings with secondary schools present within a radius of 2km. There is no priority scheme for families living near secondary schools, so if users use this filter, they are likely looking for convenience in commuting to and fro the secondary schools. As such, a distance of 2km should prove to be a short enough distance that is convenient by public transport.
* **mall (bool):** Default value is False. If True, filters listings with shopping malls present within a radius of 2km. Like secondary schools, this option is more likely for convenience in accessing amenities.
* **hdb (bool):** Default value is False. If True, filters listings that are of type *hdb* or *hdb executive*, for users looking to stay in public housing.
* **private (bool):** Default value is False. If True, filters listings that are not *hdb* or *hdb executive* (i.e., private housing), for users looking to stay in private housing.

Users can customize the input options to their preferences following this given argument schema and the output data will be filtered accordingly. Once this is done, the last remaining step is to sort the data by their cosine similarity scores in descending order and select only the top *k+1* rows in the dataframe, where *k* is the number of recommendations the user wants. The top-most row is removed as this is guaranteed to be the query listing itself, with a cosine similarity of 1. This occurs as the query listing is sampled from this dataset. The resulting dataframe with *k* listings is then returned as output.

## Results

First, we select a random row from the dataset to be the query listing as shown in Figure 21. We then do some searches to look at various results.

In the first search, we do a baseline search without any optional features as shown in Figure 22. The cosine similarity scores are very high, and the features are quite similar. The full comparison can be found in the Jupyter notebook for Task 2.

In the second search, we search for similar listings that are public housing and cost less than $500,000. Public housing are *hdb* and *hdb executive*, with column *property\_type\_cat\_0* values 0 and 1. We can, therefore, deduce that the query listing is a private property. Since no private properties cost less than $500,000, the most similar listings have a considerably lower cosine similarity score compared to the baseline, as shown in Figure 23.

In the third search, we search for similar listings that are public housing, close to primary schools and with at least 3 bedrooms, as shown in Figure 24. Like the second search, the cosine similarity score is considerably lower than the baseline as the query listing is a private property.

In the fourth search, we search for similar listings that are private housing, close to shopping malls and cost less than $500,000, as shown in Figure 25. There were no listings in the output as no private property costs less than $500,000.

In the last search, we search for similar listings that are public housing, built during or after 2015, close to primary and secondary schools, with at least 2 bedrooms and cost less than $1,000,000. The cosine similarity scores are the lowest compared to previous searches as newer houses tend to be more costly, thus fewer listings meet the condition of costing less than $1,000,000. Coupled with the fact that the housing type is different from the query listing, the cosine similarities are a lot lower, as shown in Figure 26.

## Evaluation

While our model is practical, it is not flexible. Further improvements to this model include allowing users to specify the distances of the houses from primary/secondary schools and shopping malls and allowing for range searches for price and built\_year. However, such a model would be confusing to users due to the immense possibilities of parameters, thus we would create a web page to provide a better user experience.

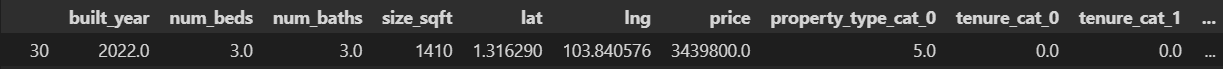


Figure 21: Query listing

Graphical user interface, text

Description automatically generated

Figure 22: Search for the most similar listings

A picture containing text, scoreboard

Description automatically generated

Figure 23: Search for similar listings that are public housing with maximum price $500,000

Graphical user interface, text

Description automatically generated

Figure 24: Search for similar listings that are public housing, close to primary schools and with at least 3 bedrooms

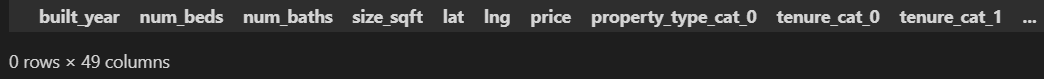


Figure 25: Search for similar listings that are private housing, close to shopping malls and cost less than $500,000

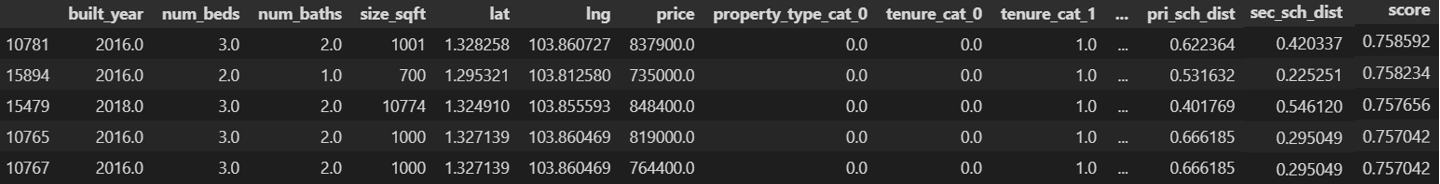


Figure 26: Search for similar listings that are public housing, built during or after 2015, close to primary and secondary schools, with at least 2 bedrooms and cost less than $1,000,000

# Task III: Out-of-Distribution Detection

* 1. *Justification*

In recent years, out-of-distribution (OOD) detection has become increasingly important following the rise in importance of machine learning models in businesses. In the real world, data distributions usually drift over time and it is often impractical to track a developing data distribution in any real-world activities. Our group chose OOD detection as our third task as we found it to be a potentially meaningful enhancement to our implementation in task II to explore and discuss. Task III theorizes how OOD detection can be implemented to improve the robustness and reliability of our model against OOD inputs by detecting and addressing such inputs with proper generalization techniques. At the same time, our group hopes to use this opportunity to improve our understanding on a critical consideration in developing any reliable neural network models in future works.

* 1. *Background*

When machine learning models are deployed in the real world, it is essential to always consider OOD inputs and how these cases should be handled, as it is impossible to cover all possibilities using training data, and we do not want to let our models confidently give incorrect predictions based on an input that has no intersections with the data that the model was trained with. Deep neural networks are usually trained with an assumption that the test data distribution is similar to that of the training data, thus when the test data is not similar, the resulting models would fail to generate the desired outputs. While high levels of accuracy and robustness against OOD inputs is not exactly necessary in the scope of task II, this is an important consideration for domains like medicine and autonomous driving, and thus deemed worth exploring.

* 1. *Consideration*

Typically, for a model to be effective even with test samples that are significantly different from the training distribution, a well-calibrated prediction uncertainty is essential in the application such that it can tell the user how “unexpected” or “surprising” the output is. One way of estimating the uncertainty of the model in for example, image classification, is the predictive entropy, calculated with the formula shown below in Fig. 7.

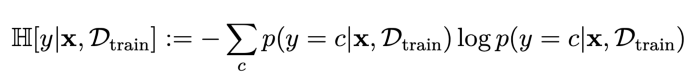


Fig. 7. Mathematical formulation of predictive entropy [2]

With predictive entropy, the model is able to reflect how “unsure” it is about its result and is applicable in a practical sense especially in the aforementioned intolerant domains such as medicine and autonomous driving where the model can reflect the level of accuracy and even throw out a line of “I’m not sure” if the uncertainty level is too high.

However, this is not applicable in our task II implementation, as firstly we are not working with image classification and secondly, we are always working with houses with a known range of features. Thus, to account for OOD input in our model, we will have to simply address scenarios of OOD inputs in our list of known features. Our theorized approach will be outlined in the next section.

* 1. *Implementation*

The effectiveness of the addition of OOD detection into our model can simply be evaluated by whether the model exhibits any unexpected behavior when given an OOD input, which means that the recommended listings based on the input listing should be relevant to the input listing in features that are not OOD, as the model should be able to tell which numbers are OOD and hence not be skewed by them. For example, the model should be capable of identifying that an input of 39.24 billion dollars for a unit is clearly wrong and should look at other features to generate recommendations.

In order to test whether our model generates unexpected recommendations based on an OOD input, we can utilize Deepchecks as a testing tool. Deepchecks is a company made up of a group of machine learning researchers who aims to build a tool that helps businesses and organizations test and detect problems in their machine learning models. According to Deepchecks, machine learning models are different from traditional software systems in the sense that machine learning systems can fail silently, and some issues might go undetected for years. The tool that we looked at is the Weak Segments Performance check to test the recommendations. With the Weak Segments Performance kit, we can separate the data into different segments and check for segments where the model’s performance, calculated by the f1 score, is below average. From our observations, our model was generally robust against moderate alterations in most features.

In the context of task II where typically recommendation systems face OOD recommendation problem is when the user features shift. For example, when the user has moved on to another product, but the recommendations are still trailing behind. For example, when the user has decided to look at condominiums instead of HDBs after getting an unexpected windfall, but the recommendation system is still suggesting HDBs based on the user’s interactions from the last visit. In a paper published on this topic [3], the authors suggested strong OOD generalization and fast OOD adaptation as the two objectives of models to have high fidelity. For our model in task II, as there is no “user interaction” present, the following is a theorized discussion in the context of a user-navigated housing website instead of our content-based filtering implementation based on the paper cited above.

## In cases of user feature shifts, i.e., users decide to change their focus, the learning representation of user preferences would become outdated and lead to a sharp performance drop in the recommendations provided to the users. OOD generalization and fast adaptation basically means that the recommendation system should be able to pick up new user features quickly and translate that into accurate recommendations. In the same example given earlier, when the user decides to look at condominiums instead of HDBs from their last visit to the website, the system should be able to pick up this signal of user feature shift and update all recommendations to be condominiums based on the user’s inputs of other features, instead of taking the entire user history to provide HDB recommendations after the user searches.

# References

1. T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proceedings of the 22nd acm sigkdd international, 2016. [https://arxiv.org/pdf/1603.02754.pdf](chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https:/arxiv.org/pdf/1603.02754.pdf)
2. V.U. Thompson, C. Panchev and M. Oakes, “Performance Evaluation of Similarity Measures on Similar and Dissimilar Text Retrieval,” 2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K), 2015, pp. 577-584.
3. Yarin G., “Uncertainty in Deep Learning”. Unversity of Cambridge. <http://mlg.eng.cam.ac.uk/yarin/thesis/thesis.pdf>
4. Wenjie W., Xinyu L., Fuli F., Xiangnan H., Min L., Tat-Seng C., “Causal Representation Learning for Out-of-Distribution Recommendation”. <http://staff.ustc.edu.cn/~hexn/papers/www22-ood-rec.pdf>

# Appendix

## Breakdown of workload

|  |  |
| --- | --- |
| **Name** | **Workload** |
| Ma Yuan | Task 1 + EDA |
| Jefferson Sie | Task 2 |
| Huang Lifu | Task 3 |
| Joel Huang | Task 3 |