

Practice Module for Graduate Certificate in Practical Language Processing

Project Title:

**Home Buddy –
Your Personal House-Search Buddy**

Group 10:

Jonathan Lim Ching Loong (A0261707E)

Pradeep Kumar Arumugam (A0261606J)

Zhao Lutong (A0249279L)

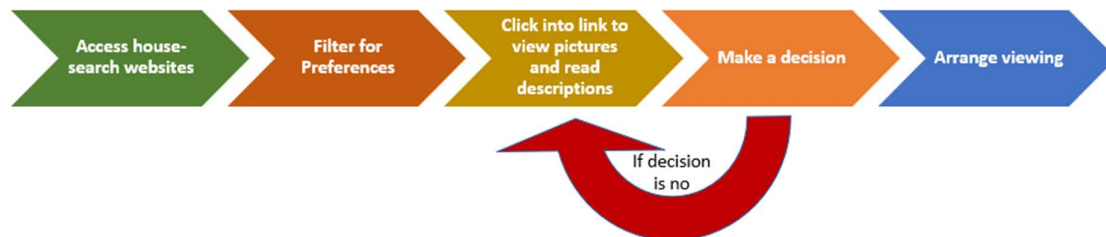
Table of Contents

Business Problem	2
Market Survey	2
Dataset	3
Solutions and Methodology	4
High level diagram	4
Chatbot (User Interface)	4
Technical explanation of the webpage and chatbot:	4
DialogFlow Chatbot Agent:	6
Robotic Process Automation (RPA)	9
Language Model.....	10
Ideation of the Language Model.....	10
Model Classes	10
Model Structure	11
Results and Performance	12
Information Flow.....	14
Results Output	15
Conclusion	17
References	17
ChatGPT Usage	18
Robotic Process Automation (RPA)	18
Language Model.....	18
UI.....	18

Business Problem

Singapore is well-known as a hub for economy and education. As such, there has been many foreigners and expatriates that come for various purposes. A common struggle faced by foreigners is the issue of finding an accommodation for their long-term stay in Singapore.

The diagram below shows a typical overview of the house-search process for any given house-search website:



While it's true that websites such as PropertyGuru has a wide range of available filters that a user can apply, there are many crucial information in the description that can't be filtered using the built-in hard filters. For example:

- i) Landlords looking for specific demographics of tenants (e.g ethnicity, gender, profession, age groups, marital status, etc)
- ii) Landlords with specific requests (e.g. no cooking, no smoking, comfortable with pets, etc)
- iii) Hidden costs (e.g. rentals not including utility bills, agent fees, etc)
- iv) Amenities provided (e.g. air-conditioning, cooking, etc)
- v) Presence of landlord in the house or number of household members in the house.

Currently, these information are usually stated only in the description and there are no quick ways to filter the descriptions except to click into the link and read. This causes the whole process is very time consuming and can take hours on end, especially if the postings that were clicked on constantly fail to meet the user's pre-requisites.

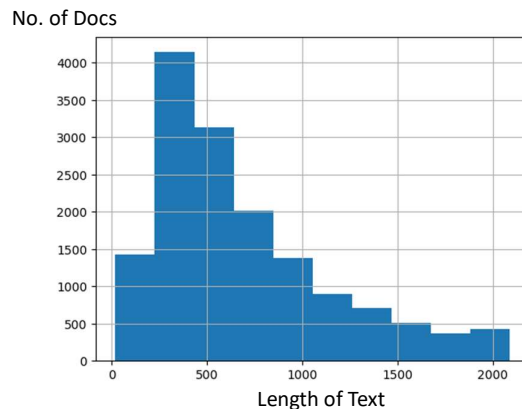
Market Survey

To understand the extent of the problem, our team conducted a quick survey to understand the needs of users who are searching for rental houses and identified that most users feel that the filter options in housing websites are insufficient in shortlisting houses.

As such, our project applied Natural Language Processing (NLP) methods to speed up this process and dynamically meet the user's needs to be more relevant than hard-filtering. This project is not limited by constraints set by hard-coded values as users can most likely write anything they want. Combined with RPA, our system performs real-time scraping and analysis of description to generate the closest shortlist given the user's constraints for the user to have a quick overview for decision-making.

Dataset

Web-scraping was performed to scrape 15000 description data from Property Guru along with the rental prices and URLs for each listing. The description data follows the distribution below:



To get a better understanding of the data, text analysis techniques such as topic modelling, ngrams, and word clouds were applied. The data was first pre-processed by the following steps:

- 1) Removal of non-English characters
- 2) Tokenization
- 3) Lowercasing
- 4) Removal of numbers
- 5) Lemmatization
- 6) Stopwords removal
- 7) Removal of single letter words and words with 2 to 2nd last letter that is numeric (to remove Agent Ref numbers, block number, and bus numbers)

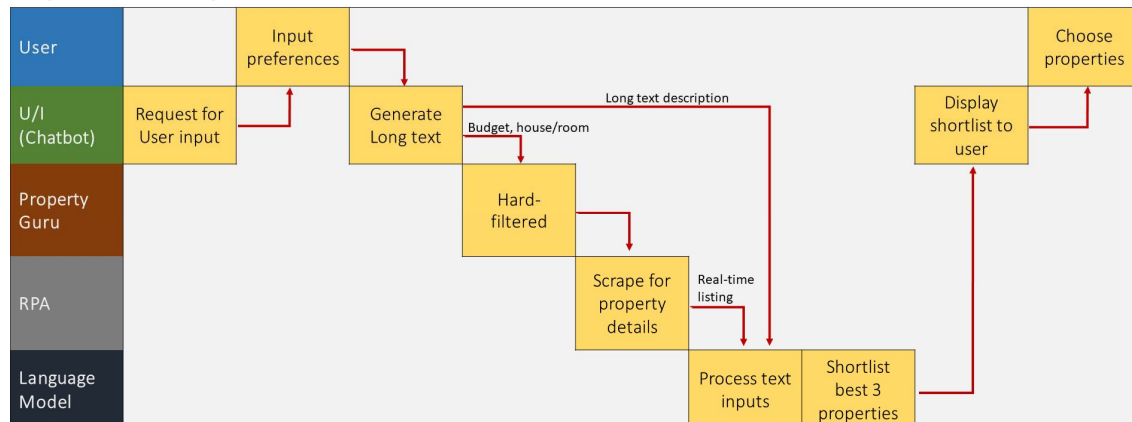
From there, we obtained the word cloud below and some topics from Topic modelling by LDA (number of topics = 7):



Topic	Topic Models
1	access away easy shopping also amenity resident residence supermarket option
2	pool living area apartment facility kitchen bathroom swimming spacious balcony
3	school primary mall park shopping centre amenity layout distance international
4	market director right shall private subject life need owner website
5	garden park bungalow development house landed land partial project estate
6	bed utility table common wardrobe air cleaning term brand studio
7	common wifi aircon tenant cooking single landlord master female utility

Solutions and Methodology

High level diagram



Our project consists of three main components to carry out its function, namely the Chatbot (User Interface), Robotic Process Automation (RPA), and Language Model. We decided to focus solely on Property Guru as it is one of the largest property website when it comes to rental housing currently and have more than enough data for us to build our model. The sections to follow will describe more on the flow of our system.

Chatbot (User Interface)

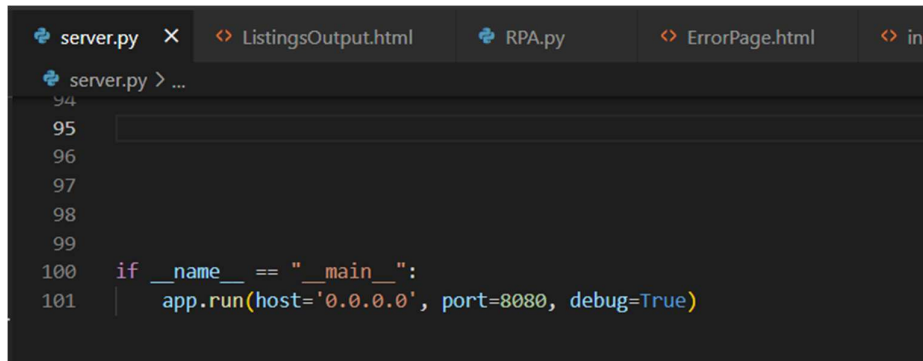
Technical explanation of the webpage and chatbot:

We have used python and flask for routing our web app. Which is the flask web framework (written in python) is used to initializing our web server in the local and navigation to other pages. It is a list of routes (web pages) and their functions in that specific route. Routes will be defined as `@app.route()` and functions below such route is the function which needs to be done with that page.

We have built the flask here. The path variables are defined and configured as below, which will be used later for specific tasks.

```
1  import os
2  from flask import Flask, render_template, request, send_from_directory, jsonify
3  import pandas as pd
4  # import tagui as t
5
6  app = Flask(__name__)
7
8
9
10
11 def dataframe(table):
12     tup = tuple(table.itertuples(index=False, name=None))
13     return tup
14
15
16 @app.route('/')
17 def index():
18     return render_template('index.html')
19
20 #####Temporary route before fitting #####
21 @app.route('/message', methods=['GET', 'POST'])
22 def message():
23     print("Inside webhook")
24     # if request.method == "GET":
25     #     return "Sample"
26     # if request.method == "POST":
27     #     payload = request.json
28     #     data = request.get_json()
29     #     response = (data['queryResult'])['fulfillmentText']
30     #     message = data.get('message')
31     #     print(data)
32     #     print(response)
33     # if response != "":
34     # if response.startswith("Okay, here's a summary of your preferences,"):
35     #     preferences = response
36     #     # if preferences.startswith("Okay, here's a summary of your preferences,"):
37     #     preferences = preferences[len("Okay, here's a summary of your preferences, "):]
38     #     # if preferences.endswith(" Please confirm below."):
39     #     #     preferences = preferences[:-len(" Please confirm below.")]
40     #     #     return jsonify(preferences)
41     #     return jsonify(preferences)
```

As seen in the below screenshot, we have defined a case statement, so whenever a user run this python file directly, python sets this variable with the value `__main__`, means we want this code to run only when this file is run directly, not when called from another python file. So, it will start the server in that specified port when run and the file will be waiting for the user to work with the web app, which will be active in our local server.



```
server.py > ...
94
95
96
97
98
99
100 if __name__ == "__main__":
101     app.run(host='0.0.0.0', port=8080, debug=True)
```

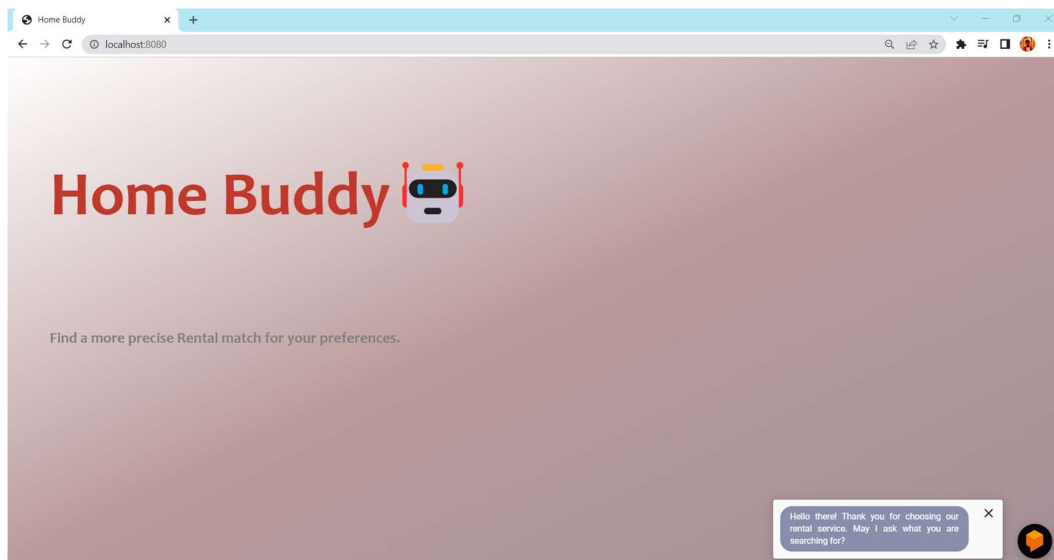
After running the file, the server will be initialized. You can just either type the localhost web url in your chrome and will be able to access the web page initialized. When you initialize the server in 8080, the web page url will be <http://localhost:8080/> . The Active page (which is the home page) will be the default "/" (route to root path) and we have rendered our home page of our webapp under that flow. This will be default in the <http://localhost:8080/>.

Hence, as soon "/" was found, flask will launch our main home page (our main page is named as index.html) will be called by our server.py script using **render_template()** function, which was defined in the route and the function as seen below



```
@app.route('/')
def index():
    return render_template('index.html')
```

The route is / and the function **render_template** is a Flask function which is used to generate output from a template file based on the Jinja2 engine that is found in the application's templates folder. Hence, we created the template folder in the same path as file, as it defaults the search location in the templates folder. And our template will be visible in the homepage like below



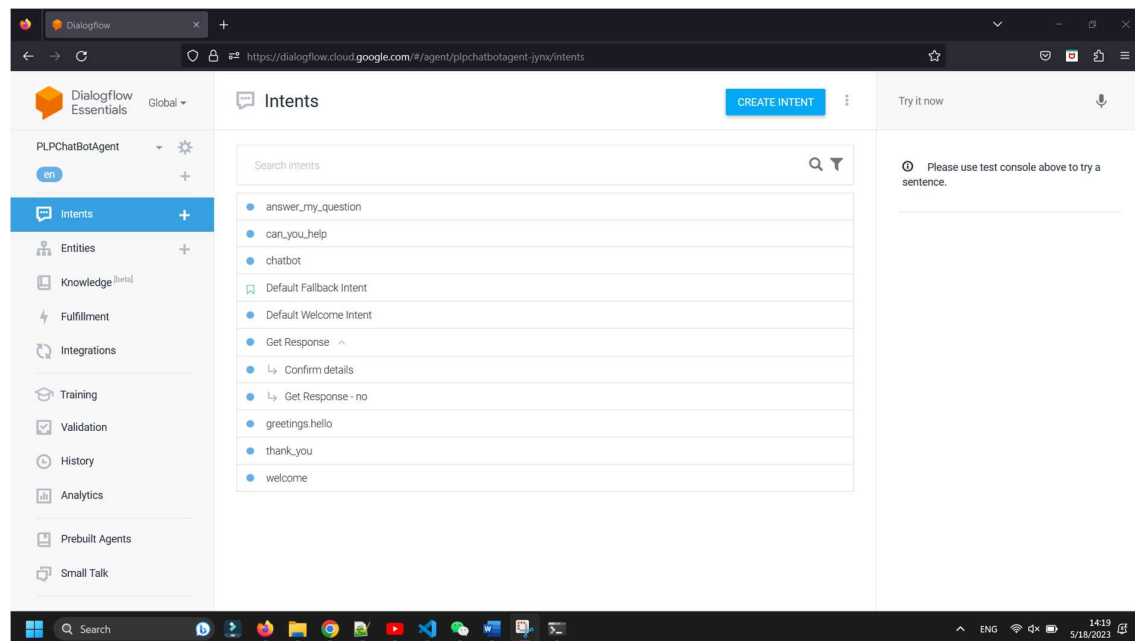
For our chat bot, we have used Google's Dialogflow to create an agent which can handle a natural conversation with the user. As this is also a task bot which will be used to perform some actions in the backend, the conversation will be more focused on getting some basic information from the users. Because of the fact this is domain specific conversation.

The DialogFlow chatbot we created is integrated into this webpage which can be seen on bottom right of the screenshot above is integrated as shown below.

```
<script src="https://www.gstatic.com/dialogflow-console/fast/messenger/bootstrap.js?v=1"></script>
<df-messenger
  intent="WELCOME"
  chat-title="Dr. House"
  agent-id="9b3d5d31-f5fd-4d1b-ad3b-f15c871498a0"
  session-id="chatsession"
  language-code="en">
</df-messenger>
```

DialogFlow Chatbot Agent:

Dialogflow also offers an integration component called the Dialogflow Messenger which brings a rich UI for Dialogflow that enables developers to easily add conversational agents to websites. Once we integrate it into our HTML website code, the chatbot gets added to the webapp with the specifications which can be finetuned to a certain degree.



We have created multiple domain specific entities and parameters (can be seen below) which the agent needs to capture from the contextual conversation with the user to get a long description with all the user preferences.

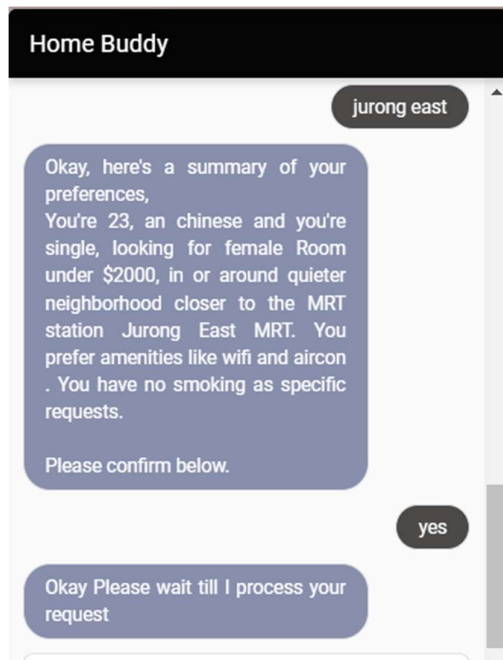
The top screenshot shows the Dialogflow 'Entities' page. The left sidebar contains navigation options: PLPChatBotAgent, Intents, Entities (selected), Knowledge, Fulfillment, Integrations, Training, Validation, History, Analytics, Prebuilt Agents, and Small Talk. The main area displays a list of entities under the 'Custom' tab, including @age, @amenities, @budget, @ethnicity, @gender, @hidden_costs, @marital_status, @mrt, @neighborhood, @propertytype, and @specific_requests. A 'CREATE ENTITY' button is visible in the top right.

The bottom screenshot shows the 'Get Response' page. The left sidebar is identical to the top screenshot. The main area displays a table of parameters and their values. The table has columns: REQUIRED, PARAMETER NAME, ENTITY, VALUE, IS LIST, and PROMPTS. The data is as follows:

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
<input checked="" type="checkbox"/>	budget	@sys.unit-currentcy	\$budget	<input checked="" type="checkbox"/>	Okay, what's yo...
<input checked="" type="checkbox"/>	marital_status	@marital_status	\$marital_status	<input type="checkbox"/>	Please tell me ...
<input checked="" type="checkbox"/>	ethnicity	@ethnicity	\$ethnicity	<input type="checkbox"/>	Where are you f...
<input checked="" type="checkbox"/>	gender	@gender	\$gender	<input checked="" type="checkbox"/>	for a gender sp...
<input checked="" type="checkbox"/>	type	@propertytype	\$type	<input checked="" type="checkbox"/>	What type of pr...
<input checked="" type="checkbox"/>	neighborhood	@neighborhood	\$neighborhood	<input checked="" type="checkbox"/>	Understood. No w...
<input checked="" type="checkbox"/>	amenities	@amenities	\$amenities	<input checked="" type="checkbox"/>	Any amenities r...
<input checked="" type="checkbox"/>	specific_requests	@specific_reque	\$specific_request	<input checked="" type="checkbox"/>	Please mention ...
<input checked="" type="checkbox"/>	age	@age	\$age	<input type="checkbox"/>	What's your age...
<input type="checkbox"/>	moveindate	@sys.date-period	\$moveindate	<input checked="" type="checkbox"/>	--
<input checked="" type="checkbox"/>	mrt	@mrt	\$mrt	<input type="checkbox"/>	Please tell me ...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	--

A '+ New parameter' link is located at the bottom of the table.

During the conversation, our "Get Response" intent will try and get all the basic and required information from the user and collate them to give a summarized sentence a long description for the user to confirm their overall expectation on what he/she is looking for in their rental needs.



We have used JavaScript, within the HTML to capture and handle the response event from the dialogflow agent. We have added an event listener `addEventListener()` in the messenger element which is triggered to capture the event type response from the chatbot agent. And these responses are loaded and posted back into our flask server response handler which is a function, which waits for a specific message from the agent, which is the summary message and alter its structure slightly to capture a string containing the user's general description of needs or expectation from the property or room which he/she is looking for. This string will be the input string to our model for the backend processing to get the equivalent listing. Whenever there is a response from the agent, this listener will past that message into our local server for further processing.

```
<script>
const dfMessenger = document.querySelector('df-messenger');
// const xhr = new XMLHttpRequest();
dfMessenger.addEventListener('df-response-received', function (event) {
  console.log(event);
  if (event.detail.response) {
    data = event.detail.response;
    console.log(data);
    //now do something with the data JSON object
    const xhr = new XMLHttpRequest();
    xhr.open('POST', '/message');
    //xhr.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
    //xhr.send(JSON.stringify({message: messages}));
    //xhr.send(JSON.stringify({message: response}));
    xhr.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
    xhr.send(JSON.stringify(data));
  }
});
```

A response handler within our flask framework will be waiting for a specific message response from the agent which has the summary of the user's preferences. This will be processed with the below codes and ready for further processing.

```
@app.route('/message', methods=['GET', 'POST'])
def message():
    print("Inside webhook")
    # if request.method == "GET":
    #     return "Sample"
    # if request.method == "POST":
    #     payload = request.json
    data = request.get_json()
    response = (data['queryResult']['fulfillmentText'])
    # message = data.get('message')
    # print(data)
    print(response)
    # if response != "":
    if response.startswith("Okay, here's a summary of your preferences"):
        preferences = response
        # if preferences.startswith("Okay, here's a summary of your preferences, "):
        preferences = preferences[len("Okay, here's a summary of your preferences, "):]
        # if preferences.endswith(" Please confirm below."):
        preferences = preferences[:-len(" Please confirm below.")]
        print ("Long Description")
        print(preferences)
```

```
Please confirm below.
Long Description
You're 24, an chinese and you're single, looking for male Room under $3500, in or around quiet neighborhood neighborhood
closer to the MRT station Jurong East MRT. You prefer amenities like wifi and aircon . You have no smoking and pets as
specific requests.
```

This string is then passed into our **RPA** and Fine-Tuned BERT **Language model** for precise matching of the descriptions on the listed property available in the website.

Robotic Process Automation (RPA)

The RPA component starts by accepting input from the Chatbot and performs a rule-based information extraction procedure to identify filter criteria that can be applied to the website.

As Property Guru incorporates the filters into the URL, the RPA function builds a URL based on the extracted filter criteria and jumps directly into a webpage with the filtered search results and begins scraping using the RPA library TagUI.

To maintain a reasonable wait time for the user, we applied a scraping rule to speed up the process. The rule is summarized in the table below:

Condition	Rule	Logic
# Price here refers to Listed Rental Price # Budget here refers to User's Budget # Results refers to Filtered Search Results		
If Number of Results < 20	Scrape all	To have a reasonable pool of options for Language Model to shortlist
Else if Price < 0.55 * Budget	Do not scrape	Users are unlikely looking for low-cost rentals if they put a high budget
Else if 0.55 * budget < Price < 0.70 * budget	Scrape 35% of the properties	This is hypothesized as the price range that a typical user would look for given their budget.
Else if 0.7 * budget < Price < 0.80 * budget	Scrape 95% of the properties	
Else if 0.8* budget < Price < budget	Scrape 55% of the properties	
Additional Rule: If Listed address was scraped before	Do not scrape	Usually a repeated

As our experiments shown that the results are more accurate with the triplication of the text, the input passed from the chatbot is triplicated to form a larger bulk of text to be passed on to the Language model together with the scraped dataframe. Our experiments also show that straightforward inputs, i.e. without the "you" and "you're", works better. As such, irrelevant words from the input template were also removed before passing into the language model.

Language Model

Ideation of the Language Model

As a potential user of this App, we think from our own pain points when searching for a place to stay. Platforms like PropertyGuru already provide very powerful filter functions by map/district/MRT and budget. However, there are some **needs that are specific but non-trivial** (e.g. gender/ethnicity specific, no landlord staying, light cooking), we could only click the listing URLs and read it to match these expectations.

We therefore come up with an idea to help with house searching process by making use of language models' strong contextual understanding ability to find the best match to user's house searching requirement in natural language. It would be a good tool when there are many listings available, especially for foreign student or professionals who are not very familiar with Singapore's housing district/MRT distribution.

For all these requirements, it is also manifestable from platform's side to add filters for each one of them. However, it will make much harder for agents/owners to post 1 listing with 35 filters to fill in. And the filter is not flexible to change if there is a shift in user needs.

Inspired by Word2Vec model, we treat this problem as a document classification problem. Where we do not use the end product of the classification result, but the embeddings which learned to be discriminative on the areas we want to focus. The model will pay more attention to the values related to these aspects in the classification tasks, while still preserving the ability to capture semantics information without explicit telling.

Model Classes

We firstly identified what are the common requirements in Singapore house searching and also discovered some new topics from topic modelling process. We finally consolidate these 35 different binary classes for model to predict. (The number in the comments are their weights for losses, which is introduced in section3)

```
[ 'female', 'male',      # 2.0 -> 2.5
  'malay', 'chinese', 'indian', 'family', '1pax', # 1.3
  'cooking_no', 'cooking_ok_0.5', 'cooking_ok_1', # 1.5 -> 2.0
  'no_smoking', 'smoking_ok', 'pets_no', 'pets_ok', 'visitor_no', 'visitor_ok', #1.0
  'utilities_excluded', 'utilities_included', 'agent_fee_excluded', 'agent_fee_included', #1.2
  'wifi_no', 'wifi', 'no_aircon', 'aircon_yes', # 1.6
  'transport_1', 'transport_0.5', 'grocery_1', 'school_1', 'gym_1', # 1.0
  'no_landlord', 'landlord', # 1.8 -> 2.0
  'low_floor', 'mid_floor_0.5', 'high_floor', 'new_house' ] # 0.5
```

Model Structure

Backbone:

The backbone used in this model is BertForSequenceClassification. We adopted the code from TPML BERT Finetuning workshop. It is finetuned from pretrained "bert-base-uncased"

Custom Weighted BCELoss

During experiments, we notice two issues:

- For certain classes like with/without landlord, no wifi, the model is not performing very well on differentiating them, as long as the relevant word appears, the model will return as relevant.
- For certain unbalanced class like "male"(compared to "female"), "pets", they are less occurred in the labelled corpus. Model is not doing well at capturing these requirement.

Therefore, we modify the BertForSequenceClassification and overwrite the original default BCELoss with different weights for each class. We assign higher weights for class that we want them to learn well and penalize more for unbalanced class. From the class image in section2, the commented number after each class is their custom weight.

```
class BertForSequenceClassificationWithWeightedLoss(BertForSequenceClassification):
    def __init__(self, config):
        super().__init__(config)
        self.config = config
        # self.label_weight = [2.0]*2 + [1.3]*5 + [1.5]*3 + [1.0]*6 + [1.2]*4 + [1.6]*4
        self.label_weight = [2.5]*2 + [1.3]*5 + [2.0]*3 + [1.0]*6 + [1.2]*4 + [2]*4 + [1
        self.num_labels = len(self.label_weight)
        self.loss_fn = nn.BCEWithLogitsLoss(pos_weight=torch.tensor(self.label_weight))

    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        labels=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):
        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )
        pooled_output = outputs[1]

        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        loss = None
        if labels is not None:
            loss = self.loss_fn(logits, labels)

        if not return_dict:
            output = (logits,) + outputs[2:]
            return ((loss,) + output) if loss is not None else output

        # return logits
        return SequenceClassifierOutput(
```

We observe **adding different and even higher weight really helped the model** to better find listings that are relevant to "male" or other unbalanced class. Here is an example after adding custom weights for different classes:

User Input:

Male preferred. A single Room under \$1000, in or around Family friendly neighborhood closer to the MRT station. Come with amenities like aircon, wifi and laundry . Allowed light cooking and no smoking.

Top1 Matched Listing's Description:

Listing 1 #####
*Ready-to-move room in a highly convenient location!
Common bedroom in a highly exclusive double storey apartment to move in ASAP!*

Looking for 1 male tenant. Preferred professional who can maintain cleanliness and is quiet.

- + Single size bed, wardrobe and study desk are provided.
- + Standard utilities, aircon and Wifi are provided.
- + Light use of washing machine.
- + Light cooking and microwaveable food are allowed.
- + Co-stay with owners. No other tenants.
- + House is extremely well-kept and beautiful.
- + Short walk to Khatib MRT. 8 mins walk to Khoo Teck Puat hospital.
- + 24 hours coffeeshop is nearby.

No pets. No smoking. No overnight visitors. Laundromat is at next block.

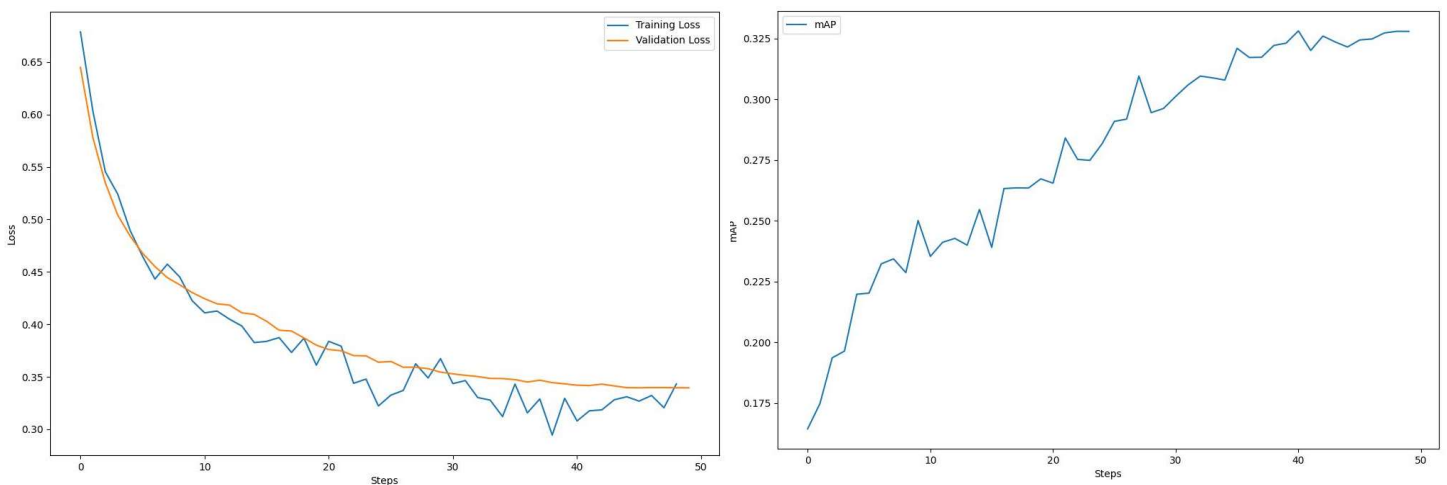
Custom compute_metrics mAP

We also added custom compute_metrics function such that it is evaluating the best validation performance on mAP (mean Average Precision) besides BCE loss. In this way it will better evaluating its ability to differentiate these requirements.

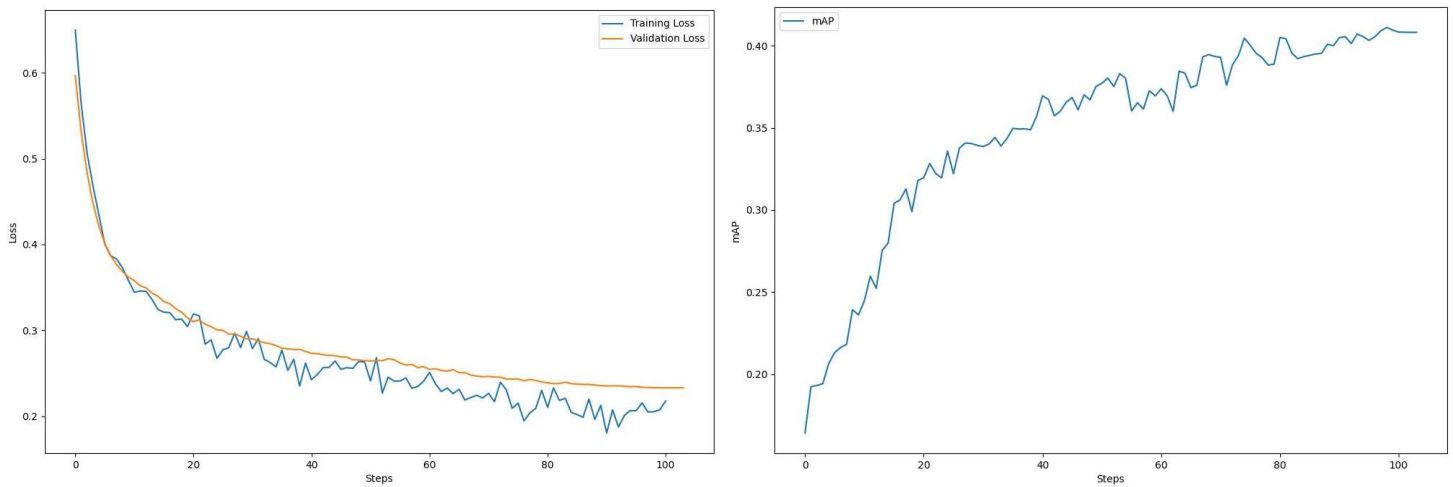
```
from sklearn.metrics import average_precision_score
from sklearn.preprocessing import LabelBinarizer
from transformers import EvalPrediction
import numpy as np

def compute_metrics(p: EvalPrediction):
    lb = LabelBinarizer()
    lb.fit(p.label_ids)
    binarized_labels = lb.transform(p.label_ids)
    preds = p.predictions
    # convert outputs logits to probabilities here
    if preds.shape[1] > 1:
        preds = np.exp(preds) / np.sum(np.exp(preds), axis=-1, keepdims=True)
    return {"average_precision": average_precision_score(binarized_labels, preds)}
```

Results and Performance



Model_v6 Val loss and mAP



Model_v3 Val loss and mAP

From the experiment results, we find out it is not always the higher the val mAP (or the lower the val loss), the better the model is. Within certain range, the model is performing better as the classification task metrics increase. However, between Model_v3 and Model_v6, we found out although Model_v3 trained with more steps and has higher mAP, the listings it is recommending are less sensible and shorter. The model is finetuning too much from BERT's original weight, such that it starts to ignoring other contextual info but only the relevant info.

Example using Model_v3 (over-trained although metrics look promising)

User Input:

You're 23, an chinese and you're single, looking for female Room under \$1000, in or around Family friendly neighborhood closer to the MRT station Hougang MRT. You prefer amenities like aircon, wifi and laundry . You have light cooking and no smoking as specific requests.

Top2 Matched Listing's Description from Model_v3:

Listing 1 #####
 Female environment near Commonwealth MRT
 female environment
 ##### Listing 2 #####
 Only Female
 Only Female

Top1 Matched Listing's Description from Model_v6: (same input, too long to show all 3)

Listing 1 #####
 Common Rm(1 Or 2 Pax) All Female Unit, Can Be Queen Bed, Avail 1 April
 Available 1st April 2023 for 1 or 2 Persons (Can be Queen Bed instead of the bunk bed shown in the pics)

 Agent Fees apply
 Name of Condo: Atrium Residences

Room type: Common Room, for 1-2 tenants, all female unit
 7-minute walk to Aljunied MRT station on the green line
 15 minutes on the MRT into the CBD, near to James Cook University, SMU, Kaplan and PSB Academy.

- No owner
- Cooking and visitors allowed
- Pet-free environment
- Facilities include swimming pool, gym & BBQ pits
- Fully furnished with single bunk bed, desk, chair and wardrobe
- All units will not exceed the number of tenants stipulated by authorities
- Can register address with the ministries.
- Fuss-free tenancy - deposits will be refunded immediately when the lease ends.

Co-living with other tenants

\$1580 - \$1680 monthly, for 1-2 tenants only, inclusive of utilities, wifi & weekly cleaning of common areas. There is a maximum cap on utilities of \$70 for 1 person; \$105 for 2 persons per month, any excess is to be shared by all tenants and paid. There is also an aircon servicing, repairs and maintenance fee of \$20 per room per month.

We can see both Model_v3 and Model_v6 results satisfy the requirement, however Model_v3 only matches 1 or 2 requirements (female, near MRT), while Models_v6 results matched all 5 requirements (female, cooking_allowed, near MRT, wifi, aircon) with other rich information about the house. In this case, Model_v6's recommendation results are preferred.

Overall, after experimenting with different number of epoch, custom weighted loss, different evaluation metrics for validation set, max_len for tokens, we found the best performing model on this task is with this setting:

Parameters	Epoch	max_len	BCELoss weights	Val Evaluation Metric
value	4	256	Unequal weights, more weights for more important class	mAP

Information Flow

For the information flow, after receiving the user input from the chatbot *HomeBuddy*, we have two methods:

- 1) The use input will be directly passed to language model, which will generate an embedding to compare with all pre-scraped and pre-computed 15,000 listings. Language model will return back top 3 recommended listing URLs.
- 2) The user input will be passed to RPA process to do real-time scrapping based on location and budget, then the shortlisted scrapped result will be passed to language model. Language model will return back top 3 recommended listing URLs.

The above results are generated using method1, here is the result using method2:

```
##### Listing 1 #####
Shared room
3 room flat. 1 common room for rent.
3 female. 1 room 1 bed space available for renta female $400.
```

Near bus stop.
 Near coffee shop.
 Near hougangmrt.
 Move in immediately.
 ##### Listing 2 #####
 Room rental for female tenants.
 Serangoon north ave 4
 Common room for rent immediately.

 Prefer ladies (1 or 2)
 750 excluding utilities (2 pax 850)
 Living with lady owner, mother and helper.

We can see that previously using method1 the language model is not hard-filtered on MRT and budget, with method2, the search results are under budget and near to Hougang. We also notice the link expires very fast on PropertyGuru. To make sure the house listing we recommended is still available, we decided to go with method2.

Results Output

And once we get the top 3 matching or shortlisted property's URL from the model, we'll be using **renderCustomCard()** function to display those results to the user which is a rich response send to the user as though it is sent from the agent. List is rendered into a json component and captured for the button to be linked with the actual links of properties. The user can just click and see the 3 matches.

```

xhr.onreadystatechange = function() {
  // dfMessenger.addEventListener('df-messenger-loaded', function (event) {
  if (xhr.readyState == 4 && xhr.status == 200 && data.queryResult.fulfillmentText == "Okay Please wait till I process your request") {
    // const payload = JSON.parse(xhr.responseText);
    // const payload = xhr.responseText;
    const respon = JSON.parse(xhr.response);
    console.log(respon);
    console.log(respon.Sug1);
    console.log(respon.Sug2);
    console.log(respon.Sug3);
    //dfMessenger.renderCustomText('Custom text');
    const payload = [
      {
        "type": "button",
        "icon": {
          "type": "home",
          "color": "#FF9800"
        },
        "text": "Suggestion 1",
        "link": respon.Sug1,
      },
      {
        "type": "button",
        "icon": {
          "type": "home",
          "color": "#FF9800"
        },
        "text": "Suggestion 2",
        "link": respon.Sug2,
      },
      {
        "type": "button",
        "icon": {
          "type": "home",
          "color": "#FF9800"
        },
        "text": "Suggestion 3",
        "link": respon.Sug3,
      }
    ];
    dfMessenger.renderCustomList(payload);
  }
}
  
```



```

    };
    console.log(payload);
    dfMessenger.renderCustomCard(payload);
  }
}

```

Home Buddy

single, looking for female Room under \$2000, in or around quieter neighborhood closer to the MRT station Jurong East MRT. You prefer amenities like wifi and aircon . You have no smoking as specific requests.

Please confirm below.

yes

Okay Please wait till I process your request

Suggestion 1

Suggestion 2

Suggestion 3

Ask something...

Common Room For Rent! Mins Walk Away From Jurong East MRT

Clean, Well Maintained & Spacious Common Bedroom!

- ▶ 1 PAX **FEMALE ONLY**
- ▶ 1 YEAR LEASE MIN
- ▶ AVAILABLE 3rd of MAY

1pax = \$1400

- Fully Furnished
- Room Includes **Aircon, Wifi** & Utilities
- Spacious, Clean & Well Maintained
- Queen size bed
- Light cooking allowed
- Newly renovated common toilet

Nearby Amenities:

- Shopping Malls, JEM, Westgate and J Cube
- Walking distance to Jurong East MRT/Interchange
- Mins walk to Coffeshop/Supermarkets/Hair Salon

No Visitors, **No Smoking**, No Pets In The House

Description

Brand New Room For Immediate Rent!

111 Jurong East Common Room for rental.

Preferably **Female** Tenant

Asking \$1500/mth

Immediate Move In!

Utilities, **Wifi, Air-conditioning** all inclusive.

5 mins walk to Jurong East Mrt. Call Razis 📞9077....👁 before you miss it. Thanks

AIR-CONDITIONING

Conclusion

House-search through a property website is a tedious process that involves many clicking and reading and tends to take a long time before coming to a decision. Our project assists the user in the process by incorporating Robotic Process Automation and Natural Language Processing, providing them a dynamic, real-time method to search for houses, on top of the website's hard-filtering, to shortlist the properties that they are looking for, all below 5 minutes!

Through our testing and experiments, the project is quite useful in shortlisting properties for users, being able to return results that meets criteria such as gender and amenities. Unfortunately the project still falls short in meeting certain criteria like the presence of landlord in a house or choosing houses without wifi and have much to improve in those aspects. However, given that the real-time method chosen (Method 2) only returns roughly 15-20 listings on average, it may be that none of the listings scraped actually satisfies all given criteria and hence the model returns the closest-fitting listings.

References

- 1) <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html>

ChatGPT Usage

Robotic Process Automation (RPA)

The RPA process initially uses TagUI as the scraping tool. We tried switching the codes over to use Selenium after hearing about Selenium's popularity and ChatGPT was used to convert the TagUI-tailored codes to become Selenium-tailored codes. Unfortunately, the idea with Selenium was not feasible due to certain restrictions on PropertyGuru specifically to prevent Selenium bots to run on them so we switched back to TagUI.

ChatGPT was largely accurate in that conversion, just a couple of mistakes initially that makes the code erroneous. However, coupled with some googling and self-troubleshooting, the errors can be rectified. It is still far from being a tool that can replace human expertise but it is a very useful tool to assist, kickstart ideas, troubleshoot, and improve efficiency.

Language Model

ChatGPT was really helpful when interacting with HuggingFace's Trainer, which is a powerful library but with many parameters. When I try to add my own custom function to be used in evaluating validation set, with clear purpose and what function I want to use, ChatGPT is able to provide help that is not covered by official HuggingFace documentation.

However, ChatGPT is not very good at debugging, since it does not have the clear picture of all the functions doing, and sometimes the example function it returned are not really functional. We will have to further adapt or debug before using.

UI

As this was the very first attempt for the UI developer of this project to use a JavaScript component such as `querySelector()`, `addEventListener()` to handle a web-agent response real-time, ChatGPT was really helpful to identify them and provide us with basic examples of how to use them in an active web-session which was more reliable and focused than the examples we find from googling as they were more generic examples across different domain and use cases. With those examples, we were able to build our functions for this specific requirement.