Perfumer

Student name: MINGZHAO DU
Student number: 24004238
05/12/2024
Link to git project repo: Mingzhao-Du/Perfumer-Comprehensive-Scent-Expert: A rule-based chatbot

Introduction

In recent years, the perfume industry has embraced personalized experiences, as perfume choices are deeply influenced by emotions, environment, and individual tastes. For newcomers, however, the complexity of scent descriptions and terminology can be overwhelming. As a perfume enthusiast, I created ‘Perfumer’, a chatbot that combines my passion with technical practice to help users explore the world of perfumes interactively.

The project is designed for beginners and enthusiasts, offers customised perfume recommendations and introduces users to perfume culture, including brands, scent categories, and history. To enhance interactivity, the chatbot includes an intent recognition module, enabling it to handle a variety of queries beyond perfume-related topics. Its goal is to offer an accessible and enjoyable platform for users to express their preferences and discover the allure of perfumes.

This paper examines the design and development of ‘Perfumer’ as a non-commercial project aimed at promoting interest in perfume culture. It discusses the chatbot’s core functions, design philosophy, and technical challenges, highlighting its potential to engage users through meaningful and informative interactions.

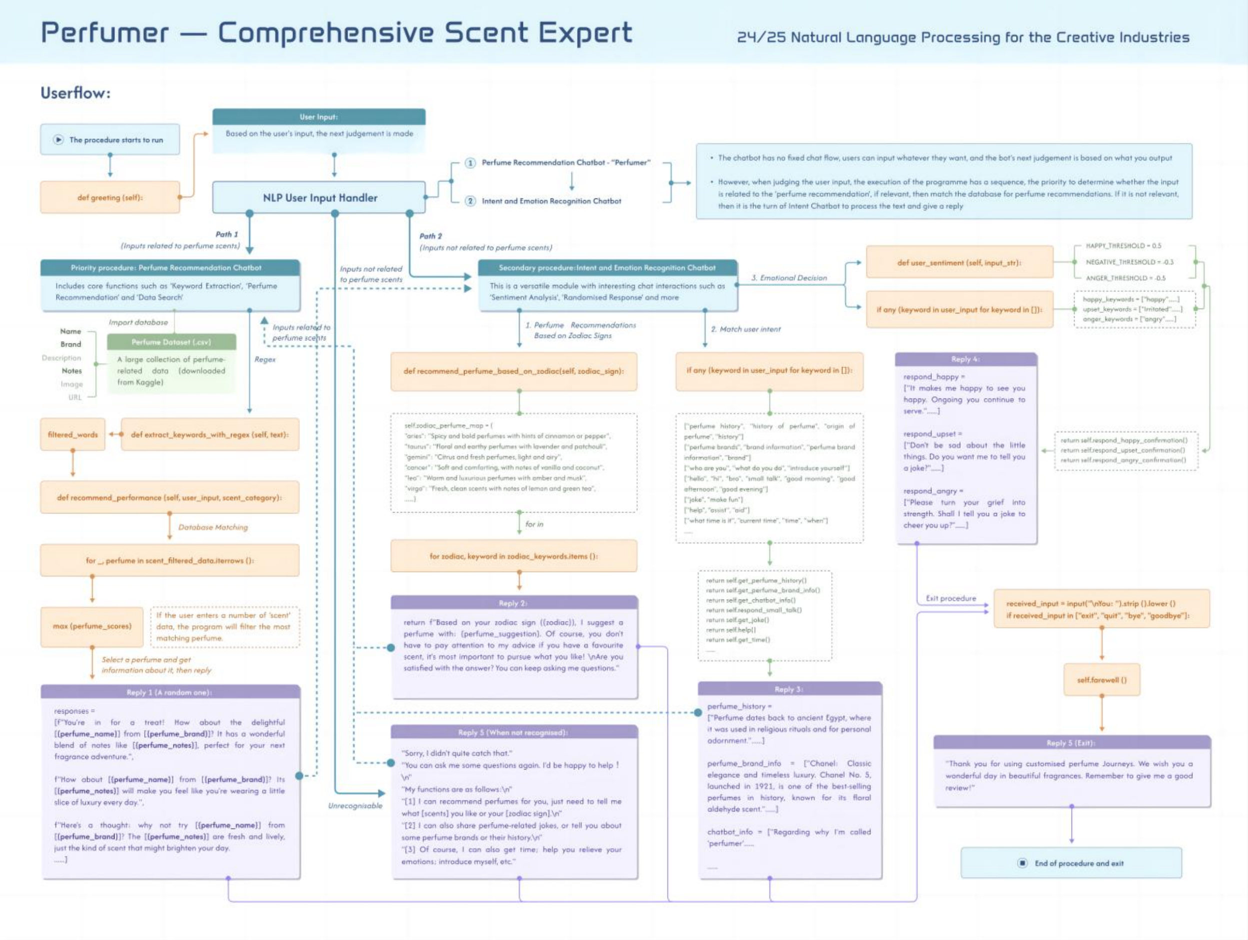
Background

The development of my project was guided by a review of relevant case studies and literature, which provided valuable insights into its design and implementation.
Initially, I sought to understand the current role of AI technology in the fashion industry. A key turning point was an article titled *Chatbots in the Fashion Industry: Benefits and Examples* [1], highlighting the significant advantages of chatbots in online retail, such as enhanced customer engagement, cost-effectiveness, improved conversion rates, and expanded market reach. This revealed the deep integration of AI technology into the fashion sector. Further research showed that similar advancements have been made in the perfume industry. For instance, *Philyra* [2], an AI system developed by *IBM* in collaboration with *Symrise*, created the first fully AI-generated perfume in 2019. *Philyra* now

collaborates with perfumers, leveraging vast datasets and advanced algorithms to craft sustainable and innovative fragrances. These examples underscore the transformative impact of AI on traditional industries and reaffirm the potential of my project.

To better understand perfume recommendation systems, I explored *Scentbird*, a subscription service that includes the *Scentbird Quiz* [3] for identifying user preferences and recommending perfumes. I found the questionnaire simple and efficient, capable of delivering personalized recommendations quickly. However, the system's unidirectional design focuses on gathering information without fostering substantial interactivity or emotional engagement, making it better suited for consumer needs than educating users about perfume culture. Drawing inspiration from its questioning approach, I refined my project goal: rather than selling perfumes, my project aims to spark users' interest, deepen their knowledge, and offer a fulfilling experience in exploring the world of perfume.

Implementation details


Perfumer — Comprehensive Scent Expert
24/25 Natural Language Processing for the Creative Industries
Userflow:
The procedure starts to run
def greeting (self):
User Input:
NLP User Input Handler
① Perfume Recommendation Chatbot - "Perfumer"
② Intent and Emotion Recognition Chatbot
Path 1
(Inputs related to perfume scents)
Path 2
(Inputs not related to perfume scents)
Priority procedure: Perfume Recommendation Chatbot
Secondary procedure:Intent and Emotion Recognition Chatbot
Inputs not related to perfume scents
Inputs related to perfume scents
Import database
Name
Brand
Description
Notes
Image
URL
Perfume Dataset (.csv)
Regex
filtered_words
def extract_keywords_with_regex (self, text):
def recommend_performance (self, user_input, scent_category):
Database Matching
for _, perfume in scent_filtered_data.iterrows ():
max (perfume_scores)
Select a perfume and get information about it, then reply
Reply 1 (A random one):
1. Perfume Recommendations Based on Zodiac Signs
def recommend_perfume_based_on_zodiac(self, zodiac_sign):
for in
for zodiac, keyword in zodiac_keywords.items ():
Reply 2:
Unrecognisable
Reply 5 (When not recognised):
2. Match user intent
if any (keyword in user_input for keyword in []):
Reply 3:
3. Emotional Decision
def user_sentiment (self, input_str):
if any (keyword in user_input for keyword in []):
Reply 4:
Exit procedure
self.farewell ()
Reply 5 (Exit):
End of procedure and exit

Based on the user's input, the next judgement is made

- The chatbot has no fixed chat flow, users can input whatever they want, and the bot's next judgement is based on what you output
- However, when judging the user input, the execution of the programme has a sequence, the priority to determine whether the input is related to the 'perfume recommendation', if relevant, then match the database for perfume recommendations. If it is not relevant, then it is the turn of Intent Chatbot to process the text and give a reply

Includes core functions such as 'Keyword Extraction', 'Perfume Recommendation' and 'Data Search'

A large collection of perfume-related data (downloaded from Kaggle)

If the user enters a number of 'scent' data, the program will filter the most matching perfume.

```
responses =
[f"You're in for a treat! How about the delightful [{perfume_name}] from [{perfume_brand}]? It has a wonderful blend of notes like [{perfume_notes}], perfect for your next fragrance adventure.",

f"How about [{perfume_name}] from [{perfume_brand}]? Its [{perfume_notes}] will make you feel like you're wearing a little slice of luxury every day.",

f"Here's a thought: why not try [{perfume_name}] from [{perfume_brand}]? The [{perfume_notes}] are fresh and lively, just the kind of scent that might brighten your day.
......]
```

This is a versatile module with interesting chat interactions such as 'Sentiment Analysis', 'Randomised Response' and more

```
self.zodiac_perfume_map = {
"aries": "Spicy and bold perfumes with hints of cinnamon or pepper",
"taurus": "Floral and earthy perfumes with lavender and patchouli",
"gemini": "Citrus and fresh perfumes, light and airy",
"cancer": "Soft and comforting, with notes of vanilla and coconut",
"leo": "Warm and luxurious perfumes with amber and musk",
"virgo": "Fresh, clean scents with notes of lemon and green tea",
......}
```

```
return f"Based on your zodiac sign ({zodiac}), I suggest a perfume with: {perfume_suggestion}. Of course, you don't have to pay attention to my advice if you have a favourite scent, it's most important to pursue what you like! \nAre you satisfied with the answer? You can keep asking me questions."
```

```
"Sorry, I didn't quite catch that."
"You can ask me some questions again. I'd be happy to help ! \n"
"My functions are as follows:\n"
"[1] I can recommend perfumes for you, just need to tell me what [scents] you like or your [zodiac sign].\n"
"[2] I can also share perfume-related jokes, or tell you about some perfume brands or their history.\n"
"[3] Of course, I can also get time; help you relieve your emotions; introduce myself, etc."
```

```
["perfume history", "history of perfume", "origin of perfume", "history"]
["perfume brands", "brand information", "perfume brand information", "brand"]
["who are you", "what do you do", "introduce yourself"]
["hello", "hi", "bro", "small talk", "good morning", "good afternoon", "good evening"]
["joke", "make fun"]
["help", "assist", "aid"]
["what time is it", "current time", "time", "when"]
......
```

```
return self.get_perfume_history()
return self.get_perfume_brand_info()
return self.get_chatbot_info()
return self.respond_small_talk()
return self.get_joke()
return self.help()
return self.get_time()
......
```

```
perfume_history =
["Perfume dates back to ancient Egypt, where it was used in religious rituals and for personal adornment."......]

perfume_brand_info = ["Chanel: Classic elegance and timeless luxury. Chanel No. 5, launched in 1921, is one of the best-selling perfumes in history, known for its floral aldehyde scent."......]

chatbot_info = ["Regarding why I'm called 'perfumer'......

......
```

```
HAPPY_THRESHOLD = 0.5
NEGATIVE_THRESHOLD = -0.3
ANGER_THRESHOLD = -0.5
```

```
happy_keywords = ["happy"......]
upset_keywords = ["irritated"......]
anger_keywords = ["angry"......]
```

```
return self.respond_happy_confirmation()
return self.respond_upset_confirmation()
return self.respond_angry_confirmation()
```

```
respond_happy =
["It makes me happy to see you happy. Ongoing you continue to serve."......]

respond_upset =
["Don't be sad about the little things. Do you want me to tell you a joke?"......]

respond_angry =
["Please turn your grief into strength. Shall I tell you a joke to cheer you up?"......]
```

```
received_input = input("\nYou: ").strip ().lower ()
if received_input in ["exit", "quit", "bye", "goodbye"]:
```

"Thank you for using customised perfume Journeys. We wish you a wonderful day in beautiful fragrances. Remember to give me a good review!"

(Fig i) Flow-chart work by Mingzhao Du (2024)

My chatbot consists of two chat modules with different functions, namely the perfume recommendation module and the intent-bot module. As shown in the above Flow-chart (fig i), it illustrates the user's service flow and the working logic of these two chat modules.

Service Flow:

1. Opening Greeting

The procedure starts with a brief functional introduction that leads the user to enter information that can be recognized.

2. Perfume Recommendation Logic

When the user inputs the content, the procedure extracts the keywords through regular expressions while filtering out common deactivated words.

If the keywords are related to perfume scents (included in the *scent_keywords* dictionary), the procedure will match these scents with the pre-downloaded text dataset and output the perfume information that meets the requirements, including the specific perfume name, brand and notes description.

If the user inputs multiple scent requests, the procedure will filter out the perfume with the highest scent overlap in the database for recommendation.

3. Intent Recognition Logic

However, If the user inputs a keyword unrelated to perfume, the Intent-Bot Module identifies it and selects a suitable response randomly. This versatile module can recognize emotions, share perfume jokes, provide perfume history and brand insights, and even suggest scents based on the user's horoscope.

The procedure works with the following logical code:

```python
        for category, keywords in scent_keywords.items():
            if any(keyword in user_input.lower() for keyword in keywords):
                result = self.recommend_performance(user_input, category)
                return result
        # If no keywords related to perfume are found, hand over to intent_chatbot
        return self.intent_chatbot.generate_response(user_input)
```

Dataset Selection:

The text dataset mentioned earlier is called *Perfume Recommendation Dataset* [4] (downloaded from Kaggle), which covers thousands of perfumes and is rich in data while the file size is only 1MB, which is the reason why I chose it. I only used the ‘Name’, ‘Brand’ and ‘Notes’ columns of the dataset, where the contents of ‘Notes’ column is used to match the scents entered by the user, and the detailed scent descriptions of each perfume are the key to the procedure's accurate recommendations.

Core Functions and Code Details:

The core functionality of ‘Perfumer’ consists of the following parts:

1. Dataset Matching:

```
    def recommend_performance(self, user_input, scent_category):
        # Ensure Notes field is a string type and handle missing values
        self.dataset['Notes'] = self.dataset['Notes'].fillna('').astype(str)

        # Extract keywords from user input using regular expressions
        keywords = self.extract_keywords(user_input)

        # Filter the dataset based on the scent category
        if scent_category:
            scent_filtered_data = self.dataset[self.dataset['Notes'].str.contains(scent_category, case=False, na=False)]
        else:
            scent_filtered_data = self.dataset

        # Initialize perfume matching score list
        perfume_scores = []
        for _, perfume in scent_filtered_data.iterrows():
            # Get the perfume's Notes field
            perfume_notes = perfume['Notes']

            # Calculate the matching score for each perfume
            score = 0
            for keyword in keywords:
                if keyword in perfume_notes:
                    score += 1

            perfume_scores.append(score)
        if perfume_scores:
            # Get the index of the perfume with the highest score
            most_similar_index = perfume_scores.index(max(perfume_scores))

            # Get the details of this perfume
            recommended_perfume = scent_filtered_data.iloc[most_similar_index]
            perfume_name = recommended_perfume['Name']
            perfume_brand = recommended_perfume['Brand']
            perfume_notes = recommended_perfume['Notes']
            responses = [
                    f"You're in for a treat! How about the delightful
[{perfume_name}] from [{perfume_brand}]? It has a wonderful blend of notes like
[{perfume_notes}], ......]
            return random.choice(responses)
        else:
            return "I'm really sorry that I couldn't find the suitable
perfume, ......"
```

The code in this section first fills in the missing values of the ‘Notes’ field and ensures that it is of string type. It then extracts the keywords entered by the user via regular expressions and filters the relevant perfume data according to the perfume category. Next, the keyword match score for each perfume is calculated and the name, brand and notes description of the highest scoring perfume is returned.

2. Intent Matching:

```
    def get_perfume_history(self):
        perfume_history = ["Perfume dates back to ancient Egypt, where it was
used in religious rituals and for personal adornment.",
            "The word 'perfume' comes from the Latin word perfumare, meaning 'to
smoke through,' as incense was often used in early fragrance-making.",
            ......"]
        return random.choice(perfume_history)
    def get_perfume_brand_info(self):
        perfume_brand_info = ["Chanel: Classic elegance and timeless luxury. Chanel No. 5, launched in 1921, is one of the best-selling perfumes in history, known for its floral aldehyde scent.",
                                "Dior: Bold and elegant fragrances that reflect French refinement. Dior perfumes are often known for their rich, complex compositions.",
                                "Guerlain: Being one of the oldest perfume houses (established in 1828). Guerlain is known for creating rich, opulent scents with a focus on floral, oriental, and powdery notes.",
                                ......]
        return random.choice(perfume_brand_info)
```

```
    def generate_response(self, user_input):
        user_input = user_input.lower()
        # Check perfume history
        if any(keyword in user_input for keyword in ["perfume history", "history of perfume", "origin of perfume", "history"]):
            return self.get_perfume_history()
        # Check perfume brands
        if any(keyword in user_input for keyword in ["perfume brands", "brand information", "perfume brand information", "brand"]):
            return self.get_perfume_brand_info()
```

This is an excerpt from some intent-matching related codes, showing how to match different keywords entered by the user with corresponding intents and generate corresponding replies. For example, when the user mentions the keyword ‘perfume history’, the code will call the ‘get_perfume_history’ method to return a random paragraph about the history of perfume; when the user mentions the keyword ‘perfume brand’, the code will call the ‘get_perfume_history’ method to return a random paragraph about the history of perfume; when the user mentions the keyword ‘perfume brand’, the code will call the ‘get_perfume_brand_info’ method to provide brand information.

Results

In week nine of user testing, I invited four participants to take the test, and the results provided valuable feedback. Below are some of the test notes from the user tests:

You: I love citrusy and fruity perfumes.
Bot: You're in for a treat! How about the delightful [11 - Semma Eau de Parfum] from [Odin]? It has a wonderful blend of notes like [Warm myrrh, Fresh chili pepper, Cinnamon bark, Herbaceous clove, Sandalwood, Tonka bean, Sweet tobacco], perfect for your next perfume adventure.

You: what's next?
Bot: Is there something else you’d like to discuss or learn about?

You: I love the smell of the ocean.
Bot: It makes me happy to see you happy. Ongoing you continue to serve.

You: tom ford perfume
Bot: Sorry, I didn’t quite get that.
You can ask me some questions again. I'd be happy to help！
My functions are as follows:
[1] I can recommend perfumes for you, just need to tell me what [scents] you like or your [zodiac sign].
[2] I can also share perfume-related jokes, or tell you about some perfume brands or their history.
[3] Of course, I can also get time; help you relieve your emotions; introduce myself, etc.

From the results of the test, the successes were that the procedure successfully recognized the inputs of ‘citrusy’ and ‘fruity’ perfumes and was able to make perfume recommendations. However, when the user mentions ‘ocean scent’, the application fails to provide relevant recommendations, which reflects the lack of certain common perfume notes in my data dictionary. In addition, when the user asked for a brand (e.g., ‘Tom Ford’), the procedure failed to recognize and make a recommendation, suggesting that I had not designed brand recognition into the system. The tests also revealed that the procedure lacked sufficient guiding text and that the user was not clear on what to do next after receiving a response, leading to an interruption in the chat.

Discussion

Based on the results of the user testing, I gained a lot of valuable feedback and did some deep reflection. Firstly, I realized that I had been in a bit of a hurry to write the textual content in the code and had overlooked some user experience details. For example, the procedure successfully recognized and responded to inputs of citrus and fruit perfumes, but failed to identify common scents such as ‘Ocean Scent’, which was not covered by my dictionary data. To solve this problem, I have further expanded my perfume notes dictionary. I have categorised oceanic scents into the ‘fresh’ perfume type, which the procedure now recognizes and provides recommendations for; the text lacks sufficient lead-in phrases, so I have also reintroduced lead-in text where it might be needed, to help the user have a better chat.

However, the main features were identified at the beginning of the project and most of the code was already done, so I didn’t have enough time to add some additional features such as ‘brand keyword recognition' in the later stages of testing. Nevertheless, I plan to implement these features in subsequent releases and improve the project by learning about coding.

It is certain that as my coding skills continue to improve, I will be able to add more functionality to the project and provide a richer interactive experience for users. I will continue to optimise the system and explore more innovative points to ensure that the project can be closer to the user's needs.

Conclusion

In my view, the 'Perfumer' project has largely achieved its goal of creating an interactive, user-friendly chatbot for perfume enthusiasts and beginners. It offers personalized perfume recommendations while promoting knowledge of perfume brands, categories, and history, fostering a deeper interest in perfume culture.

However, 'Perfumer' is still somewhat short of my ideal, as there are certain functional limitations in its current use. If I had more time for development, I would add additional features related to perfume recommendations, such as personalized suggestions based on the user's mood and preferred perfume brands. As my coding skills continue to improve, I look forward to expanding 'Perfumer' with more engaging and useful modules, ultimately providing users with a richer and more immersive perfume exploration experience.

Ethical considerations

The project should be used only for personal learning, and exploration of interests, and should not be used for commercial sales or personal privacy data collection. My program involves simple emotion recognition and responses; the bot will not ask for more in-depth emotional information, nor will it actively ask for private information. Therefore, the project does not raise ethical issues.

LLM disclaimer

For this project, I am using the large-scale language model ChatGPT as an aid. I used it only for the following aspects:
Code part: When implementing the 'Perfume Recommendation' function, I used ChatGPT as an aid to find and fix some of the code errors.
Reporting: I only used ChatGPT to check the English grammar errors of parts of the article when writing the project report.

Bibliography

[1] Tiffany. (2023) *Chatbots In the Fashion Industry: Benefits and Examples.* Available at: https://www.chatinsight.ai/chatbots/fashion-chatbots/ (Accessed: 23/11/2024).

[2] Symrise. (no date) *The most inspiring perfumers combined with the most powerful AI fragrance tool* (*Philyra) – makes perfect scents.* Available at: https://www.symrise.com/scent-and-care/competence-platforms/philyra/ (Accessed: 23/11/2024).

[3] Ljubisavljevic, M. (2021) *Find My Perfect Perfume With The New Scentbird Quiz.* Available at: https://www.scentbird.com/blog/find-your-perfect-scent-with-the-new-scentbird-quiz/ (Accessed: 22/11/2024).

[4] Bansal, N. (2020) *Perfume Recommendation Dataset.* Available at: https://www.kaggle.com/datasets/nandini1999/perfume-recommendation-dataset/data (Accessed: 17/11/2024).