AN INTERACTIVE NLP-BASED AI SYSTEM

- Flight booking chatbot -

1. Introduction

This report discusses the development and evaluation of an interactive NLP-based chatbot, which is designed to handle a flight booking transaction. The chatbot is built to perform tasks such as intent matching, identity management, transactional dialogues, question answering, and small talk. It leverages modular architecture to integrate core functionalities and provide a user-friendly experience. Conversational design principles, such as prompt design, personalization and error handling have been implemented to increase usability. This report outlines the system's architecture and conversational design of the chatbot, then mentions the evaluating results and reflects on the overall development.

2. Chatbot Architecture:

a. Text Pre-processing

The most important and most used piece of code is about pre-processing the text. It is at the heart of natural language preprocessing. It transforms raw text into a clean structured form for further NLP tasks, such as classification, matching, and extraction. Not only this functionality is used to analyse the user inputs, but also be used to process the data in every dataset.

This process contains 5 steps, they are:

- Lowercasing: The text is normalised by converting all characters to lowercase, which ensures case-insensitive comparisons.
- Tokenization: Then, the text is split into smaller units called tokens, which are words and punctuation. This step allows individual word-level processing.
- Stopword Removal: Next, we filter out common words that usually don't contribute much to the meaning.
- Lemmatisation: We reduce words to their base form, for instance, "running" is reduced to "run". This maintains consistency in word representation.
- Non-Alphanumeric Removal: Finally non-alphanumeric characters are removed to focus on meaningful textual content.

Together, these steps reduce noise, standardize raw text and improve the performance of NLP model by focusing on relevant linguistic information.

The reason we need to lowercase input is to maintain the equality of words. If the words aren't lowercase, NLP models will treat them with different cases as distinct, which could lead to inefficiencies. Tokeniser handles punctuation, and special characters better than simple string splitting. Because stopwords don't usually contribute much to the meaning of a sentence, they are often eliminated. However, in the TextPreprocessor class, there are 2 preprocessors. One of them removes stopwords, the other one doesn't. Next, the words are lemmatised to their base form. The reason why lemmatization is used widely instead of stemming is that lemmatisation provides linguistically correct base forms, unlike stemming, which may produce invalid words. Lastly, non-alphanumeric characters are removed from the text. This ensures that the text does not have any unusual characters, so that further processing steps run predictively and smoothly.

There are 2 distinct preprocessing methods. One of them removes stopwords, the other one doesn't. These methods are used for different NLP pipelines, which offers flexibility and avoids unnecessary re-implementation.

The preprocess_text() method with stopwords removal, is used to process the datasets, where a large number of text needs to be processed. This way the data in datasets becomes cleaner and more efficient to do similarity matching, because it doesn't have to deal with unnecessary words.

However, the preprocess_text_not_remove_stopwords() method tends to be more accurate when processing user inputs, which are short sentences. This method is used in small talk and handling identity and it gives much higher accuracy compared to the preprocess_text() method. Stopwords removal may remove common words like "I", "are", "my" and "you". Therefore, the processed text has no difference when user inputs "Who I am" and "Who are you". In this case, not removing stopwords is a better option.

b. Intent Matching

Intent matching is one of the most important parts of the project. It classifies the intent of the user, it detects whether user wants to book a flight ticket, have a small talk or just ask some general questions. In this project, the chatbot identifies and classifies the user inputs into 4 intents, which are:

- Book a flight ticket
- Have a small talk
- Ask for their personal information (identity management)
- Ask general questions

To correctly identify the intents and achieve the user goals, TF-IDF vectorizer and cosine similarity matching are widely used in the program. Every input from the user must go through the intent classifier first to detect the most suitable intent and pass it to specialized classes to process; so that the request can be handled correctly and efficiently. Although, during the booking process, inputs do not go through this intent classifier, they must go through another classifier to determine whether user inputs are negative, which may tell they want to change information or stop the booking process. Therefore, we can handle the error and unexpected situations.

In the program, there is a separate Python script dedicated to classifying the intents, named "intent_classifier.py". It works based on textual similarity. It matches user queries to the sample inputs in the dataset, which is tagged with the intent, and find the most relevant sample input to get the most relevant intent. This approach enables the chatbot to respond appropriately.

TF-IDF vectoriser is used to transform user input and data in datasets into numerical vectors. Before doing the intent matching, we had already pre-processed sample inputs in the dataset. TF-IDF weigh the terms by their frequency in the document and their rarity across the corpus. This way, it increases the meaning of matches. Next, cosine similarity is applied to measure the similarity between the vector of the user input and the dataset vectors. After identifying the best match, we compare similarity score with the threshold of 0.3. If the score exceeds 0.3, the program returns the corresponding intent; otherwise, it labels the intent as unknown.

The reason why we use TF-IDF and cosine similarity is that these methods are lightweight and effective for classifying intents in a controlled dataset. This way we can avoid the need for computationally expensive machine learning models. The threshold of 0.3 ensures that the weak matches are filtered out, which reduces false positives and improves response quality. Because even when it is the best match, it can be irrelevant to user question, in this case, the dataset does not have answer to this question.

c. Identity management

This feature is specifically designed to handle user's information. With IndentityManager class, chatbot can recognise, acquire and memorise user's details. It can handle different identity attributes like age and date of birth dynamically. Chatbot can dynamically store and track user attributes by using a dictionary. For instance, when user mentions about their age for the first time, it automatically expands the size of dictionary to store the new attribute. Dictionary structure allows flexible storage and retrieval of user attributes without hardcoding specific keys.

By tagging data in dataset, the intent and kinds of personal details are detected. Chatbot can create new attributes to store the kinds of personal details (etc. name, age, favourite colour) in dictionary. Together with TF-IDF and cosine similarity, it matches user input to a pre-defined dataset to retrieve the most appropriate response. By extracting and storing user details, the chatbot can respond in a more personalized way. Apart from that, system stores important information like username in Person object, that way it ensures consistency across chatbot components.

Additionally, chatbot supports both implicit and explicit updates, like "Call me Bill" and "Bill is my name" by detecting the attributes using Part of Speech (POS) and Named Entity Recognition (NER). By leveraging NER, the person's name is detected right after being tokenized, no matter where the name is in the sentence. In case every word is in lowercase, the last word of input is extracted and capitalized to check if it's a person's name. If it is not, chatbot will prompt user to capitalize their name. Similarly, POS and NER are also used to detect user's age.

d. Information retrieval & question answering

Retrieving information to answer questions is one of the most straightforward tasks to implement. The QuestionAnswering class matches user questions to the questions in a predetermined dataset. After doing similarity matching, the system retrieves the best corresponding answer. This approach provides direct and contextually relevant answers.

First of all, QuestionAnswering class receives a pre-processed dataset dedicated to answering general questions from the main Python script, which contains instances of all classes and controls every aspect of the chatbot. The pre-processed dataset has a column where every question is pre-processed via 5-step process in "text_preprocessor.py". At the same time, the system receives the input and processes it: lowercase it, tokenise it, lemmatise, remove stopwords and non-alphanumeric characters. Then, the TF-IDF vectoriser converts dataset questions and user input into vectors. After calculating similarity scores between the input question and the dataset questions, it returns the answer corresponding to the most similar question if the similarity exceeds 0.3. Otherwise, a fallback message is provided.

Because the input for Q&A can be long and complex, that's why stopwords in the input need to be removed. This approach eliminates unnecessary words, keep the input clean and meaningful, which allows the matching become more effective. TF-IDF and cosine similarity matching is efficient for matching user queries to existing questions in a predefined FAQ-style dataset. Finally, a threshold and fallback message ensures that the chatbot can gracefully handle questions outside the scope of its knowledge base, without giving any non-relevant answers to user questions.

e. Small talk

Chatbot handles small talk interactions by matching user input to a dataset of predefined small talk responses. The process of handling small talk is quite similar to handling answering questions. After input is classified belongs small talk, the main Python script ("chatbot.py") allocates small talk handler a dataset, which is specially for small talk, and user input. The input and data are pre-processed and then encoded as vectors, then cosine similarity determines the closest match to a predefined small talk response based on textual similarity.

In small talk handling, the preprocessing doesn't contain stopwords removal. This is because the input for this intent is usually short and contains many stopwords. Removing them decreases the meaningfulness of the input, which reduces the accuracy of matching.

The difference between small talk handler and question answering handle can be noticed in datasets. Dataset for Q&A only contains pairs of questions and answers, while the dataset for handling small talk also contains a number of sample inputs from other intents and tags them.

The reason for this is, in a small talk, user can talk about many aspects, including asking questions belonging to other intents. In case the intent classifier fails to detect the correct intent at the very beginning, system can recognise and redirect the input to the corresponding handler.

f. Transactions

This project handles the flight booking process. Therefore, the main purpose of it is to guide the user through the booking process and handles interruptions which may occur.

The system implements a stack to track user processes and handles situations when user wants to go back to previous step, restart the process or they just need to know where in the process they are at. Whenever user moves on to a new stage of booking process, that stage is added to the stack. When user wants to go back to the previous stage, stack pops the current stage out and returns back the beginning of the previous stage.

In case user wants to restart the booking process, the stack will be emptied, and the current stage is assigned as the beginning stage. As the user moves on the process, the stages are added to the stack again. In the transition between the stages, there are conversational markers to indicate the end of a stage, and it is about to move to a new stage. Therefore, user can navigate through the booking process easier.

To detect when the user wants to go back, restart or cancel the transaction, there is a handling negation function which is specifically designed to detect the negative input and classify them into go back, restart or cancel the transaction. It leverages negative keywords like "no", "not", "don't", and "wrong" to detect the user input. Every input has to go through this classifier before it is interpreted and executed.

Finally, to correctly identify the valid locations, the system uses POS and NER to detect places. It leverages place detection method in text_preprosser.py to validate the destination and departure location. This is a quick and efficient way to do, which is suitable for the long BookingProcess class which it handles many stages of transactions.

3. Conversation Design

a. Prompt design

When the user indicates they want to book a flight, the system identifies the intent and activates a booking system processor. The system displays guidance and quick notes, so that user know what to expect during the process and how to navigate. This approach is important as it makes the user feel informed and more confident to process the transaction.

```
Chatbot: How can I assist you today? 💖
User: book
Chatbot: I'm ready to help you book your flight, Bill! 😊 Let's get started.
Chatbot: I'll guide you through the process step by step.
        Step 1: Gathering Flight Information
        Step 2: Filling Personal Details
        Step 3: Confirming Booking Details
        Step 4: Processing Payment

Quick note:
        You can type 'back' to go BACK to the previous step.
        You can type 'restart' to RESTART the booking process again.
        You can type 'cancel' to CANCEL the booking process at any time.
```

Moreover, the system displays **conversational markers at the beginning and** the end of every stage. For example: “Shall we begin?” is used at the beginning, just right after the guidance, to indicate the start of the booking process. “Let's move to the 2nd step 🚀: Gathering personal information” informs user about the transition between stages. This allows users to keep track of the process and know what to expect next.

The design implements constrained responses. These prompts allow the user to understand easily and give a more predictive response. Some prompts even help users to decide more easily. For example: "Will this be a one-way trip or a return ticket? “and “How would you like to pay for your ticket? We accept Credit Card, PayPal, or Apple Pay", these prompts give users choices to choose from. This results in a predictive response and reduces the fraction to users finishing the transaction.

During the conversation, key phrases are often used, such as “Got it”, “Great” and “Perfect choice”. These key phrases make the experience more conversational. On top of that, it shows user that the chatbot has received and understood their inputs.

b. Error handling

Reprompt strategy is widely used to handle errors and interruptions. The system does not use silence because it is not good practice for doing high-stakes tasks like booking a flight. Users need to know whether chatbot interprets and do the transactions correctly, that’s why keeping silence is not preferred.

Reprompt strategy is used for detecting the departure locations and destinations. If user type any place where services are not supported, it will prompt user to choose again: “I'm sorry, we only support flights to England, Singapore, France, and Spain. Could you please specify again?”. During handling errors for departure date, return date and date of birth, reprompt is used when the input is invalid.

Chatbot implements global commands, such as “back”, “restart” and “cancel”, therefore users can go back, restart or terminate the process anytime. Handling negation is also used to categorize negative inputs and handle them appropriately. This approach can detect whether user wants to change some information or stop the transaction, by saying "This is wrong" or "No, I want to edit personal details".

c. Discoverability & Personalisation

At the beginning of every stage, the prompt displays what the user does in that stage. At the end of the stage, it informs the user what to do in the next stage.

During the booking process, if the user wants to do something else, and is not sure what they can do, they can ask chatbot "What can you do?" at any time. It will displays what it can do and guide user when it detects the user’s intent.

The chatbot remembers the names of users and often mentions them throughout the conversation. The name is stored in Person object and withdrawn to use in conversation. Not only limited to names, but system also remembers and restates flight information during the booking process.

```
User: Spain
Chatbot: Got it - Spain it is! And which country will you be flying from?
User: England
Chatbot: Nice, Bill. Will this be a one-way trip or a return ticket?
User: one
Chatbot: Gotcha, that's a one-way ticket. What’s the departure date? (DD/MM/YYYY)
```

d. Confirmation

When it comes to getting the user to confirm an action, explicit confirmation is the most used. Nearly after everytime the system collects information from the user, it restates that information. For example, after user choose the type of ticket, system reponses “Gotcha, that's a one-way ticket.” Sometimes it asks user for confirmation after getting all the details for the flight ticket: “Does everything looks good to you?”.

In addition, visual confirmation method is used twice when confirming the ticket details. System prints out the details in ticket, so that user can check and confirm. This increases the confidence and of the users. Occasionally, generic confirmation is used after users share their name or update their details. The popular response is: “I have updated your <details> to <new information>”.

e. Context

The chatbot implements a stack to keep track of the stages of transaction. This approach allows users to be aware of which stages they have completed, where they are at the process and how many steps left, they need to complete. Using a stack is convenient to do backtracking when user wants to go back or restart the process.

4. Evaluation

a. Usability

To evaluate the usability, the chatbot has been testing by individuals with different demographics and levels of technical expertise. There are 4 people involved in the usability testing.

The first test was to test the sample dialog. The flow of dialogs and conversations was scripted using flowchart. The second test is when intent matching is developed with other features such as answering questions and small talk are fully developed, but not the booking process. The third test happened when the booking process is built and the chatbot is fully developed

Users were asked to interact with the chatbot for tasks such as booking a flight, asking general information, and having small talk. The key findings are:

- Ease of use: Most users found the chatbot intuitive because of the clear prompts and step-by-step guidance. They found features like automatically updating personal details, dynamic responses, and handling errors enhancing the usability.
- Discoverability: Users like the prompts indicating available commands like, “type 'exit' or 'quit' to stop the program” and the clear explanation of stages in the booking process. They also love how chatbot respond when being asked “What can you do”. However, some users suggested mentioning those commands more frequently during the conversation.
- Personalization: The chatbot’s ability to remember user-specific information like names, age and other details was highly evaluated. Users are impressed on how well the chatbot is trained to recognise the different input for an intent. This created a more human-like and user-friendly experience.
- Error Handling: When users entered invalid inputs, such as incorrectly formatted dates, the chatbot politely provided specific guidance for correction.

b. Performance

Performance was evaluated in terms of response accuracy, speed, and system robustness.

- Intent matching: the TF-IDF and cosine similarity approach achieved an 85% success rate in correctly identifying the user intents during tests. That means within 12 tests, there are only 3 times the classifier failed to recognise. Therefore, there are some limitations when dealing with advanced and tricky prompts which have ambiguous inputs.
- Transaction Handling: Booking processes were completed without significant delays, even when managing complex inputs, such as multiple passengers and updating their information. The chatbot handled restarts, backtracking, and cancellations without errors.
- Resource Efficiency: Memory usage remained stable during the test. The preprocessing and vectorisation methods optimise the speed of execution with the tasks like intent matching and retrieving information.
- Robustness: The chatbot performed well under varied conditions, such as missing user attributes, invalid dates, or unexpected commands. The fallback mechanisms ensured the conversation flows continuously.

5. Discussion

a. Reflection on results:

The evaluation results reveal the strengths of chatbot in usability and performance. The chatbot's structured architecture and conversational design enables a seamless user experience. User really like:

- How the prompts are design to guide user through booking process
- The effective error messages that are clear and easy to follow, which reduces user frustration.
- Personalisation features, such as remembering names and providing tailored responses. This incredibly satisfies users.

However, there are some areas for improvement. Although intent matching worked well overall, probabilistic thresholds or multi-intent suggestions could improve the handling of tricky inputs. While help commands were explained at the start, users sometimes forgot them. So, those key commands should be constantly mentioned in the conversation. Despite the quick response in intent matching, the responses in booking process still have a delay.

b. Reflection on the project overall

This project provided valuable insights about developing NLP-based systems. The key things to consider are:

- NLP-based systems need an effective preprocessing: Tokenisation, lemmatisation, and stopwords handling significantly impact the performance of intent matching and information retrieval.
- Always involve agile development and user testing in the developing process. Iterative testing with real users ensures the chatbot meets the usability expectations.

- The design and modular architecture of the system significantly affect future improvement. With a good design, we can implement machine learning models to improve intent matching or expand transactional capabilities, with little changes in the original code.

6. Conclusion:

To sum up, this chatbot project has successfully achieved its goal as an interactive conversational AI chatbot which can handle a transaction. By utilising modular architecture and conversational design principles, it provides a human-like and robust user experience. Usability testing demonstrated high levels of user satisfaction. However, areas like handling ambiguous inputs and improving help discoverability remain room for improvement. Overall, this project successfully illustrates the natural language processing architecture and conversational design.