Assignment 7: Dialogue Systems

Johan Östling - 20h
Hanna Olsson - 20h

March 7, 2023

1 Reading and Reflection

GUS is natural language dialog systems designed for use in restricted domains. In the article, GUS is used as a travel agent for helping people to book flights and provide travel information. They mentioned that the restriction in what the dialog system provides helped improve the system, since it is easier to focus on a specific domain than doing a general dialog system. Nevertheless, the innovative part of GUS was that it uses frames to represents knowledge and possible dialog scenarios. For example, a frame could be constructed to represent the users goal. To fill in this frame, GUS searched through the users responses to find for example destination and departure cities. The key takeaway from this article is that using a frame-driven dialog system helps in organising knowledge, handling errors and managing dialog structure.

2 Implementation

2.1 Description of the implementation

For the implementation of the dialogue system we decided to keep it as basic as possible while still allowing the system to scale well.

The first mission was to get the system to recognize the intent of the user input, i.e. if the user wants to find a restaurant, the next bus, be provided with a weather forecast or if the request is something unrelated to these three topics. To get this functionality we decided to use support vector classification (SVC) that we used to classify the requests. We decided to use SVC as it is a popular machine learning algorithm for text classification tasks because of its ability to handle high-dimensional and sparse data, such as text data. It works by seperate sentences into different classes based on features. It could for example be specific words. Such that the word 'restaurant' often shows up in sentences where someone wants to find a restaurant. When the SVC is trained it has set up decision boundaries and when provided with a new sentence, it looks at its features to decide in which class to assign it to. The dataset used consisted of 312 sentences and their corresponding intent. We do recognize that a larger dataset would be preferable, but we did not find an appropriate one available, so we decided to make the sentences ourselves, using chatGPT, which is the reason for the lacking size of the data. The model did however reach an accuracy of 98% when provided with test data and when trying manually with different requests it could identify the intent correctly in most of the cases. The accuracy of 98% is not a good measurement though. Because the sentences were provided using chatGPT, most of the sentences were quite similar. The sentences were short-to-the-point sentences. So when testing the intent classifier, it was tested on data that was very similar to its training data. This resulted in a very high accuracy. So, in our case, using a keyword matching algorithm to identify intent would most likely work just as good. But our classifier is much more scalable and more accurate if we would train in on better data. In a big and diverse set of data of sentences with labled intentions, our intent classifier would be able to find the correct intention from underlying intentions. This is much better than using a keyword matching algorithm.

We then proceeded with creating an abstract class Frame that has the basic structure of having a dictionary of tasks and their corresponding value and fetching appropriate data based on the values of the tasks. The tasks are types of information that is needed to complete the request. So for each intent we implemented a concrete class, which in our case was RestaurantFrame, WeatherFrame and BusFrame, with its own unique set of tasks. For example, the WeatherFrame had tasks "Location", "Time" and "Day" which essentially means that for the frame to provide a weather forecast, it needs to have information about the location, time and day. The choice of making an abstract class was based on wanting to reuse as much code as possible and allow for easy scaling. Every instance of Frame has the method find_everything(sentence) that tries to fill in the missing tasks based on the sentence it takes as an argument. However, the exact implementation of the method varies between the concrete classes since there are different tasks to find in each class. If we look at the implementation of find_everything() in the WeatherFrame we see that it uses three helper methods that are specific for this class, namely find_day(), find_time() and find_location() which are directly a consequence of day, time and location being the tasks in this class, see figure 1. Other methods such as missing_tasks() are shared among the concrete classes and are thus implemented in the abstract class. This method simply returns a list of the tasks that still have no information.

```python
def find_day(self, sentence):
  day_regex = r"(Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sun
  match = re.search(day_regex, sentence, re.IGNORECASE)

  if match:
    day = match.group(0)
    self.tasks["Day"] = day
  else:
    return False

def find_time(self, sentence):
  time_regex = r"([1-9]|1[0-2])(:[0-5]\d)? ?([ap]m|AM|PM)"
  match = re.search(time_regex, sentence)

  if match:
      time = match.group(0)
      self.tasks["Time"] = time
  else:
      return False

def find_location(self, sentence):
  print("finding location")
  doc = nlp(sentence)
  for ent in doc.ents:
      if ent.label_ == "GPE" or ent.label_ == "LOC":
          print("true")
          location = ent.text
          self.tasks["Location"] = location
          return True
  return False

def find_everything(self, sentence):
  if self.tasks["Location"] == None:
    self.find_location(sentence)
  if self.tasks["Time"] == None:
    self.find_time(sentence)
  if self.tasks["Day"] == None:
    self.find_day(sentence)
```

Figure 1: Methods of concrete class WeatherFrame

The mission for the digital assistant then becomes to fill in the tasks based on the user input, and to ask relevant questions to get the information needed. In the implementation of the actual chat it starts by taking in a user input and tries to identify the intent based on the input. If the intent cannot be mapped to any of the three requests this digital assistant can handle, the digital assistant asks the user to rephrase the request until an intent can be identified. When an intent is identified the digital assistant repeats the request and possible tasks that have been identified and asks if it has the correct interpretation of the request. If the answer is no, the chat restarts and the user is prompted to rephrase the request and if the answer is yes the digital assistant asks the user to input information about the missing tasks. The latter continues until all tasks are filled in and then it repeats the entire request with all the given information and fetches data from a database based on the values of the tasks.

Some of our find methods that extract the right information from the user's input, work by using the python module Spacy. Spacy provides different natural language processing functions. These function are pre-trained using large amount of annotated text data, and can therefore perform various natural language processing tasks without needing additional training. So in our case, we used Spacy to identify, for example, a location from a sentence.

In other cases we used our own natural language techniques to extract the right information. For example, to extract time, we searched for a number in the sentence followed by "AM" or "PM".

2.2 Sample outputs

In figures 2 through 5 some conversations between the implemented digital assistant and a user is shown.

```
Digital assistant: What can I assist with?
User input: I am craving Italian food, do you know a good restaurant?
Digital assistant: I understand that you want help with finding a restaurant with the following criterias: Cuisine:Italian
Digital assistant: Is this correct, Yes/No?
User input: Yes
Digital assistant: Good! Please specify informating about the following: Day and Time and Location.
User input: Friday would fit me best, around 7:30PM
Digital assistant: Good! Please specify informating about the following: Location.
User input: Somewhere in New York
Digital assistant: You want to find a restaurant in New York on Friday at 7:30PM with a Italian cuisine.
Digital assistant: I have found the following restaurant for you: Resturant Bongo
```

Figure 2: An example conversation about finding a restaurant

```
Digital assistant: What can I assist with?
User input: Will it be cold in New York on Friday?
Digital assistant: I understand that you want a weather forecast with the following criterias: Day:Friday, Location:New York
Digital assistant: Is this correct, Yes/No?
User input: Yes
Digital assistant: Good! Please specify informating about the following: Time.
User input: 11:30AM
Digital assistant: The weather in New York on Friday at 11:30AM will be: sunny and 20 deg celcius
```

Figure 3: An example conversation about a weather forecast

```
Digital assistant: What can I assist with?
User input: I need to find the next bus from Boston
Digital assistant: I understand that you want to find the next bus with the following criterias: Start:Boston, Time:12:02
Digital assistant: Is this correct, Yes/No?
User input: Yes
Digital assistant: Good! Please specify informating about the following: Destination.
User input: New York
Digital assistant: The next bus leaving from Boston with desitination New York is: Bus 57 at 13:30
```

Figure 4: An example conversation about finding the next bus

```
Digital assistant: What can I assist with?
User input: Do you know how I can go from Boston to New York?
Digital assistant: I am sorry, I don't understand your request. Please rephrase.
User input When is the next bus leaving from Boston to New York?
Digital assistant: I understand that you want to find the next bus with the following criterias: Start:Boston, Destination:New York, Time:12:04
Digital assistant: Is this correct, Yes/No?
User input: Yes
Digital assistant: The next bus leaving from Boston with desitination New York is: Bus 57 at 13:30
```

Figure 5: An example conversation about finding the next bus where the system fails to identify the initial intent

One thing to notice about the system is that it is pretty good at identifying the intent even when the user intent is not explicit. This can be seen in figure 3 when the user asks if it will be cold in New York and the digital assistant still identifies it as a weather forecast request even though the request does not contain the keyword "weather". This is one of the reasons for why we choose to use machine learning for the intent identifier instead of using keyword matching. This approach also allows the system to scale easier since only new data is needed when adding a new intent instead of manually adding appropriate keywords. However, it is not perfect as we can see that it fails to identify the request of " Do you know how I can go from Boston to New York" as a bus request in figure 5. But this depends highly on our small data set, used to train our intention classifier.

2.3 Limitations of the system

To continue discussing the limitations of our system, an obvious shortcoming is that the digital assistant is not flexible in its answers. When the intent is identified and confirmed the digital keeps outputting the same sentence of "Good! Please specify information about the following criteria..." until all information is found. The only flexible part is that the criteria that are asked to be specified are updated based on the missing information. This, of course, reduces the "human-like" factor of the system to zero as it is very clear that these are just standardized answers. However, we chose this approach as it, in a time efficient way, allows the system to scale well. To extend the system to be able to handle more requests, you only need to create a new frame and implement the abstract methods. The other alternative would be to write many if-statements and outputting more tailored answers to the current situation, but this could be both time-consuming for this implementation and it would make scaling very difficult.

Another limitation of the system is, as discussed earlier, the dataset used for training the intent recognition model. First of all, it should be much larger, but due to lack of time, this was not possible. The sentences in the dataset also need to be more diverse, as the current sentences are only a few words and are generally straight to the point. This results in the model not identifying the right intent when the user input is longer sentences and it has in most cases only connected some common keywords to the different intents. As we could see in figure 5 the model only identifies the right intent (of finding the next bus) when the user input contains the word "bus". However, with a larger dataset with more diverse sentences, the system should learn to identify even vague sentences.

Additionally, the system has very little error handling since the only error handling that exists is when the user inputs a request where the system identifies the intent as something other than the three tasks it can perform. The system is now reliant on that the user spells correctly and after the first confirmation of the request, there is no way to tell the digital agent that it has been misunderstood.

2.4 Suggested Improvements

If more time was available, there are several improvements that could be made to the system to make it more advanced. Firstly, we suggest using more rule-based natural language techniques like the chatbot ELIZA to make the responses to the user more conversationally fluent. ELIZA used knowledge of natural language to respond to user inputs in a rule-based fashion. One example is that if the user inputs "I am happy", ELIZA responds with "You are happy". These techniques make the chatbot more human-like since this is how normal human conversation often is like.

Another improvement that could be made is to incorporate a sentiment analysis component. This would enable the system to analyze the user's mood or emotions and provide recommendations based on their current state of mind. For example, if a user is feeling sad, the system could recommend uplifting movies or music to improve their mood. Another way sentiment analysis could be used is to change the responses of the dialogue systems depending on the mood of the user. This increases the human-like interpretation of the system's responses. One way to implement a sentiment analysis component in a dialogue system is to use a machine learning algorithm trained on a large dataset of labelled data. The labelled data would consist of examples of text labelled with the corresponding sentiment (positive, negative, or neutral). The machine learning algorithm would learn to recognize patterns in the data and use those patterns to classify new examples of text.

Another improvement that could be made is to incorporate a feedback mechanism. This would allow users to provide feedback on the recommendations they received and enable the system to learn from the feedback to make better recommendations in the future. This would also help the system adapt to changing user preferences and behaviour over time.

Finally, the system could be extended to include more domains or areas of interest, such as sports, politics, or food. This would require expanding the system's knowledge base and training the algorithms on new data, but it would enable the system to provide more diverse and comprehensive recommendations to users with varied interests. The easiest way to do this with the current implementation is to add more frames, but if even more time was available more machine learning models could be incorporated to make the system even more flexible.

3 Summary and Reflection

3.1 Hanna

3.1.1 Dialogue systems 28/2

Developing a dialogue system presents numerous challenges, as there are many aspects of human conversation that need to be mimicked. For instance, people take turns when speaking, and they have an innate ability to know when to start and stop talking, which can be difficult for a system to learn. Additionally, conversations can be divided into various speech acts, such as constatives, directives, commissives, and acknowledgments. Grounding is another crucial concept, indicating that the participants have understood each other. A simple token for grounding is the word "OK." Initiative in conversations is another characteristic of human interactions that can either be user-initiated, system-initiated, or mixed-initiative. Inference and implicature are other essential features of human conversation.

In the early stages of chatbot development, rules were used to create conversations. For example, certain keywords, such as "I'm" and "my," were replaced with "you are" and "your," respectively, resulting in more natural-sounding conversations. On the other hand, corpus-based chatbots learn from large datasets of existing conversations and do not rely on rules. Finally, ML-based encoder-decoder chatbots use neural response generation for dialogue.

In task-based dialogue systems, speech is often used to communicate with the system. Examples of such systems include Google Home and Amazon's Alexa. A task manager is responsible for knowing how to perform the task, which is then communicated via the dialogue manager.

3.1.2 Reflection

Through last weeks assignment, I gained a deeper understanding of how AI algorithms work in practice and how they can be applied to real-world problems. I also learned about the importance of good software design principles, such as writing clean, modular, and well-documented code.

I found the assignment to be challenging, but also rewarding. While Tic-tac-toe is a relatively simple game, I realized that implementing an AI agent to play it optimally still requires careful attention to detail and thoughtful design choices. Overall, I was pleased with my implementation, but I also identified areas for improvement and future work. For example, I would like to experiment with different policies and algorithms, and explore how my implementation could be adapted to work with larger or more complex games.

3.2 Johan

3.2.1 Dialogue systems 28/2

There are two different concepts of dialogue systems, chatbots and digital assistant. Where chatbots tries to mimic human behaviour for arbitrary conversation. And digital assistans are made for tasks.

- Properties of human conversation

A dialogue is a sequence of turns. To know when turns happen, you need to predict endpoints. In conversation, sayings can be divided into four different speech acts; constatives, directives, commissives, acknowledgements. Further properties are grounding and sub-dialogue. Adjacency pair, for example "nice jacket" → "oh, this thing!", and request → "Yes/No". Correction sub-dialogue. Presequences. We also use initatives in conversation. The last propertiy that was presented was inference and implicature, which is dialogue that is not directly an answer to a statement, but they implicilty answer the statement.

- Chatbots

ELIZA was presented which is a chatbot created in 1966. It was a rule-based AI that identified keywords to give a response. For example if it read "I'm", it responded with "you are". Corpus-based chatbots are based on very large datasets of real conversations. They respond, not by rules, by based on user's last turn (or two). IR-based chatbots, are a type of chatbot that uses preexisting information sources to provide responses to a user. ML-based encoder-decoder chatbot uses ML techniques to generate responses, these consists of two main components, and encoder and a decoder.

- Task-based dialogue systems for digital assistants

Task-based dialogue systems often contains speech recognition. Then follow different parts of a commercial dialogue system. So after speech recognition follows natural language understanding, dialogue manager, and then task manager. From the task manager follows the dialogue manager again, then natural language generation and lastly text-to-speech synthesis.

3.2.2 Reflection

The previous module was very interesting, and I find the concept of game-playing systems very fascinating. I enjoyed trying to implement a Monte Carlo Search Tree and to understand how it worked was a perfect level of difficulty for me. It was not trivial, but still, it was not so difficult to understand. It gave me extra motivation to truly understand Monte Carlo Tree Search since it is used for a variety of different games and used in much more complex game-playing systems like AlphaGo.

Nonetheless, the code implementation was much more difficult than anticipated. We represented the graph with objects as nodes, and then we created the search tree as an object with methods in a standard Monte Carlo Tree Search manner. Meaning we defined methods for selection, expansion, simulation and backpropagation. The biggest difficulty was to decide on the selection policy and also to decide the Q function for the nodes. We eventually ended up with defining Q as the number of wins + the number of ties - the number of losses. And for the selection policy we used the UCT formula.

My key takeaway was the knowledge of how a Monte Carlo Tree Search works. It would be intresting to try and implement a MCTS to build an AI capable of becoming good at chess.