COMP3074 Human-AI Interaction
Lab 4: Conversational design

Joel Fischer

October 2023

Contents

1 Introduction

This week's lab is focused on conversational design in order to build a **classifier**. In order to do so, we will go over the conversational design principles that are expected of your coursework.

2 Conversational Design

Conversational design refers to the process of creating efficient and engaging dialogue systems for artificial intelligence applications, such as chatbots, voice assistants, and other interactive AI systems. The goal is to design these systems in a way that they can communicate with users in a manner that feels as natural and intuitive as a human conversation. This lab is focused on the conversational design principles that you must implement in your coursework. It presents some pseudo-code examples to inspire you to implement the principles in your chatbot.

WARNING: The pseudocode examples provided in this lab are not meant to be copied. They are meant to illustrate the concepts we talk about. They will not run by themselves.

2.1 Confirmation

It is a good idea to get the user to confirm certain important actions, for instance those that have a monetary consequence for the user. For example, get the user to "hit enter or type cancel". Here, we also generate a random booking number that we may use later, for instance, to cancel the booking.

```
print('These are your booking details: You have booked ' + str(assigned) + ' seat(s). \nIf you are happy to proceed with your booking click enter or type \'cancel\' to cancel your booking')
while True:
    reply = input()
    if reply == '':
        booking_reference = random.randint(1000, 9999) #assigns a random booking reference for the user
        print('Booking confirmed! Booking reference: ' + str(booking_reference))
```

Over to you. What kind of confirmation strategies can you think of to improve your chatbot? Recall that while the above example demonstrates explicit confirmations, we also learnt about implicit and more complex forms of confirmations (confidence-level-based confirmations). Have a go at implementing at least one of these types of confirmation strategy for your chatbot.

2.2 Input validation and error handling

In order to handle potential user input errors and provide the user with a pathway to recovery, it is a good idea to validate (check) the user input where you are expecting a specific kind of input to progress with the transaction.

2.2.1 Verifying and helping users to generate well-formed input

```
# Simple string checking
if answer.isdigit():
    print(f"{chatbot_name}: Your answer should not be a number!")
elif answer == '':
    print(f"{chatbot_name}: Your answer should not be empty!")
```

You may want to verify that the user has chosen a seat row that is actually in your cinema.

```
if seats_reserve not in range(seats_in_row):
    print('Please choose a seat within our cinema :')
```

2.2.2 More complex checking

Here's a more complex specific case - checks for the age of the child(ren) on the booking, handles error input.

```
# Stops inputting children if skip
if child_input in ["skip", "no", "none"]:
    have_child=False
# Due to URl formatting, the first child is 1_(AGE), assigned using count, and
    the subsequent ones are 2C1_(AGE)
elif (child_input.isdigit()):
    child_string += f"1_{child_input}" if count < 1 else f"%2C1_{child_input}"
else:
    print("Sorry! I don't understand. What is the age of the child? Or \"skip\"?"
    )
```

2.2.3 Returning alternative answers

Sometimes it can be effective to return an answer from an alternative source when your database does not appear to have a suitable answer.

```
# Compares similarity value to threshold, if less then google for an answer
if max(cosine_matrix.flatten()) <= threshold:
    return google_result if (google_result := google_search(query)) else "I can't
     find what you are looking for, my liege. Try something else."
```

Over to you. There are many useful ways - from simple to complex - to validate user input and provide more specific error handling help in case the input is not formatted as expected/required by your chatbot. Go ahead and find some parts in your chatbot code where this would be useful and implement it.

2.3 Personalisation

Using a simple form of pattern-level sentiment matching can be a powerful way to personalise the response to something the user said.

```
intent_feelings = compare_intent(feelings_response.lower(), intent_threshold)
# Prints feeling according to user emotion inputted
if ('feelings_user_positive' in intent_feelings) or ('feelings_user_negative' in
    intent_feelings):
    print(f"{chatbot_name}: {intent_feelings[1]}")
```

Of course, there are many other ways in which you could personalise your chatbot response to the user, for instance, using datetime.datetime.now() to greet the user with the appropriate greeting for the time of day, remembering when they last used the chatbot, or what their favourite order is.

Over to you! Have a go at implementing some personalisation, beyond merely using the name of the user for your chatbot.

2.4 Contextualisation

One way in which you may want to think about context is in terms of the user's progress through the transaction. Tracking the progress allows you to, for instance, tell the user whether they are ready to check out or have some remaining steps to complete before doing so.

```
if booking.completed
  print('All done! Ready to check out?')
else
  print('There are ' + str(booking.steps_remaining) + ('steps remaining to
    complete your booking.'))
```

Over to you! Think about ways in which you might use the progress through the transactional sequence with your chatbot to contextualise the user experience. Of course, there are more advanced ways of thinking about context tracking. For example, to track the user's previous turn to be able to handle coreference resolution. This could be used, for instance, to understand what concept the user is asking about when they use a pronoun to refer to a person referred to in the previous turn.

2.5 Conversational markers

Context tracking as per the above example also provides you with the resources to use appropriate conversational markers. For example,

```
if booking.halfway
  print('Halfway there!...')
elif booking.steps_remaining == 1
  print('Only one more step, ...')
```

Over to you. Think about places in your chatbot control flow where you can use conversational markers to dynamically improve the user experience. Of course, conversational markers can generally improve the experience of your chatbot, as discussed in the lectures.

2.6 Discoverability

We have learnt that we can use 'discoverability' information to help the user understand what they can ask the chatbot, such as typing 'what can you do?'. For example, by answering 'you can ask me questions, chat with me, or ask me to make a reservation'.

However, we may also provide more contextual discoverability help, depending on what the user is asking for. For example, if they are asking for a hotel for which we do not have a good match, we can provide a list of hotels to the user to choose from.

```
if similarity_index < threshold:
    if 'hotel' in input or 'accommodation' in input:
        print('Sorry, we currently only have the following hotels available: \n'
    + read_txt())
```

You may say that this kind of discoverability support could also be understood as a type of error handling. That is correct! It fulfils both functions - however, here the focus is on enabling the user to 'discover' what they can do next (e.g., which hotels to ask for).

Over to you! In addition to 'generic' discoverability such as you implemented for the checkpoint, what kind of contextually relevant discoverability support could you implement for your chatbot?