Lab 04 Spacy Chatbot

Note: Refer to last lab 03 before continuing with this. This is just a prototype exercise. You can extend its interaction to watsapp, telegram, facebook, web interface, etc. separately using tutorials on the internet.

Note: A full extended implementation of a chatbot is through RASA[1]. They use spacy at its backend.

Task 5: Finding Multiple Patterns automatically in Sentences

Designing a manual pattern matching function is cumbersome as everythin needs to be hard-coded. We can try to automate it using Matchers in spaCy. Consider the following code, which is used to find text that matches the "Subject Verb Object SVO" pattern:

```
from spacy.matcher import Matcher

matcher = Matcher(nlp.vocab)

pattern1 = [{"DEP": "nsubj"}, {"DEP": "ROOT"}, {"DEP": "dobj"}]
matcher.add("SubRootObject", [pattern1])

doc = nlp("The big dog chased everybody")
matches = matcher(doc)

displacy.render(doc, style='dep')                # Not needed, only for illustration

for pattern_id, start, end in matches:

    print("Matching Sentence: ", doc[start:end])
    print("Pattern Type:      ", doc.vocab.strings[pattern_id])

    for token in doc[start:end]:

        print("Dependency:          {}-{}".format(token, token.dep_))
```

Attempt the following questions:

1. What text and dependencies did the above code catch for the sentence "The big dog chased everybody".
2. Change the sentence to "The big dog chased the cat". Does the pattern catch the SVO pattern? If not, add another pattern2 to the matcher. The pattern should be DEP: nsubj, DEP: ROOT, DEP: det, DEP: dobj. When done, update matcher.add("SubRootDetObject", [pattern2])
3. Now design a third pattern for the sentence "The big dog chased the small cat".

If we analyze what we have done, you will notice that we are still in fact doing manual pattern matching. We can design patterns of any length by introducing wild cards. See below:

```
pattern1 = [{"DEP": "nsubj"}, {"DEP": "ROOT"}, {"OP": "*"}, {"DEP": "dobj"}]
```

To have clarity on what else is possible, visit https://spacy.io/api/matcher. Then, attempt the following:

1. Design a pattern to identify a noun at least one time.
2. Design a pattern to identify a noun of length >= 10 characters.
3. Design a pattern to identify vulgar language (Hint: you will need usage of IN, or NOT_IN).

1 Rasa.com

Task 6: Getting Replies

A system now needs to be made to fetch replies from the chatbot. We are not going to go for an open world chatbot which can talk on all topics. But will restrict to a scenario of ordering a pizza. The chatbot should be able to capture a flow in the conversation. We can design it to be greeting, questions on pizza types, placing an order, and closing. Note that this chatbot is not generative, it will simply be going with a flow through matching. We will first start with the greeting. The baseline code for getting a reply can be done through the following test function:

```
def utterance(msg):

    nlp = spacy.load('en_core_web_sm')
    doc = nlp(msg)
    matcher = Matcher(nlp.vocab)

    pattern1 = [{"LEMMA": {"IN": ["salam", "assalam", "hi", "hello"]}}]
    matcher.add("greeting", [pattern1])

    matches = matcher(doc)

    if (len(matches) == 0):
        print('Please rephrase your request. Be as specific as possible!')
        return

    for pattern_id, start, end in matches:

        if doc.vocab.strings[pattern_id] == "greeting":
            print("Welcome to Pizza ordering system")
            return
```

Which can be called from:

```
msg = nlp("Hi")

utterance(msg)
```

An interaction loop can be designed for calling the utterance again and again (See below), however, it is interruptive to prototyping.

```
while True:

    message = input("You: ")

    if message.lower() == "quit":

        break

    else:

        print("Bot:", utterance(nlp(message)))
```

Now attempt the following:

1. Extend the code by adding pattern and matches if a user enters: “I would like to order a pizza”. The bot should ask about which pizza type he/she wants.
2. Extend the code by adding pattern and matches if a user enters: “I would like to complain about an order”.
3. In respone to what pizza type user wants, the user may want to enter “Chief Special Pizza”. Use the .lefts (mentioned in Lab 03) to get the pizza type. Ask about quantity. Use Cardinal as ent type to get the quantity, and place the order. Ask for address, and confirm the user with address.

In the next lab, we will address generative text instead of flow based text, along with prompt engineering.