NLP_Lab#04

December 28, 2023

0.0.1 Task 05

```
[ ]:
```

```
[7]: import spacy
     from spacy.matcher import Matcher
     from spacy import displacy
     nlp = spacy.load('en_core_web_sm')
     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_))
```

```
<IPython.core.display.HTML object>


Matching Sentence:  dog chased everybody
Pattern Type:     SubRootObject
Dependency: dog-nsubj
Dependency: chased-ROOT
Dependency: everybody-dobj
```

What text and dependencies did the above code catch for the sentence “The big dog chased everybody”.

```
Dependency: dog-nsubj
Dependency: chased-ROOT
Dependency: everybody-dobj
```

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])

```
Matching Sentence:  The big dog chased the cat
Pattern Type: SubRootDetObject
Dependency: dog-nsubj
Dependency: chased-ROOT
Dependency: cat-dobj
```

```
Matching Sentence:  The big dog chased the small cat
Pattern Type: SubRootSmallObject
Dependency: dog-nsubj
Dependency: chased-ROOT
Dependency: small-det
Dependency: cat-dobj
```

Design a pattern to identify a noun at least one time.

```
pattern_noun_at_least_one = [{"POS": "NOUN"}]
```

Design a pattern to identify a noun of length >= 10 characters.

```
pattern_long_noun = [{"POS": "NOUN", "LENGTH": {"min": 10}}]
```

Design a pattern to identify vulgar language (Hint: you will need usage of IN, or NOT_IN).

```
vulgar_words = ["Damn", "Bloody", "Shit"]

pattern_vulgar_language = [{"LOWER": {"IN": vulgar_words}}]
```

[10]:
```
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
```

```
msg = nlp("Hi")
utterance(msg)
```

```
Welcome to Pizza ordering system
```

[12]:
```
while True:
    message = input("You: ")
    if message.lower() == "quit":
        break
    else:
        print("Bot:", utterance(nlp(message)))
```

```
You: quit
```

[ ]:
```
```

[ ]:
```
```

[ ]:
```
import spacy
from spacy.matcher import Matcher

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

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

    # Order pizza pattern
    pattern_order_pizza = [{"LEMMA": {"IN": ["order"]}}, {"LOWER": "pizza"}]
    matcher.add("order_pizza", [pattern_order_pizza])

    # Complaint pattern
    pattern_complaint = [{"LEMMA": {"IN": ["complain", "complaint"]}}, {"LOWER":
 ↪ {"IN": ["about", "regarding"]}}, {"LOWER": "order"}]
    matcher.add("complaint", [pattern_complaint])

    matches = matcher(doc)

    if not matches:
        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")
        elif doc.vocab.strings[pattern_id] == "order_pizza":
            handle_order_pizza(doc)
        elif doc.vocab.strings[pattern_id] == "complaint":
            print("I'm sorry to hear that. Please provide more details about␣
↪your complaint.")
            # Additional logic for handling complaints can be added here

def handle_order_pizza(doc):
    pizza_type = get_pizza_type(doc)
    if pizza_type:
        quantity = get_quantity(doc)
        if quantity:
            address = get_address(doc)
            if address:
                print(f"Thank you for your order!\nYou ordered {quantity}␣
↪{pizza_type}(s).\nYour order will be delivered to {address}.")

def get_pizza_type(doc):
    for token in doc:
        if token.text.lower() == "pizza":
            pizza_type = " ".join([left.text for left in token.lefts])
            print(f"Sure, you would like to order {pizza_type} pizza.")
            return pizza_type
    return None

def get_quantity(doc):
    for ent in doc.ents:
        if ent.label_ == "CARDINAL":
            print(f"Great! How many {ent.text} would you like to order?")
            return ent.text
    return None

def get_address(doc):
    address = input("Please provide your delivery address: ")
    print(f"Thank you! Your order will be delivered to {address}.")
    return address

msg = "Hi"
utterance(msg)

while True:
    message = input("You: ")
    if message.lower() == "quit":
        break
    else:
        print("Bot:")
        utterance(message)
```