NLP_Lab#03

December 27, 2023

0.1 Lab # 03

[ ]:

0.1.1 Task 2

[ ]:

[4]:
```
from spacy.tokens.doc import Doc
from spacy.vocab import Vocab
doc = Doc(Vocab(), words = [u'Hello', u'World!'])
print(doc)

print("The Vocab() object belong to class = ", type(doc))

print(doc.vocab)
for token in doc:
    lexeme = doc.vocab[token.text]
    print(lexeme.text)
```

```
Hello World!
The Vocab() object belong to class =  <class 'spacy.tokens.doc.Doc'>
<spacy.vocab.Vocab object at 0x7f9a29f14280>
Hello
World!
```

The Lexeme object in spaCy represents a single entry in the vocabulary. It is part of spaCy's

[ ]:

[5]:
```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp(u'I want to learn spaCy.')
token_text1 = [token.text for token in doc]
token_text2 = [doc[i].text for i in range(len(doc))]
print(token_text1)
print(token_text2)
```

```
['I', 'want', 'to', 'learn', 'spaCy', '.']
['I', 'want', 'to', 'learn', 'spaCy', '.']
```

[ ]:

What is en_core_web_sm?

"en_core_web_sm" is a pre-trained English language model provided by spaCy. It includes word ve

What is the size of en_core_web_sm?

It is small and is of 12 MB

What other variations can be used?

en_core_web_md: A medium-sized English language model with more features and a larger size thar
en_core_web_lg: A large-sized English language model with even more features and a larger size
en_core_web_trf: A transformer-based English language model,

[ ]:

[ ]:

[7]:
```
doc = nlp(u'I want to learn spaCy.')
for i in range(len(doc)):
    print([t for t in doc[i].lefts])
```

```
[]
[I]
[]
[to]
[]
[]
```

[8]:
```
doc = nlp(u'I want to learn spaCy.')
for i in range(len(doc)):
    print([t for t in doc[i].rights])
    print([t for t in doc[i].children])
```

```
[]
[]
[learn, .]
[I, learn, .]
[]
[]
[spaCy]
[to, spaCy]
[]
[]
```

```
[]
[]
```

[9]:
```
from spacy import displacy
displacy.render(doc, style='dep')
```

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

[ ]:
```

```

Draw the left and right dependencies for the sentence: I want to learn spaCy.

[10]:
```
doc = nlp(u'I want to learn spaCy.')
for i in range(len(doc)):
    print([t for t in doc[i].lefts])
    print([t for t in doc[i].rights])
```

```
[]
[]
[I]
[learn, .]
[]
[]
[to]
[spaCy]
[]
[]
[]
[]
```

Draw the children for the sentence: I want to learn spaCy.

[11]:
```
doc = nlp(u'I want to learn spaCy.')
for i in range(len(doc)):
    print([t for t in doc[i].children])
```

```
[]
[I, learn, .]
[]
[to, spaCy]
[]
[]
```

Draw the left and right dependencies for the sentence: I would very much want to eat a hot dinner.

[12]:
```
doc = nlp(u'I would very much want to eat a hot dinner.')
for i in range(len(doc)):
    print([t for t in doc[i].lefts])
```

```
    print([t for t in doc[i].rights])
```

```
[]
[]
[]
[]
[]
[]
[very]
[]
[I, would, much]
[eat, .]
[]
[]
[to]
[dinner]
[]
[]
[]
[]
[a, hot]
[]
[]
[]
```

Present a list of all dependency grammars of your sentences above.

```
`'I'`: No dependencies (root of the sentence).
 `'want'`: No dependencies (root of the sentence).
 `'to'`: Dependent on 'I' and 'want' (lefts: 'I', 'want').
 `'learn'`: Dependent on 'to' (lefts: 'to').
 `'spaCy'`: Dependent on 'learn' (lefts: 'to', 'learn').
 `'.'`: Dependent on 'spaCy' (lefts: 'spaCy').
 `'I'`: No dependencies (root of the sentence).
 `'would'`: No dependencies (root of the sentence).
 `'very'`: No dependencies (root of the sentence).
 `'much'`: Dependent on 'I', 'would', 'very' (lefts: 'I', 'would', 'very').
 `'want'`: Dependent on 'much' (lefts: 'much').
 `'to'`: Dependent on 'want' (lefts: 'want').
 `'eat'`: Dependent on 'to' (lefts: 'to').
 `'a'`: Dependent on 'eat' (lefts: 'to', 'eat').
 `'hot'`: Dependent on 'a' (lefts: 'a').
 `'dinner'`: Dependent on 'hot' (lefts: 'to', 'eat', 'hot').
 `'.'`: Dependent on 'dinner' (lefts: 'to', 'eat', 'hot', 'dinner').
```

0.1.2 Task 03

[1]:
```
import nltk
texts = [u"We are nearing the end of the semester at Peshawar. Final exams of␣
 ↪the Fall 2023 semester will start soon."]
for text in texts:
    sentences = nltk.sent_tokenize(text)
    print(sentences)
```

```
['We are nearing the end of the semester at Peshawar.', 'Final exams of the Fall
2023 semester will start soon.']
```

[2]:
```
import nltk
texts = [u"We are nearing the end of the semester at Peshawar. Final exams of␣
 ↪the Fall 2023 semester will start soon."]
for text in texts:
    sentences = nltk.sent_tokenize(text)
    print(sentences)
    for sentence in sentences:
        words = nltk.word_tokenize(sentence)
        print(words)
```

```
['We are nearing the end of the semester at Peshawar.', 'Final exams of the Fall
2023 semester will start soon.']
['We', 'are', 'nearing', 'the', 'end', 'of', 'the', 'semester', 'at',
'Peshawar', '.']
['Final', 'exams', 'of', 'the', 'Fall', '2023', 'semester', 'will', 'start',
'soon', '.']
```

[3]:
```
import nltk
texts = [u"We are nearing the end of the semester at Peshawar. Final exams of␣
 ↪the Fall 2023 semester will start soon."]
for text in texts:
    sentences = nltk.sent_tokenize(text)
    print(sentences)
    for sentence in sentences:
        words = nltk.word_tokenize(sentence)
        print(words)
        tagged_words = nltk.pos_tag(words)
        print(tagged_words)
```

```
['We are nearing the end of the semester at Peshawar.', 'Final exams of the Fall
2023 semester will start soon.']
['We', 'are', 'nearing', 'the', 'end', 'of', 'the', 'semester', 'at',
'Peshawar', '.']
[('We', 'PRP'), ('are', 'VBP'), ('nearing', 'VBG'), ('the', 'DT'), ('end',
'NN'), ('of', 'IN'), ('the', 'DT'), ('semester', 'NN'), ('at', 'IN'),
('Peshawar', 'NNP'), ('.', '.')]
```

```
['Final', 'exams', 'of', 'the', 'Fall', '2023', 'semester', 'will', 'start',
'soon', '.']
[('Final', 'JJ'), ('exams', 'NN'), ('of', 'IN'), ('the', 'DT'), ('Fall', 'NN'),
('2023', 'CD'), ('semester', 'NN'), ('will', 'MD'), ('start', 'VB'), ('soon',
'RB'), ('.', '.')]
```

[4]:
```
import nltk
texts = [u"We are nearing the end of the semester at Peshawar. Final exams of␣
 ↪the Fall 2023 semester will start soon."]
for text in texts:
    sentences = nltk.sent_tokenize(text)
    print(sentences)
    for sentence in sentences:
        words = nltk.word_tokenize(sentence)
        print(words)
        tagged_words = nltk.pos_tag(words)
        print(tagged_words)
        ne_tagged_words = nltk.ne_chunk(tagged_words)
        print(ne_tagged_words)
```

```
['We are nearing the end of the semester at Peshawar.', 'Final exams of the Fall
2023 semester will start soon.']
['We', 'are', 'nearing', 'the', 'end', 'of', 'the', 'semester', 'at',
'Peshawar', '.']
[('We', 'PRP'), ('are', 'VBP'), ('nearing', 'VBG'), ('the', 'DT'), ('end',
'NN'), ('of', 'IN'), ('the', 'DT'), ('semester', 'NN'), ('at', 'IN'),
('Peshawar', 'NNP'), ('.', '.')]
(S
  We/PRP
  are/VBP
  nearing/VBG
  the/DT
  end/NN
  of/IN
  the/DT
  semester/NN
  at/IN
  (ORGANIZATION Peshawar/NNP)
  ./.)
['Final', 'exams', 'of', 'the', 'Fall', '2023', 'semester', 'will', 'start',
'soon', '.']
[('Final', 'JJ'), ('exams', 'NN'), ('of', 'IN'), ('the', 'DT'), ('Fall', 'NN'),
('2023', 'CD'), ('semester', 'NN'), ('will', 'MD'), ('start', 'VB'), ('soon',
'RB'), ('.', '.')]
(S
  Final/JJ
  exams/NN
  of/IN
```

```
the/DT
Fall/NN
2023/CD
semester/NN
will/MD
start/VB
soon/RB
./.)
```

[7]:
```
import spacy
from spacy.vocab import Vocab
nlp = spacy.load('en_core_web_sm')
doc = nlp(u'We are nearing the end of the semester at Peshawar. Final exams of␣
 ↪the Fall 2023 semester will start soon.')
displacy.render(doc, style='ent')
```

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

[9]:
```
for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
the end of the semester DATE
Peshawar GPE
```

[ ]:
```

```

[1]:
```
import spacy
nlp = spacy.load('en_core_web_sm')
nlp.pipe_names
```

[1]:
```
['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
```

[ ]:
```

```

What is the default pipeline structure of spaCy?

The default pipeline for the English language model ('en_core_web_sm' or 'en_core_web_lg') typi

Tokenizer (tok2vec): The tokenizer breaks the input text into individual words or tokens. The t

Part-of-Speech Tagger (tagger): The part-of-speech tagger assigns grammatical parts-of-speech t

Dependency Parser (parser): The dependency parser analyzes the grammatical structure of the sen

Named Entity Recognizer (ner): The named entity recognizer identifies entities such as persons

Text Categorizer (textcat): This component assigns categories or labels to the entire document

Entity Linker (entity_linker): This component associates recognized entities with knowledge bas

Similarity (text_similarity): This component computes the similarity between two documents. It

[ ]:

0.1.3 Task 4

[3]:
```
import spacy
from spacy import displacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'I want to learn spaCy.')
displacy.render(doc, style='dep', options={'distance': 90})
```

<IPython.core.display.HTML object>

[4]:
```
doc = nlp(u'How do I learn spaCy.')
displacy.render(doc, style='dep')
```

<IPython.core.display.HTML object>

[8]:
```
import spacy
nlp = spacy.load('en_core_web_sm')
def dep_pattern(doc):
    for i in range(len(doc)-1):
        print(doc[i].dep_)
        if doc[i].dep_ == 'nsubj' and doc[i+1].dep_ == 'ROOT' and doc[i+2].dep_␣
 ↪== 'acomp':
            return True
            return False
doc = nlp(u'How do I learn spaCy.')
if dep_pattern(doc):
    print('Found')
else:
    print('Not found')
```

```
advmod
aux
nsubj
Found
```

[ ]: