Natural Language Processing Basics

June 12, 2021

Table of Contents

1 spaCy basics

- For more info, visit: https://spacy.io

1.1 Installation and setup

- For more info, visit: https://spacy.io/usage

1.1.1 From the command line or terminal

```
conda install -c conda-forge spacy
```

or

```
pip install -U spacy
```

1.1.2 Alternatively, create a virtual environment

```
conda create -n spacyenv python spacy
```

1.1.3 Next, download the specific model of language

```
python -m spacy download en_core_web_sm
```

1.2 Working with spaCy in Python

[1]:
```python
# Import spaCy and load the language library
import spacy
```

[2]:
```python
nlp = spacy.load('en_core_web_sm')
```

[3]:
```python
# Create a Doc object
doc_1 = nlp(u'Tesla is looking at buying a U.S. startup for $6 million.')
```

[4]:
```python
row_format = "{:>10}" * 2
# Print each token separately
for token in doc_1:
    print(row_format.format(token.text, token.pos))
```

```
     Tesla        96
        is        87
   looking       100
        at        85
    buying       100
         a        90
      U.S.        96
   startup        92
       for        85
         $        99
         6        93
   million        93
         .        97
```

[5]:
```python
row_format = "{:>10}" * 3
for token in doc_1:
    print(row_format.format(token.text, token.pos_, token.dep_))
```

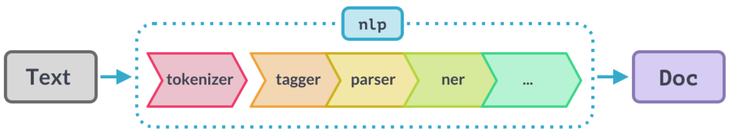
```
     Tesla     PROPN     nsubj
        is       AUX       aux
   looking      VERB      ROOT
        at       ADP      prep
    buying      VERB     pcomp
         a       DET       det
      U.S.     PROPN  compound
   startup      NOUN      dobj
```

```
       for       ADP      prep
         $       SYM  quantmod
         6       NUM  compound
   million       NUM      pobj
         .     PUNCT     punct
```

1.3 spaCy objects

1.3.1 Pipeline

- Image source: https://spacy.io/usage/processing-pipelines



[6]:
```python
nlp.pipeline
```

[6]:
```
[('tok2vec', <spacy.pipeline.tok2vec.Tok2Vec at 0x7f9b379e72f0>),
 ('tagger', <spacy.pipeline.tagger.Tagger at 0x7f9b379f7950>),
 ('parser', <spacy.pipeline.dep_parser.DependencyParser at 0x7f9b379bd210>),
 ('ner', <spacy.pipeline.ner.EntityRecognizer at 0x7f9b379bd360>),
 ('attribute_ruler',
  <spacy.pipeline.attributeruler.AttributeRuler at 0x7f9b372cad70>),
 ('lemmatizer',
  <spacy.lang.en.lemmatizer.EnglishLemmatizer at 0x7f9b37a12230>)]
```

[7]:
```python
nlp.pipe_names
```

[7]:
```
['tok2vec', 'tagger', 'parser', 'ner', 'attribute_ruler', 'lemmatizer']
```

1.3.2 Tokenization

[8]:
```python
doc_2 = nlp(u"Tesla isn't looking into startups anymore.")

row_format = "{:>10}" * 3
for token in doc_2:
    print(row_format.format(token.text, token.pos_, token.dep_))
```

```
     Tesla     PROPN     nsubj
        is       AUX       aux
       n't      PART       neg
   looking      VERB      ROOT
      into       ADP      prep
  startups      NOUN      pobj
   anymore       ADV    advmod
         .     PUNCT     punct
```

[9]:
```
doc_2 = nlp(u"Tesla isn't   looking into startups anymore.")

row_format = "{:>10}" * 3
for token in doc_2:
    print(row_format.format(token.text, token.pos_, token.dep_))
```

```
    Tesla     PROPN     nsubj
       is       AUX       aux
      n't      PART       neg
              SPACE     nsubj
  looking      VERB      ROOT
     into       ADP      prep
 startups      NOUN      pobj
  anymore       ADV    advmod
        .     PUNCT     punct
```

[10]:
```
doc_2
```

[10]:
```
Tesla isn't   looking into startups anymore.
```

[11]:
```
type(doc_2)
```

[11]:
```
spacy.tokens.doc.Doc
```

[12]:
```
doc_2[0]
```

[12]:
```
Tesla
```

[13]:
```
doc_2[0].text
```

[13]:
```
'Tesla'
```

1.3.3 Part-of-speech tagging (POS)

- For more info, visit: https://spacy.io/usage/linguistic-features#pos-tagging

[14]:
```
doc_2[0].pos_
```

[14]:
```
'PROPN'
```

[15]:
```
spacy.explain('PROPN')
```

[15]:
```
'proper noun'
```

1.3.4 Dependencies

- For more info, visit: https://spacy.io/usage/linguistic-features#dependency-parse
- Here, there is a good explanation of typed dependencies.

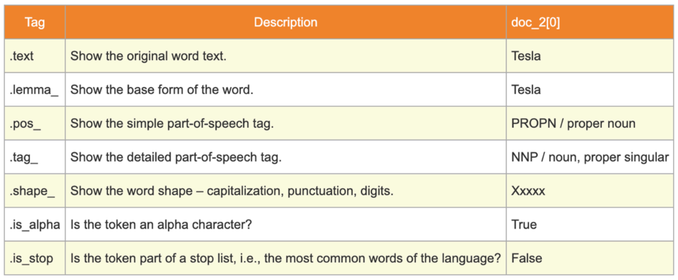
[16]:
```
doc_2[0].dep_
```

[16]:
```
'nsubj'
```

[17]:
```
spacy.explain('nsubj')
```

[17]:
```
'nominal subject'
```

1.3.5 Additional token attributes



Tag	Description	doc_2[0]
.text	Show the original word text.	Tesla
.lemma_	Show the base form of the word.	Tesla
.pos_	Show the simple part-of-speech tag.	PROPN / proper noun
.tag_	Show the detailed part-of-speech tag.	NNP / noun, proper singular
.shape_	Show the word shape – capitalization, punctuation, digits.	Xxxxx
.is_alpha	Is the token an alpha character?	True
.is_stop	Is the token part of a stop list, i.e., the most common words of the language?	False

[18]:
```
# Lemmas (the base form of the word)
print(doc_2[0].text)
print(doc_2[0].lemma_)
```

```
Tesla
Tesla
```

[19]:
```
print(doc_2[4].text)
print(doc_2[4].lemma_)
```

```
looking
look
```

[28]:
```
# Simple parts-of-speech & detailed tags
print(doc_2[0].pos_ + ' / ' + spacy.explain(doc_2[0].pos_))
print(doc_2[0].tag_ + ' / ' + spacy.explain(doc_2[0].tag_))
```

```
PROPN / proper noun
NNP / noun, proper singular
```

[27]:
```
print(doc_2[4].pos_ + ' / ' + spacy.explain(doc_2[4].pos_))
print(doc_2[4].tag_ + '  / ' + spacy.explain(doc_2[4].tag_))
```

```
VERB / verb
VBG  / verb, gerund or present participle
```

[31]:
```
# Word shapes
print(doc_2[0].text + ': ' + doc_2[0].shape_)
print(doc_1[6].text + ' : ' + doc_1[5].shape_)
```

```
Tesla: Xxxxx
U.S. : x
```

[33]:
```
# Boolean values
print(doc_2[0].is_alpha)
print(doc_2[0].is_stop)
```

```
True
False
```

1.3.6 Spans

[ ]:
```
doc3 = nlp(
    u'Although commonly attributed to John Lennon from his song "Beautiful␣
 ↪Boy", \
    the phrase "Life is what happens to us while we are making other plans" \
    was written by cartoonist Allen Saunders and published in Reader\'s Digest \
    in 1957 when Lennon was 17.'
)
```

[ ]:
```
life_quote = doc3[16:30]
print(life_quote)
```

[ ]:
```
type(life_quote)
```

In upcoming lectures we'll see how to create Span objects using `Span()`. This will allow us to assign additional information to the Span.

1.4 Sentences

Certain tokens inside a Doc object may also receive a "start of sentence" tag. While this doesn't immediately build a list of sentences, these tags enable the generation of sentence segments through `Doc.sents`. Later we'll write our own segmentation rules.

[ ]:
```
doc4 = nlp(
    u'This is the first sentence. This is another sentence. This is the last␣
 ↪sentence.'
)
```

[ ]:
```
for sent in doc4.sents:
    print(sent)
```

[ ]:
```
doc4[6].is_sent_start
```

1.5 Next up: Tokenization