NLP_spacy

Tuesday, February 1, 2022 11:47 PM

Chapter 1: Finding words, phrases, names and concepts

1.Introduction to spaCy

```
# Import spaCy
import spacy

# Create a blank English nlp object
nlp = spacy.blank("en")
```

The Doc object

```
# Created by processing a string of text with the nlp object
doc = nlp("Hello world!")

# Iterate over tokens in a Doc
for token in doc:
    print(token.text)
```

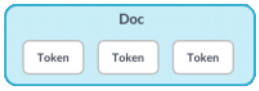
```
Hello
world
!
```

The Token object

```
doc = nlp("Hello world!")

# Index into the Doc to get a single Token
token = doc[1]

# Get the token text via the .text attribute
print(token.text)
```

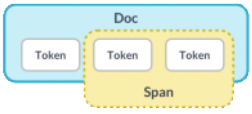



```
world
```

The Span object
It is a view and does not contain any data itself.

```
doc = nlp("Hello world!")
# A slice from the Doc is a Span object
span = doc[1:3]

# Get the span text via the .text attribute
print(span.text)
```




```
world!
```

Lexical Attributes(relating to any entry in vocabulary)

```
doc = nlp("It costs $5.")
print("Index:   ", [token.i for token in doc]) #gives the index
of words

print("Text:    ", [token.text for token in doc]) #give the
words

print("is_alpha:", [token.is_alpha for token in doc]) #checks
if alphabet

print("is_punct:", [token.is_punct for token in doc]) #checks
if punctuation

print("like_num:", [token.like_num for token in doc]) #checks
if numeric
```

```
Index:    [0, 1, 2, 3, 4]
Text:     ['It', 'costs', '$', '5', '.']
is_alpha: [True, True, False, False, False]
is_punct: [False, False, False, False, True]
like_num: [False, False, False, True, False]
```

2.Getting started

```
# Import spaCy
import spacy

# Create the English nlp object
nlp = spacy.blank("en")

# Process a text
doc = nlp("This is a sentence.")

# Print the document text
print(doc.text)
```

```
This is a sentence.
```

3.Documents,spans and tokens

```
# Import spaCy and create the English nlp object
import spacy
nlp = spacy.blank("en")

# Process the text
```

```
I
```

```
doc = nlp("I like tree kangaroos and narwhals.")

# Select the first token
first_token = doc[0]

# Print the first token's text
print(first_token.text)
```

Spans

```
# Import spaCy and create the English nlp object
import spacy
nlp = spacy.blank("en")

# Process the text
doc = nlp("I like tree kangaroos and narwhals.")

# A slice of the Doc for "tree kangaroos"
tree_kangaroos = doc[2:4]
print(tree_kangaroos.text)

# A slice of the Doc for "tree kangaroos and narwhals" (without the ".")
tree_kangaroos_and_narwhals = doc[2:6]
print(tree_kangaroos_and_narwhals.text)
```

```
tree kangaroos
tree kangaroos and narwhals
```

4.Lexical attributes

```
import spacy
nlp = spacy.blank("en")

# Process the text
doc = nlp(
    "In 1990, more than 60% of people in East Asia were in extreme poverty. "
    "Now less than 4% are.")

# Iterate over the tokens in the doc
for token in doc:
    # Check if the token resembles a number
    if token.like_num:
        # Get the next token in the document
        next_token = doc[token.i + 1]
        # Check if the next token's text equals "%"
        if next_token.text == "%":
            print("Percentage found:", token.text)
```

```
Percentage found: 60
Percentage found: 4
```

5.Trained pipelines

- Models that enable spaCy to predict linguistic attributes *in context*
  - Part-of-speech tags
  - Syntactic dependencies
  - Named entities
- Trained on labeled example texts
- Can be updated with more examples to fine-tune predictions

Predicting Part-of-speech Tags

```
import spacy

# Load the small English pipeline
nlp = spacy.load("en_core_web_sm")

# Process a text
doc = nlp("She ate the pizza")

# Iterate over the tokens
for token in doc:
    # Print the text and the predicted part-of-speech tag
    print(token.text, token.pos_)
```

```
She PRON
ate VERB
the DET
pizza NOUN
```

Predicting Syntactic Dependencies

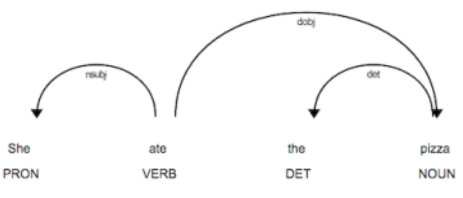
```
for token in doc:
    print(token.text, token.pos_, token.dep_, token.head.text)
```

```
She PRON nsubj ate
ate VERB ROOT ate
the DET det pizza
pizza NOUN dobj ate
```

Dependency label scheme

Label	Description	Example
nsubj	nominal subject	She
dobj	direct object	pizza
det	determiner (article)	the

```
Visualization of the dependency graph for 'She ate the pizza'
```



Predicting Named Entities

```
# Process a text
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
# Iterate over the predicted entities
for ent in doc.ents:
    # Print the entity text and its label
    print(ent.text, ent.label_)
```

```
Visualization of the named entities in 'Apple is looking at buying U.K. startup for $1 billion'
```


```
Apple ORG
U.K. GPE
$1 billion MONEY
```

```
Tip: the spacy.explain method
GPE:geopolitical entity.

Get quick definitions of the most common tags and labels.

spacy.explain("GPE")

spacy.explain("NNP")

spacy.explain("dobj")
```

```
'Countries, cities, states'


'noun, proper singular'


'direct object'
```

6.Training data and binary weights:

Trained pipelines allow you to generalize based on a set of **training examples**. Once they're trained, they use **binary weights** to make predictions. That's why it's not necessary to ship them with their training data.

7.Loading Pipeline

```
import spacy

# Load the "en_core_web_sm" pipeline
nlp = spacy.load("en_core_web_sm")
text = "It's official: Apple is the first U.S. public company to reach a $1 trillion market value"

# Process the text
doc = nlp(text)

# Print the document text
print(doc.text)
```

```
It's official: Apple is the first U.S. public company to reach a $1 trillion market value
```

8.Predicting Linguistic Annotations

You'll now get to try one of spaCy's trained pipeline packages and see its predictions in action. Feel free to try it out on your own text! To find out what a tag or label means, you can call spacy.explain in the loop. For example: spacy.explain("PROPN") or spacy.explain("GPE").

Part 1

Process the text with the nlp *object* and create a doc. For each token, print the token text, the token's **.pos_** (part-of-speech tag) and the token's **.dep_** (dependency label).

In spacy attributes that are turn in string end with underscore("_") other are integers.

```
import spacy

nlp = spacy.load("en_core_web_sm")

text = "It's official: Apple is the first U.S. public company to reach a $1 trillion market value"

# Process the text
doc = nlp(text)

for token in doc:
    # Get the token text, part-of-speech tag and dependency label
    token_text = token.text
    token_pos = token.pos_
    token_dep = token.dep_
    # This is for formatting only
    print(f"{token_text:<12}{token_pos:<10}{token_dep:<10}")
```

```
It          PRON      dep
's          INTJ      intj
official    ADJ       amod
:           PUNCT     punct
Apple       PROPN     nsubj
is          AUX       ROOT
the         DET       det
first       ADJ       amod
U.S.        PROPN     nmod
public      ADJ       amod
company     NOUN      attr
to          PART      aux
reach       VERB      relcl
a           DET       det
$           SYM       quantmod
1           NUM       compound
trillion    NUM       nummod
market      NOUN      compound
value       NOUN      dobj
```

Part 2

Process the text and create a doc object.
Iterate over the doc.ents and print the entity text and label_ attribute.

```
import spacy
```

```
Apple ORG
first ORDINAL
U.S. GPE
$1 trillion MONEY
```

```
nlp = spacy.load("en_core_web_sm")

text = "It’s official: Apple is the first U.S. public company
to reach a $1 trillion market value"

# Process the text
doc = nlp(text)

# Iterate over the predicted entities
for ent in doc.ents:
    # Print the entity text and its label
    print(ent.text, ent.label_)
```

9.Predicting named entities in context

Models are statistical and not always right. Whether their predictions are correct depends on the training data and the text you’re processing. Let’s take a look at an example.

- Process the text with the nlp *object*.
- Iterate over the entities and print the entity text and label.
- Looks like the model didn’t predict “iPhone X”. Create a span for those tokens manually.

```
import spacy
nlp = spacy.load("en_core_web_sm")
text = "Upcoming iPhone X release date leaked as Apple reveals
pre-orders"

# Process the text
doc = nlp(text)

# Iterate over the entities
for ent in doc.ents:
    # Print the entity text and label
    print(ent.text, ent.label_)

# Get the span for "iPhone X"
iphone_x = doc[1:3]

# Print the span text
print("Missing entity:", iphone_x.text)
```

```
Apple ORG
Missing entity: iPhone X
```

10.Rule-based matching

Why not just regular expressions?
Match on Doc objects, not just strings
Match on tokens and token attributes
Use a model's predictions
Example: "duck" (verb) vs. "duck" (noun)

Match patterns

Lists of dictionaries, one per token

```
Match exact token texts
[{"TEXT": "iPhone"}, {"TEXT": "X"}]

Match lexical attributes
[{"LOWER": "iphone"}, {"LOWER": "x"}]

Match any token attributes
[{"LEMMA": "buy"}, {"POS": "NOUN"}]
```

Using the Matcher (1)

- match_id: hash value of the pattern name
- start: start index of matched span
- end: end index of matched span

```
import spacy
# Import the Matcher
from spacy.matcher import Matcher

# Load a pipeline and create the nlp object
nlp = spacy.load("en_core_web_sm")

# Initialize the matcher with the shared vocab
matcher = Matcher(nlp.vocab)

# Add the pattern to the matcher
pattern = [{"TEXT": "iPhone"}, {"TEXT": "X"}]
matcher.add("IPHONE_PATTERN", [pattern])
```

```
iPhone X
```

```
# Process some text
doc = nlp("Upcoming iPhone X release date leaked")

# Call the matcher on the doc, it'll return a list of tuples.
matches = matcher(doc)

# Iterate over the matches
for match_id, start, end in matches:
    # Get the matched span
    matched_span = doc[start:end]
    print(matched_span.text)
```

Matching lexical(words) attributes

```
pattern = [
    {"IS_DIGIT": True},
    {"LOWER": "fifa"},
    {"LOWER": "world"},
    {"LOWER": "cup"},
    {"IS_PUNCT": True}
]
matcher.add("FIFA_PATTERN", [pattern])
doc = nlp("2018 FIFA World Cup: France won!")
```

```
2018 FIFA World Cup:
```

Matching other token attributes

```
pattern = [
    {"LEMMA": "love", "POS": "VERB"},
    {"POS": "NOUN"}
]
matcher.add("IPHONE_PATTERN", [pattern])
doc = nlp("I loved dogs but now I love cats more.")
```

```
loved dogs
love cats
```

Using operators and quantifiers (1)

```
pattern = [
    {"LEMMA": "buy"},
    {"POS": "DET", "OP": "?"},  # optional: match 0 or 1 times
    {"POS": "NOUN"}
]
doc = nlp("I bought a smartphone. Now I'm buying apps.")
```

```
bought a smartphone
buying apps
```

Using operators and quantifiers (2)

```
"OP":operators

Example Description
{"OP": "!"} Negation: match 0 times
{"OP": "?"} Optional: match 0 or 1 times
{"OP": "+"} Match 1 or more times
{"OP": "*"} Match 0 or more times
```

11.Using the Matcher

```
• Import the Matcher from spacy.matcher.
• Initialize it with the nlp object's shared vocab.
• Create a pattern that matches the "TEXT" values of two
tokens: "iPhone" and "X".
• Use the matcher.add method to add the pattern to the matcher.
• Call the matcher on the doc and store the result in the
variable matches.
• Iterate over the matches and get the matched span from
the start to
the end index.

import spacy

# Import the Matcher
from spacy.matcher import Matcher

nlp = spacy.load("en_core_web_sm")
doc = nlp("Upcoming iPhone X release date leaked as Apple
reveals pre-orders")

# Initialize the Matcher with the shared vocabulary
matcher = Matcher(nlp.vocab)

# Create a pattern matching two tokens: "iPhone" and "X"
pattern = [{"TEXT": "iPhone"}, {"TEXT": "X"}]

# Add the pattern to the matcher
matcher.add("IPHONE_X_PATTERN", [pattern])

# Use the matcher on the doc
matches = matcher(doc)
print("Matches:", [doc[start:end].text for match_id, start, end
```

```
Matches: ['iPhone X']
```

```
in matches])
```

12.Writing match patterns

Part 1

- Write **one** pattern that only matches mentions of the *full* iOS versions: “iOS 7”, “iOS 11” and “iOS 10”.

```
Total matches found: 3
Match found: iOS 7
Match found: iOS 11
Match found: iOS 10
```

Part 2

- Write **one** pattern that only matches forms of “download” (tokens with the lemma “download”), followed by a token with the part-of-speech tag "PROPN" (proper noun).

```
import spacy
from spacy.matcher import Matcher

nlp = spacy.load("en_core_web_sm")
matcher = Matcher(nlp.vocab)

doc = nlp(
    "i downloaded Fortnite on my laptop and can't open the game
at all. Help? "
    "so when I was downloading Minecraft, I got the Windows
version where it "
    "is the '.zip' folder and I used the default program to
unpack it... do "
    "I also need to download Winzip?"
)

# Write a pattern that matches a form of "download" plus proper
noun
pattern = [{"LEMMA": "download"}, {"POS": "PROPN"}]

# Add the pattern to the matcher and apply the matcher to the
doc
matcher.add("DOWNLOAD_THINGS_PATTERN", [pattern])
matches = matcher(doc)
print("Total matches found:", len(matches))

# Iterate over the matches and print the span text
for match_id, start, end in matches:
    print("Match found:", doc[start:end].text)
```

```
Total matches found: 3
Match found: downloaded Fortnite
Match found: downloading Minecraft
Match found: download Winzip
```

Part 3

- Write **one** pattern that matches adjectives ("ADJ") followed by one or two "NOUN"s (one noun and one optional noun).

```
import spacy
from spacy.matcher import Matcher
nlp = spacy.load("en_core_web_sm")
matcher = Matcher(nlp.vocab)
doc = nlp(
    "Features of the app include a beautiful design, smart
search, automatic "
    "labels and optional voice responses."
)

# Write a pattern for adjective plus one or two nouns
pattern = [{"POS": "ADJ"}, {"POS": "NOUN"}, {"POS": "NOUN",
"OP": "?"}]

# Add the pattern to the matcher and apply the matcher to the
doc
matcher.add("ADJ_NOUN_PATTERN", [pattern])
matches = matcher(doc)
print("Total matches found:", len(matches))

# Iterate over the matches and print the span text
for match_id, start, end in matches:
    print("Match found:", doc[start:end].text)
```

```
Total matches found: 5
Match found: beautiful design
Match found: smart search
Match found: automatic labels
Match found: optional voice
Match found: optional voice responses
```

Chapter 2: Large-scale data analysis with spaCy

In this chapter, you'll use your new skills to extract specific information from large volumes of text. You'll learn how to make the most of spaCy's data structures, and how to effectively combine statistical and rule-based approaches for text analysis.
