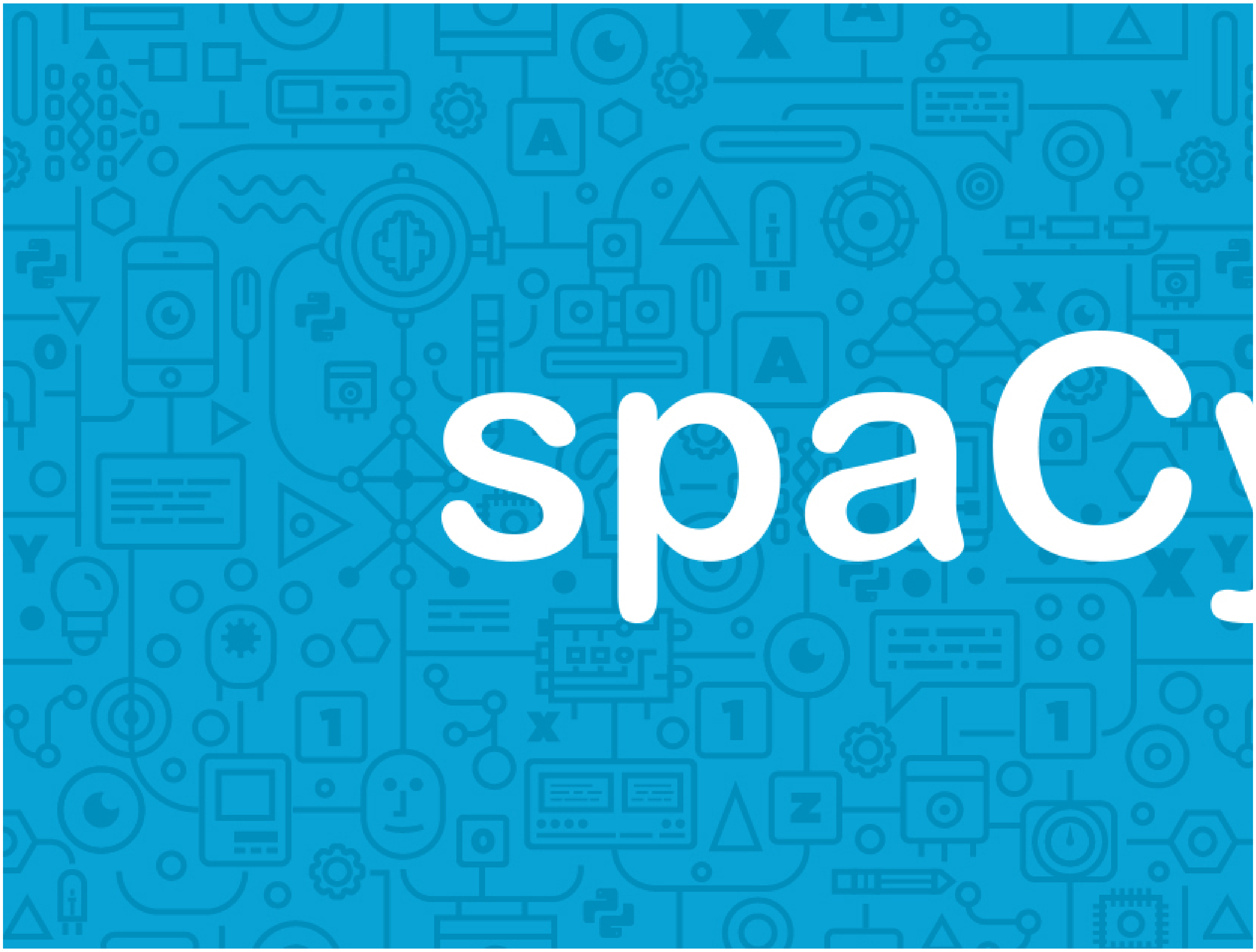
[Tutorial] Text Processing with spaCy



Picture from spacy.io (https://spacy.io/)

Install spaCy with pip :

```
pip install -U spacy
```

Download statistical models :

- English

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

---

1. Introduction to spaCy

1.1. What is spaCy?

spaCy is a relatively new package for **performant and advanced Natural Language Processing tasks**. It is a free, open-source library developed for Python.

Differently from NLTK, which was conceived for **teaching and research** purposes, spaCy was designed with the **applied data science concepts** in mind.

spaCy spares users time and work : it does not **ask to choose between multiple algorithms** that deliver equivalent functionality for each specific NLP task. Keeping the menu small lets spaCy deliver generally better performance and developer experience.

Also, spaCy is extremely **efficient and fast**.

1.2. A summary of spaCy features

NAME	DESCRIPTION
Tokenization	Segmenting text into words, punctuations marks etc.
Part-of-speech (POS) **Tagging**	Assigning word types to tokens, like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Lemmatization	Assigning the base forms of words. For example, the lemma of "was" is "be", and the lemma of "rats" is "rat".
Sentence Boundary Detection (SBD)	Finding and segmenting individual sentences.
Named Entity Recognition (NER)	Labelling named "real-world" objects, like persons, companies or locations.
Similarity	Comparing words, text spans and documents and how similar they are to each other.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Rule-based Matching	Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions.
Training	Updating and improving a statistical model's predictions.
Serialization	Saving objects to files or byte strings.

2. Getting started with spaCy - Preprocessing

2.1. Preprocessing English text

```
# Import spacy and load the English version of the model
import spacy

# Create the nlp object -- this object will parse the text and preprocess it automatically
nlp = spacy.load("en_core_web_md")

# Toy sentence
sent = "I bought 3 beers for 2€ each!"
```

```
# Document created by preprocessing the text with the nlp object
doc = nlp(sent)

# Iterate over the tokens in the doc and print them
for token in doc:
    print(token.text)
```

```
I
bought
3
beers
for
2
€
each
!
```

```
# We can also print a "slice" of the text -- we call it a text span
span = doc[1:4]
print(span.text)
```

```
bought 3 beers
```

The nlp object does not only parse the tokens in the text, but it also stores many charateristics and attributes for each token, including:

- Whether it's an alphabetic token
- Whether it's a number
- Whether it's a punctuation
- Token's lemma
- And others (we will see them later in the course)

> 🔦 Hint: spaCy **does not provide stemming**, as usually lemmatization generalizes better.

➡️ Let's see how spaCy outputs these features:

```
# Printing the index of each token in the text
print('Index:   ', [token.i for token in doc])

# Printing the raw text representation of each token
print('Text:    ', [token.text for token in doc])

# Pritting the token's lemma
print('Lemma:   ', [token.lemma_ for token in doc])  # -PRON- is used as the lemma for all personal
 pronouns

# Whether each token is alphabetic or not
print('is_alpha:', [token.is_alpha for token in doc])

# Whether each token is a number or not
print('like_num:', [token.like_num for token in doc])
```

```
# Whether each token is a punctuation or not
print('is_punct:', [token.is_punct for token in doc])
```

```
Index:    [0, 1, 2, 3, 4, 5, 6, 7, 8]
Text:     ['I', 'bought', '3', 'beers', 'for', '2', '€', 'each', '!']
Lemma:    ['-PRON-', 'buy', '3', 'beer', 'for', '2', '€', 'each', '!']
is_alpha: [True, True, False, True, True, False, False, True, False]
like_num: [False, False, True, False, False, True, False, False, False]
is_punct: [False, False, False, False, False, False, False, False, True]
```

2.2. Other preprocessing functionalities

2.2.1 Preprocessing several documents with the `.pipe` method

```
texts = ["I love cats", "Alice loves dogs", "Bob doesn't like animals"]
docs = nlp.pipe(texts)

for i, doc in enumerate(docs):
    print("Document {}:".format(i), [token.text for token in doc])
```

```
Document 0: ['I', 'love', 'cats']
Document 1: ['Alice', 'loves', 'dogs']
Document 2: ['Bob', 'does', "n't", 'like', 'animals']
```

2.2.2 Managing with stop words

spaCy already comes with a **predefined** set of stop words:

```
print([w for w in spacy.lang.en.stop_words.STOP_WORDS][:10])
```

```
['become', 'beyond', 'nothing', 'now', 'too', 'no', 'each', 'few', 'after', 'meanwhile']
```

However, depending on the use-case we might need to **remove / add some stop words**.

```
sent = "My glass is empty"
doc = nlp(sent)
print(doc[1].text, ':', doc[1].is_stop)
print(doc[3].text, ':', doc[3].is_stop)
```

```
glass : False
empty : True
```

```
nlp.vocab["empty"].is_stop = False
nlp.vocab["glass"].is_stop = True
```

```
print(doc[1].text, ':', doc[1].is_stop)
print(doc[3].text, ':', doc[3].is_stop)
```

```
glass : True
empty : False
```

---

3. Linguistic annotations

The spaCy library also provides **powerful statistical models** that can **extract linguistic annotations** such as:

- Part of Speech Tagging (POS tagging)
- Dependency Parsing
- Named Entity Recognition (NER)

Let's understand what these linguistic features mean and how to extract them with spaCy.

3.1. Part of Speech Tagging (POS)

Part of Speech (POS) is a category of words with **similar grammatical properties**. Examples: noun, verb, adjective, adverb.

Part of Speech tagging, also called grammatical tagging, is the process of marking up a word in a text as **corresponding to a particular part of speech based on both its definition and its context**.

POS Tagging with spaCy:

```
doc = nlp('Jonas is going to a party with two friends')

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

```
Jonas PROPN NNP
is AUX VBZ
going VERB VBG
to ADP IN
a DET DT
party NOUN NN
with ADP IN
two NUM CD
friends NOUN NNS
```

The attribute `token.pos_` corresponds to the simple part-of-speech tag and the attribute `token.tag_` corresponds to the detailed part-of-speech tag.

🔦 In order to get the definitions of each tag we can use the method `.explain()`:

```
print('PROPN:', spacy.explain('PROPN'))
print('NNP:', spacy.explain('NNP'))
```

```
PROPN: proper noun
NNP: noun, proper singular
```

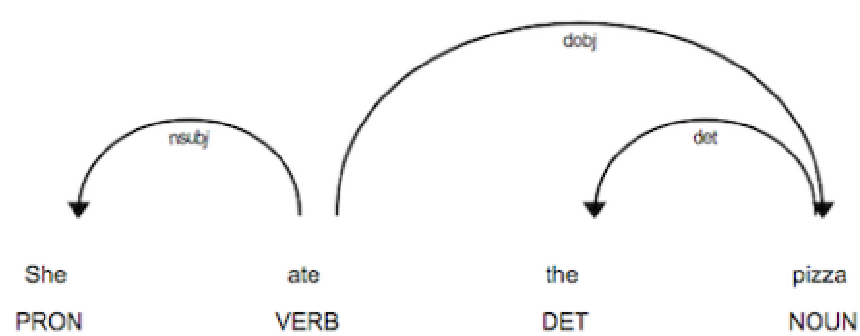
```
for token in doc:
    print(token.text + ' - ' +
          token.pos_ + ' (' + spacy.explain(token.pos_) + ') - ' +
          token.tag_ + ' (' + spacy.explain(token.tag_) + ')')
```

```
Jonas - PROPN (proper noun) - NNP (noun, proper singular)
is - AUX (auxiliary) - VBZ (verb, 3rd person singular present)
going - VERB (verb) - VBG (verb, gerund or present participle)
to - ADP (adposition) - IN (conjunction, subordinating or preposition)
a - DET (determiner) - DT (determiner)
party - NOUN (noun) - NN (noun, singular or mass)
with - ADP (adposition) - IN (conjunction, subordinating or preposition)
two - NUM (numeral) - CD (cardinal number)
friends - NOUN (noun) - NNS (noun, plural)
```

3.2. Dependency Parsing

Dependency Parsing is the task of extracting **sintactical dependencies between the words in a sentence**. It represents the grammatical structure of the sentence and defines the **relationships between "head" words and words**, which modify those heads.

Dependency label scheme



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

Dependency parsing with spaCy:

```
doc = nlp("Marie's sister liked my cats")

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

```
Marie poss sister
's case Marie
sister nsubj liked
liked ROOT liked
my poss cats
cats dobj liked
```

Visualizing the dependencies:

```
from spacy import displacy

displacy.render(doc, style='dep')
```

Marie PROPN 's PART sister NOUN liked VERB my DET cats NOUN

poss case nsubj poss dobj

🔦 We can also use the method `.explain()` to get the definition of each dependency:

```
print('nsubj:', spacy.explain('nsubj'))
print('poss:', spacy.explain('poss'))
```

```
nsubj: nominal subject
poss: possession modifier
```

3.3. Named Entity Recognition (NER)

Named entity recognition (NER) is the task of **tagging entities in text with pre-defined categories** such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

It is a **really usefull** sub-task of Information Extraction and can help us answer questions such as:

- Which companies were mentioned in the news article?
- What are the financial amounts mentioned in a company's annual report?
- Which products were mentioned in complaints or reviews?
- Does the tweet contain the name of a person? Does the tweet contain this person's location?

NER with spaCy:

```
doc = nlp('Apple is looking at buying a U.K. startup for $1 billion')
```

```
for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_, ' - ', spacy.explain(ent.label_))
```

```
Apple 0 5 ORG  -  Companies, agencies, institutions, etc.
U.K. 29 33 GPE  -  Countries, cities, states
$1 billion 46 56 MONEY  -  Monetary values, including unit
```

spaCy visualization tool for NER:

```
displacy.render(doc, style="ent")
```

Apple ORG is looking at buying U.K. GPE startup for $1 billion MONEY

---

4. Word vectors and similarity in spaCy

Like the Gensim library, spaCy also provides **pre-trained word vectors**, which are easily accessible since spaCy was designed to facilitate user's experience.

```
doc = nlp("King is similar to Queen, but it is not similar to Doctor")

king = doc[0]
queen = doc[4]
doctor = doc[-1]

print(king.has_vector, queen.has_vector, doctor.has_vector)
```

```
True True True
```

```
print("\nKing's vector:\n", king.vector)
```

```
King's vector:
 [ 3.1542e-01 -3.5068e-01  4.2923e-01 -5.3825e-01 -1.8480e-01 -3.1082e-01
  2.9196e-01 -7.1030e-01 -2.3867e-01  1.8471e+00 -3.6446e-01 -5.1282e-01
  1.2210e-01  3.8909e-01 -7.3204e-02  3.5462e-02  3.3289e-01  6.6466e-01
  2.7175e-02  4.2021e-01 -1.4520e-01  3.7991e-01 -6.0520e-01  1.0695e-01
 -6.4716e-01 -1.0739e-02 -3.9754e-01  3.8857e-01 -2.0134e-01  6.9813e-01
 -3.2411e-01  7.3085e-01 -1.0930e-01 -2.3511e-01  1.8482e-01 -1.1595e-01
 -7.1003e-01 -2.2974e-01 -4.1979e-01  8.1004e-03 -1.0504e-01 -4.4802e-01
 -7.3928e-02 -4.2380e-01  2.8482e-01 -7.4517e-02  9.8161e-02  6.4602e-01
 -2.5832e-01 -2.0452e-02 -6.6863e-02  5.1501e-01  1.6758e-01  1.2329e-01
  1.9636e-01  1.1958e-01 -1.8296e-01 -1.4325e-01 -2.7758e-01  5.0597e-02
 -6.6122e-02 -1.8920e-01  3.3300e-01  2.5319e-01  6.6355e-01  6.6735e-01
  4.9969e-01  1.5481e-01 -8.4247e-02 -2.2947e-01 -6.8367e-01 -2.9783e-01
 -1.8651e-01 -4.7121e-01  1.8272e-01 -3.2604e-01 -6.8030e-02  7.0073e-01
  3.3159e-01  7.0393e-02 -7.6987e-01  5.9069e-01  2.0592e-01  1.7976e-01
  6.9525e-03  5.7855e-02  7.2047e-01 -7.7249e-01 -5.4188e-01 -1.2189e-01
 -3.1734e-03 -1.5960e-01  1.6970e-01 -1.2546e-01  8.7069e-01 -4.6478e-01
 -1.9302e-01 -4.5618e-01 -1.5419e-01  8.1190e-01 -2.0544e-01  3.9454e-01
```

```
 -3.1178e-01 -6.4318e-02 -4.4443e-02 -5.8338e-01 -1.4792e-01  1.7083e-02
  8.3239e-01 -1.1280e-01  5.7826e-02  1.7024e-01 -1.3635e-01 -2.8894e-01
 -4.0590e-01 -5.0685e-02  4.9856e-01  6.0885e-02  1.9437e-01 -1.9811e-01
 -2.2335e-01 -2.5909e-02  3.9846e-01  4.4087e-01  2.3195e-02  9.8666e-02
 -1.3004e-01 -2.0339e-01 -4.2958e-01 -7.9760e-03 -3.2016e-01 -4.1094e-01
 -1.0304e-01 -7.5565e-01  1.7748e-02 -2.0037e-01  1.7185e-01  2.1787e-01
 -3.1685e-01  2.2068e-02 -2.5559e+00 -9.9115e-02  1.8434e-01  1.2448e-01
 -5.9413e-02 -4.5649e-02  7.9018e-01  2.4556e-01 -1.5059e-02 -7.8996e-01
  2.9087e-01 -3.9419e-01  3.7617e-01  1.5718e-01  5.1356e-01 -3.4219e-01
  5.0628e-02 -3.3254e-01 -1.4157e-01  3.3355e-01  4.4398e-01 -2.5451e-01
 -3.3201e-02 -2.0958e-01  3.8870e-01 -2.4565e-01  5.2391e-01  4.3247e-01
 -4.1701e-01  2.9031e-01 -7.8001e-01  3.0100e-02 -6.1446e-02 -1.4029e-01
 -5.5354e-01 -1.9175e-01  6.7279e-01 -1.1104e-01 -3.5486e-01 -2.8601e-01
  1.1720e-01 -4.5021e-01  1.4004e-01 -5.7484e-01 -2.2531e-01  4.1572e-01
 -1.5950e-01 -2.7877e-01  7.9785e-02  1.9120e-02 -9.8357e-01 -5.6998e-01
 -3.4023e-02  1.7382e-02 -1.7157e-02 -2.8211e-01  1.5573e-01 -1.3556e-01
 -2.6296e-01 -7.4571e-01  1.2015e-01  5.4234e-01  5.6783e-02 -7.5675e-02
  2.1820e-01 -2.5679e-01  2.3552e-01 -2.7111e-02 -1.9342e-01 -3.1088e-01
 -1.0600e-01  4.9512e-01  5.7932e-02  3.8773e-01  9.3160e-02 -1.3782e-01
  2.4244e-01  3.8098e-01  9.1109e-04  8.8338e-01  4.3823e-01 -7.7041e-02
  1.1541e-01  3.4702e-01  5.9785e-01  6.7012e-01 -6.0953e-02 -4.3872e-02
 -4.0800e-01  7.5721e-01  2.4773e-01  8.8926e-02 -1.8493e-01 -5.2339e-01
  8.5809e-02 -6.0880e-01 -7.7463e-02 -2.6829e-01 -3.9021e-01 -1.5002e-01
  5.4297e-01 -4.1076e-01 -9.5215e-02 -2.9787e-01  1.0041e-01 -3.7774e-01
  7.5511e-01 -4.3910e-01 -6.1722e-01 -1.0360e+00  6.9651e-01  1.4157e-01
 -4.4533e-01  3.2702e-01  3.8306e-02  2.6765e-01  5.4242e-02 -3.0242e-02
 -4.5133e-01  6.2505e-03  2.7504e-01 -5.2413e-02 -1.9870e-01 -1.7869e-01
 -2.4658e-01 -3.7369e-01  2.6174e-01  4.1482e-01 -5.9277e-01  6.1446e-02
  6.6261e-02  1.0970e-01 -1.4388e-01 -3.2442e-01 -3.9016e-04 -2.1392e-01
  3.2963e-01  5.0402e-01  1.3454e-01 -5.6133e-01  1.0422e+00  5.8985e-01
  1.4473e-01  1.7745e-01  1.6160e-01  3.3230e-01  2.2909e-01  1.5774e-01
 -3.5463e-01 -4.7642e-01 -2.5822e-01  2.3677e-01 -4.0255e-01 -3.5364e-01
 -1.6697e-01  7.0677e-01  8.4272e-02  1.1427e-01  5.8221e-01 -1.0559e-01]
```

```
print("king <-> queen", king.similarity(queen))
print("king <-> doctor", king.similarity(doctor))
```

```
king <-> queen 0.72526103
king <-> doctor 0.22130153
```