datacamp

Python For Data Science
spaCy Cheat Sheet

Learn spaCy online at www.DataCamp.com

spaCy

spaCy is a free, open-source library for advanced Natural Language processing (NLP) in Python. It's designed specifically for production use and helps you build applications that process and "understand" large volumes of text. Documentation: spacy.io

```
>>> $ pip install spacy
>>> import spacy
```

> Statistical models

Download statistical models

Predict part-of-speech tags, dependency labels, named entities and more. See here for available models: spacy.io/models

```
>>> $ python -m spacy download en_core_web_sm
```

Check that your installed models are up to date

```
>>> $ python -m spacy validate
```

Loading statistical models

```
>>> import spacy
>>> nlp = spacy.load("en_core_web_sm") # Load the installed model "en_core_web_sm"
```

> Documents and tokens

Processing text

Processing text with the `nlp` object returns a `Doc` object that holds all information about the tokens, their linguistic features and their relationships

```
>>> doc = nlp("This is a text")
```

Accessing token attributes

```
>>> doc = nlp("This is a text")
>>>[token.text for token in doc] #Token texts
 ['This', 'is', 'a', 'text']
```

> Label explanations

```
>>> spacy.explain("RB")
 'adverb'
>>> spacy.explain("GPE")
 'Countries, cities, states'
```

> Spans

Accessing spans

Span indices are exclusive. So `doc[2:4]` is a span starting at token 2, up to – but not including! – token 4.

```
>>> doc = nlp("This is a text")
>>> span = doc[2:4]
>>> span.text
 'a text'
```

Creating a span manually

```
>>> from spacy.tokens import Span #Import the Span object
>>> doc = nlp("I live in New York") #Create a Doc object
>>> span = Span(doc, 3, 5, label="GPE") #Span for "New York" with label GPE (geopolitical)
>>> span.text
 'New York'
```

> Linguistic features

Attributes return label IDs. For string labels, use the attributes with an underscore. For example, `token.pos_` .

Part-of-speech tags

Predicted by Statistical model

```
>>> doc = nlp("This is a text.")
>>> [token.pos_ for token in doc] #Coarse-grained part-of-speech tags
 ['DET', 'VERB', 'DET', 'NOUN', 'PUNCT']
>>> [token.tag_ for token in doc] #Fine-grained part-of-speech tags
 ['DT', 'VBZ', 'DT', 'NN', '.']
```

Syntactic dependencies

Predicted by Statistical model

```
>>> doc = nlp("This is a text.")
>>> [token.dep_ for token in doc] #Dependency labels
 ['nsubj', 'ROOT', 'det', 'attr', 'punct']
>>> [token.head.text for token in doc] #Syntactic head token (governor)
 ['is', 'is', 'text', 'is', 'is']
```

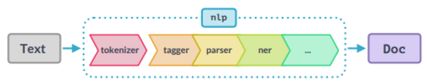
Named entities

Predicted by Statistical model

```
>>> doc = nlp("Larry Page founded Google")
>>> [(ent.text, ent.label_) for ent in doc.ents] #Text and label of named entity span
 [('Larry Page', 'PERSON'), ('Google', 'ORG')]
```

> Pipeline components

Functions that take a `Doc` object, modify it and return it.



Pipeline information

```
>>> nlp = spacy.load("en_core_web_sm")
>>> nlp.pipe_names
 ['tagger', 'parser', 'ner']
>>> nlp.pipeline
 [('tagger', <spacy.pipeline.Tagger>),
 ('parser', <spacy.pipeline.DependencyParser>),
 ('ner', <spacy.pipeline.EntityRecognizer>)]
```

Custom components

```
def custom_component(doc): #Function that modifies the doc and returns it
    print("Do something to the doc here!")
    return doc
nlp.add_pipe(custom_component, first=True) #Add the component first in the pipeline
```

Components can be added `first` , `last` (default), or before or after an existing component.

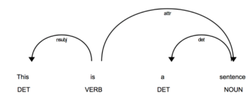
> Visualizing

If you're in a Jupyter notebook, use `displacy.render` otherwise, use `displacy.serve` to start a web server and show the visualization in your browser.

```
>>> from spacy import displacy
```

Visualize dependencies

```
>>> doc = nlp("This is a sentence")
>>> displacy.render(doc, style="dep")
```



Visualize named entities

```
>>> doc = nlp("Larry Page founded Google")
>>> displacy.render(doc, style="ent")
```

Larry Page PERSON founded Google ORG

> Word vectors and similarity

To use word vectors, you need to install the larger models ending in `md` or `lg` , for example `en_core_web_lg` .

Comparing similarity

```
>>> doc1 = nlp("I like cats")
>>> doc2 = nlp("I like dogs")
>>> doc1.similarity(doc2) #Compare 2 documents
>>> doc1[2].similarity(doc2[2]) #Compare 2 tokens
>>> doc1[0].similarity(doc2[1:3]) # Comparetokens and spans
```

Accessing word vectors

```
>>> doc = nlp("I like cats") #Vector as a numpy array
>>> doc[2].vector #The L2 norm of the token's vector
>>> doc[2].vector_norm
```

> Syntax iterators

Sentences

Ususally needs the dependency parser

```
>>> doc = nlp("This a sentence. This is another one.")
>>> [sent.text for sent in doc.sents] #doc.sents is a generator that yields sentence spans
 ['This is a sentence.', 'This is another one.']
```

Base noun phrases

Needs the tagger and parser

```
>>> doc = nlp("I have a red car")
#doc.noun_chunks is a generator that yields spans
>>> [chunk.text for chunk in doc.noun_chunks]
 ['I', 'a red car']
```

> Extension attributes

Custom attributes that are registered on the global `Doc`, `Token` and `Span` classes and become available as `._`.

```
>>> from spacy.tokens import Doc, Token, Span
>>> doc = nlp("The sky over New York is blue")
```

Attribute extensions — With default value

```
# Register custom attribute on Token class
>>> Token.set_extension("is_color", default=False)
# Overwrite extension attribute with default value
doc[6]._.is_color = True
```

Property extensions — With getter and setter

```
# Register custom attribute on Doc class
>>> get_reversed = lambda doc: doc.text[::-1]
>>> Doc.set_extension("reversed", getter=get_reversed)
# Compute value of extension attribute with getter
>>> doc._.reversed
'eulb si kroY weN revo yks ehT'
```

Method extensions — Callable Method

```
# Register custom attribute on Span class
>>> has_label = lambda span, label: span.label_ == label
>>> Span.set_extension("has_label", method=has_label)
# Compute value of extension attribute with method
>>> doc[3:5].has_label("GPE")
True
```

> Rule-based matching

Using the matcher

```
# Matcher is initialized with the shared vocab
>>> from spacy.matcher import Matcher
# Each dict represents one token and its attributes
>>> matcher = Matcher(nlp.vocab)
# Add with ID, optional callback and pattern(s)
>>> pattern = [{"LOWER": "new"}, {"LOWER": "york"}]
>>> matcher.add("CITIES", None, pattern)
# Match by calling the matcher on a Doc object
>>> doc = nlp("I live in New York")
>>> matches = matcher(doc)
# Matches are (match_id, start, end) tuples
>>> for match_id, start, end in matches:
    # Get the matched span by slicing the Doc
    span = doc[start:end]
    print(span.text)
    'New York'
```

Token patterns

```
# "love cats", "loving cats", "loved cats"
>>> pattern1 = [{"LEMMA": "love"}, {"LOWER": "cats"}]
# "10 people", "twenty people"
>>> pattern2 = [{"LIKE_NUM": True}, {"TEXT": "people"}]
# "book", "a cat", "the sea" (noun + optional article)
>>> pattern3 = [{"POS": "DET", "OP": "?"}, {"POS": "NOUN"}]
```

Operators and quantifiers

Can be added to a token dict as the `"OP"` key

`!`	Negate pattern and match exactly 0 times
`?`	Make pattern optional and match 0 or 1 times
`+`	Require pattern to match 1 or more times
`*`	Allow pattern to match 0 or more time

> Glossary

Tokenization

Segmenting text into words, punctuation etc

Lemmatization

Assigning the base forms of words, for example:
"was" → "be" or "rats" → "rat".

Sentence Boundary Detection

Finding and segmenting individual sentences.

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.

Named Entity Recognition (NER)

Labeling named "real-world" objects,
like persons, companies or locations.

Text Classification

Assigning categories or labels to a whole
document, or parts of a document.

Statistical model

Process for making predictions based on examples.

Training

Updating a statistical model with new examples.