PYTHON FOR DATA SCIENCE | SPACY CHEAT SHEET

DATA-DRIVEN SCIENCE

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.

Getting started

```
$ 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.

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

DOCUMENTS, TOKENS AND SPANS

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 texts
[token.text for token in doc]
# ['This', 'is', 'a', 'text']
```

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

```
# Import the Span object
from spacy.tokens import Span
# Create a Doc object
doc = nlp("I live in New York")
# Span for "New York" with label GPE (geopolitical)
span = Span(doc, 3, 5, label="GPE")
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.")
# Coarse-grained part-of-speech tags
[token.pos_ for token in doc]
# ['DET', 'VERB', 'DET', 'NOUN', 'PUNCT']
# Fine-grained part-of-speech tags
[token.tag_ for token in doc]
# ['DT', 'VBZ', 'DT', 'NN', '.']
```

SYNTACTIC DEPENDENCIES (Predicted by statistical model)

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

NAMED ENTITIES (Predicted by statistical model)

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

SYNTAX ITERATORS

SENTENCES (Usually needs the dependency parser)

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

LABEL EXPLANATIONS

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

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")
# Compare 2 documents
doc1.similarity(doc2)
# Compare 2 tokens
doc1[2].similarity(doc2[2])
# Compare tokens and spans
doc1[0].similarity(doc2[1:3])
```

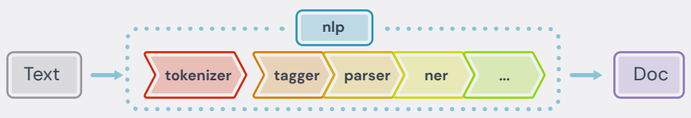
ACCESSING WORD VECTORS

You can also do the internal data alignment yourself with the help of the fill methods:

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

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>)]
```

LEARN DATA SKILLS ONLINE AT WWW.DATADRIVENSCIENCE.COM

CUSTOM COMPONENTS

```
from spacy.language import Language

@Language.component("custom_component")
def custom_component(doc): #Function that modifies the doc and returns it
        #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.

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 & 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', [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.

OP	Description	OP	Description
!	Negate pattern and match exactly 0 times.	+	Require pattern to match 1 or more times.
?	Make pattern optional and match 0 or 1 times.	*	Allow pattern to match 0 or more times.

spaCy GLOSSARY

Name	Description
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.