spaCy

A Cheat Sheet for General Use

Getting Started

spaCy is a free, open-source library for advanced Natural Language Processing that helps you process text in Python. Documentation at https://spacy.io.

Get started by installing spaCy and a trained pipeline. There's a complete list of pipelines at https://spacy.io/usage/models.

```
>>> $ python -m pip install —upgrade pip
>>> $ python -m pip install spacy
>>> $ python -m spacy download en_core_web_sm
>>> $ python -m spacy validate
```

Then, import spaCy and load the trained pipeline.

```
# import spaCy
import spacy

# load the language model
nlp = spacy.load("en_core_web_sm")
# initalize the parsed document
doc = nlp("spaCy is used for Natural Language Processing")

print([[token.text, token.shape_] for  token in doc])
# [['spaCy', 'xxxXx'], ['is', 'xx'], ['used', 'xxxx'],
# ['for', 'xxx'], ['Natural', 'Xxxxx'], ['Language', 'Xxxxx'],
# ['Processing', 'Xxxxx']]
```

Named Entities

NER (Named Entity Recognition)

```
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
for ent in doc.ents:
    print(ent.text, ent.label_)

# ('Apple', 'ORG'), ('U.K.', 'GPE'), ('$1 billion', 'MONEY')
```

Setting Entities

```
from spacy.tokens import Span

doc = nlp("Apple is hiring developers.")
print([(e.text, e.label_) for e in doc.ents])
# [('Apple', 'ORG')]

# Create a span for the new entity
corp_ent = Span(doc, 3, 4, label="JOB")
orig_ents = list(doc.ents)

# Option 1: Modify the provided entity spans, leaving the rest
doc.set_ents(orig_ents + [corp_ent])
# Option 2: Assign a complete list of ents to doc.ents
doc.ents = orig_ents + [corp_ent]

print([(e.text, e.label_) for e in doc.ents])
# [('Apple', 'ORG'), ('developers', 'JOB')]
```

Linguistic Features

Part-of-speech tagging

```
doc = nlp("This is a simple sentence.")

print([token.pos_ for token in doc]) # more fine-grained: token.tag_
# ['PRON', 'AUX', 'DET', 'ADJ', 'NOUN', 'PUNCT']
```

You can use **spacy.explain()** for a label explanation.

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

Morphology — Get features

```
doc = nlp("She and I are reading the paper.")
token_1 = doc[0] # She
token_2 = doc[2] # I

print(token_1.morph, token_2.morph)
# Case=Nom|Gender=Fem|Number=Sing|Person=3|PronType=Prs
# Case=Nom|Number=Sing|Person=1|PronType=Prs

print(token_1.morph.get("Person"), token_2.morph.get("Person"))
# ['3'] ['1']
```

Morphology — Lemmatization

```
doc = nlp("I was reading the books.")
print([token.lemma_ for token in doc])
# ['I', 'be', 'read', 'the', 'book', '.']
```

Dependency Parsing — The Parse Tree

```
doc = nlp("Many books were piled on the desk.")
for token in doc:
    print(token.text, token.dep_, token.head.text, token.head.pos_,
          list(token.children))
```

Text	Dependency	Head	Head POS	Children
Many	amod	books	NOUN	[]
books	nsubjpass	piled	VERB	[Many]
were	auxpass	piled	VERB	[]
piled	ROOT	piled	VERB	[books,were, on,.]
on	prep	piled	VERB	[desk]
the	det	desk	NOUN	[]
desk	pobj	on	ADP	[the]
.	punct	piled	VERB	[]

Dependency Parsing — Noun Chunks

```
for chunk in doc.noun_chunks:
    print(chunk.text, chunk.root.text, chunk.root.dep_,
chunk.root.head.text)
```

Text	Root text	Dependency	Head
Many books	books	nsubjpass	piled
the desk	desk	pobj	on

Sentence Segmentation

```
doc = nlp("This is the first sentence. This is another sentence")
print([sent.text for sent in doc.sents])
# ['This is the first sentence.', 'This is another sentence.']
```

Word Vectors and Similarity

Getting Started

```
# download the larger pipeline
 >>> $ python -m spacy download en_core_web_md
```

```
# load the larger pipeline for vectors!
nlp = spacy.load("en_core_web_md")
tokens = nlp("dog banana afskfsd")

for token in tokens:
    print(token.text, token.has_vector, token.vector_norm)

# dog True 7.443447
# banana True 6.895898
# afskfsd False 0.0

print(tokens[0].vector)

# [-0.72483    0.42538    0.025489  -0.39807    0.037463  -0.29811
#  -0.28279    0.29333    0.57775    1.2205    -0.27903    0.80879
# ...]
```

Word, Span and Sentence Similarity

```
# use the larger pipeline for better similarity scores!
nlp = spacy.load("en_core_web_md")
doc1 = nlp("I love french fries.")
doc2 = nlp("I hate hamburgers.")

# Accessing word vector
doc1[0].vector

# Similarity of tokens and spans
french_fries = doc1[2:4]
burgers = doc2[2]

print(french_fries.similarity(burgers))
# 0.46016860008239746

# Similarity of documents/sentences
print(doc1.similarity(doc2))
# 0.801...
```

Similarity scores are high for words that appear in similar contexts, but they might have different or opposite meanings.

```
love = doc1[1]
hate = doc2[1]

print(love.similarity(hate))
# Even though they have opposite meanings, they are similar
# 1.0
```

Adding External Word Vectors

```
 >>> $ wget https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/
cc.en.300.vec.gz
 >>> $ python -m spacy init vectors en cc.en.300.vec.gz
/project/vectors/en_vectors_wiki_lg
```

```
# load new vectors
nlp_latin = spacy.load("/project/vectors/en_vectors_wiki_lg")
```

Matching

Token Matcher

```
from spacy.matcher import Matcher

matcher = Matcher(nlp.vocab)
# Add match ID "HelloWorld" with no callback and one pattern
pattern = [{"LOWER": "hello"}, {"IS_PUNCT": True},
{"POS":"NOUN","LEMMA":"world"}]

matcher.add("HelloWorld", [pattern])

doc = nlp("Hello, world! Hello world!")
matches = matcher(doc)
for match_id, start, end in matches:
    span = doc[start:end]  # The matched span
    print(span.text)

# Hello, world
```

Phrase Matcher

Although less flexible, PhraseMatcher is faster than Matcher.

```
from spacy.matcher import PhraseMatcher

matcher = PhraseMatcher(nlp.vocab)
matcher.add("OBAMA", [nlp("Barack Obama")])
doc = nlp("Barack Obama lifts America one last time")
matches = matcher(doc)

for match_id, start, end in matches:
    span = doc[start:end]  # The matched span
    print(span.text)
# Barack Obama
```

Entity Matcher

```
ruler = nlp.add_pipe("entity_ruler")
# patterns to add to doc.ents
patterns = [{"label": "ORG", "pattern": "MyCorp Inc."},
{"label": "GPE", "pattern": [{"LOWER": "san"},
{"LOWER": "francisco"}]}]
ruler.add_patterns(patterns)

doc = nlp("MyCorp Inc. is a company in San Francisco")
print([(ent.text, ent.label_) for ent in doc.ents])

# [('MyCorp Inc.', 'ORG'), ('San Francisco', 'GPE')]
```

Extensions

```
from spacy.tokens import Doc, Token, Span
doc = nlp("Bob says San Francisco is often grey and cloudy")
```

Attribute Extensions — with Token

```
Token.set_extension("is_color", default=False)
doc[6]._.is_color = True
```

Property Extensions — with Doc

```
get_reversed = lambda doc: doc.text[::-1]
Doc.set_extension("reversed", getter=get_reversed)

print(doc._.reversed)
# yduolc dna yerg netfo si ocsicnarF naS syas boB
```

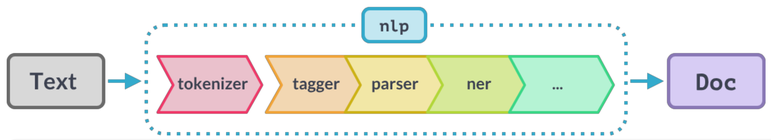
Method Extensions — with Span

```
def pos_filter(span, pos):
    return [token for token in span if token.pos_ in pos]

Span.set_extension("pos_filter", method=pos_filter)

doc[5:]._.pos_filter(["ADJ"])
# [grey, cloudy]
```

Pipeline




```
nlp.pipe_names
# ['tagger','parser','ner']
```

Enable Pipes

```
# Enable only the parser
with nlp.select_pipes(enable="parser"):
    doc = nlp("I will only be parsed")
```

Disable Pipes

```
# Load pipeline without entity recognizer (faster if not needed)
nlp = spacy.load("en_core_web_sm", exclude=["ner"])

# Load the tagger and parser but don't enable them
nlp = spacy.load("en_core_web_sm", disable=["tagger", "parser"])

# Explicitly enable the tagger later on
nlp.enable_pipe("tagger")
```

Replace Pipes

```
def my_custom_tagger(doc):
  print("Do something to the doc here!")
  return doc
nlp.replace_pipe("tagger", "my_custom_tagger")
# If no tagger was present, you can add one
nlp.add_pipe(my_custom_tagger, first=True)
```

Glossary

Statistical Models make predictions based on patterns from examples.

Parsing is the analysis of the document, which tokenizes the text into meaningful chunks and adds features to the tokens to describe their syntactic roles.

POS (Parts-of-speech) describe the syntactic and inflectional category of words, e.g., nouns or verbs.

Lemmatized word (a lemma) is the word without any inflectional morphology, or prefixes or suffixes that change the grammatical function of the word but not its part of speech.

The lemma of 'teaches' would be 'teach', but the lemma of 'weaken' (VERB) would not be 'weak' (ADJ), because they have different POS tags.

Dependency Relationships
The parse tree allows us to see the relationships between words. Every word has exactly one head, in which the word is dependent upon for its meaning within the sentence.

For example, in the phrase "the small cat", "small" is dependent on "cat" for its meaning, so "cat" is the head of that phrase.

You can think of the root of a phrase like the center, it's what everything else is dependent on.
Different types of dependency relationships can be described by what types

NER (Named Entity Recognition) recognizes named and numeric entities, like companies, locations, money terms or products.

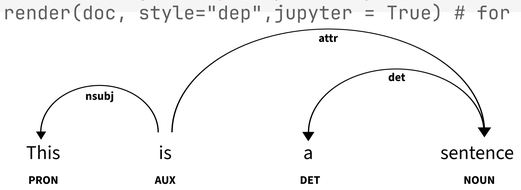
Word Vectors / Word embeddings, are multi-dimensional encodings that are used by models to represent what a word means in relation to other words.

Pipeline is a series of steps that a model takes to do a function, like create a Doc object.

Visualization

Dependencies

```
from spacy import displacy
doc = nlp("This is a sentence.")
opts= {"compact":True,"bg":"#09a3d5","color":"white","font":"Arial"}
displacy.serve(doc, style="dep", options=opts) # for web browser
#displacy.render(doc, style="dep",jupyter = True) # for Jupyter NB
```



Entities

```
from spacy import displacy

text = "When Sebastian Thrun started working on self-driving cars
at Google in 2007, people laughed."
doc = nlp(text)
displacy.serve(doc, style="ent")
```

Spans

```
from spacy import displacy
from spacy.tokens import Span

nlp = spacy.blank("en")
doc = nlp("Welcome to the Bank of China.")

doc.spans["sc"] = [Span(doc, 3, 6, "ORG"),
                   Span(doc, 5, 6, "GPE"),]

displacy.serve(doc, style="span")
```

Additional Information

Hash vs Text Values

spaCy encodes all of the strings into hash values, which means if you call **token.pos**, **ent.label**, etc., you're going to get a number, not a word. Instead, add underscore to the end "**token.pos_**" to get the word (string) value.

```
doc = nlp("Microsoft is buying startup for $3.4 billion")

print(doc[0].pos, doc[0].pos_)
# 96 PROPN
```