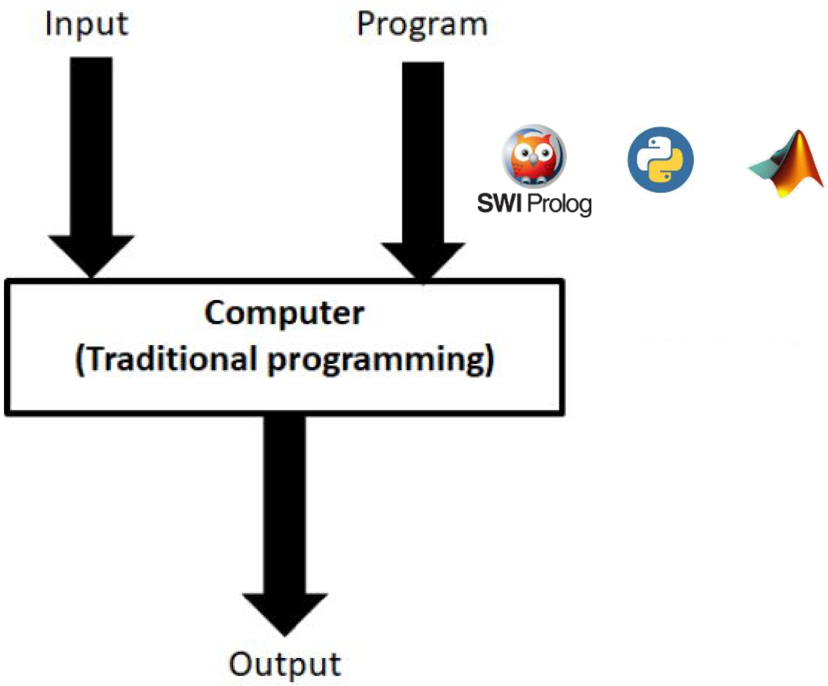
Language Models

- for Text Analysis

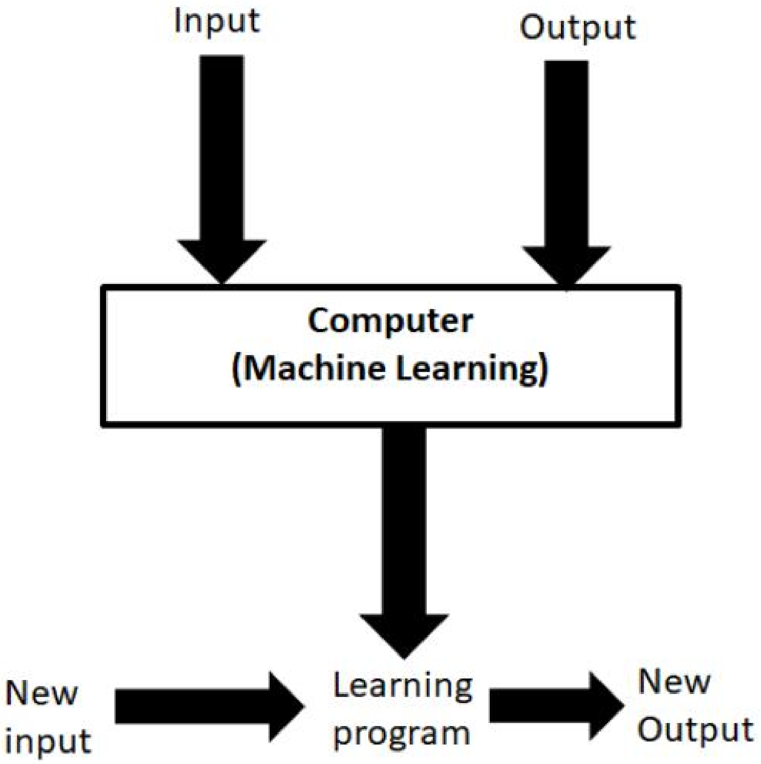
Outline

- Machine Learning vs GOFAI (Symbolic artificial intelligence)
- About Text Analysis
- Language Models(spaCy)
- Part-of-speech (POS) – tagging
- Named entity recognition
- Rule-based matching
- Preprocessing

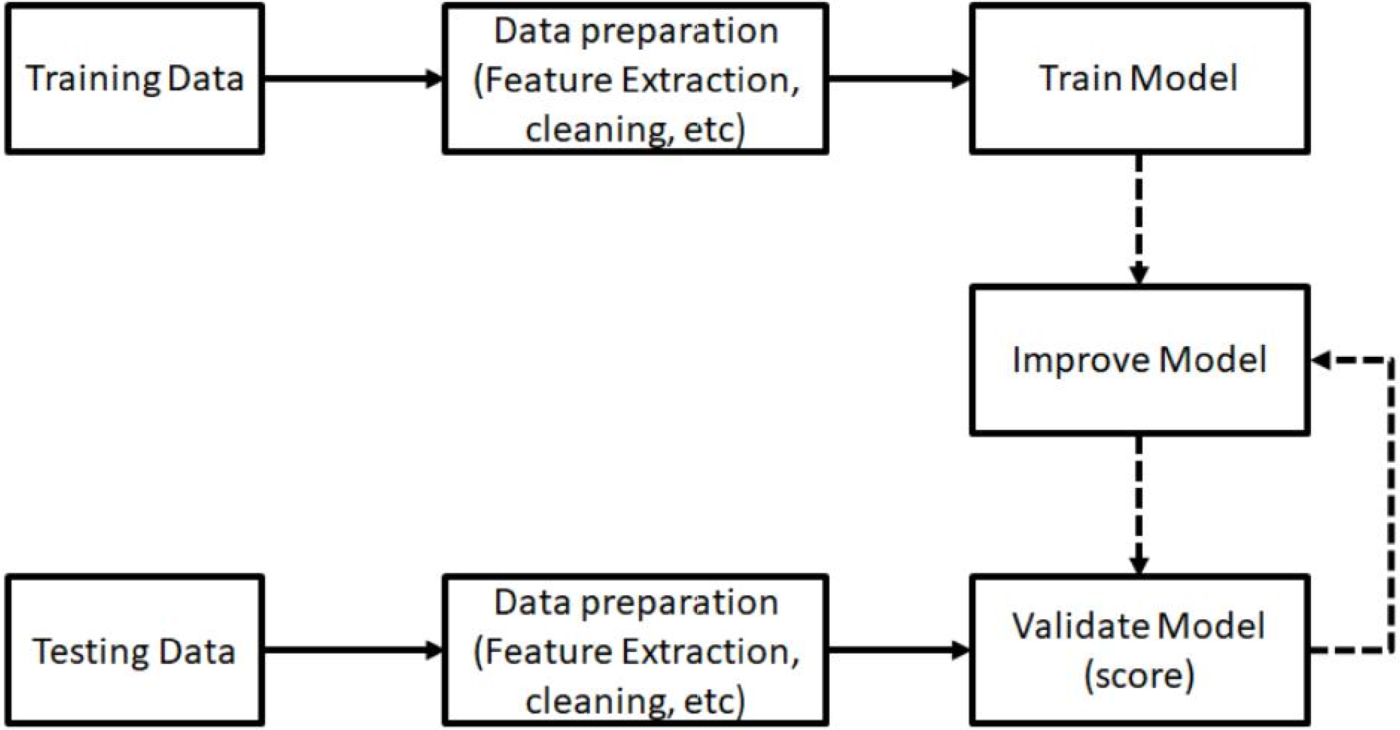
GOFAI


Input
Program
SWI Prolog
Computer
(Traditional programming)
Output

Machine Learning



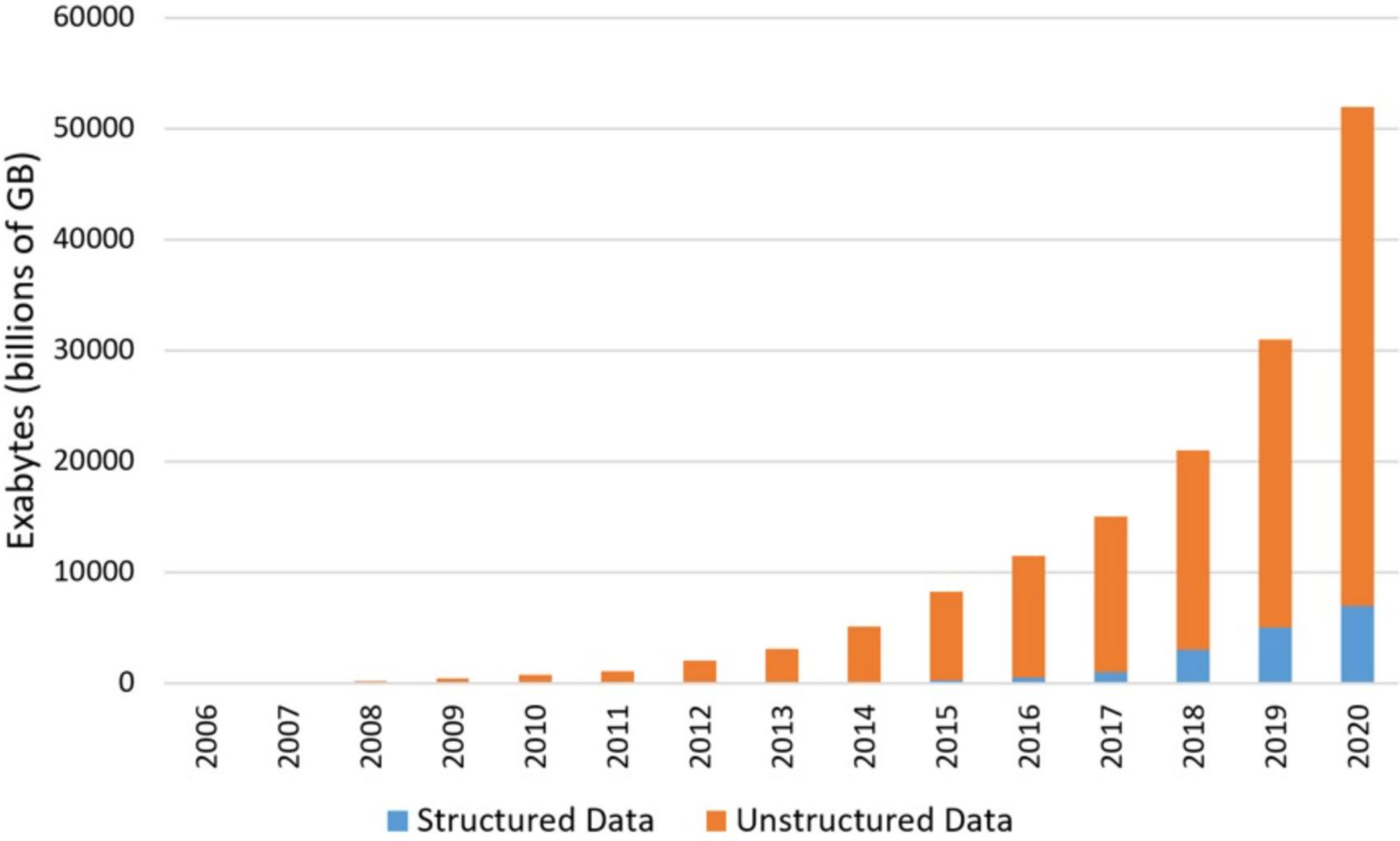
Machine Learning


Training Data
Data preparation (Feature Extraction, cleaning, etc)
Train Model
Improve Model
Testing Data
Data preparation (Feature Extraction, cleaning, etc)
Validate Model (score)

About Text Analysis

- Textual data also has huge business value, and companies can use this data to help profile customers and understand customer trends. This can either be used to offer a more personalized experience for users or as information for targeted marketing.

The Cambrian Explosion…of Data



About Text Analysis

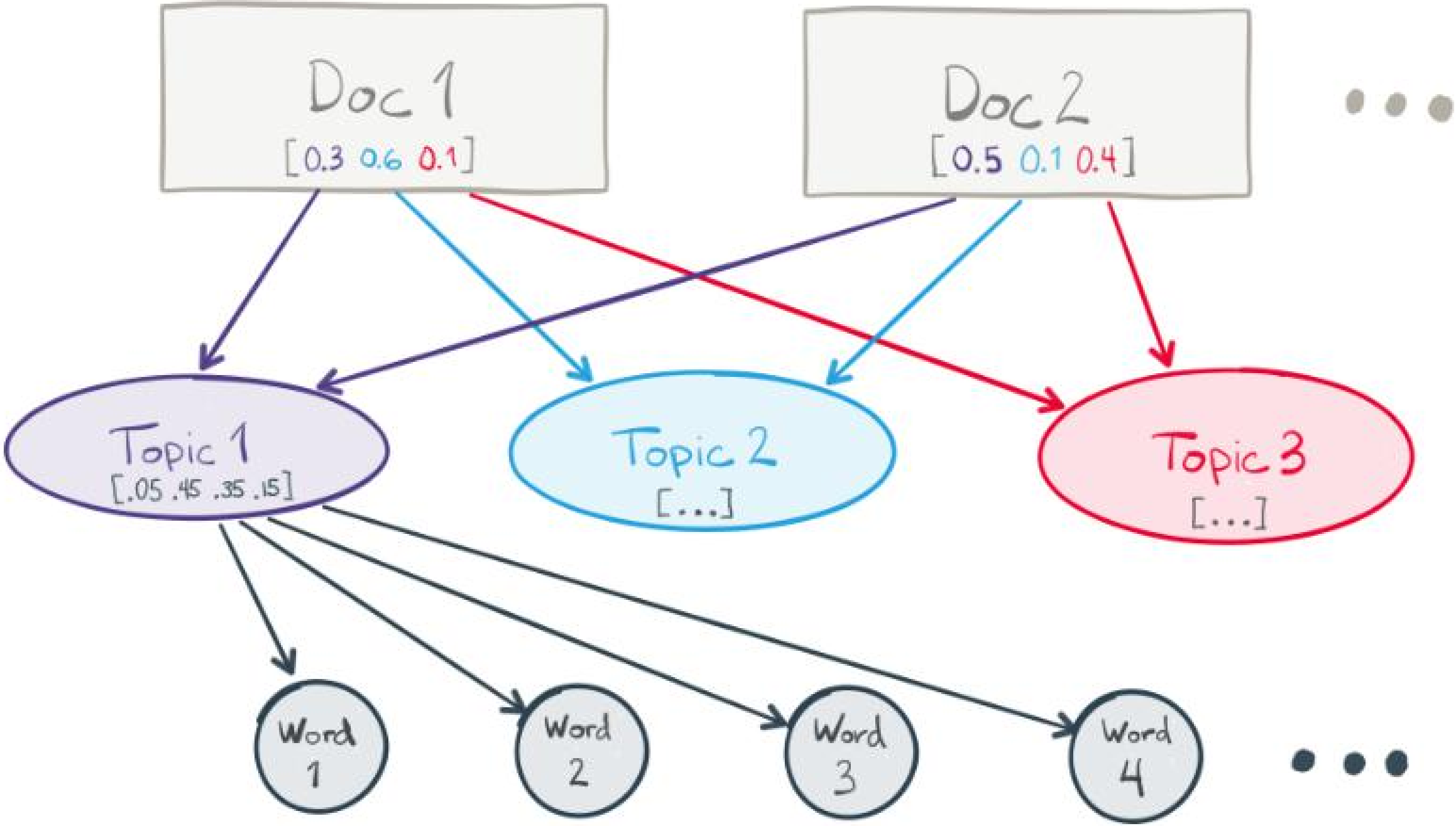
- Text analysis can be understood as the technique of gleaning useful information from text. This can be done through various techniques, and we use **Natural Language Processing** (**NLP**), **Computational Linguistics** (**CL**), and numerical tools to get this information.

About Text Analysis

- Natural language processing (NLP) refers to the use of a computer to process natural language. For example, removing all occurrences of the word *thereby* from a body of text is one such example, albeit a basic example.
- Computational linguistics (CL), as the name suggests, is the study of linguistics from a computational perspective. This means using computers and algorithms to perform linguistics tasks such as marking your text as a part of speech (such as noun or verb), instead of performing this task manually.

About Text Analysis

- Information retrieval (IR) builds on statistical approaches in text processing and allows us to classify, cluster, and retrieve documents. Methods such as topic modeling can help us identify key topics in large, unstructured bodies of text. Identifying these topics goes beyond searching for keywords, and we use statistical models to further understand the underlying nature of bodies of text.


Doc 1
[0.3 0.6 0.1]
Doc 2
[0.5 0.1 0.4]
...
Topic 1
[.05 .45 .35 .15]
Topic 2
[...]
Topic 3
[...]
Word 1
Word 2
Word 3
Word 4
...

Language Models(spaCy)

- In this section, we will introduce spaCy and how we can use spaCy to help us in our text analysis tasks, as well as talk about some of its more powerful functionalities, such as Part of Speech-tagging and Named Entity Recognition-tagging. We will finish up with an example of how we can preprocess data quickly and efficiently using the natural language processing Python library, spaCy.

Language Models(spaCy)

- spaCy describes itself as Industrial Strength Natural Language Processing – and it most certainly does its best to live up to this promise. Focused on getting things done rather than a more academic approach, spaCy ships with only one part-of-speech tagging algorithm and only one named-entity-recognizer (per language).

Language Models(spaCy)

- Following are some features of spaCy:

1. Non-destructive tokenization
2. Support for 21+ natural languages
3. 6 statistical models for 5 languages
4. Pretrained word vectors
5. Easy deep learning integration
6. Part-of-speech tagging
7. Named entity recognition
8. Labeled dependency parsing
9. Syntax-driven sentence segmentation

Language Models(spaCy)

Installation

— pip install -U spacy

— virtualenv env

— source env/bin/activate

— pip install spacy

Language Models(spaCy)

One of spaCy's most interesting features is its language models. A language model is a statistical model that lets us perform the NLP tasks we want to, such as POS-tagging and NER-tagging. These language models do not come packaged with spaCy, but need to be downloaded.

Language Models(spaCy)

Different languages have different models to perform these tasks, and there are also different models for the same language – the difference between these models is mostly statistical, and you can use different models based on your use case. A different model would just be trained on a different dataset. It is still the same underlying algorithm. The spaCy documentation on their models gives us some more insight into how they work.

Language Models(spaCy)

Installing language models

#download best-matching default model

—spacy download en # english model

—spacy download de # german model

—spacy download en_core_web_sm

Language Models(spaCy)

```
pip install spacy
spacy download en


import spacy
nlp = spacy.load('en')


doc = nlp(u'This is a sentence.')
```

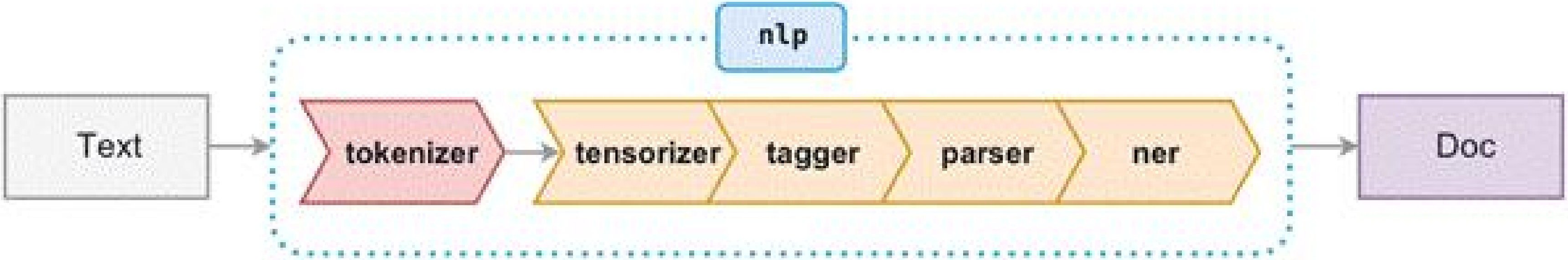
Language Models(spaCy)

Now that we know exactly how to get the models on our systems, let's start asking more questions about these models – how does it perform the POS-tagging or NER-tagging?

We'll attempt to answer these in the coming section, while also discussing the other possibilities spaCy has to offer with regard to its models, such as training our own models.

Language Models(spaCy)

doc = nlp(u'This is a sentence.')

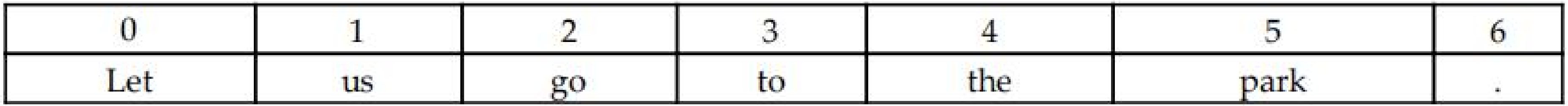


Language Models(spaCy)

Tokenization is the task of splitting a text into meaningful segments, called tokens. These segments could be words, punctuation, numbers, or other special characters that are the building blocks of a sentence. In spaCy, the input to the tokenizer is a Unicode text, and the output is a Doc object.

Language Models(spaCy)

For the sentence – Let us go to the park., it's quite straightforward, and would be broken up as follows, with the appropriate numerical indices:



0	1	2	3	4	5	6
Let	us	go	to	the	park	.

Language Models(spaCy)

So, once we pass our sentence to the nlp pipeline, the first step was tokenization – once this is done, we are now dealing with Doc objects, which are comprised of tokens – which we described before as the basic parts of our sentence. Once we have our tokens in the doc, each token is then worked on by the other components of the pipeline.

Part-of-speech (POS) – tagging

- The second component of the default pipeline was the tensorizer. A tensorizer encodes the internal representation of the doc as an array of floats. This is a necessary step because spaCy's models are neural network models, and only speak tensors – every Doc object is expected to be tenzorised. After this step, we start with our first annotation – part of speech tagging.

Part-of-speech (POS) – tagging

- POS-tagging makes each token of the sentence with its appropriate part of speech, such as noun, verb, and so on. spaCy uses a statistical model to perform its POS-tagging. To get the annotation from a token, we simply look up the pos_ attribute on the token.

```
doc = nlp(u'John and I went to the park")
for token in doc:
  print((token.text, token.pos_))
```

Part-of-speech (POS) – tagging

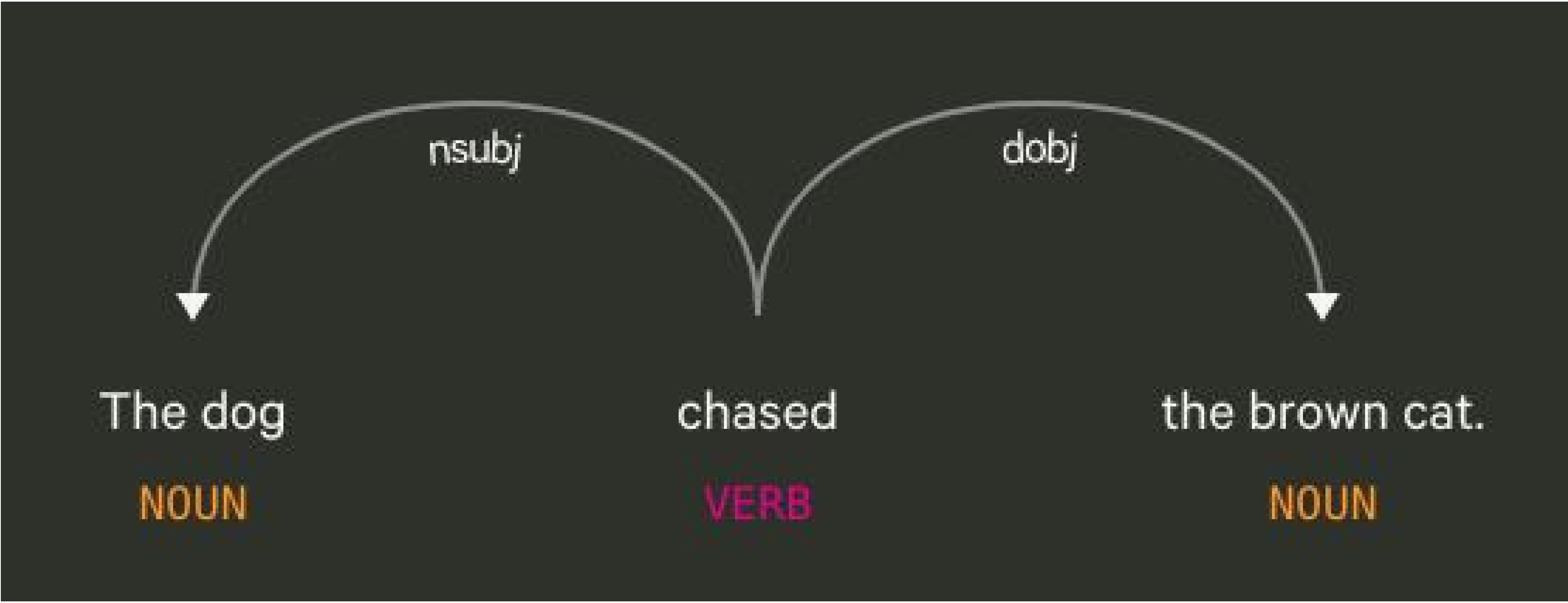
```
doc = nlp(u'John and I went to the park'')
for token in doc:
   print((token.text, token.pos_))
```

```
(u'John', u'PROPN')   (u'and', u'CCONJ')   (u'I', u'PRON')
(u'went', u'VERB')   (u'to', u'ADP')   (u'the', u'DET')
(u'park', u'NOUN')   (u'.', u'PUNCT')
```

Part-of-speech (POS) – tagging

The next part of our pipeline is the parser, which performs dependency parsing. While parsing refers to any kind of analysis of a string of symbols to understand relationships between the symbols, dependency parsing refers to the understanding of dependencies between these symbols. For example, in the English language, this could be for describing the relations between individual tokens, such as subject or object. spaCy has a rich API for navigating parse trees.

Part-of-speech (POS) – tagging



Named entity recognition

- We now have the last part of our pipeline, where we perform named entity recognition. A named entity is a real-world object that is assigned a name – for example, a person, a country, a product, or organization. spaCy can recognize various types of named entities in a document, by asking the model for a prediction. We have to remember that since models are statistical, they don't always work perfectly and might need tuning.

Named entity recognition

- Named entities are available as the ents property of a Doc:

```
doc = nlp(u'Microsoft has offices all over Europe.')
for ent in doc.ents:
   print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
(u'Microsoft', 0, 9, u'ORG')
(u'Europe', 31, 37, u'LOC')
```

Named entity recognition

spaCy has the following built-in entity types:

- PERSON: People, including fictional ones
- NORP: Nationalities or religious or political groups
- FACILITY: Buildings, airports, highways, bridges, and so on
- ORG: Companies, agencies, institutions, and so on
- GPE: Countries, cities, and states
- LOC: Non GPE locations, mountain ranges, and bodies of water
- PRODUCT: Objects, vehicles, foods, and so on (not services)
- EVENT: Named hurricanes, battles, wars, sports events, and so on
- WORK_OF_ART: Titles of books, songs, and so on

Rule-based matching

SpaCy's default pipeline also performs rule-based matching. This further annotates tokens with more information and is valuable during preprocessing. The following token attributes are available:

- ORTH: The exact verbatim text of a token
- LOWER, UPPER: The lowercase and uppercase form of the token
- IS_ALPHA: Token text consists of alphanumeric chars
- IS_ASCII: Token text consists of ASCII characters
- POS, TAG: The token's simple and extended POS tag
- DEP, LEMMA, SHAPE: The token's dependency label, lemma, and shape

Preprocessing

- The wonderful thing about preprocessing text is that it almost feels intuitive – we get rid of any information which we think won't be used in our final output and keep what we feel is important. Here, our information is words – and some words do not always provide useful insights. In the text mining and natural language processing community, these words are called stop words.

Preprocessing

- Stop words are words that are filtered out of our text before we run any text mining or NLP algorithms on it. Again, we would like to draw attention to the fact this is not in every case – if we intend to find stylistic similarities or understand how writers use stop words, we would obviously need to stop words!

Preprocessing

- There is no universal stop words list for each language, and it largely depends on the use case and what kind of results we expect to be seeing. Usually, it is a list of the most common words in the language, such as of, the, want, to, and have.
- With spaCy, stop words are very easy to identify – each token has an IS_STOP attribute, which lets us know if the word is a stop word or not.

Preprocessing

- We can also add our own stop words to the list of stop words. For example:

```
my_stop_words = [u'say', u'be', u'said', u'says', u'saying', 'field']
for stopword in my_stop_words:
   lexeme = nlp.vocab[stopword]
   lexeme.is_stop = True
```

Preprocessing

- You might have noticed in the preceding example how the words say, saying, and says all pretty much provide the same information to us – grammatical differences aside, it won't hurt our results to only see one representation of these words.

Preprocessing

- There are two popular techniques to achieve this, **stemming** and **lemmatization**. Stemming usually involves chopping off the end of the word, following some basic rules. For example, the words say, saying, and says would all become say. Stemming is contextless and does not rely on part of speech, for example, to make its decisions. Lemmatization, on the other hand, conducts morphological analysis to find the root word.

Preprocessing

- In spaCy, the lemmatized form of a word is accessed with the **.lemma_** attribute.
- Now, with what we know, we can do some basic preprocessing. Let's clean up this sentence: **the horse galloped down the field and past the 2 rivers.** We would like to get rid of stop words, numbers, and convert our string into a list so that we can use it later.

Preprocessing

```
doc = nlp(u'the horse galloped down the field and past the river.')
sentence = []
for w in doc:
    # if it's not a stop word or punctuation mark, add it to our article!
    if w.text != 'n' and not w.is_stop and not w.is_punct and not w.like_num:
        # we add the lematized version of the word
        sentence.append(w.lemma_)
print(sentence)
```

```
[u'horse', u'gallop', u'past', u'river']
```

Preprocessing

We can further remove or not remove words based on our use-case. In our example, it is deemed that numbers are not important information, but in some cases, it might be. For example, it might be that we want to remove all verbs from a sentence – in which case we can, by simply checking the POS tag of that particular token.

Preprocessing

spaCy's pipeline annotates text in such a way that we can very easily use that information to process our text. The handy thing is that we can further use that information later on in our text-processing, and not just in preprocessing. It makes sense to start any of our NLP tasks by running it through a spaCy pipeline, custom or otherwise, just for the large amount of information and annotation we will get, in almost just five lines of code.

Summary

spaCy offers us an easy way to annotate your text data very easily, and with the language model, we annotate your text data with a lot of information – not just tokenizing and whether it is a stop word or not, but also the part of speech, named entity tag, and so on – we can also train these annotating models on our own, giving a lot of power to the language model and processing pipeline!

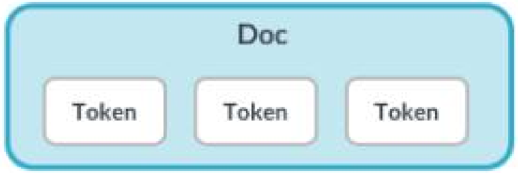
The Doc object

```
# 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

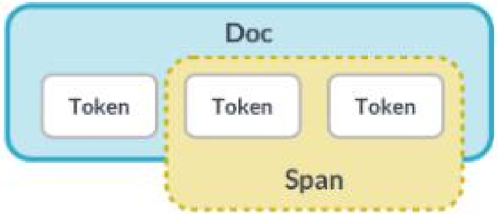


```
doc = nlp("Hello world!")

# 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



```
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
```

Lexical Attributes

```
doc = nlp("It costs $5.")
print("Index:   ", [token.i for token in doc])
print("Text:    ", [token.text for token in doc])

print("is_alpha:", [token.is_alpha for token in doc])
print("is_punct:", [token.is_punct for token in doc])
print("like_num:", [token.like_num for token in doc])
```

```
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]
```

Predicting Part-of-speech Tags

```
import spacy

# Load the small English model
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_)
```

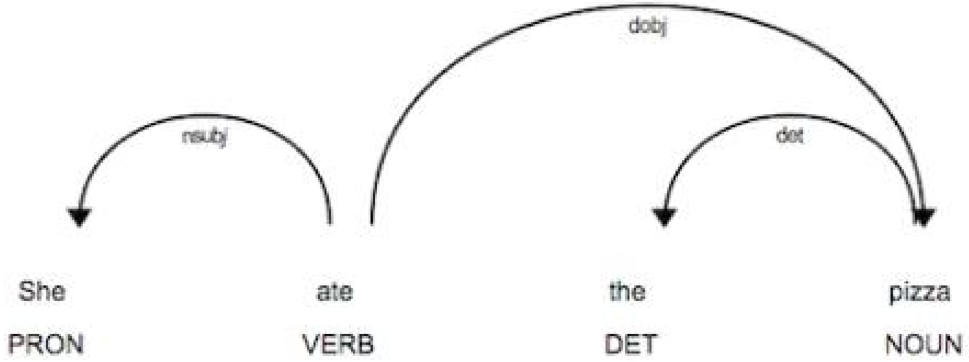
```
She PRON
ate VERB
the DET
pizza NOUN
```

Predicting Syntactic Dependencies

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

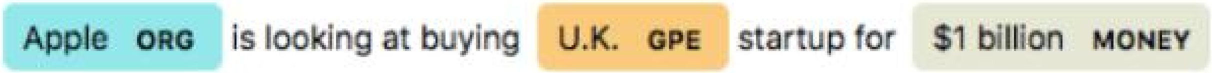
```
She PRON nsubj ate
ate VERB ROOT ate
the DET det pizza
pizza NOUN dobj ate
```

Dependency label scheme



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

Predicting Named Entities



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

```
# 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_)
Apple ORG
U.K. GPE
$1 billion MONEY
```

Tip: the spacy.explain method

Get quick definitions of the most common tags and labels.

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

spacy.explain("NNP")
'noun, proper singular'

spacy.explain("dobj")
'direct object'
```

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)

```
import spacy
# Import the Matcher
from spacy.matcher import Matcher
# Load a model 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", None, pattern)
# Process some text
doc = nlp("Upcoming iPhone X release date leaked")
# Call the matcher on the doc
matches = matcher(doc)
```

Using the Matcher (2)

```
# Call the matcher on the doc
doc = nlp("Upcoming iPhone X release date leaked")
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)
```

```
iPhone X
```

match_id: hash value of the pattern name

start: start index of matched span

end: end index of matched span

Matching lexical attributes

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

```
2018 FIFA World Cup:
```

Matching other token attributes

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

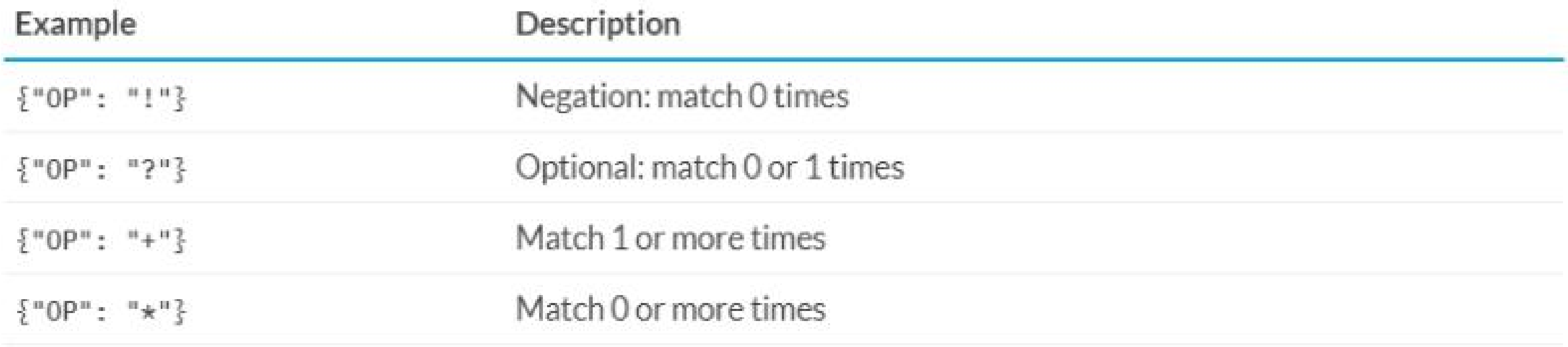
```
loved dogs
love cats
```

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)



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

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

doc = nlp(
    "After making the iOS update you won't notice a radical system-wide "
    "redesign: nothing like the aesthetic upheaval we got with iOS 7. Most of "
    "iOS 11's furniture remains the same as in iOS 10. But you will discover "
    "some tweaks once you delve a little deeper.")

# Write a pattern for full iOS versions ("iOS 7", "iOS 11", "iOS 10")
pattern = [{"TEXT": "iOS"}, {"IS_DIGIT": True}]
# Add the pattern to the matcher and apply the matcher to the doc
matcher.add("IOS_VERSION_PATTERN", None, 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)
```

```
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", None, 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)
```