Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 2

Representing Text

Nicola Capuano and Antonio Greco

DIEM – University of Salerno

Outline

- Tokenization
- Bag of Words Representation
- Token Normalization
- Stemming and Lemmatization
- Part of Speech Tagging
- Introducing spaCy

Tokenization

Prepare the Environment

For most exercises, we will use **Jupyter notebooks**

- Install the **Jupyter Extension for Visual Studio Code**
- `pip install jupyter`
- Create and activate a **virtual environment**:
  - `python -m venv .env`
  - `source .env/bin/activate`
- Alternative: **Google Colab notebooks**
  https://colab.research.google.com/

For this section we also need some **Python package…**

- `pip install numpy pandas`

Text Segmentation

The process of **dividing a text into meaningful units**

- **Paragraph Segmentation**: breaking a document into paragraphs
- **Sentence Segmentation**: breaking a paragraph into sentences
- **Word Segmentation**: breaking a sentence into words

Tokenization:

- A specialized form of text segmentation
- Involves breaking text into **small units** called **tokens**

What is a Token?

A unit of text that is treated as a **single, meaningful element**

- **Words**: the most common form of tokens
- **Punctuation Marks**: symbols that punctuate sentences (e.g., periods, commas)
- **Emojis**: visual symbols representing emotions or concepts
- **Numbers**: digits and numerical expressions
- **Sub-words**: smaller units within words, such as **prefixes** (re, pre, ...) or **suffixes** (ing, ...) that have intrinsic meaning
- **Phrases**: multiword **expressions** treated as single units (e.g., "ice cream")

Tokenizer

Idea: use **whitespaces** as the "delimiter" of words

- Not suitable for languages with a **continuous orthographic system** (Chinese, Japanese, Thai, etc.)

```
sentence = "Leonardo da Vinci began painting the Mona Lisa at the age of 51."
token_seq = sentence.split()
token_seq
```

```
['Leonardo',
 'da',
 'Vinci',
 'began',
 'painting',
 'the',
 'Mona',
 'Lisa',
 'at',
 'the',
 'age',
 'of',
 '51.']
```

Here a good tokenizer should separate **51** and **.**

We'll tackle **punctuation** and other challenges later

Bag of Words Representation

Turning Words into Numbers

One-hot Vectors

- A **vocabulary** lists all unique tokens that we want to keep track of
- Each word is represented by a vector with all **0**s except for a **1** corresponding to the index of the word

```
vocab = sorted(set(token_seq))
vocab
```

```
['51.',
 'Leonardo',
 'Lisa',
 'Mona',
 'Vinci',
 'age',
 'at',
 'began',
 'da',
 'of',
 'painting',
 'the']
```

```
import numpy as np
onehot_vectors = np.zeros((len(token_seq), len(vocab)), int)
for i, word in enumerate(token_seq):
  onehot_vectors[i, vocab.index(word)] = 1
onehot_vectors
```

```
array([[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

Turning Words into Numbers

```
import pandas as pd
pd.DataFrame(onehot_vectors, columns = vocab, index = token_seq)
```

	51.	Leonardo	Lisa	Mona	Vinci	age	at	began	da	of	painting	the
Leonardo	0	1	0	0	0	0	0	0	0	0	0	0
da	0	0	0	0	0	0	0	0	1	0	0	0
Vinci	0	0	0	0	1	0	0	0	0	0	0	0
began	0	0	0	0	0	0	0	1	0	0	0	0
painting	0	0	0	0	0	0	0	0	0	0	1	0
the	0	0	0	0	0	0	0	0	0	0	0	1
Mona	0	0	0	1	0	0	0	0	0	0	0	0
Lisa	0	0	1	0	0	0	0	0	0	0	0	0
at	0	0	0	0	0	0	1	0	0	0	0	0
the	0	0	0	0	0	0	0	0	0	0	0	1
age	0	0	0	0	0	1	0	0	0	0	0	0
of	0	0	0	0	0	0	0	0	0	1	0	0
51.	1	0	0	0	0	0	0	0	0	0	0	0

One-hot Vectors

Positive features:

- **No information is lost**: you can reconstruct the original document from a table of one-hot vectors

Negative Features:

- One-hot vectors are **super-sparse**, this results in a large table even for a short sentence
- Moreover, a language **vocabulary** typically contains at least **20,000** common words
- This number increases to **millions** when you consider **word variations** (conjugations, plurals, etc.) and **proper nouns** (names of people, places, organizations, etc.)

One-hot Vectors

Let's assume you have:

- A **million tokens** in your vocabulary
- A small library of **3.000** short books with **3.500** sentences each and **15** words per sentence

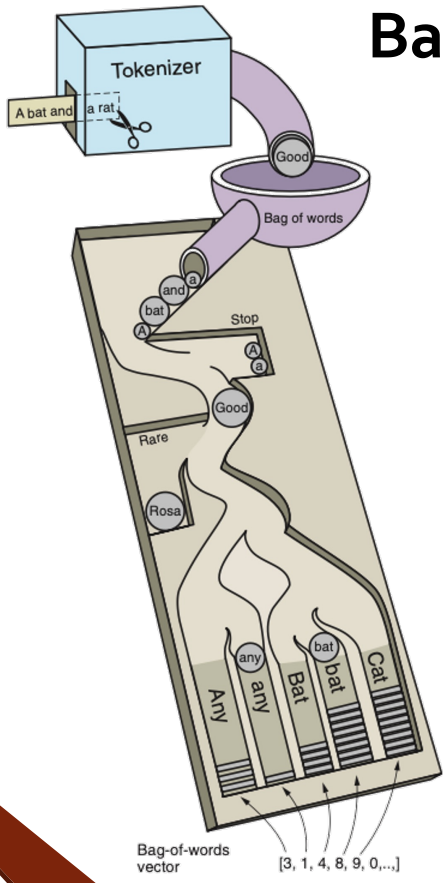
15 x 3.500 x 3.000 = 157.500.000 tokens

10^6 bits per tokens x **157.500.000** = **$157{,}5 \times 10^{12}$** bits

$157{,}5 \times 10^{12} / (8 \times 1024^4) \approx 17{,}9$ TB

Not practical!

Bag-of-Words



BoW: a vector obtained by **summing all the one-hot vectors**

- One bag for each sentence or short document
- Compresses a document down to a **single vector** representing its **essence**
- **Lossy transformation**: you can't reconstruct the initial text

Binary BoW: each word presence is marked as **1** or **0**, regardless of its frequency

Binary BoW: Example

Generating a **vocabulary** for a **text corpus**...

```
sentences = [
    "Leonardo da Vinci began painting the Mona Lisa at the age of 51.",
    "Leonardo was born in Vinci, Italy, in 1452.",
    "In addition to being a painter, Leonardo da Vinci was also a skilled engineer.",
    "Tennis is played on a rectangular court with a net in the middle.",
    "The four Grand Slam tournaments are the most prestigious events in tennis.",
    "A tennis match is typically played as a best of three or best of five sets."
]

all_words = " ".join(sentences).split()
vocab = sorted(set(all_words))
vocab
```

```
['1452.', '51.', 'A', 'Grand', 'In', 'Italy,', 'Leonardo', 'Lisa', 'Mona', 'Slam',
'Tennis', 'The', 'Vinci', 'Vinci,', 'a', 'addition', 'age', 'also', 'are', 'as', 'at',
'began', 'being', 'best', 'born', 'court', 'da', 'engineer.', 'events', 'five',
'four', 'in', 'is', 'match', 'middle.', 'most', 'net', 'of', 'on', 'or', 'painter,',
'painting', 'played', 'prestigious', 'rectangular', 'sets.', 'skilled', 'tennis',
'tennis.', 'the', 'three', 'to', 'tournaments', 'typically', 'was', 'with']
```

Binary BoW: Example

Generating a **BoW vector** for each text…

```
import numpy as np, pandas as pd
bags = np.zeros((len(sentences), len(vocab)), int)
for i, sentence in enumerate(sentences):
  for j, word in enumerate(sentences[i].split()):
    bags[i, vocab.index(word)] = 1
pd.DataFrame(bags, columns = vocab).transpose()
```

for display purposes only

	0	1	2	3	4	5
1452.	0	1	0	0	0	0
51.	1	0	0	0	0	0
A	0	0	0	0	0	1
Grand	0	0	0	0	1	0
In	0	0	1	0	0	0
Italy,	0	1	0	0	0	0
Leonardo	1	1	1	0	0	0
Lisa	1	0	0	0	0	0
Mona	1	0	0	0	0	0
Slam	0	0	0	0	1	0
Tennis	0	0	0	1	0	0
The	0	0	0	0	1	0
Vinci	1	0	1	0	0	0
Vinci,	0	1	0	0	0	0

	0	1	2	3	4	5
a	0	0	1	1	0	1
addition	0	0	1	0	0	0
age	1	0	0	0	0	0
also	0	0	1	0	0	0
are	0	0	0	0	1	0
as	0	0	0	0	0	1
at	1	0	0	0	0	0
began	1	0	0	0	0	0
being	0	0	1	0	0	0
best	0	0	0	0	0	1
born	0	1	0	0	0	0
court	0	0	0	1	0	0
da	1	0	1	0	0	0
engineer.	0	0	1	0	0	0

	0	1	2	3	4	5
events	0	0	0	0	1	0
five	0	0	0	0	0	1
four	0	0	0	0	1	0
in	0	1	0	1	1	0
is	0	0	0	1	0	1
match	0	0	0	0	0	1
middle.	0	0	0	1	0	0
most	0	0	0	0	1	0
net	0	0	0	1	0	0
of	1	0	0	0	0	1
on	0	0	0	1	0	0
or	0	0	0	0	0	1
painter,	0	0	1	0	0	0
painting	1	0	0	0	0	0

	0	1	2	3	4	5
played	0	0	0	1	0	1
prestigious	0	0	0	0	1	0
rectangular	0	0	0	1	0	0
sets.	0	0	0	0	0	1
skilled	0	0	1	0	0	0
tennis	0	0	0	0	0	1
tennis.	0	0	0	0	1	0
the	1	0	0	1	1	0
three	0	0	0	0	0	1
to	0	0	1	0	0	0
tournaments	0	0	0	0	1	0
typically	0	0	0	0	0	1
was	0	1	1	0	0	0
with	0	0	0	1	0	0

Binary BoW: Example

- You can see little **overlap** in word usage for some sentences…
- We can use this overlap to **compare documents** or search for **similar documents**

	0	1	2	3	4	5
1452.	0	1	0	0	0	0
51.	1	0	0	0	0	0
A	0	0	0	0	0	1
Grand	0	0	0	0	1	0
In	0	0	1	0	0	0
Italy,	0	1	0	0	0	0
Leonardo	1	1	1	0	0	0
Lisa	1	0	0	0	0	0
Mona	1	0	0	0	0	0
Slam	0	0	0	0	1	0
Tennis	0	0	0	1	0	0
The	0	0	0	0	1	0
Vinci	1	0	1	0	0	0
Vinci,	0	1	0	0	0	0

	0	1	2	3	4	5
a	0	0	1	1	0	1
addition	0	0	1	0	0	0
age	1	0	0	0	0	0
also	0	0	1	0	0	0
are	0	0	0	0	1	0
as	0	0	0	0	0	1
at	1	0	0	0	0	0
began	1	0	0	0	0	0
being	0	0	1	0	0	0
best	0	0	0	0	0	1
born	0	1	0	0	0	0
court	0	0	0	1	0	0
da	1	0	1	0	0	0
engineer.	0	0	1	0	0	0

	0	1	2	3	4	5
events	0	0	0	0	1	0
five	0	0	0	0	0	1
four	0	0	0	0	1	0
in	0	1	0	1	1	0
is	0	0	0	1	0	1
match	0	0	0	0	0	1
middle.	0	0	0	1	0	0
most	0	0	0	0	1	0
net	0	0	0	1	0	0
of	1	0	0	0	0	1
on	0	0	0	1	0	0
or	0	0	0	0	0	1
painter,	0	0	1	0	0	0
painting	1	0	0	0	0	0

	0	1	2	3	4	5
played	0	0	0	1	0	1
prestigious	0	0	0	0	1	0
rectangular	0	0	0	1	0	0
sets.	0	0	0	0	0	1
skilled	0	0	1	0	0	0
tennis	0	0	0	0	0	1
tennis.	0	0	0	0	1	0
the	1	0	0	1	1	0
three	0	0	0	0	0	1
to	0	0	1	0	0	0
tournaments	0	0	0	0	1	0
typically	0	0	0	0	0	1
was	0	1	1	0	0	0
with	0	0	0	1	0	0

Bag-of-Words Overlap

Measuring the bag of words overlap for two texts…

- we can get a (**good?**) estimate of **how similar they are in the words they use**
- and this is a (**good?**) estimate of **how similar they are in meaning**

Idea: use the **dot product**

```
#0: "Leonardo da Vinci began painting the Mona Lisa
#    at the age of 51."
#2: "In addition to being a painter, Leonardo da Vinci
#    was also a skilled engineer."
np.dot(bags[0], bags[2])
```

```
3
```

	0	2
51.	1	0
In	0	1
Leonardo	1	1
Lisa	1	0
Mona	1	0
Vinci	1	1
a	0	1
addition	0	1
age	1	0
also	0	1
at	1	0
began	1	0
being	0	1
da	1	1
engineer.	0	1
of	1	0
painter,	0	1
painting	1	0
skilled	0	1
the	1	0
to	0	1
was	0	1

Bag-of-Words Overlap

Measuring the bag of words overlap for two texts…

- we can get a (**good?**) estimate of **how similar they are in the words they use**
- and this is a (**good?**) estimate of **how similar they are in meaning**

Idea: use the **dot product**

```
#3: "Tennis is played on a rectangular court with a net in
#    the middle.",
#5: "A tennis match is typically played as a best of three
#    or best of five sets."
np.dot(bags[3], bags[5])
```

```
3
```

	3	5
A	0	1
Tennis	1	0
a	1	1
as	0	1
best	0	1
court	1	0
five	0	1
in	1	0
is	1	1
match	0	1
middle.	1	0
net	1	0
of	0	1
on	1	0
or	0	1
played	1	1
rectangular	1	0
sets.	0	1
tennis	0	1
the	1	0
three	0	1
typically	0	1
with	1	0

?

Bag-of-Words Overlap

Measuring the bag of words overlap for two texts…

- we can get a (**good?**) estimate of **how similar they are in the words they use**
- and this is a (**good?**) estimate of **how similar they are in meaning**

Idea: use the **dot product**

```
#2: "In addition to being a painter, Leonardo da Vinci
#    was also a skilled engineer."
#5: "A tennis match is typically played as a best of three
#    or best of five sets."
np.dot(bags[2], bags[5])
```

```
1
```

	2	5
A	0	1
In	1	0
Leonardo	1	0
Vinci	1	0
a	1	1
addition	1	0
also	1	0
as	0	1
being	1	0
best	0	1
da	1	0
engineer.	1	0
five	0	1
is	0	1
match	0	1
of	0	1
or	0	1
painter,	1	0
played	0	1
sets.	0	1
skilled	1	0
tennis	0	1
three	0	1
to	1	0
typically	0	1
was	1	0

Bag-of-Words Overlap

Measuring the bag of words overlap for two texts…

- we can get a (**good?**) estimate of **how similar they are in the words they use**
- and this is a (**good?**) estimate of **how similar they are in meaning**

Idea: use the **dot product**

```
#0: "Leonardo da Vinci began painting the Mona Lisa at the
#.   age of 51."
#1: "Leonardo was born in Vinci, Italy, in 1452."
np.dot(bags[0], bags[1])
```

```
1
```

	0	1
1452.	0	1
51.	1	0
Italy,	0	1
Leonardo	1	1
Lisa	1	0
Mona	1	0
Vinci	1	0
Vinci,	0	1
age	1	0
at	1	0
began	1	0
born	0	1
da	1	0
in	0	1
of	1	0
painting	1	0
the	1	0
was	0	1

] ?

Token Normalization

Tokenizer Improvement

Not only spaces are used to **separate words**

- \t (tab), \n (newline), \r (return), …
- **punctuation** (commas, periods, quotes, semicolons, dashes, …)

We can improve our tokenizer with **regular expressions**

```
import re
sentence = "Leonardo was born in Vinci, Italy, in 1452."
token_seq = re.split(r'[-\s.,;!?]+', sentence) # remove punctuation
token_seq = [token for token in token_seq if token] # remove void tokens
token_seq
```

```
['Leonardo', 'was', 'born', 'in', 'Vinci', 'Italy', 'in', '1452']
```

Tokenizer Improvement

But… what would happen with these **sentences**?

- The company's revenue for 2023 was $1,234,567.89.
- The CEO of the U.N. (United Nations) gave a speech.
- It's important to know the basics of A.I. (Artificial Intelligence).
- He didn't realize the cost was $12,345.67.
- Her new book, 'Intro to NLP (Natural Language Processing)', is popular.
- The temperature in Washington, D.C. can reach 100°F in the summer.

Tokenizers can easily become complex …

… but NLP libraries can help us (we will see them later)

Case Folding

Consolidates multiple "spellings" of a word that differ only in their **capitalization** under a **single token**

- **Tennis** → **tennis**, **A** → **a**, **Leonardo** → **leonardo**, …
- A.K.A. Case normalization

Advantages:

- Improves text **matching** and **recall** in search engines

Disadvantages:

- Loss of distinction between **proper** and **common** nouns
- May **alter** the original **meaning** (e.g., **US** → **us**)

Case Folding

```
import re
sentence = "Leonardo was born in Vinci, Italy, in 1452."
token_seq = re.split(r'[-\s.,;!?]+', sentence) # remove punctuation
token_seq = [token for token in token_seq if token] # remove void tokens
token_seq = [x.lower() for x in token_seq] # case folding
token_seq
```

```
['leonardo', 'was', 'born', 'in', 'vinci', 'italy', 'in', '1452']
```

A lot of **meaningful capitalization** is "normalized" away

- We can just normalize **first-word-in-sentence** capitalization …
- … but the first word can be a proper noun
- We can **first detect proper nouns** and then normalizing only the remaining words …
- … we will see **Named Entity Recognition** later

Stop Words

Common words that occur with a **high frequency** but carry **little information** about the meaning of a sentence

- Articles, prepositions, conjunctions, forms of the verb "to be", …
- These words are can be **filtered** out to reduce noise

```
stop_words = [
  "a", "about", "after", "all", "also", "an", "and", "any", "are", "as", "at", "be", "because",
  "been", "but", "by", "can", "co", "corp", "could", "for", "from", "had", "has", "have", "he",
  "her", "his", "if", "in", "inc", "into", "is", "it", "its", "last", "more", "most", "mr",
  "mrs", "ms", "mz", "no", "not", "of", "on", "one", "only", "or", "other", "out", "over", "s",
  "says", "she", "so", "some", "such", "than", "that", "the", "their", "there", "they", "this",
  "to", "up", "was", "we", "were", "when", "which", "who", "will", "with", "would"
]
sentence = "Leonardo was born in Vinci, Italy, in 1452."
token_seq = re.split(r'[-\s.,;!?]+', sentence) # remove punctuation
token_seq = [token for token in token_seq if token] # remove void tokens
token_seq = [x.lower() for x in token_seq] # case folding
token_seq = [x for x in token_seq if x not in stop_words] # remove stop words
token_seq
```

```
['leonardo', 'born', 'vinci', 'italy', '1452']
```

Stop Words

Disadvantages:

- Even though the stop words carry little information, thy can provide important relational information
  - **Mark reported to the CEO** → **Mark reported CEO**
  - **Suzanne reported as the CEO to the board** → **Suzanne reported CEO board**

Italian stop words:

a, affinché, agl', agli, ai, al, all', alla, alle, allo, anziché, avere, bensì, che, chi, cioè, come, comunque, con, contro, cosa, da, dacché, dagl', dagli, dai, dal, dall', dalla, dalle, dallo, degl', degli, dei, del, dell', delle, dello, di, dopo, dove, dunque, durante, e, egli, eppure, essere, essi, finché, fino, fra, giacché, gl', gli, grazie, i, il, in, inoltre, io, l', la, le, lo, loro, ma, mentre, mio, ne, neanche, negl', negli, nei, nel, nell', nella, nelle, nello, nemmeno, neppure, noi, nonché, nondimeno, nostro, o, onde, oppure, ossia, ovvero, per, perché, perciò, però, poiché, prima, purché, quand'anche, quando, quantunque, quasi, quindi, se, sebbene, sennonché, senza, seppure, si, siccome, sopra, sotto, su, subito, sugl', sugli, sui, sul, sull', sulla, sulle, sullo, suo, talché, tu, tuo, tuttavia, tutti, un, una, uno, voi, vostro

Putting All Together

```
stop_words = [
  "a", "about", "after", "all", "also", "an", "and", "any", "are", "as", "at", "be", "because",
  "been", "but", "by", "can", "co", "corp", "could", "for", "from", "had", "has", "have", "he",
  "her", "his", "if", "in", "inc", "into", "is", "it", "its", "last", "more", "most", "mr",
  "mrs", "ms", "mz", "no", "not", "of", "on", "one", "only", "or", "other", "out", "over", "s",
  "says", "she", "so", "some", "such", "than", "that", "the", "their", "there", "they", "this",
  "to", "up", "was", "we", "were", "when", "which", "who", "will", "with", "would"
]

sentences = [
    "Leonardo da Vinci began painting the Mona Lisa at the age of 51.",
    "Leonardo was born in Vinci, Italy, in 1452.",
    "In addition to being a painter, Leonardo da Vinci was also a skilled engineer.",
    "Tennis is played on a rectangular court with a net in the middle.",
    "The four Grand Slam tournaments are the most prestigious events in tennis.",
    "A tennis match is typically played as a best of three or best of five sets."
]

def tokenize (sentence):
  token_seq = re.split(r'[-\s.,;!?]+', sentence) # remove punctuation
  token_seq = [token for token in token_seq if token] # remove void tokens
  token_seq = [x.lower() for x in token_seq] # case folding
  token_seq = [x for x in token_seq if x not in stop_words] # remove stop words
  return token_seq

tok_sentences = [tokenize(sentence) for sentence in sentences]
all_tokens = [x for tokens in tok_sentences for x in tokens]
vocab = sorted(set(all_tokens))

bags = np.zeros((len(tok_sentences), len(vocab)), int)
for i, sentence in enumerate(tok_sentences):
  for j, word in enumerate(tok_sentences[i]):
    bags[i, vocab.index(word)] = 1
pd.DataFrame(bags, columns = vocab).transpose()
```

Putting All Together

	0	1	2	3	4	5
1452	0	1	0	0	0	0
51	1	0	0	0	0	0
addition	0	0	1	0	0	0
age	1	0	0	0	0	0
began	1	0	0	0	0	0
being	0	0	1	0	0	0
best	0	0	0	0	0	1
born	0	1	0	0	0	0
court	0	0	0	1	0	0
da	1	0	1	0	0	0
engineer	0	0	1	0	0	0
events	0	0	0	0	1	0

	0	1	2	3	4	5
five	0	0	0	0	0	1
four	0	0	0	0	1	0
grand	0	0	0	0	1	0
italy	0	1	0	0	0	0
leonardo	1	1	1	0	0	0
lisa	1	0	0	0	0	0
match	0	0	0	0	0	1
middle	0	0	0	1	0	0
mona	1	0	0	0	0	0
net	0	0	0	1	0	0
painter	0	0	1	0	0	0
painting	1	0	0	0	0	0

	0	1	2	3	4	5
played	0	0	0	1	0	1
prestigious	0	0	0	0	1	0
rectangular	0	0	0	1	0	0
sets	0	0	0	0	0	1
skilled	0	0	1	0	0	0
slam	0	0	0	0	1	0
tennis	0	0	0	1	1	1
three	0	0	0	0	0	1
tournaments	0	0	0	0	1	0
typically	0	0	0	0	0	1
vinci	1	1	1	0	0	0

Using NLTK

The **Natural Language Toolkit** is a popular NLP library

- `pip install nltk`
- It includes more refined **tokenizers**

```
import nltk

nltk.download('punkt') # download the Punkt tokenizer models
text = "Good muffins cost $3.88\nin New York. Please buy me two of them.\n\nThanks."

print(nltk.tokenize.word_tokenize(text)) # word tokenization
print(nltk.tokenize.sent_tokenize(text)) # sentence tokenization
```

```
['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.', 'Please', 'buy',
'me', 'two', 'of', 'them', '.', 'Thanks', '.']
['Good muffins cost $3.88\nin New York.', 'Please buy me two of them.', 'Thanks.']
```

Using NLTK

The **Natural Language Toolkit** is a popular NLP library

- `pip install nltk`
- It includes extended **stop-word lists** for many languages

```
import nltk

nltk.download('stopwords') # download the stop words corpus
text = "This is an example sentence demonstrating how to remove stop words using NLTK."

tokens = nltk.tokenize.word_tokenize(text)
stop_words = set(nltk.corpus.stopwords.words('english'))
filtered_tokens = [x for x in tokens if x not in stop_words]

print("Original Tokens:", tokens)
print("Filtered Tokens:", filtered_tokens)
```

```
Original Tokens: ['This', 'is', 'an', 'example', 'sentence', 'demonstrating', 'how', 'to',
'remove', 'stop', 'words', 'using', 'NLTK', '.']
Filtered Tokens: ['This', 'example', 'sentence', 'demonstrating', 'remove', 'stop', 'words',
'using', 'NLTK', '.']
```

Stemming and Lemmatization

Stemming

Identifies a **common stem** among various forms of a word

- E.g., **Housing** and **houses** share the same stem: **house**

Function:

- **Removes suffixes** to combine words with similar meanings under the same token (stem)
- A **stem** isn't required to be a properly spelled word: **Relational**, **Relate**, **Relating** all stemmed to **Relat**

Benefits:

- Helps **generalize your vocabulary**
- Important for **information retrieval** (improves recall)

A Naïve Stemmer

```
def simple_stemmer(word):
    suffixes = ['ing', 'ly', 'ed', 'ious', 'ies', 'ive', 'es', 's', 'ment']
    for suffix in suffixes:
        if word.endswith(suffix):
            return word[:-len(suffix)]
    return word

words = ["running", "happily", "stopped", "curious", "cries", "effective", "runs", "management"]
[simple_stemmer(word) for word in words]
```

```
['runn', 'happi', 'stopp', 'cur', 'cr', 'effect', 'run', 'manage']
```

Very basic example…

- Doesn't handle **exceptions**, **multiple suffixes**, or words that require more **complex modifications**

NPL libraries include more accurate stemmers

Porter Stemmer

- **Step 1a:** Remove **s** and **es** endings
  - **cats** → **cat**, **buses** → **bus**
- **Step 1b:** Remove **ed**, **ing**, and **at** endings
  - **hoped** → **hope**, **running** → **run**
- **Step 1c:** Change **y** to **i** if preceded by a consonant
  - **happy** → **happi**, **cry** → **cri**
- **Step 2:** Remove "nounifying" endings such as **ational**, **tional**, **ence**, and **able**
  - **relational** → **relate**, **dependence** → **depend**

Porter Stemmer

- **Step 3:** Remove adjective endings such as **icate**, **ful**, and **alize**
  - **duplicate** → **duplic**, **hopeful** → **hope**
- **Step 4:** Remove adjective and noun endings such as **ive**, **ible**, **ent**, and **ism**
  - **effective** → **effect**, **responsible** → **respons**
- **Step 5a:** Remove stubborn **e** endings
  - **probate** → **probat**, **rate** → **rat**
- **Step 5b:** Reduce trailing **double** consonants ending in **l** to a single **l**
  - **controlling** → **controll** → **control**, **rolling** → **roll** → **rol**

Using NLTK Porter Stemmer

```
import nltk

texts = [
    "I love machine learning.",
    "Deep learning is a subset of machine learning.",
    "Natural language processing is fun.",
    "Machine learning can be used for various tasks."
]

nltk.download('punkt') # Download the Punkt tokenizer models
stemmer = nltk.stem.PorterStemmer() # Initialize the Porter Stemmer

stemmed_texts = []
for text in texts:
    tokens = nltk.tokenize.word_tokenize(text.lower())
    stemmed_tokens = [stemmer.stem(token) for token in tokens if token.isalpha()]
    stemmed_texts.append(' '.join(stemmed_tokens))

for text in stemmed_texts:
    print(text)
```

```
i love machin learn
deep learn is a subset of machin learn
natur languag process is fun
machin learn can be use for variou task
```

Snowball Project

Provides stemming algorithms for several languages

- https://snowballstem.org/

Italian stemming examples:

word	stem	word	stem
abbandonata	abbandon	pronto	pront
abbandonate	abbandon	pronuncerà	pronunc
abbandonati	abbandon	pronuncia	pronunc
abbandonato	abbandon	pronunciamento	pronunc
abbandonava	abbandon	pronunciare	pronunc
abbandonerà	abbandon	pronunciarsi	pronunc
abbandoneranno	abbandon	pronunciata	pronunc
abbandonerò	abbandon	pronunciate	pronunc
abbandono	abband	pronunciato	pronunc
abbandonò	abbandon	pronunzia	pronunz
abbaruffato	abbaruff	pronunziano	pronunz
abbassamento	abbass	pronunziare	pronunz
abbassando	abbass	pronunziarle	pronunz
abbassandola	abbass	pronunziato	pronunz
abbassandole	abbass	pronunzio	pronunz
abbassar	abbass	pronunziò	pronunz
abbassare	abbass	propaga	propag
abbassarono	abbass	propagamento	propag

Supported in NLTK

Lemmatization

Determines the **dictionary form** (lemma) of a word

- Considers the **context** of the word
- Uses **dictionaries** and **language rules** (**morphological analysis**)
- Requires to **prior identify the part of speech (PoS)** of the word (verb, noun, adjective, …)

Lemmatization vs Stemming:

- Lemmatization always produces a **valid lemma**; stemming may produce roots that are not actual words
- Lemmatization is **slower**; stemming is faster

Lemmatization

Lemmatization:

- **went** → **go**
- **ate** → **eat**
- **better** → **good**
- **best** → **good**
- **children** → **child**
- **andato** → **andare**
- **migliore** → **buono**
- **corre** → **correre**
- **connessione** → **connettere**

Stemming:

- **went** → **went**
- **ate** → **at**
- **better** → **better**
- **best** → **best**
- **children** → **children**
- **andato** → **andat**
- **migliore** → **miglior**
- **corre** → **corr**
- **connessione** → **conness**

Part of Speech Tagging

Part of Speech Tagging

- **PoS tagging** is the operation of labeling tokens with respect to their **lexical category**
  - Noun, Adjective, Article, Verb, Preposition, …
- Each category in turn admits different **subcategories** and **morphological variants**
  - **Gender** and **number** in the case of nouns
  - **Tense**, **person**, and **number** in the case of verbs
- PoS tagging is a **prerequisite for lemmatization**
  - It is also important for many other NLP tasks (parsing, information extraction, …)

Main PoS Tags

POS Tag	Description	Example(s)
ADJ	Adjective, describes or modifies a noun	"big", "yellow", "quick"
ADP	Adposition, shows relationship between a noun or pronoun and another word	"in", "on", "at"
ADV	Adverb, modifies a verb, an adjective, or another adverb	"quickly", "very", "well"
AUX	Auxiliary verb, used to form tenses, moods, aspects, and voices	"is", "have", "will"
CCONJ	Coordinating conjunction, links words, phrases, or clauses of equal rank	"and", "but", "or"
DET	Determiner, specifies a noun in terms of quantity, possession, specificity, etc.	"the", "a", "an", "some"
INTJ	Interjection, expresses emotion or a spontaneous reaction	"oh", "wow", "ouch"
NOUN	Noun, person, place, thing, or idea	"dog", "city", "happiness"
NUM	Numeral, expresses a number	"one", "two", "first"
PRON	Pronoun, replaces a noun	"he", "she", "they"
PROPN	Proper noun, a specific name of a person, place, or organization	"John", "Paris", "Google"
PUNCT	Punctuation mark	".", ",", "!"
SCONJ	Subordinating conjunction, introduces a subordinate clause	"because", "if", "while"
SYM	Symbol	"%", "&", "$"
VERB	Verb, describes an action, state, or occurrence	"run", "is", "seems"

Specific PoS Tags

Tag	Description	Example
CC	conjunction, coordinating	*and, or, but*
CD	cardinal number	*five, three, 13%*
DT	determiner	*the, a, these*
EX	existential there	*there were six boys*
FW	foreign word	*mais*
IN	conjunction, subordinating or preposition	*of, on, before, unless*
JJ	adjective	*nice, easy*
JJR	adjective, comparative	*nicer, easier*
JJS	adjective, superlative	*nicest, easiest*
LS	list item marker	
MD	verb, modal auxillary	*may, should*
NN	noun, singular or mass	*tiger, chair, laughter*
NNS	noun, plural	*tigers, chairs, insects*
NNP	noun, proper singular	*Germany, God, Alice*
NNPS	noun, proper plural	*we met two Christmases ago*
PDT	predeterminer	*both his children*
POS	possessive ending	*'s*
PRP	pronoun, personal	*me, you, it*
PRP$	pronoun, possessive	*my, your, our*
RB	adverb	*extremely, loudly, hard*
RBR	adverb, comparative	*better*

Specific PoS Tags

Tag	Description	Example
RBS	adverb, superlative	*best*
RP	adverb, particle	*about, off, up*
SYM	symbol	*%*
TO	infinitival to	*what to do?*
UH	interjection	*oh, oops, gosh*
VB	verb, base form	*think*
VBZ	verb, 3rd person singular present	*she thinks*
VBP	verb, non-3rd person singular present	*I think*
VBD	verb, past tense	*they thought*
VBN	verb, past participle	*a sunken ship*
VBG	verb, gerund or present participle	*thinking is fun*
WDT	*wh*-determiner	*which, whatever, whichever*
WP	*wh*-pronoun, personal	*what, who, whom*
WP$	*wh*-pronoun, possessive	*whose, whosever*
WRB	*wh*-adverb	*where, when*
.	punctuation mark, sentence closer	*.;?**
,	punctuation mark, comma	*,*
:	punctuation mark, colon	*:*
(	contextual separator, left paren	*(*
)	contextual separator, right paren	*)*

PoS Tagging Algorithms

PoS tagging is a complex task due to the **ambiguity of natural language**

- The same term can represent **different parts of speech**

Example:

- The word **light** can be a **noun** or a **verb**:
  - In the sentence **The light is bright**, it is a **noun**
  - In **Please light the candle**, it is a **verb**
- Only the **context** can help to discriminate

PoS Tagging Algorithms

PoS Tagging Algorithms

- Use **dictionaries** of words annotated with all their **possible PoS**
- Use **statistical models** to choose the most appropriate tag for a token based on the **surrounding context**

Simple models assign the PoS tag **based on the PoS tag assigned to the previous token**

- let $w_1, \ldots, w_n$ be the sequence of tokens to be analyzed…
- … assign to the token w_i the tag t_i which maximizes the probability:

$$P(t_i) = P(w_i|t_i)\, P(t_i|t_{i-1})$$

where probabilities are learned on **corpora of annotated text**

Example

"The light is bright"

After determining that t_1 = **DET** (**article**) we estimate:

- $P(t_2 = \text{NOUN}) = P(\text{light}|\text{NOUN}) \times P(\text{NOUN}|\text{DET})$
- $P(t_2 = \text{VERB}) = P(\text{light}|\text{VERB}) \times P(\text{VERB}|\text{DET})$

where:

- P(light|NOUN) is the **marginal probability** that light is a **noun**
- P(NOUN|DET) is the **conditional probability** that a token is a **noun** when the preceding is an **article**
- P(light|VERB) is the **marginal probability** that light is a **verb**
- P(VERB|DET) is the **conditional probability** that a token is a **verb** when the preceding is an **article**

Morphological Analysis

More complex models consider **longer sequences of tokens** and use morphological analysis

Morphological Analysis

- Analyze the **grammatical structure** of the word
- Identify **inflectional morphemes** (suffixes that indicate verb conjugation or noun declension)
- **Example**: **Running is good for health**
  - The word **running** is recognized as a verb in the gerund form based on the suffix **-ing**
- Recognizing inflectional morphemes allows estimating the conditional probability for an **unknown word**

POS Tagging with NLTK

NLTK includes several **pre-trained POS tagger models**

```
import nltk

nltk.download('punkt') # Download the Punkt tokenizer models
nltk.download('averaged_perceptron_tagger') # Download a pre-trained tagger model

sentence = "The quick brown foxes are jumping over the lazy dogs."
tokens = nltk.tokenize.word_tokenize(sentence)
pos_tags = nltk.pos_tag(tokens)

pos_tags
```

```
[('The', 'DT'),
 ('quick', 'JJ'),
 ('brown', 'NN'),
 ('foxes', 'NNS'),
 ('are', 'VBP'),
 ('jumping', 'VBG'),
 ('over', 'IN'),
 ('the', 'DT'),
 ('lazy', 'JJ'),
 ('dogs', 'NNS'),
 ('.', '.')]
```

Lemmatization with NLTK

POS Tagging enables Lemmatization

- To obtain the lemma **we must know the POS tag** of a token first
- NLTK also include some **lemmatizer models**

Wordnet Lemmatizer

- Uses the **Wordnet lexicon**
- Includes about **155.000 words** divided in four **lexical categories**
  - **Nouns, Verbs, Adjectives, Adverbs**
- **https://wordnet.princeton.edu/**

Lemmatization with NLTK

```
import nltk
from nltk.corpus import wordnet

nltk.download('wordnet') # Download the WordNet corpus
lemmatizer = nltk.stem.WordNetLemmatizer() # Initialize the WordNet lemmatizer

# Function to convert NLTK POS tags to WordNet POS tags
def wordnet_pos(tag):
    if tag.startswith('J'): return wordnet.ADJ
    if tag.startswith('V'): return wordnet.VERB
    if tag.startswith('N'): return wordnet.NOUN
    if tag.startswith('R'): return wordnet.ADV
    return wordnet.NOUN

lemmatized_tokens = []
for token, pos in pos_tags:
    lemmatized_token = lemmatizer.\
        lemmatize(token, pos = wordnet_pos(pos))
    lemmatized_tokens.append((token, pos, lemmatized_token))

lemmatized_tokens
```

```
[('The', 'DT', 'The'),
 ('quick', 'JJ', 'quick'),
 ('brown', 'NN', 'brown'),
 ('foxes', 'NNS', 'fox'),
 ('are', 'VBP', 'be'),
 ('jumping', 'VBG', 'jump'),
 ('over', 'IN', 'over'),
 ('the', 'DT', 'the'),
 ('lazy', 'JJ', 'lazy'),
 ('dogs', 'NNS', 'dog'),
 ('.', '.', '.')]
```

Introducing spaCy

spaCy

spaCy is a free, open-source **library for NLP** in Python

- Supports **25 languages** (including **Italian**)
- Different models for each language
  - **it_core_news_sm** (12 MB)
  - **it_core_news_md** (40 MB)
  - **it_core_news_lg** (541 MB)
  - **en_core_web_sm** (12 MB)
  - **en_core_web_md** (40 MB)
  - **en_core_web_lg** (560 MB)

Installation:

```
$ pip install -U pip setuptools wheel
$ pip install -U spacy
$ python -m spacy download en_core_web_sm
$ python -m spacy download it_core_news_sm
```

spaCy

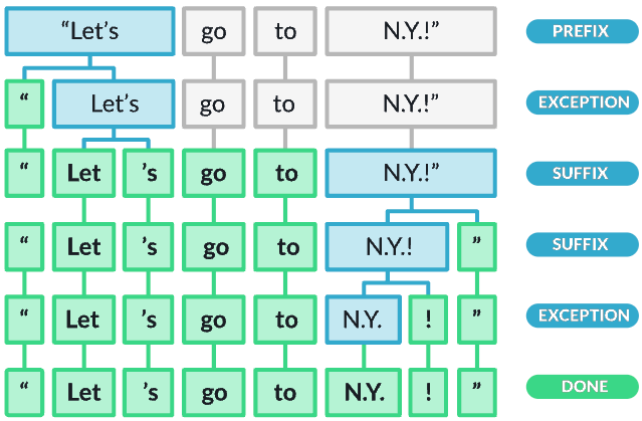
https://spacy.io/

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.
Entity Linking (EL)	Disambiguating textual entities to unique identifiers in a knowledge base.
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.

Tokenization

- First, the text is **split on whitespace** characters
- Then, on each substring, the following **checks** are performed:
  - Does the substring match a tokenizer **exception rule**?
  - Can a prefix, **suffix or infix** be split off?
- Matching rules are applied, and the tokenizer continues its loop



Tokenization

```
sentences = [
    "Leonardo da Vinci began painting the Mona Lisa at the age of 51.",
    "Leonardo was born in Vinci, Italy, in 1452.",
    "Tennis is played on a rectangular court with a net in the middle.",
    "The four Grand Slam tournaments are the most prestigious events in tennis.",
    "The company's revenue for 2023 was $1,234,567.89.",
    "The CEO of the U.N. (United Nations) gave a speech.",
    "It's important to know the basics of A.I. (Artificial Intelligence).",
    "He didn't realize the cost was $12,345.67.",
    "The temperature in Washington, D.C. can reach 100°F in the summer."
]

import spacy
nlp = spacy.load("en_core_web_sm") # load the language model

for sentence in sentences:
  doc = nlp(sentence)
  tokens = [tok.text for tok in doc]
  print(tokens)
```

```
['Leonardo', 'da', 'Vinci', 'began', 'painting', 'the', 'Mona', 'Lisa', 'at', 'the', 'age', 'of', '51', '.']
['Leonardo', 'was', 'born', 'in', 'Vinci', ',', 'Italy', ',', 'in', '1452', '.']
['Tennis', 'is', 'played', 'on', 'a', 'rectangular', 'court', 'with', 'a', 'net', 'in', 'the', 'middle', '.']
['The', 'four', 'Grand', 'Slam', 'tournaments', 'are', 'the', 'most', 'prestigious', 'events', 'in', 'tennis', '.']
['The', 'company', "'s", 'revenue', 'for', '2023', 'was', '$', '1,234,567.89', '.']
['The', 'CEO', 'of', 'the', 'U.N.', '(', 'United', 'Nations', ')', 'gave', 'a', 'speech', '.']
['It', "'s", 'important', 'to', 'know', 'the', 'basics', 'of', 'A.I.', '(', 'Artificial', 'Intelligence', ')', '.']
['He', 'did', "n't", 'realize', 'the', 'cost', 'was', '$', '12,345.67', '.']
['The', 'temperature', 'in', 'Washington', ',', 'D.C.', 'can', 'reach', '100', '°', 'F', 'in', 'the', 'summer', '.']
```

Additional Token Info

```
import spacy
nlp = spacy.load("en_core_web_sm") # load the language model

def token_info(sentence):
  doc = nlp(sentence)
  print([tok.text for tok in doc]) # tokens
  print([tok.text for tok in doc if not tok.is_stop]) # tokens without sotp words
  print([tok.pos_ for tok in doc]) # PoS tag (generic)
  print([tok.tag_ for tok in doc]) # PoS tag (specific)
  print([tok.lemma_ for tok in doc]) # lemmas
  print()

token_info("Leonardo da Vinci began painting the Mona Lisa at the age of 51.")
token_info("The company's revenue for 2023 was $1,234,567.89.")
```

```
['Leonardo', 'da', 'Vinci', 'began', 'painting', 'the', 'Mona', 'Lisa', 'at', 'the', 'age', 'of', '51', '.']
['Leonardo', 'da', 'Vinci', 'began', 'painting', 'Mona', 'Lisa', 'age', '51', '.']
['PROPN', 'PROPN', 'PROPN', 'VERB', 'VERB', 'DET', 'PROPN', 'PROPN', 'ADP', 'DET', 'NOUN', 'ADP', 'NUM', 'PUNCT']
['NNP', 'NNP', 'NNP', 'VBD', 'VBG', 'DT', 'NNP', 'NNP', 'IN', 'DT', 'NN', 'IN', 'CD', '.']
['Leonardo', 'da', 'Vinci', 'begin', 'paint', 'the', 'Mona', 'Lisa', 'at', 'the', 'age', 'of', '51', '.']

['The', 'company', "'s", 'revenue', 'for', '2023', 'was', '$', '1,234,567.89', '.']
['company', 'revenue', '2023', '$', '1,234,567.89', '.']
['DET', 'NOUN', 'PART', 'NOUN', 'ADP', 'NUM', 'AUX', 'SYM', 'NUM', 'PUNCT']
['DT', 'NN', 'POS', 'NN', 'IN', 'CD', 'VBD', '$', 'CD', '.']
['the', 'company', "'s", 'revenue', 'for', '2023', 'be', '$', '1,234,567.89', '.']
```

Other Token Attributes

A spaCy token has many **other attributes**

- Full list here: https://spacy.io/api/token#attributes

We can also ask spaCy to **explain a tag**...

```
import spacy
nlp = spacy.load("en_core_web_sm") # load the language model

print(spacy.explain("PROPN")) # explain a generic PoS tag
print(spacy.explain("NNP")) # explain a specific PoS tag
```

```
proper noun
noun, proper singular
```

Sentence Splitting

spaCy can easily determine sentence boundaries

```
import spacy

nlp = spacy.load("en_core_web_sm")
text = "This is the first sentence. Here is another one! Can you see how it works?"
doc = nlp(text)

for sent in doc.sents:
    print(sent.text)
```

```
This is the first sentence.
Here is another one!
Can you see how it works?
```

Creating BoWs with spaCy

```
sentences = [
    "Leonardo da Vinci began painting the Mona Lisa at the age of 51.",
    "Leonardo was born in Vinci, Italy, in 1452.",
    "In addition to being a painter, Leonardo da Vinci was also a skilled engineer.",
    "Tennis is played on a rectangular court with a net in the middle.",
    "The four Grand Slam tournaments are the most prestigious events in tennis.",
    "A tennis match is typically played as a best of three or best of five sets."
]

import spacy, numpy as np, pandas as pd
nlp = spacy.load("en_core_web_sm") # load the language model

def tokenize (sentence):
  doc = nlp(sentence)
  return [tok.lemma_ for tok in doc if not tok.is_stop and not tok.is_punct]

tok_sentences = [tokenize(sentence) for sentence in sentences]
all_tokens = [x for tokens in tok_sentences for x in tokens]
vocab = sorted(set(all_tokens))

bags = np.zeros((len(tok_sentences), len(vocab)), int)
for i, sentence in enumerate(tok_sentences):
  for j, word in enumerate(tok_sentences[i]):
    bags[i, vocab.index(word)] = 1
pd.DataFrame(bags, columns = vocab).transpose()
```

Creating BoWs with spaCy

With lemmatization:

- Each term is reduced to its lemma
- Smaller vocabulary
- Preserves almost the same discriminative information

	0	1	2	3	4	5
1452	0	1	0	0	0	0
51	1	0	0	0	0	0
Grand	0	0	0	0	1	0
Italy	0	1	0	0	0	0
Leonardo	1	1	1	0	0	0
Lisa	1	0	0	0	0	0
Mona	1	0	0	0	0	0
Slam	0	0	0	0	1	0
Tennis	0	0	0	1	0	0
Vinci	1	1	1	0	0	0
addition	0	0	1	0	0	0
age	1	0	0	0	0	0
bear	0	1	0	0	0	0
begin	1	0	0	0	0	0
court	0	0	0	1	0	0
da	1	0	1	0	0	0

	0	1	2	3	4	5
engineer	0	0	1	0	0	0
event	0	0	0	0	1	0
good	0	0	0	0	0	1
match	0	0	0	0	0	1
middle	0	0	0	1	0	0
net	0	0	0	1	0	0
paint	1	0	0	0	0	0
painter	0	0	1	0	0	0
play	0	0	0	1	0	1
prestigious	0	0	0	0	1	0
rectangular	0	0	0	1	0	0
set	0	0	0	0	0	1
skilled	0	0	1	0	0	0
tennis	0	0	0	0	1	1
tournament	0	0	0	0	1	0
typically	0	0	0	0	0	1

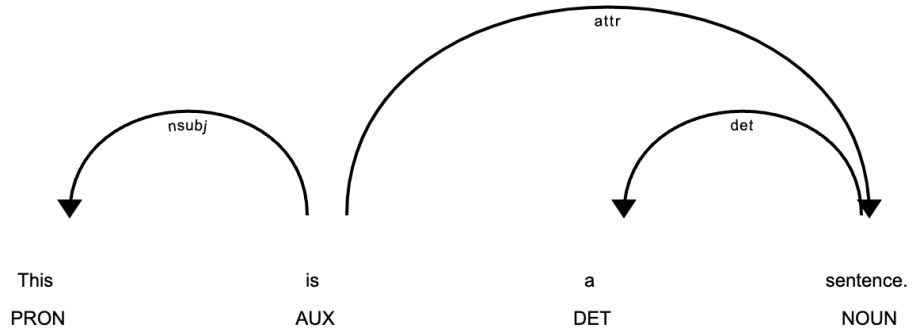
Dependency Parsing

spaCy can determine **syntactic dependencies** between tokens (like subject, object, etc.)

```
import spacy
from spacy import displacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("This is a sentence.")
displacy.render(doc, style="dep", jupyter=True)
```

Displacy is the built-in spaCy visualizer



Named Entity Recognition

A **named entity** is a real-world object that's assigned a name (a person, a country, a product, a book title, …)

- spaCy can recognize various types of named entities

```
from spacy import displacy

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

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

```
Apple 0 5 ORG
U.K. 27 31 GPE
$1 billion 44 54 MONEY
```

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

Named Entity Recognition

Entity Labels

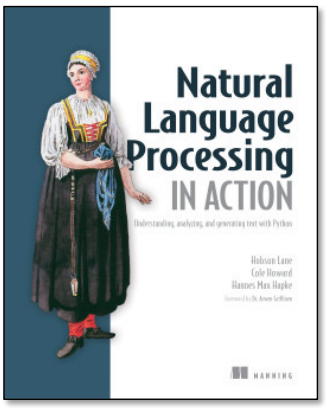
- **CARDINAL:** Numerals that don't refer to a specific quantity of something, like "three", "100", "thousands".
- **DATE:** References to dates, such as exact days, months, or years, e.g., "January 1st", "2019", "Monday".
- **EVENT:** Named occurrences of events like "Olympics", "World War II", or "Super Bowl".
- **FAC (Facility):** Buildings, airports, highways, bridges, etc., such as "Eiffel Tower", "Golden Gate Bridge".
- **GPE (Geo-political Entity):** Countries, cities, states, or regions, e.g., "France", "New York", "Asia".
- **LANGUAGE:** Any mention of a language, e.g., "English", "Spanish", "Mandarin".
- **LAW:** Legal documents, laws, treaties, or regulations, such as "First Amendment", "Treaty of Versailles".
- **LOC (Location):** Non-political locations such as mountain ranges, bodies of water, and general areas, e.g., "Sahara Desert", "Pacific Ocean".
- **MONEY:** Monetary values, including currency symbols, e.g., "$10", "€500", "five dollars".
- **NORP:** Nationalities, religious, or political groups, e.g., "American", "Buddhists", "Republicans".
- **ORDINAL:** Words indicating order or rank, such as "first", "second", "10th".
- **ORG (Organization):** Companies, institutions, agencies, etc., e.g., "Google", "United Nations", "NASA".
- **PERCENT:** Percentage values, e.g., "50%", "twelve percent".
- **PERSON:** Names of people, including fictional, e.g., "John", "Marie Curie", "Sherlock Holmes".
- **PRODUCT:** Objects, vehicles, foods, etc., e.g., "iPhone", "Tesla Model S", "Big Mac".
- **QUANTITY:** Measurable quantities, including units of measurement, e.g., "10 kilograms", "two liters".
- **TIME:** Specific times or periods, e.g., "10:30 AM", "two hours", "last night".

References

Natural Language Processing IN ACTION

Understanding, analyzing, and generating text with Python

Chapter 2 (2.3 excluded)



Further Readings...

- **spaCy 101:** Everything you need to know

  https://spacy.io/usage/spacy-101

- **NLTK Documentation**

  https://www.nltk.org/

Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 2

Representing Text

Nicola Capuano and Antonio Greco

DIEM – University of Salerno