COMP7630 – Web Intelligence and its Applications

Natural Language Processing pipelines

Valentino Santucci

(valentino.santucci@unistrapg.it)

Outline

- <u>NLP and NLP pipelines</u>
- The Spacy library
- Pretrained LLM

Natural Language Processing (NLP)

- NLP involves using computational techniques to understand, analyze, and generate human language.

- Some NLP tasks: text classification, sentiment analysis, topic modeling, machine translation, text generation, text summarization, …

- NLP techniques are used in WI for tasks such as:
  - Extracting structured data from unstructured text found on web pages
  - Identifying named entities and extracting information about them
  - Analyzing the sentiment and emotion expressed in web content
  - Understanding the intent behind user queries and search phrases
  - Summarizing web pages and articles
  - …

NLP pipeline

- It is possible to identify some basic processing steps which are required by many complex NLP and WI tasks

- These basic steps form a NLP pipeline and they can vary depending on the task at hand, but generally a NLP pipeline includes some combination of the following steps:
  - Tokenization
  - Sentence Segmentation
  - Part-of-Speech Tagging
  - Lemmatization
  - Stemming
  - Morphological Analysis
  - Dependency Parsing
  - Named Entity Recognition
  - Token Vectorization
  - …

Tokenization

- Divide a text into tokens, i.e., words, punctuation marks, etc.
- This is done by applying rules specific to each language.
  - For example, punctuation at the end of a sentence should be split off – whereas “U.K.” should remain one token.

- Example

"Apple is looking at buying U.K. startup for $1 billion"

0	1	2	3	4	5	6	7	8	9	10
Apple	is	looking	at	buying	U.K.	startup	for	$	1	billion

Sentence Segmentation

- Segment a text into sentences
- <u>Example</u>

"Alan Mathison Turing (23 June 1912 – 7 June 1954) was an English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist. Turing was highly influential in the development of theoretical computer science, providing a formalisation of the concepts of algorithm and computation with the Turing machine, which can be considered a model of a general-purpose computer. He is widely considered to be the father of theoretical computer science and artificial intelligence."

<u>Sentence #1</u>

"Alan Mathison Turing (23 June 1912 – 7 June 1954) was an English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist."

<u>Sentence #2</u>

"Turing was highly influential in the development of theoretical computer science, providing a formalisation of the concepts of algorithm and computation with the Turing machine, which can be considered a model of a general-purpose computer."

<u>Sentence #3</u>

"He is widely considered to be the father of theoretical computer science and artificial intelligence."

Part-of-speech Tagging

- The process of marking up a token in a text as corresponding to a particular part of speech, based on both its definition and its context

TEXT	POS
Apple	PROPN
is	AUX
looking	VERB
at	ADP
buying	VERB
U.K.	PROPN
startup	NOUN
for	ADP
$	SYM
1	NUM
billion	NUM

Text Normalization

- Text normalization is the process of transforming text into a consistent and standardized format. It involves various techniques to reduce words to their base or root form, enhancing the efficiency of text analysis.

- Two possibilities:
  - Lemmatization
  - Stemming

- … plus their combination which is sometimes useful:
  - analyze the text where every *word* is replaced with *stem*(*lemma*(*word*))

Lemmatization

- Extract the lemma of a word, i.e. the base form of a word
- Base form = dictionary form
- Useful in order to group up together tokens with the same "meaning"

TEXT	LEMMA
Apple	apple
is	be
looking	look
at	at
buying	buy
U.K.	u.k.
startup	startup
for	for
$	$
1	1
billion	billion

Stemming

- The process of reducing a word to its root form
- Considering the stems or the lemmas of the words allows to group together words which have the same semantic meaning
- Example: stem("runs") = stem("running") = "run"
- Stemming is a crude heuristic process that chops off the end of a word using a set of predefined rules. It does not consider the context of the word and often results in non-real words, known as stemmed words. The Porter stemmer is an example of an algorithm used for stemming.
- Lemmatization, on the other hand, is a more sophisticated process that involves understanding the context of a word and reducing it to its base form using a dictionary or morphological analysis. This results in real words, known as lemmas. Lemmatization is more accurate than stemming but also more computationally expensive.
- Example:
  - Word = *composition*
  - Lemma(*composition*) = *compose*
  - Stem(*composition*) = *compos*
  - Stem(Lemma(*composition*)) = *compos*

Morphological Analysis

- Inflectional morphology is the process by which a root form of a word is modified by adding prefixes or suffixes that specify its grammatical function but do not change its part-of-speech.

- Example

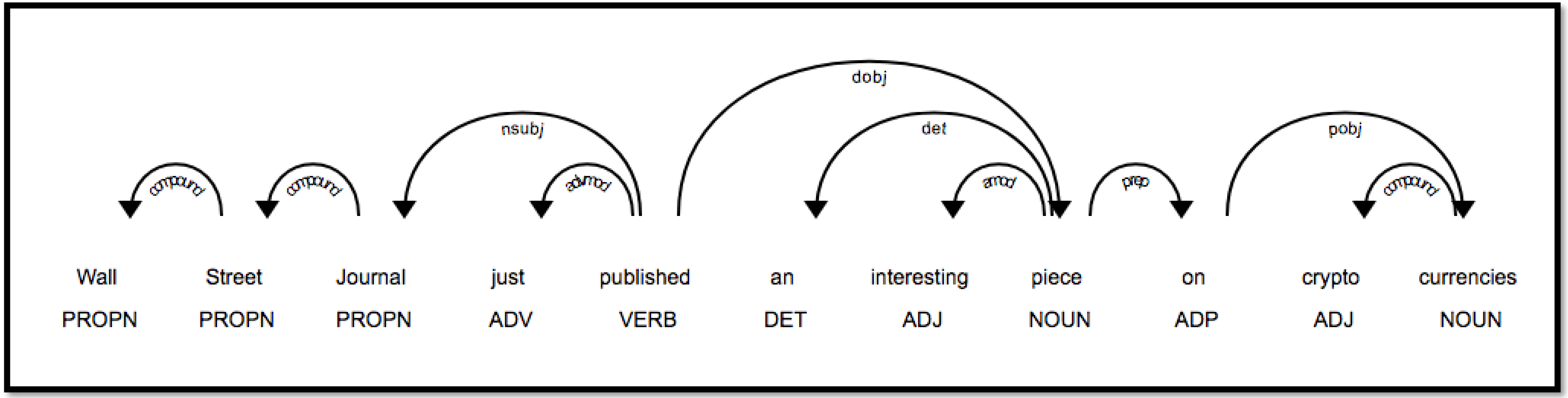
"I was reading the paper"

Number=Sing
Person=1
PronType=Prs

VerbForm=Ger

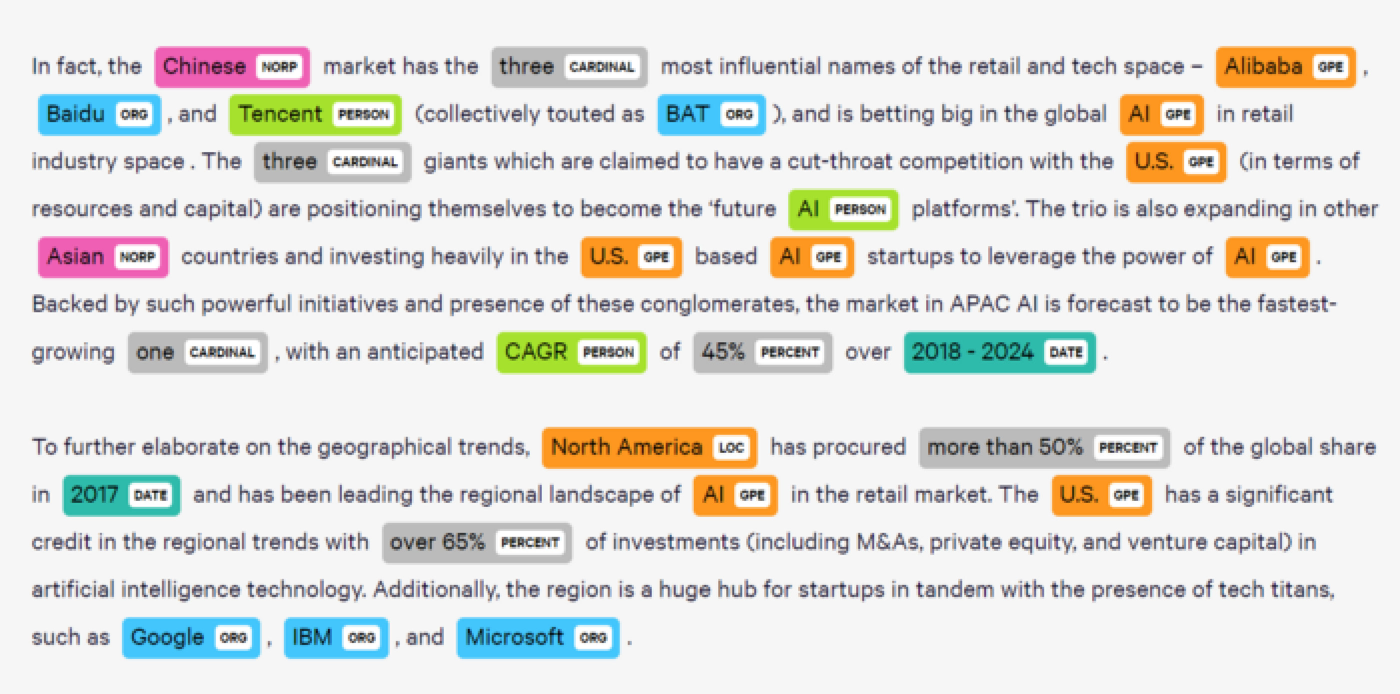
Dependency Parsing

- Extract the dependency parse tree of a sentence
- Any sentence is represented by a tree where:
  - the nodes are the token in the sentence,
  - the edges represent relationships among the tokens.



Named Entity Recognition (NER)

- NER is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc.



Token Vectorization

- Extract the word vector (or word embedding) of every token
- A word vector is a multi-dimensional mathematical representation of a word
- Semantically similar words have vectors located close to each other (cosine distance or similarity is usually adopted)
- Word vectors are computed by training a neural network to predict a word given its surrounding context (such as the words that appear before or after it in a sentence), and then using the weights of the neural network's hidden layers as the word's embedding
- Usually, it is better to use pretrained word vectors (transfer learning) such as Word2Vec, FastText, Glove, …
- Generally, word vectors are high dimensional (usually $\mathbb{R}^{100}$ or $\mathbb{R}^{300}$)

How NLP pipeline steps work?

- Usually, the NLP steps seen before are implemented by using some form of Neural Network
- We will use them out-of-the-box by exploiting a Python's library called Spacy

Outline

- NLP and NLP pipelines
- The Spacy library
- Pretrained LLM

Install Spacy and NLTK

- If you have created a Conda environment for our scripts, activate it with the following command:

```
conda activate webintelligence
```

- Install Spacy and NLTK:

```
pip install spacy nltk
```

- Download a prebuilt NLP English pipeline for Spacy

```
python -m spacy download en_core_web_md
```

There are also:

- `en_core_web_sm` (but it misses some processing steps)
- `en_core_web_lg` (but it is quite slow)

The pipeline of the Spacy model `en_core_web_md`

```
In [1]: import spacy

In [2]: nlp = spacy.load('en_core_web_md')

In [3]: nlp.pipe_names
Out[3]: ['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
```

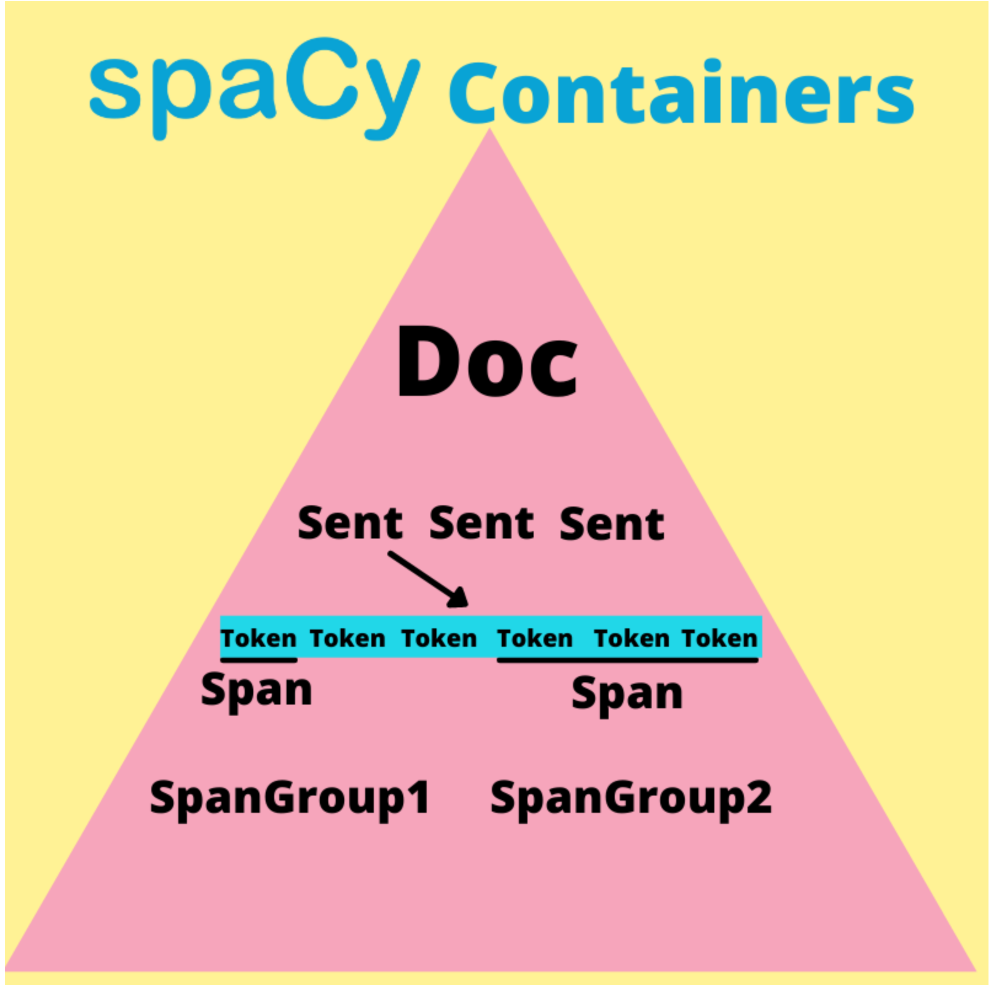
Sentence Segmentantion with Spacy

```
In [5]: import spacy

In [6]: #define a text to be analyzed

In [7]: txt = 'Alan Mathison Turing (23 June 1912 – 7 June 1954) was an English mathematician, computer scientist, logi
   ...: cian, cryptanalyst, philosopher, and theoretical biologist. Turing was highly influential in the development of
   ...:  theoretical computer science, providing a formalisation of the concepts of algorithm and computation with the
   ...: Turing machine, which can be considered a model of a general-purpose computer. He is widely considered to be th
   ...: e father of theoretical computer science and artificial intelligence.'

In [8]: #create a pipeline object to use with English texts

In [9]: nlp = spacy.load('en_core_web_md')

In [10]: #apply the pipeline to the text and collect the results in the doc object

In [11]: doc = nlp(txt)

In [12]: #print all the sentences in the text

In [13]: i = 0

In [14]: for sent in doc.sents:
    ...:     i += 1
    ...:     print(f'Sentence #{i}:')
    ...:     print(sent)
    ...:
Sentence #1:
Alan Mathison Turing (23 June 1912 – 7 June 1954) was an English mathematician, computer scientist, logician, cryptanaly
st, philosopher, and theoretical biologist.
Sentence #2:
Turing was highly influential in the development of theoretical computer science, providing a formalisation of the conce
pts of algorithm and computation with the Turing machine, which can be considered a model of a general-purpose computer.
Sentence #3:
He is widely considered to be the father of theoretical computer science and artificial intelligence.
```

Tokenization in Spacy

```
In [16]: import spacy

In [17]: nlp = spacy.load('en_core_web_md')

In [18]: txt = 'Hong Kong is a beautiful city!'

In [19]: doc = nlp(txt)

In [20]: for token in doc:
    ...:     print(token.text)
    ...:
Hong
Kong
is
a
beautiful
city
!
```

Spacy container objects: Doc, Token, Span




```
In [37]: doc
Out[37]: Hong Kong is a beautiful city!

In [38]: type(doc)
Out[38]: spacy.tokens.doc.Doc

In [39]: doc[0]
Out[39]: Hong

In [40]: type(doc[0])
Out[40]: spacy.tokens.token.Token

In [41]: doc[0:2]
Out[41]: Hong Kong

In [42]: type(doc[0:2])
Out[42]: spacy.tokens.span.Span

In [43]: sent = list(doc.sents)[0]

In [44]: sent
Out[44]: Hong Kong is a beautiful city!

In [45]: type(sent)
Out[45]: spacy.tokens.span.Span
```

Token basic properties in Spacy

- `is_alpha` is True if the token is a proper word

- `is_stop` is True if the token is among the English stopwords, i.e. those words that usually do not bring any semantic meaning and can be removed for semantic analyses

- `shape_` shows orthographic features of the token. Alphabetic characters are replaced by x or X, and numeric characters are replaced by d, and sequences of the same character are truncated after length 4.

```
In [49]: doc
Out[49]: Hong Kong is a beautiful city!

In [50]: for token in doc:
    ...:     print(token.text, token.is_alpha, token.is_stop, token.shape_)
    ...:
Hong True False Xxxx
Kong True False Xxxx
is True True xx
a True True x
beautiful True False xxxx
city True False xxxx
! False False !
```

See https://spacy.io/api/token for a complete reference

Lemma vs Stem of a Token

- Spacy has lemmatization but not stemming, so we use NLTK for stemming
- Lemmas are clearly more useful than stems!

```
In [52]: doc
Out[52]: Hong Kong is a beautiful city!

In [53]: from nltk.stem import PorterStemmer

In [54]: stemmer = PorterStemmer()

In [55]: for token in doc:
    ...:     print(token.text, token.lemma_, stemmer.stem(token.text))
    ...:
Hong Hong hong
Kong Kong kong
is be is
a a a
beautiful beautiful beauti
city city citi
! ! !
```

POS tag of a Token with Spacy

```
In [57]: doc
Out[57]: Hong Kong is a beautiful city!

In [58]: for token in doc:
    ...:     print(token.text, token.pos_)
    ...:
Hong PROPN
Kong PROPN
is AUX
a DET
beautiful ADJ
city NOUN
! PUNCT
```

Morphological features of a Token with Spacy

```
In [4]: doc = nlp('I was reading a paper.')

In [5]: for token in doc:
   ...:     print(token.text, token.pos_, token.morph)
   ...:
I PRON Case=Nom|Number=Sing|Person=1|PronType=Prs
was AUX Mood=Ind|Number=Sing|Person=3|Tense=Past|VerbForm=Fin
reading VERB Aspect=Prog|Tense=Pres|VerbForm=Part
a DET Definite=Ind|PronType=Art
paper NOUN Number=Sing
. PUNCT PunctType=Peri
```

Dependency Parse Tree with Spacy

- Any single sentence is formed by exactly one parse tree

- A node of a tree has only one parent (or zero if it is the root), so Spacy defines two attributes for each token:
  - `head` which points to the parent token,
  - `dep_` which provides the label of the edge (i.e. the type of dependency)

```
In [60]: doc
Out[60]: Hong Kong is a beautiful city!

In [61]: for token in doc:
    ...:     print(token.text, token.dep_, token.head.text)
    ...:
Hong compound Kong
Kong nsubj is
is ROOT is
a det city
beautiful amod city
city attr is
! punct is
```

Noun Chunks with Spacy

- Noun chunks are “base noun phrases” – flat phrases that have a noun as their head. You can think of noun chunks as a noun plus the words describing the noun – for example, “the lavish green grass” or “the world’s largest tech fund”.

```
In [6]: doc = nlp('Hong Kong is a beautiful city')

In [7]: for nc in doc.noun_chunks:
   ...:     print(nc)
   ...:
Hong Kong
a beautiful city
```

Named Entities with Spacy

```
In [75]: doc
Out[75]: Hong Kong is a special administrative regione of China and Bruce Lee was from Hong Kong!!!

In [76]: for ent in doc.ents:
    ...:     print(ent.text, ent.label_, ent.start_char, ent.end_char)
    ...:
Hong Kong GPE 0 9
China GPE 49 54
Bruce Lee PERSON 59 68
Hong Kong GPE 78 87
```

Common transformation of a text

- For further semantic processing of a text, sometimes it is useful to:
  - remove stop words and non-alphabetical tokens
  - replace token text with its lemma
  - merge the words of a compound named entity

```
In [122]: doc
Out[122]: Hong Kong is a special administrative region of China and Bruce Lee was from Hong Kong!!!

In [123]: lst = []

In [124]: for i in range(len(doc)):
     ...:     token = doc[i]
     ...:     if token.is_alpha and not token.is_stop:
     ...:         if token.ent_iob_ == 'O': #outside, i.e. not belonging to a named entity
     ...:             lst.append(token.lemma_)
     ...:         elif token.ent_iob_ == 'B': #begin, i.e. initial token of a named entity
     ...:             lst.append(token.text)
     ...:         else: #token.ent_iob_ == 'I' #inside, i.e. token inside a compound named entity
     ...:             lst[-1] = lst[-1] + '_' + token.text
     ...:

In [125]: new_text = ' '.join(lst)

In [126]: new_text
Out[126]: 'Hong_Kong special administrative region China Bruce_Lee Hong_Kong'
```

Find the most common lemmas in a corpus

- Corpus is synonym of "set of texts"... we may also call "dataset"

```
In [58]: import spacy

In [59]: from collections import Counter

In [60]: texts = [ 'Hong Kong is a beautiful city!',
    ...:           'Bruce Lee was from Hong Kong',
    ...:           'Hong Kong and Macau are two Chinese special administrative regions',
    ...:           'Macau has a very beatiful historical center!',
    ...:           'Hong Kong and Macau are two cities',
    ...:           'Perugia is a city as well' ]

In [61]: docs = [ nlp(text) for text in texts ]

In [62]: lemmas = [ token.lemma_ for doc in docs
    ...:                             for token in doc
    ...:                                 if token.is_alpha and not token.is_stop ]

In [63]: lemmas[:5]
Out[63]: ['Hong', 'Kong', 'beautiful', 'city', 'Bruce']

In [64]: lemmas_counter = Counter(lemmas)

In [65]: lemmas_counter.most_common(10)
Out[65]:
[('Hong', 4),
 ('Kong', 4),
 ('city', 3),
 ('Macau', 3),
 ('beautiful', 1),
 ('Bruce', 1),
 ('Lee', 1),
 ('chinese', 1),
 ('special', 1),
 ('administrative', 1)]
```

Spell Checking

- We need another Python library: `pip install pyspellchecker`
- It uses a classical spellchecking method that uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. It then compares all permutations (insertions, deletions, replacements, and transpositions) to known words in a word frequency list. Those words that are found more often in the frequency list are more likely the correct results.
- Just for reference, have a look to https://norvig.com/spell-correct.html

```
In [7]: import spacy

In [8]: from spellchecker import SpellChecker

In [9]: nlp = spacy.load('en_core_web_md')

In [10]: doc = nlp('Tuday is beutiful dya')

In [11]: spell = SpellChecker()

In [12]: newtokens = [ spell.correction(tok.text) for tok in doc ]

In [13]: newtext = ' '.join(newtokens)

In [14]: newtext
Out[14]: 'today is beautiful day'
```

Word Vectors in Spacy

- The attribute `vector` of every token is a numpy array of dimensionality 300
- The word embeddings have been pretrained on a large corpus using Word2Vec

```
In [138]: doc[0].vector
Out[138]:
array([ 1.2330e+00,  4.2963e+00, -7.9738e+00, -1.0121e+01,  1.8207e+00,
        1.4098e+00, -4.5180e+00, -5.2261e+00, -2.9157e-01,  9.5234e-01,
        6.9880e+00,  5.0637e+00, -5.5726e-03,  3.3395e+00,  6.4596e+00,
       -6.3742e+00,  3.9045e-02, -3.9855e+00,  1.2085e+00, -1.3186e+00,
       -4.8886e+00,  3.7066e+00, -2.8281e+00, -3.5447e+00,  7.6888e-01,
        1.5016e+00, -4.3632e+00,  8.6480e+00, -5.9286e+00, -1.3055e+00,
        8.3870e-01,  9.0137e-01, -1.7843e+00, -1.0148e+00,  2.7300e+00,
       -6.9039e+00,  8.0413e-01,  7.4880e+00,  6.1078e+00, -4.2130e+00,
       -1.5384e-01, -5.4995e+00,  1.0896e+01,  3.9278e+00, -1.3601e-01,
        7.7732e-02,  3.2218e+00, -5.8777e+00,  6.1359e-01, -2.4287e+00,
        6.2820e+00,  1.3461e+01,  4.3236e+00,  2.4266e+00, -2.6512e+00,
        1.1577e+00,  5.0848e+00, -1.7058e+00,  3.3824e+00,  3.2850e+00,
        1.0969e+00, -8.3711e+00, -1.5554e+00,  2.0296e+00, -2.6796e+00,
       -6.9195e+00, -2.3386e+00, -1.9916e+00, -3.0450e+00,  2.4890e+00,
        7.3247e+00,  1.3364e+00,  2.3828e-01,  8.4388e-02,  3.1480e+00,
       -1.1128e+00, -3.5598e+00, -1.2115e-01, -2.0357e+00, -3.2731e+00,
       -7.7205e+00,  4.0948e+00, -2.0732e+00,  2.0833e+00, -2.2803e+00,
       -4.9850e+00,  9.7667e+00,  6.1779e+00, -1.0352e+01, -2.2268e+00,
        2.5765e+00, -5.7440e+00,  5.5564e+00, -5.2735e+00,  3.0004e+00,
       -4.2512e+00, -1.5682e+00,  2.2698e+00,  1.0491e+00, -9.0486e+00,
        4.2936e+00,  1.8709e+00,  5.1985e+00, -1.3153e+00,  6.5224e+00,
        4.0113e-01, -1.2583e+01,  3.6534e+00, -2.0961e+00,  1.0022e+00,
       -1.7873e+00, -4.2555e+00,  7.7471e+00,  1.0173e+00,  3.1626e+00,
        2.3558e+00,  3.3589e-01, -4.4178e+00,  5.0584e+00, -2.4118e+00,
       -2.7445e+00,  3.4170e+00, -1.1574e+01, -2.6568e+00, -3.6933e+00,
       -2.0398e+00,  5.0976e+00,  6.5249e+00,  3.3573e+00,  9.5334e-01,
       -9.4430e-01, -9.4395e+00,  2.7867e+00, -1.7549e+00,  1.7287e+00,
```

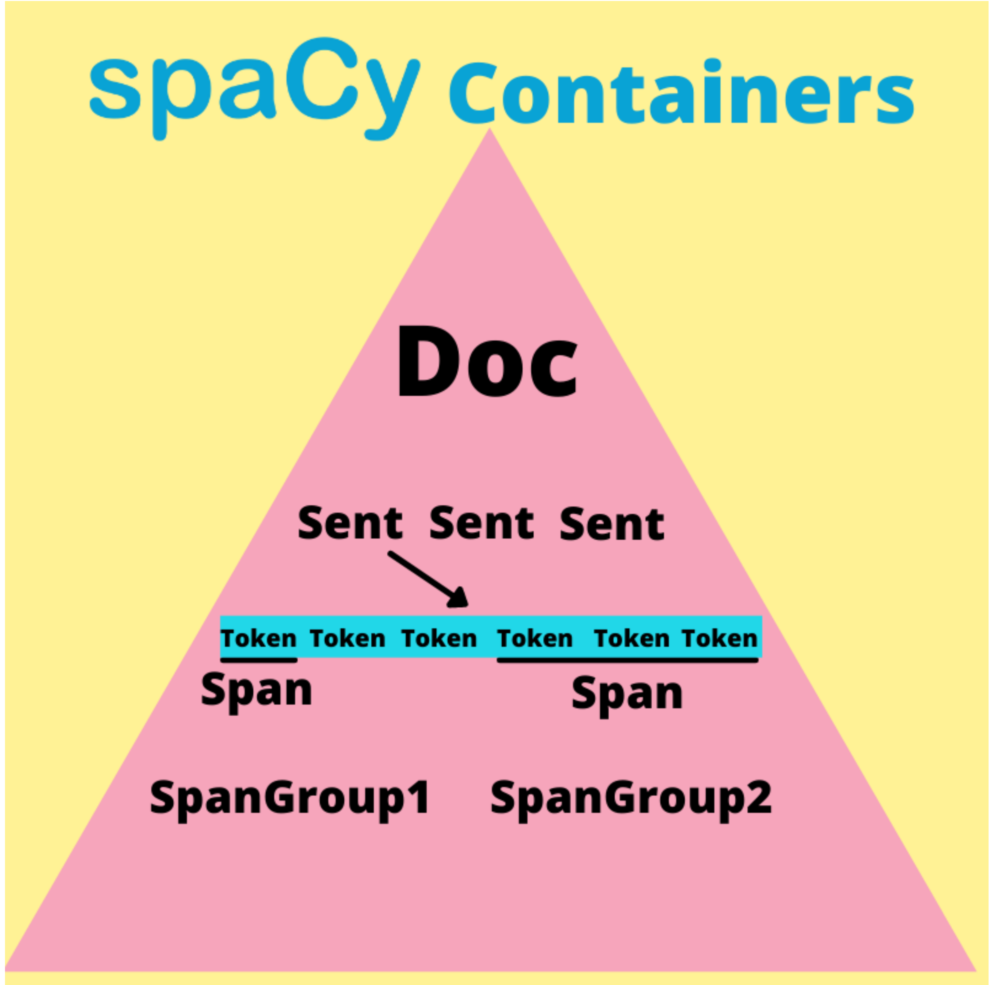
Other vector-related attributes

```
In [142]: txt = "dog cat banana afskfsd"

In [143]: doc = nlp(txt)

In [144]: for token in doc:
     ...:     print(token.text, token.has_vector, token.vector_norm, token.is_oov)
     ...:
dog True 75.254234 False
cat True 63.188496 False
banana True 31.620354 False
afskfsd False 0.0 True
```

Similarity between tokens with Spacy

```
In [148]: from scipy.spatial.distance import cosine

In [149]: doc = nlp('dog cat')
     ...: tok1, tok2 = doc[0], doc[1]
     ...: print()
     ...: print('*** SIMILARITY BETWEEN "DOG" AND "CAT" ***')
     ...: print(f'Similarity = {tok1.similarity(tok2)}')
     ...: print(f'Cosine distance = {cosine(tok1.vector,tok2.vector)}')
     ...: print(f'Similarity and cosine distance sum to {tok1.similarity(tok2)+cosine(tok1.vector,tok2.vector)}')

*** SIMILARITY BETWEEN "DOG" AND "CAT" ***
Similarity = 0.8220816850662231
Cosine distance = 0.17791831493377686
Similarity and cosine distance sum to 1.0
```

Vectorized form also for Doc, Sent, Span



- Word2vec vectors are attributes of the tokens
- Doc, Sent and Span are objects containing several tokens
- Spacy introduces a `vector` attribute for all the container in such a way that its value is the average of the tokens' vectors (where `has_token` is True)

Vectorization and Similarity for Containers

```
In [151]: doc = nlp('dog cat mango papaya')

In [152]: doc
Out[152]: dog cat mango papaya

In [153]: doc[0:2]
Out[153]: dog cat

In [154]: doc.has_vector
Out[154]: True

In [155]: doc.vector_norm
Out[155]: 40.07046503460941

In [156]: doc.vector.size
Out[156]: 300

In [157]: doc[0:2].has_vector
Out[157]: True

In [158]: doc[0:2].vector_norm
Out[158]: 66.09522

In [159]: doc[0:2].vector.size
Out[159]: 300

In [160]: doc.similarity(doc[0:2])
Out[160]: 0.887249661864046

In [161]: doc[0:2].similarity(doc)
Out[161]: 0.887249661864046
```

```
In [163]: doc1 = nlp('dog cat mango papaya')

In [164]: doc2 = nlp('cat papaya mango dog')

In [165]: doc1.similarity(doc2)
Out[165]: 1.0000000281837924
```

Since the vector of a container is the average of the vectors of its tokens, then two containers which are a permutation of each other tokens have the same vectorization, so their similarity is 1.

For practical purposes this is not a significant issue.

Anyway, to avoid this problem, it is possible to use modern Large Language Model, such as the Transformed-based models available from the HuggingFace repository, like BERT.

Outline

- NLP and NLP pipelines
- The Spacy library
- <u>Pretrained LLM</u>

Pretrained encoder-based LLM

- To obtain proper embedding for entire sentences, let's use a pretrained Sentence Transformer

  (https://www.sbert.net/docs/usage/semantic_textual_similarity.html)

  ```
  pip install sentence-transformers
  ```

- As for word embeddings, the principle is: "sentences with similar meanings have vectors/embeddings which are close under cosine similarity"

Pretrained encoder-based LLM

```
In [123]: from sentence_transformers import SentenceTransformer, util
     ...: model = SentenceTransformer('all-MiniLM-L6-v2')
     ...:
     ...: # Two lists of sentences
     ...: sentences1 = ['The cat sits outside',
     ...:              'A man is playing guitar',
     ...:              'The new movie is awesome']
     ...:
     ...: sentences2 = ['The dog plays in the garden',
     ...:               'A woman watches TV',
     ...:               'The new movie is so great']
     ...:
     ...: #Compute embedding for both lists
     ...: embeddings1 = model.encode(sentences1)
     ...: embeddings2 = model.encode(sentences2)
     ...:
     ...: #Compute cosine-similarities
     ...: cosine_scores = util.cos_sim(embeddings1, embeddings2)
     ...:
     ...: #Output the pairs with their score
     ...: for i in range(len(sentences1)):
     ...:     print(f'{sentences1[i]} \t {sentences2[i]} \t Score: {cosine_scores[i][i]:.3f}')
     ...:
The cat sits outside     The dog plays in the garden     Score: 0.284
A man is playing guitar          A woman watches TV      Score: -0.033
The new movie is awesome         The new movie is so great       Score: 0.894
```

First time you run this instruction, a pretrained model (of some megabytes) is downloaded from the web.

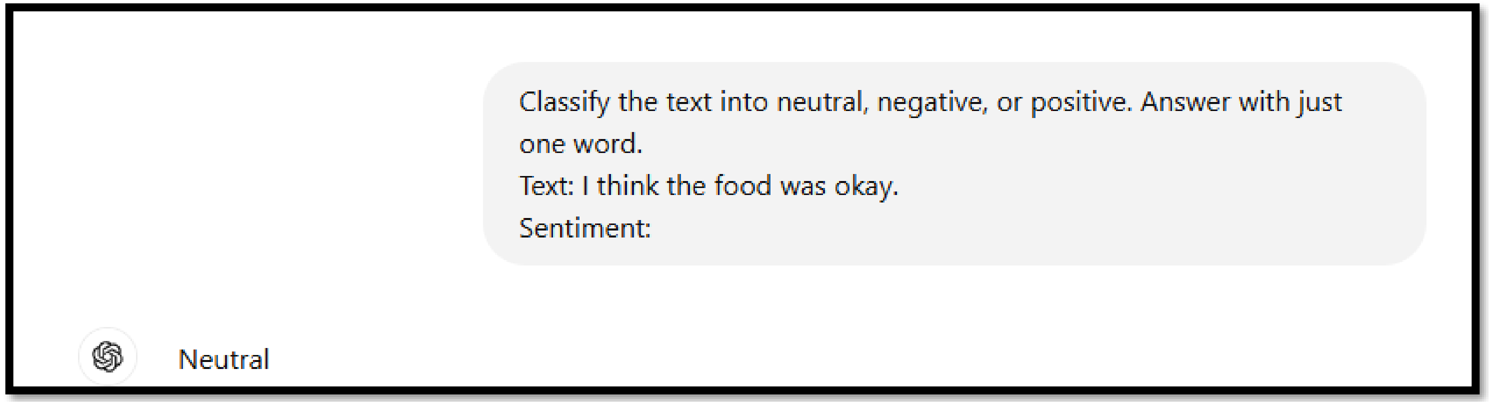
Contain the vectors/embeddings of the sentences.

Not shown in the example, but you can use the Sentence Tokenizer in Spacy and then calculate the embeddings of any sentence using a Sentence Transformer. If you additionally want the embedding for a longer text, averaging the sentence embeddings of the text is usually a good solution.

Pretrained decoder-based LLM

- Modern decoder-based LLMs (such as GPT, Gemma, LLAMA, etc.) can be used for a wide range of WI applications:
  - Classify texts in a zero or few shots way (without a proper classifier)
  - Give labels to cluster of documents
  - Mixed with information retrieval for question answering
  - …

- What is required for that? A good prompt, which is formed by, not necessarily all, the following parts:
  - Instruction - a specific task or instruction you want the model to perform
  - Context - external information or additional context that can steer the model to better responses
  - Input Data - the input or question that we are interested to find a response for
  - Output Indicator - the type or format of the output.



References

- Spacy website contains manuals, tutorials and examples https://spacy.io/
- Introduction to Spacy 3 (online tutorial): http://spacy.pythonhumanities.com/intro.html
- Sentence Transformers website contains tutorials and examples https://www.sbert.net/
- Prompt Engineering Guide https://www.promptingguide.ai/