
# NLP: Text analysis with Spacy using Python

💻 A hands-on Natural Language Processing practical session

## Agenda

- Intro to NLP and Spacy
- Pre-processing
- Sentence segmentation
- Tokenisation
- POS tagging
- Named Entity Recognition
- Stop word removal
- Removing punctuation and stripping
- Lemmatisation
- Dependency visualisation
- Frequency analysis
- Multiprocessing pipelines

## Natural Language Processing (NLP)

- subset of Artificial Intelligence (AI)
- aims to enable computers to understand human language (text and spoken content)
- forms part of computational linguistics that
  - make use of rule based matching with statistical analysis
  - machine learning and deep learning models
  - to comprehend the human language
- With NLP, natural language can be
  - analysed
  - quantified
  - understood
  - derived meaning from (context)
- Recent research by OpenAi lead to the creation of chatGPT (one of the most advanced natural language models)

## Application of NLP

- text analysis and classification
- sentiment analysis
- automatic summarisation
- chatbots
- speech recognition
- translation
- predictive text

## Spacy

- NLP library

- Support for over 72 languages
- offers pre-trained models
- allows named-entity recognition, part-of-speech tagging, dependency parsing, sentence segmentation, text classification, lemmatisation, morphological analysis, entity linking and more
- has built-in visualiser
- support for custom pipelines (tokenisation, tagging, parsing, ner, …)
- efficient library (memory usage)

## Installation of tools

**Tools**

- Vs Code
- Vs Code Extensions (Jupyter Notebooks: Top 4 extensions)

**Libraries**

- Pandas

  Allows data manipulation and analysis

  ```
  pip install pandas
  ```

- Spacy

  NLP library that enables text analysis

  ```
  pip install -U spacy
  python3 -m spacy download en_core_web_sm
  ```

- WordCloud

  Generates word cloud images

  ```
  pip install wordcloud
  ```

- Matplotlib

  A practical visualisation library

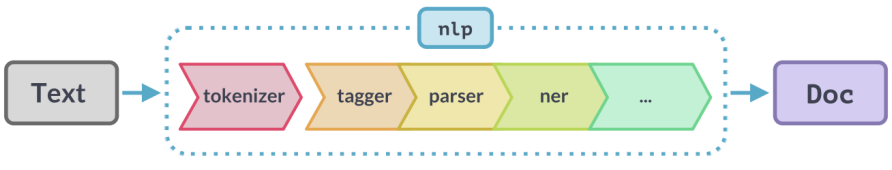
  ```
  pip install matplotlib
  ```

## How can we analyse text with spacy?

- Split sentences
- Tokenise text
- POS Tagging
- Remove stop words
- Lemmatise words
- Analyse word frequency
- Visualise dependencies
- Named-entity visualisation

**Spacy language processing pipeline**

- Default pipeline:




## Getting started with Spacy

```
#importing spacy and loading the model
import spacy
nlp = spacy.load('en_core_web_sm')
```

## Reading from csv file

```
#using pandas to read csv
import pandas as pd
df = pd.read_csv('text.csv')

print(df.head())
```

```
   id                                              para
0   1  Until recently, the conventional wisdom was th...
1   2  This mindset can create new opportunities for ...
```

Csv metadata

## Sentence segmentation

- Sometimes we have big chunks of text consisting of multiple sentences or paragraphs
- For text analysis, we need to split the text apart into separate sentences
- Often considered as preprocessing

```
#Splitting paragraph into sentences
paragraph=df.para[0]

processed_para=nlp(paragraph) #processed_para is a spacy doc object
# print(processed_para.sents)

sentences=[]
for i,sent in enumerate(processed_para.sents):
    print("{0} : {1}".format(i,sent))
    sentences.append(sent)

sentence=str(sentences[0]) #converting spacy span object to string
print('\n'+sentence)
```

```
0 : Until recently, the conventional wisdom was that while AI was better than humans at data-driven decision making tasks, it was still inferior to humans for cognitive and creative ones.
1 : But in the past two years language-based AI has advanced by leaps and bounds, changing common notions of what this technology can do.
2 : The most visible advances have been in natural language processing (NLP), the branch of AI focused on how computers can process language like humans do.
3 : It has been used to write an article for The Guardian, and AI-authored blog posts have gone viral — feats that were not possible a few years ago.
4 : AI even excels at cognitive tasks like programming where it is able to generate programs for simple video games from human instructions.
```

Segmented sentences

## Tokenisation

- First step in most NLP pipelines
- Splits text into discrete elements (words and punctuation)
- A token can be a word, punctuation, verb, noun, etc

```
#Tokenisation
doc = nlp(sentence)

for token in doc:
    print("{0}\t{1}".format(token.text,token.idx))
    #idx property gives index of word in sentence
```

```
the     16
conventional    20
wisdom  33
was     40
that    44
while   49
AI      55
was     58
better  62
```

Some tokens from sentence

## How are tokens encoded ?

- The vocabulary is encoded uses hashes in spacy
- Strings are encoded as hashes before being saved in a hashmap

```
#How are tokens encoded in spacy?
print(doc.vocab.strings) #uses a hashmap to store strings
print(doc.vocab.strings['creative']) #convert string to hash
nlp.vocab.strings[1433653077910583464] #convert hash to string
```

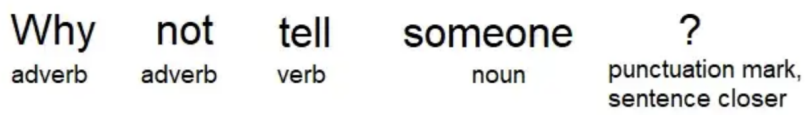
```
<spacy.strings.StringStore object at 0x147284450>
1433653077910583464

'creative'
```

Output from the code

## Part-of-speech tagging

- process of categorising words depending on
  - definition of word
  - context
- Example:




POS tags at the bottom of the words

- POS tags are encoded into shorter for by spacy
- used to understand the context and meaning of the word in a sentence

```
#Token attributes and Part-of-speech tagging
for token in doc:
    print("{0} \tTag: {1} \tPOS:{2} \nDescription: {3}\n".format(token,token.tag_,token.pos
_,spacy.explain(token.tag_)))
```

```
the     POS:DET
Description: determiner

conventional    POS:ADJ
Description: adjective (English), other noun-modifier (Chinese)

wisdom  POS:NOUN
Description: noun, singular or mass
```

Example of POS tagging

### Types of POS tags

| POS | DESCRIPTION |
|---|---|
| ADJ | adjective |
| ADP | adposition |
| ADV | adverb |
| AUX | auxiliary |
| CONJ | conjunction |
| CCONJ | coordinating conjunction |
| DET | determiner |
| INTJ | interjection |
| NOUN | noun |
| NUM | numeral |
| PART | particle |
| PRON | pronoun |
| PROPN | proper noun |
| PUNCT | punctuation |
| SCONJ | subordinating conjunction |
| SYM | symbol |
| VERB | verb |
| X | other |
| SPACE | space |

## Named Entity Recognition (NER)

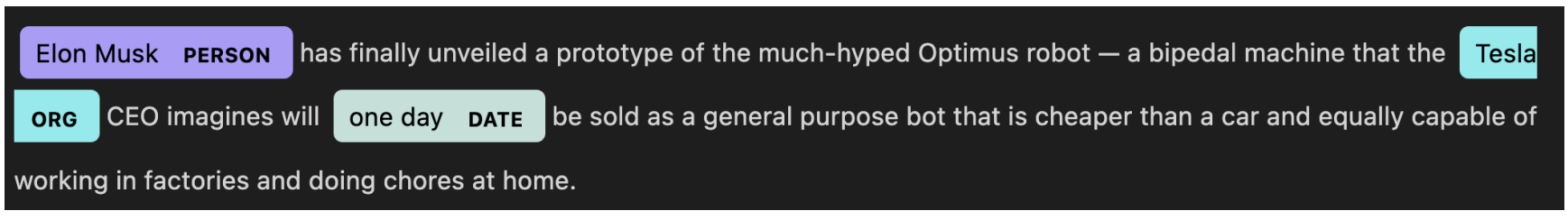
- A named-entity is a “real-life object” such as a person, a organisation, a location, a time or a product
- NER is the process of predicting and identifying named entities

```
#Named entity recogntion
for entity in doc.ents:
    print("{0}    \t{1}\nDescription: {2}\n".format(entity.text,entity.label_,spacy.explain
(entity.label_)))
```

```
Elon Musk       PERSON
Tesla           ORG
```

### Visualising Named Entities

```
from spacy import displacy
displacy.render(doc,style='ent',jupyter=True)
```




visualisation

## Stop word removal

- Stop word removal is the process of removing stop words so that stop words do not hinder other processes such as frequency analysis
- Stop words are the most common words in a language that are not significant in a sentence
- Examples: a, the, are, but

```
#Stop word removal
for token in doc: #displaying stop words
    if token.is_stop:
        print(token)

new_doc=[token for token in doc if not token.is_stop] #removing stop words
print(new_doc)
```

```
[recently, ,, conventional, wisdom, AI, better, humans, data, -, driven, decision, making, tasks, ,, inferior, humans, cognitive, creative, ones, .]
```

result

## Removing punctuation and stripping

- Stripping the tokens will remove empty spaces
- Removing punctuation will remove punctuation symbols such as (. , -)

```
#Removing punctuation
new_doc=[str(token).strip() for token in doc if not token.is_punct] #filtering punctuation and stripping words
print(new_doc)
```

```
['Until', 'recently', 'the', 'conventional', 'wisdom', 'was', 'that', 'while', 'AI', 'was', 'better', 'than', 'humans', 'at', 'data', 'driven', 'decision', 'making', 'tasks', 'it', 'was', 'still', 'inferior', 'to', 'humans', 'for', 'cognitive', 'and', 'creative', 'ones']
```

result

## Lemmatisation

- process of reducing a word to its base word (root)
- the base word(root) is called the lemma
- for example: For eats, ate, eating , the lemma is eat
- Helps in normalising text

```
new_doc=[token.lemma_ for token in doc] #lemmatisation
print(new_doc)
```

```
['until', 'recently', ',', 'the', 'conventional', 'wisdom', 'be', 'that', 'while', 'AI', 'be',
'well', 'than', 'human', 'at', 'data', '-', 'drive', 'decision', 'make', 'task', ',', 'it', 'be',
'still', 'inferior', 'to', 'human', 'for', 'cognitive', 'and', 'creative', 'one', '.']
```

## Dependencies

- process of extracting the dependency graph of a sentence
- to represent its grammatical structure
- defines the dependency relationship between headwords and their dependents
- Graph can be explained as follows:
  - words/tokens are nodes
  - dependencies are edges (relationships/arrows)

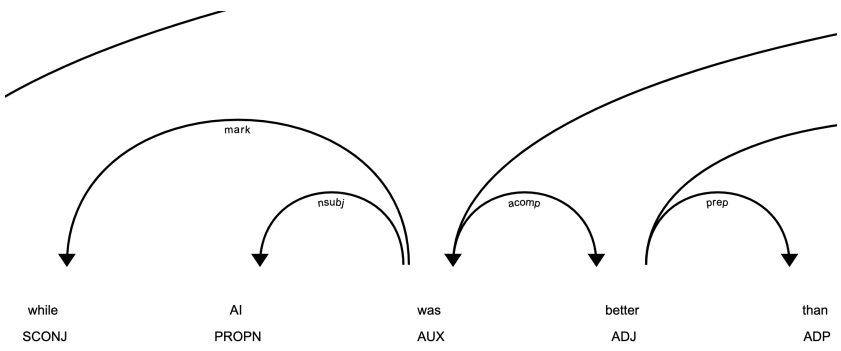
### Analysing Dependencies

```
for token in doc:
    print("{0} \tHead: {1} \tDependency: {2} \tHead POS:{3}".format(token,token.head.text,token.dep_,token.head.pos_))
```

```
better          Head: was           Dependency: acomp          Head POS:AUX
than            Head: better        Dependency: prep           Head POS:ADJ
humans          Head: than          Dependency: pobj           Head POS:ADP
at              Head: better        Dependency: prep           Head POS:ADJ
data            Head: driven        Dependency: npadvmod       Head POS:VERB
```

### Dependency Visualisation

```
displacy.render(doc, style='dep', jupyter = True)
```




Part of dependencies

## Frequency analysis

- The number of times a word appears can be analysed
- The Counter library can be used for that purpose

```
[('the', 16), ('to', 12), ('.', 11), (',', 10), ('new', 9)]
```

```
#Frequency analysis
from collections import Counter
words=[word.text for word in nlp(df.para[3])]
common=Counter(words).most_common(5) #returns top 5 most common words
print(common)
```

## Generating a WordCloud

- data visualisation technique
- represents text frequency and importance
- bigger words have higher frequency


```
#Generating a wordcloud
import matplotlib.pyplot as plt
from wordcloud import WordCloud

joined_text = " ".join(token for token in words)
# print(joined_text)
wordcloud = WordCloud().generate(joined_text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.savefig('wordcloud.png')
plt.show()
```

## Multiprocessing pipelines

- Spacy allows multiprocessing
- More than 1 process can be spawned at a time
- It is inefficient for small datasets but far more useful on large datasets with large batches

```
#Multiprocessing with spacy
texts=df.para.tolist() #converting dataframe to list
docs = nlp.pipe(texts, n_process=4) #using 4 processes to process texts

for doc in docs:
    print(doc)
```

## Concept analysis

- pre-trained model
- concept-based
- sentiment prediction

```
#Bonus
#Sentiment analysis library (Senticnet)
from senticnet.senticnet import SenticNet
sn = SenticNet()

for token in doc:
    try:
        if token.pos_ == 'ADJ':
            print("\n{0}".format(token.text))
            print(sn.concept(token.text))
    except:
        pass
```

```
smart
{'polarity_label': 'positive', 'polarity_value': '0.921', 'sentics': {'introspection': '0.995',
'temper': '0.801', 'attitude': '0', 'sensitivity': '0.968'}, 'moodtags': ['#joy', '#eagerness'],
'semantics': ['acquire_knowledge', 'comprehend', 'great', 'accumulate_knowledge', 'increase_knowledge']}
```

## Future Sessions

📌 NLP: Sentiment Analysis with Python