NLP Python Basics

Natural Language Processing

- Section Goals
  - Set up Spacy and Language Library
  - Understand Basic NLP Topics
    - Tokenization
    - Stemming
    - Lemmatization
    - Stop Words
    - Spacy for Vocabulary Matching

Spacy Setup

What is Spacy?

- Spacy is an open-source natural language processing library that is designed to handle a wide range of NLP tasks such as tokenization, part-of-speech tagging, named entity recognition, and more. It provides efficient implementations of common NLP algorithms and is known for its speed and accuracy. The library is also easily customizable and can be integrated into various applications and systems.

What is NLTK?

- NLTK (Natural Language Toolkit) is a Python library for working with human language data.
- It provides a wide range of tools for tasks such as tokenization, stemming, lemmatization, part-of-speech tagging, and more.
- It also includes a large collection of text data, such as books and other resources, for use in NLP projects and research.

NLTK vs Spacy

- NLTK and Spacy are both popular natural language processing libraries for Python, but they have some key differences in terms of their design and functionality.
- NLTK has a larger set of features and functionalities and is more general-purpose, it has more extensive libraries for Text Analysis, Machine Learning, and Deep Learning.
- Spacy, on the other hand, is designed for speed and performance and is geared towards more specific NLP tasks such as named entity recognition, part-of-speech tagging, and dependency parsing.

NLTK vs SpaCy

Aspect	NLTK	spaCy
Features and Functionalities	Comprehensive toolkit with a wide range of NLP tasks.	Focused on core NLP tasks like NER, POS tagging, and parsing.
Libraries	Extensive libraries for Text Analysis, ML, and DL.	Emphasizes efficiency and speed over a wide range of tasks.
Speed and Performance	Generally, may not be as fast and efficient as spaCy.	Known for its speed and performance.
Use Cases	Suitable for both simple text analysis and complex language processing.	Ideal for real-time or high-speed processing.
Modularity	Offers a broad set of tools and functionalities.	Built as a modular system for scalability and efficiency.

NLTK vs Spacy Processing Tests

	ABSOLUTE (MS PER DOC)			RELATIVE (TO SPACY)		
SYSTEM	TOKENIZE	TAG	PARSE	TOKENIZE	TAG	PARSE
spaCy	0.2ms	1ms	19ms	1x	1x	1x
CoreNLP	0.18ms	10ms	49ms	0.9x	10x	2.6x
ZPar	1ms	8ms	850ms	5x	8x	44.7x
NLTK	4ms	443ms	*n/a*	20x	443x	*n/a*

Natural Language Processing

	SPACY	SYNTAXNET	NLTK	CORENLP
Programming language	Python	C++	Python	Java
Neural network models	✓	✓	✗	✓
Integrated word vectors	✓	✗	✗	✗
Multi-language support	✓	✓	✓	✓
Tokenization	✓	✓	✓	✓
Part-of-speech tagging	✓	✓	✓	✓
Sentence segmentation	✓	✓	✓	✓
Dependency parsing	✓	✓	✗	✓
Entity recognition	✓	✗	✓	✓
Coreference resolution	✗	✗	✗	✓

https://spacy.io/usage/facts-figures

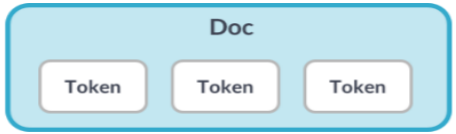
Spacy Basics

Natural Language Processing

- There are a few keys steps for working with Spacy that we will cover in this lecture:
  - Loading the Language Library
  - Building a Pipeline Object
  - Using Tokens
  - Parts-of-Speech Tagging
  - Understanding Token Attributes

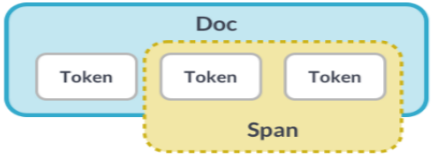
NLP objects

- The Doc Object → doc = nlp(“Hello world”)
- The Token Object →



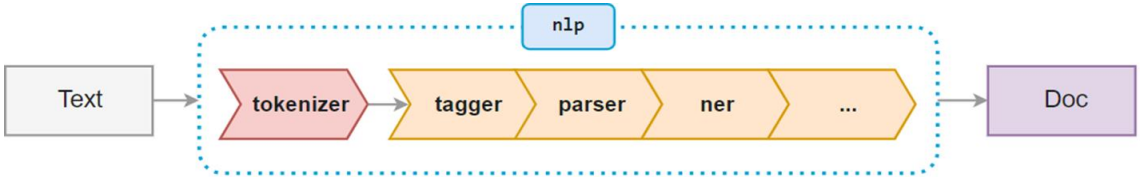
Token = doc[1]
Print (token.text)

- The Span object →



span= doc[1:3]
Print (token.text)

- The Pipeline Object →



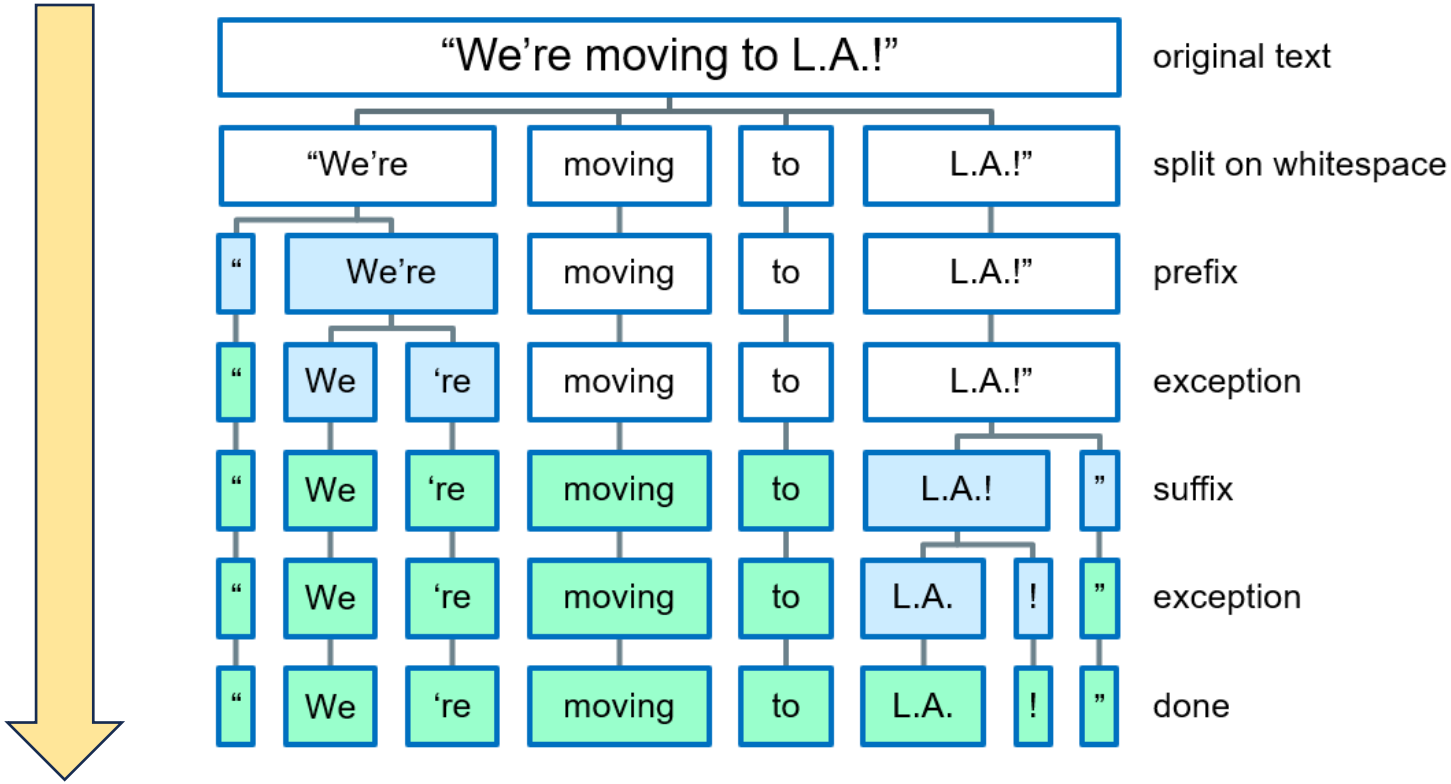
Source: Spacy.io

Tokenization

- Tokenization in Spacy refers to the process of breaking down a text into individual words, phrases, or other meaningful elements called tokens. The Spacy library provides a powerful tokenizer that can handle a wide range of languages and text formats. It can also handle complex tasks such as handling contractions, punctuation, and special characters. The tokenization process is the first step in most NLP pipelines and is used to prepare text data for further analysis and processing.

- Prefix: Character(s) at the beginning → $ (“ ¿
- Suffix: Character(s) at the end → km) , . ! ”
- Infix: Character(s) in between → - -- / ...
- Exception: Special-case rule to split a string into several tokens or prevent a token from being split when punctuation rules are applied → St. U.S.

Tokenization



Tokenization Code sample

```
# Create a string that includes opening and closing quotation marks
mystring = '"We\'re moving to L.A.!"'
print(mystring)
```

```
"We're moving to L.A.!"
```

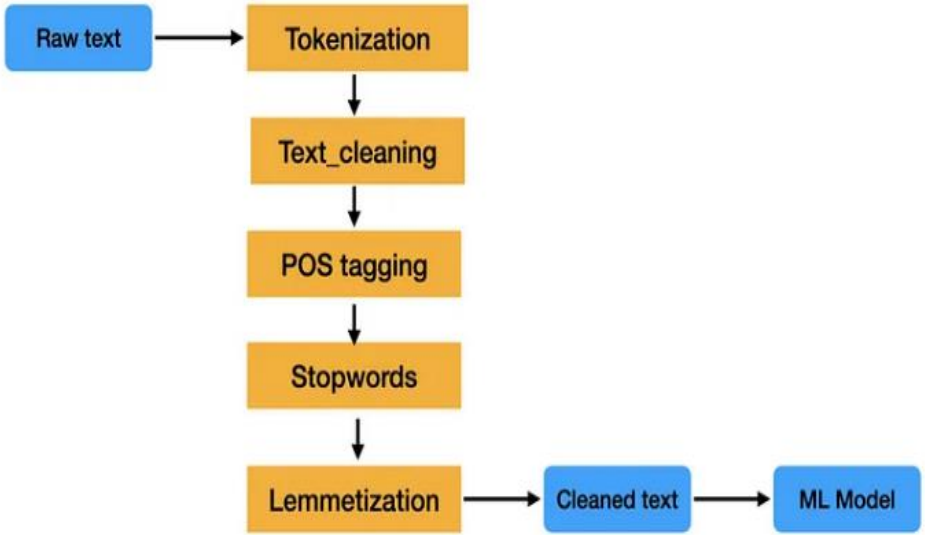
```
# Create a Doc object and explore tokens
doc = nlp(mystring)

for token in doc:
    print(token.text, end=' | ')
```

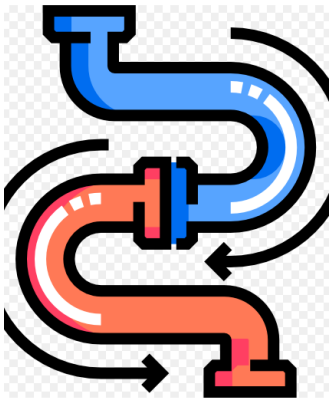
```
" | We | 're | moving | to | L.A. | ! | " |
```

Tokenization

- Notice that tokens are pieces of the original text.
- We don't see any conversion to word stems or lemmas (base forms of words) and we haven't seen anything about organizations/places/money etc.
- Tokens are the basic building blocks of a Doc object - everything that helps us understand the meaning of the text is derived from tokens and their relationship to one another.

Trained pipelines





Trained pipelines

Pipeline Package

- Binary weights
- Vocabulary
- Meta information
- Configuration file

en_core_web_sm

```
In [ ]:  import spacy

         # Load the language model with all components including the full vocabulary
         nlp = spacy.load("en_core_web_sm")
```

Trained pipelines

Predicting Part-of-speech Tags

Using spaCy to predict part-of-speech tags, the word types in context

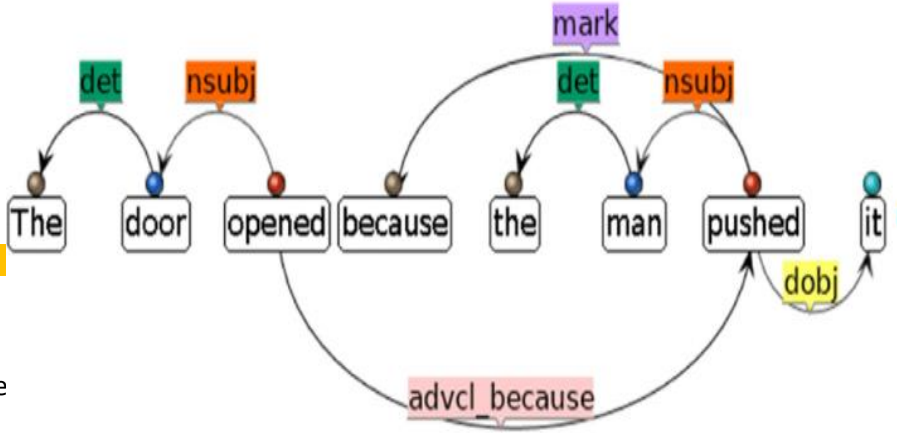
In [1]:

```
import spacy

# Load the small English pipeline
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
```

Trained pipelines

Predicting Syntactic Dependencies



In addition to the part-of-speech tags, we can also predict how the words are

- For example, whether a word is the subject of the sentence or an object.
- The .dep_ attribute returns the predicted dependency label.
- The .head attribute returns the syntactic head token. You can also think of it as the parent token this word is attached to.

In [2]:
```
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
```

Universal POS tags Definitions:
https://universaldependencies.org/u/pos/

Named Entity Recognition

Organizations,
Quantities,
Monetary values,
Percentages, and more.
People's names
Company names
Geographic locations (Both physical and political)
Product names
Dates and times
Amounts of money
Names of events

1. Named Entity Recognition is one of the key entity detection methods in NLP.

2. Named entity recognition is a natural language processing technique that can automatically scan entire articles and pull out some fundamental entities in a text and classify them into predefined categories. Entities may be, (see table of the left)

3. In simple words, Named Entity Recognition is the process of detecting the named entities such as person names, location names, company names, etc from the text.

4. It is also known as entity identification or entity extraction or entity chunking.

Named Entity Recognition

Use Case

Summarizing Resumes

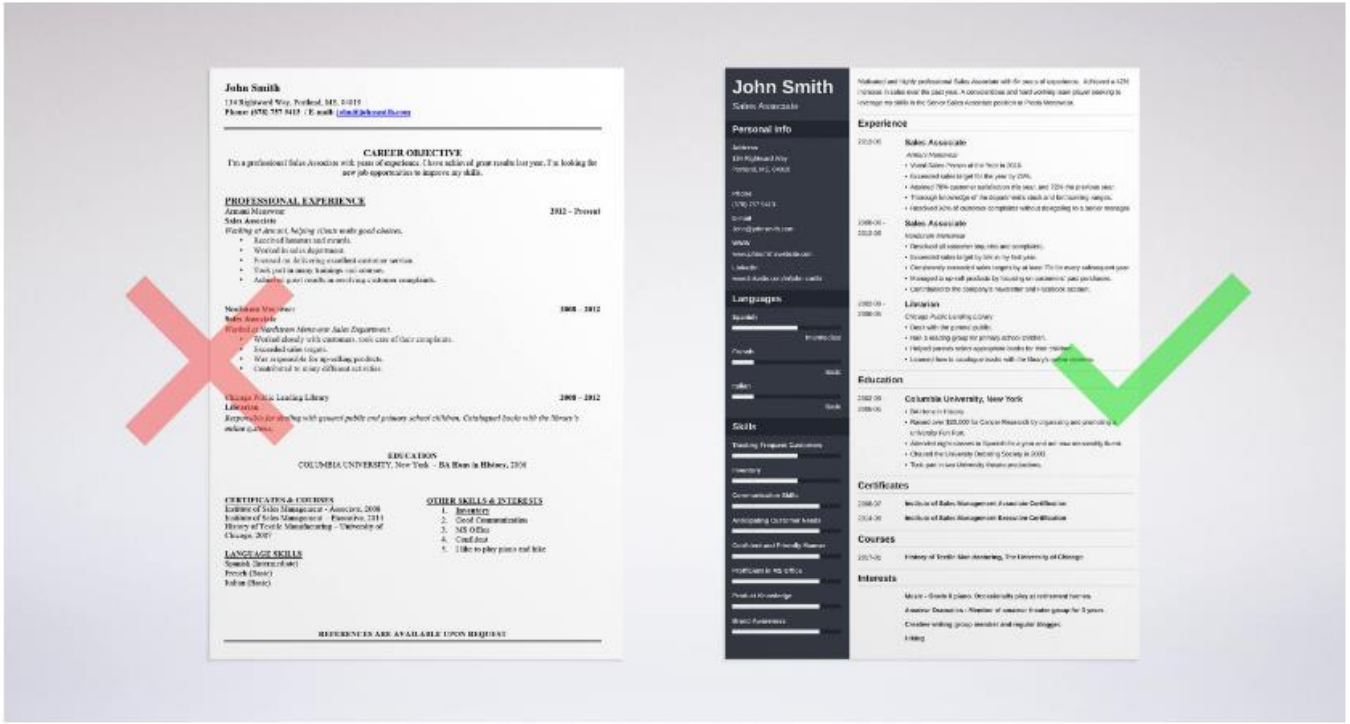


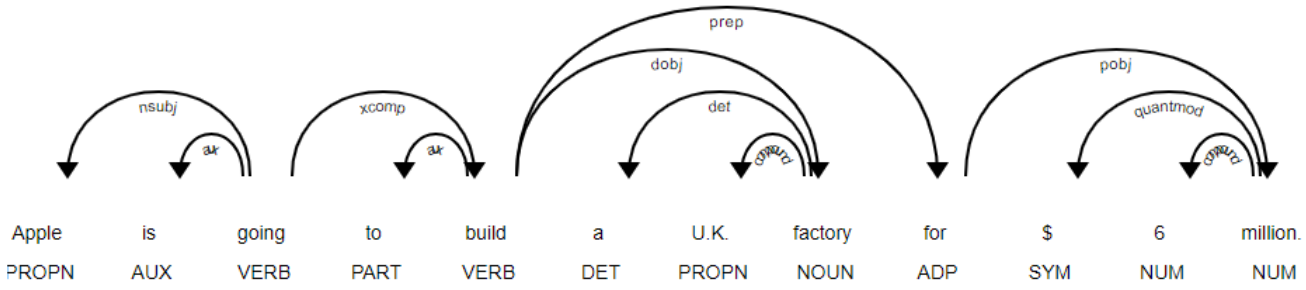
Image Source: Google Image

Visualizing the entity recognizer

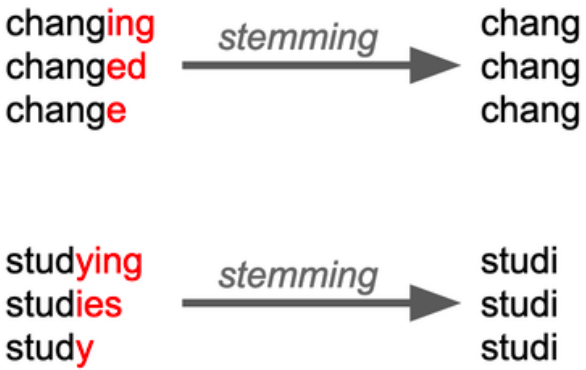
```
doc = nlp(u'Over the last quarter Apple sold nearly 20 thousand iPods for a profit of $6 million.')
displacy.render(doc, style='ent', jupyter=True)
```

Over the last quarter DATE Apple ORG sold nearly 20 thousand CARDINAL iPods PRODUCT for a profit of $6 million MONEY .

```
from spacy import displacy

doc = nlp(u'Apple is going to build a U.K. factory for $6 million.')
displacy.render(doc, style='dep', jupyter=True, options={'distance': 80})
```



Stemming



Stemming in NLP

Why use Stemming in NLP?

- The benefits of using the stemming algorithm in an NLP project can be summarized as follows:
- It reduces the number of words that serve as an input to the Machine Learning/Deep Learning model.
- It minimizes the confusion around words that have similar meanings.
- It lowers the complexity of the input space.
- When creating applications that search a specific text in a document, using stemming for indexing assists in retrieving relevant documents.
- It assists in eliminating the out-of-vocabulary (OOV) problem. For example, if the vocabulary does not contain the word ‘oranges’, one can use the stem word ‘orange’ as a proxy.
- It enhances the accuracy of the ML/DL model as the model does not have to deal with inflected word forms.
- You will be able to realize these advantages once you dive deeper into stemming and its various types. So, without any further ado, let’s get started.

Stemming

Porter Stemmer

- One of the most common - and effective - stemming tools is Porter's Algorithm developed by Martin Porter in 1980.
- The algorithm employs five phases of word reduction, each with its own set of mapping rules.

Stemming

S1		S2	word		stem
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
			ties	→	ti
SS	→	SS	caress	→	caress
S	→		cats	→	cat

- In the first phase, simple suffix mapping rules are defined.

Stemming

S1		S2	word		stem
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
			ties	→	ti
SS	→	SS	caress	→	caress
S	→		cats	→	cat

- From a given set of stemming rules only one rule is applied, based on the longest suffix S1. Thus, caresses reduces to caress but not cares.

Stemming

S1		S2	word		stem
(m>0) ATIONAL	→	ATE	relational	→	relate
			national	→	national
(m>0) EED	→	EE	agreed	→	agree
			feed	→	feed

- More sophisticated phases consider the length/complexity of the word before applying a rule. For example:

Stemming

- Snowball is the name of a stemming language also developed by Martin Porter.
- The algorithm used here is more accurately called the "English Stemmer" or "Porter2 Stemmer".
- It offers a slight improvement over the original Porter stemmer, both in logic and speed.

Natural Language Processing

- spaCy does not include built-in support for stemmers, it uses lemmatization which is a more powerful and accurate way to process text rather than stemmer. Lemmatization is the process of reducing words to their base form while taking into account the context and meaning of the word, this approach is considered to be more accurate and effective than stemming, as it takes into account the word's part of speech and its role in the sentence, which can help to preserve the meaning of the text.

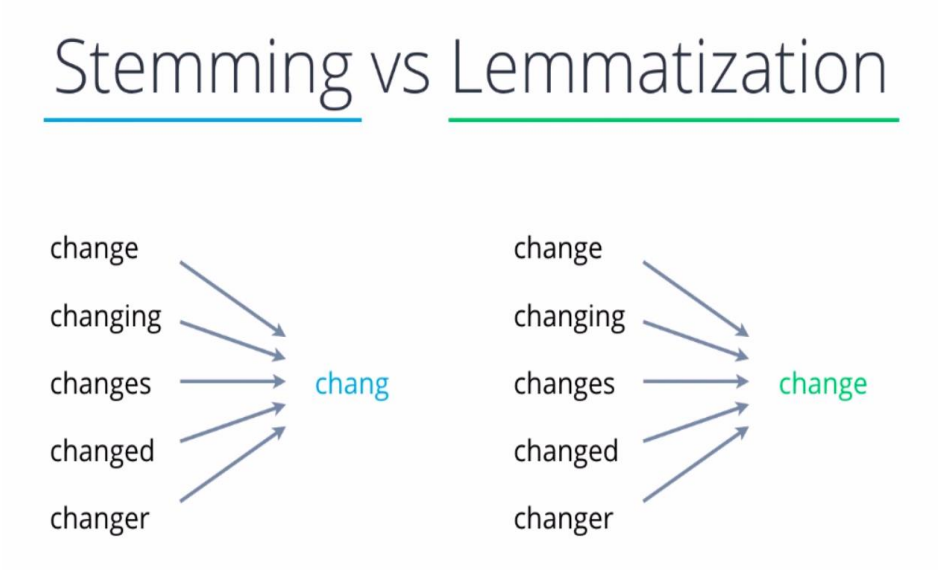
Lemmatization

Natural Language Processing

- Unlike stemming which only focuses on reducing words to their base form, lemmatization takes into account the context and meaning of a word by analyzing its morphological structure and considering the full vocabulary of the language.

- For example, the base form of 'was' is 'be' and the base form of 'mice' is 'mouse'. Additionally, the base form of 'meeting' could be 'meet' or 'meeting' depending on how it is used in a sentence.

Lemmatization



- Lemmatization is typically seen as much more informative than simple stemming, which is why Spacy has opted to only have Lemmatization available instead of Stemming.

Stemming vs. Lemmatization

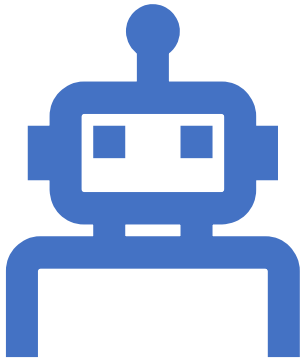
Word Before Lemmatization	After Lemmatization	
	SpaCy	NLTK
Walking	walk	Walking
is	be	is
main	main	main
gaits	gait	gait
terrestrial	terrestrial	terrestrial
locomotion	locomotion	locomotion
among	among	among
legged	legged	legged
animals	animal	animal
foxes	fox	fox
are	be	are
quick	quick	quick
they	they	they
are	be	are
jumping	jump	jumping
sleeping	sleep	sleeping
lazy	lazy	lazy
dogs	dog	dog

Common Use Cases for Lemmatization

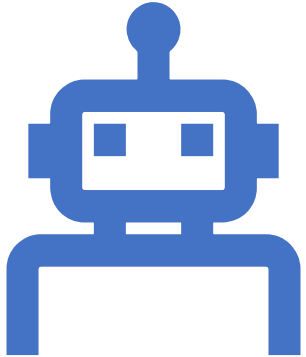
Use Case	Description
Information Retrieval	Enhancing search engines by matching queries with base forms of words, improving document retrieval accuracy.
Sentiment Analysis	Accurately determining sentiment by reducing words to their base forms, critical in understanding opinions and emotions.
Machine Translation	Improving the quality of machine translation by generating grammatically correct and contextually accurate translations.
Text Classification and NLP	Foundational preprocessing step in NLP tasks, including text classification, part-of-speech tagging, and syntactic analysis.
Document Summarization	Facilitating the identification of main topics and generating concise summaries for large volumes of text.

Lemmatization

- Lemmatization examines the context in which a word is used to identify its part of speech, unlike phrases that are not categorized.
- In the next lecture, we will delve into the concepts of word vectors and similarity.
- For now, let's focus on learning how to implement lemmatization using spaCy.



Lemmatization - Exercise

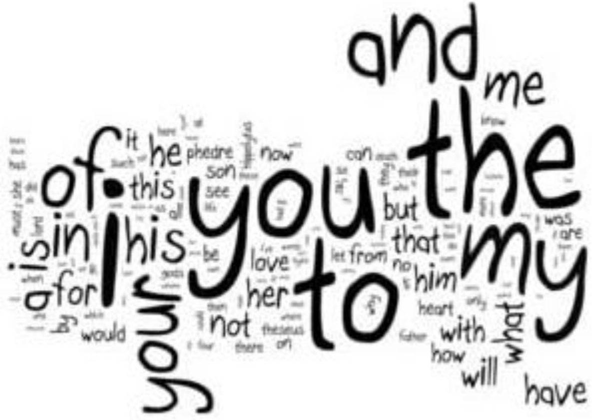


Q) Write a Python code to Lemmatize the following sentence

I am a runner running in a race because I love to run since I ran today.

<u>Use the following:</u>
token.text, token.pos_, token.lemma_, token.lemma

Stop Words


and
me
the
you
of
I
his
in
is
a
for
your
to
my
this
he
it
her
not
that
but
him
with
what
will
have
how
love
from
now
son
see
phedre
would
be
heart
theseus
there
was
are
can
all
let
no
on
father
only
by
such

Natural Language Processing

- Words like "a" and "the" appear so frequently that they don't require tagging as thoroughly as nouns, verbs and modifiers.
- We call these stop words, and they can be filtered from the text to be processed.
- Spacy holds a built-in list of some 305 English stop words.

Stop Words

Reason	Explanation
Reducing Noise	Stop words are frequently occurring words that contribute little to the meaning of a sentence.
Memory and Storage	Removing stop words reduces data size, saving memory and storage space.
Speeding Up Processing	Removing stop words reduces the number of tokens to process, improving efficiency.
Improving Model Performance	For tasks like TF-IDF and bag-of-words, excluding stop words can enhance important word representation.
Focusing on Content Words	Removal of stop words directs attention to more meaningful content words.

Vocabulary and Matching

Natural Language Processing

- In natural language processing (NLP), the vocabulary refers to the set of unique words or tokens that appear in a given text or corpus. These words or tokens are usually represented by a numerical index, and are used as the basic building blocks for many NLP tasks such as text classification, language translation, and text generation.

- Vocabulary matching is the process of identifying words or phrases that are present in a given text or corpus, and comparing them to a predefined set of words or phrases. This process is often used in NLP tasks such as named entity recognition, sentiment analysis, and text summarization.

Vocab and Matching Example

For example, let's say we have a corpus of text consisting of customer reviews for a particular product. We want to identify any mentions of the product's features in the reviews.

We can define a vocabulary of product features, such as "battery life", "camera quality", "display resolution", etc. and then match the words in the reviews against this vocabulary. Any words or phrases that match our vocabulary will be flagged as mentions of product features.

Another example, let's say you have a customer service chatbot that is designed to respond to customer inquiries. You would need a matching algorithm that can match customer inquiries to predefined questions or templates. This way the chatbot can provide the appropriate answer for each inquiry.

In both examples, the vocabulary matching process helps to extract relevant information from the text and make it easier to analyze and understand

```
import spacy
from spacy.matcher import Matcher

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

pattern1 = [{'LOWER': 'solarpower'}]
pattern2 = [{'LOWER': 'solar'}, {'LOWER': 'power'}]
pattern3 = [{'LOWER': 'solar'}, {'IS_PUNCT': True}, {'LOWER': 'power'}]

matcher.add("SolarPower", None, pattern1, pattern2, pattern3)
```

Applying the matcher to a Doc object

```
doc = nlp(u'The Solar Power industry continues to grow as demand \
for solarpower increases. Solar-power cars are gaining popularity.')
```

```
found_matches = matcher(doc)
print(found_matches)
```

```
[(8656102463236116519, 1, 3), (8656102463236116519, 10, 11), (8656102463236116519, 13, 16)]
```

`matcher` returns a list of tuples. Each tuple contains an ID for the match, with start & end tokens that map to the span `doc[start:end]`

```
for match_id, start, end in found_matches:
    string_id = nlp.vocab.strings[match_id]  # get string representation
    span = doc[start:end]                    # get the matched span
    print(match_id, string_id, start, end, span.text)
```

```
8656102463236116519 SolarPower 1 3 Solar Power
8656102463236116519 SolarPower 10 11 solarpower
8656102463236116519 SolarPower 13 16 Solar-power
```