Contents

[What is NLP? 2](#_Toc203289711)

[Common tasks in NLP 2](#_Toc203289712)

[What is spaCy? 2](#_Toc203289713)

[What can spaCy do? 2](#_Toc203289714)

[spaCy’s Pipeline 3](#_Toc203289715)

[Pretrained Models 3](#_Toc203289716)

[Containers 4](#_Toc203289717)

[Doc 4](#_Toc203289718)

[Token 6](#_Toc203289719)

[Word vectors 7](#_Toc203289720)

[Spacy Pipes 8](#_Toc203289721)

[Attribute ruler 8](#_Toc203289722)

[Rule-based approach 9](#_Toc203289723)

[Entity Ruler 9](#_Toc203289724)

[Matcher 10](#_Toc203289725)

[Matcher vs Entity Ruler 10](#_Toc203289726)

## What is NLP?

NLP (Natural Language Processing) is a field at the intersection of computer science, artificial intelligence (AI), and linguistics that focuses on enabling machines to understand, interpret, generate, and interact using human language.

## Common tasks in NLP

Text Classification: Assign categories to text

Tokenization: Break text into words or sentences

Named Entity Recognition (NER): Find names, dates, etc. in text.

Part-of-Speech Tagging: Label words by grammatical role

Machine Translation: Translate between languages

Text Generation: Generate human-like text

Question Answering: Find answers from text

Speech Recognition: Convert speech to text

## What is spaCy?

SpaCy is natural language processing library for python. It designed to help computer understand human language in efficient, powerful and production ready way.

## What can spaCy do?

spaCy can analyze and process text with variety of built-in capabilities:

1. Tokenization: split text into tokens (words, punctuation, etc.)
2. Part-of-speech (POS) tagging: identify each word’s grammatical role (noun, verb, etc.)
3. Named Entity Recognition (NER): Detect names of people, places companies, dates, etc.
4. Dependency Parsing: Understand sentence structure (who did what to whom)
5. Lemmatization: Get the base form of a word (“running” to “run”)
6. Sentence Segmentation: Detect where sentences begin and end.
7. Word vectors & similarity: Measure how similar words/sentences are (using embeddings)
8. Text classification: Categorize text into topics, labels or sentiments (trainable)

## spaCy’s Pipeline

spaCy takes the text and runs it through a sequence of processing steps, known as the NLP pipeline is modular -each step performs a specific task and adds information to the Doc object

when loading a model like en\_core\_web\_sm, the default pipeline looks like this:

[Tokenizer] → [Tagger] → [Parser] → [NER] → [Attribute Ruler] → [Lemmatizer]

1. Tokenizer: it splits the raw text into tokens (words, punctuation, etc.) every other step relies on this
2. Tagger (Part-of-speech tagger): assigns a part of speech (noun, verb, etc.) to each token it uses grammar analysis, syntactic pattern matching, etc.
3. Parser (Dependency Parser): analyzes the syntactic structure of the sentence, it finds which words depend on which (subject-verb-object relationships)
4. NER (Named Entity Recognizer): detects named entities in text: people, companies, money, dates, etc. and it adds .nets to the Doc, which is a list of all entities found.
5. Attribute Ruler: overrides or corrects token attributes (like POS, lemma) using rules. It used to fix common errors or customize behavior
6. Lemmatizer: reduces each word to its base form (lemma).

You can customize the pipeline (add or remove steps), even train your own models.

## Pretrained Models

spaCy provides many models out of the box, like:

* "en\_core\_web\_sm" → English, small (fast but less accurate)
* "en\_core\_web\_md" → English, medium (has word vectors)
* "xx\_ent\_wiki\_sm" → Multilingual NER model

## Containers

Containers are spacy objects that contain a large quantity of data about a text.

When we analyze texts with the spacy framework, we create different container objects to do that.

List of spacy containers:

* Doc
* DocBin
* Exampling
* Language
* Lexeme
* Span
* SpanGroup
* Token

### Doc

Doc is the main container for a processed text in spaCy. It holds all the annotations, like tokens, entities, sentences, and more. It is container for Tokens and spans

Spans: A slice of the Doc made up of consecutive tokens.

It is like parsed, intelligent version of the text.

Key components of a Doc:

1. Token: each word, punctuation, or symbol in the text.

for token in doc[0:10]:

    print (token)

#This is a tokenized version of text

#It is split into tokens (words, punctuation, numbers, etc.) according to spaCy's tokenizer

# can access each token

for token in text.split()[0:10]:

    print (token)

#The difference between the spaCy and split function that spaCy separate things like () from the words but the split function not

1. Text: the original full text.
2. Sentences: automatically segmented sentences.

for sent in doc.sents:

print(sent)

sentence1 = doc.sents[0] #doc.sents is a generator (not a list).

**TypeError**: '\_cython\_3\_1\_1.generator' object is not subscriptable

Why doc.sents a generator?

Using generator saves memory and improves performance especially when processing large documents. It avoids creating all sentences at once unless you need them

sentence1 = list(doc.sents)[0]

print (sentence1)

The solution is to convert the generator to a list

1. Entities: Named entities (like names, locations, organizations)
2. Noun chunks: Base noun phrases.
3. Vectors / Similarity: if using models with word vectors.

Internally, Doc is optimized using Cython and acts like a read-only container, efficiently storing annotations while being memory-friendly.

### Token

A Token is an object in spaCy that represents a **word**, **punctuation mark**, **number**, or **symbol** in your text.

It can be accessed like:

for token in doc:

    print(token.text)

Common token attributes:

| **Attribute** | **Description** | **Example** |
| --- | --- | --- |
| .text | The original word or symbol | "Apple" |
| .lemma\_ | The base form of the word | "running" → "run" |
| .pos\_ | Part-of-speech (broad category) | "NOUN", "VERB" |
| .tag\_ | Detailed part-of-speech | "NNP", "VBD" |
| .dep\_ | Syntactic dependency label | "nsubj", "dobj" |
| .head | The syntactic parent of this token | Token object |
| .shape\_ | Word shape (capitalization, punctuation) | "Xxxx", "d" |
| .is\_alpha | Is the token alphabetic? | True/False |
| .is\_stop | Is the token a stop word? | True/False |
| .is\_punct | Is it a punctuation mark? | True/False |
| .ent\_type\_ | Named entity type (if any) | "PERSON", "ORG" |
| .like\_num | Does it look like a number? | True for "42" |
| .n\_lefts, .n\_rights | Number of children in the dependency tree | Useful for parsing |
| .idx | Character offset of the token in the original text | e.g., 10 |

## Word vectors

Word vectors or word embeddings are numerical representation of words in multidimensional space through matrices typically a list of 100 to 300 numbers — that captures its semantic meaning.

Instead of thinking about words as just strings like “cat” or “car” word vectors can map them into vector space

The purpose of the word vector is to get a computer system to understand a word. Computers cannot understand text efficiently. They can process numbers quickly and will, so it is important to convert a word into a number.

The methods for creating word vectors in a pipeline take all words in a corpus and convert them into a single, unique number. These words are stored in a dictionary that would look like this: {“the”:1, “a”, 2} etc. this is known as a bag of words, this approach to representing words numerically however, only allow a computer to understand words numerically to identify unique words. It doesn’t allow a computer to understand meaning

Word vectors are used because they capture meaning and context

| **Words** | **Are Close In Vector Space** |
| --- | --- |
| "king" and "queen" | Similar in meaning |
| "Paris" and "France" | Related place and country |
| "run" and "walk" | Both are verbs of motion |
| "apple" (fruit) vs "Apple" (company) | Context helps distinguish them (in modern models like BERT) |

Visual Intuition (simplified to 2D):

king

|

man ---+--- woman

|

Queen

This geometric structure is what allows:

vector("king") - vector("man") + vector("woman") ≈ vector("queen")

## Spacy Pipes

Spacy is much more than an NLP framework. It is also a way of designing and implementing complex pipelines.

A pipeline is a sequence of pipes, or actors on data, that make alterations to the data or extract information from it.

i.e. Pipeline is a sequence of processing steps that take raw text and transform it into something useful like structured data, tokens, POS tags, entities, etc.

In some cases, later pipes require the output from earlier pipes, in other cases a pipe can exist entirely on its own. An example can be seen in the graph below.

**Sample pipeline for NER**

Entity ruler

Output sentence with entities annotated

Entity linker

Input sequence

### Attribute ruler

Attribute\_ruler in spaCy is a special pipeline component that lets you override or define custom rules for attributes like lemmas, part of speech tags or morphological features

It is especially useful when:

* The model makes consistent mistakes.
* You are working with domain-specific language (e.g., legal, medical)
* You want to fix or normalize certain words regardless of context.

We use attribute\_ruler when we want to inject specific rule, as when spaCy incorrectly lemmatizes “USA” to “usa” but you want it to stay “USA” so instead of training a new model, you can inject a rule like:

nlp.get\_pipe("attribute\_ruler").add(

    patterns=[{"TEXT": "USA"}],

    attrs={"LEMMA": "USA"}

)

## Rule-based approach

Rule-based system follows manually written rules to analyze or understand text.

If you can write a clear IF this happens → THEN do that rule, you’re using a rule-based approach.

Examples of rule-based logic:

* If the word is “USA”. Label it as a country.
* If a sentence ends in “?”, label it as a question
* If the word before “Street” is capitalized, it’s likely a location

Pros:

* Easy to understand and debug.
* No training data needed
* Works well in narrow, predictable domains (e.g., medical, legal)

Cons:

* Doesn’t generalize well
* Can’t handle ambiguity or variation (e.g., “Apple” as fruit vs company)
* Becomes hard to maintain as complexity grows.

### Entity Ruler

Entity ruler finds patterns and directly labels them as named entities (GPE, ORG, PERSON, etc.)

It also integrates into the spaCy pipeline and automatically populates doc.ents. and it can override or supplement the NER model

Note:

| **Situation** | **Outcome** |
| --- | --- |
| Add EntityRuler **before** ner | ✅ Best practice — rules take effect |
| Add EntityRuler **after** ner | ❌ Too late — doc.ents already filled |
| Run nlp(text) **before adding ruler** | ❌ Won’t see your new rule |
| Want to override NER completely | Must **replace doc.ents manually** |

### Matcher

Matcher is rule-based engine that lets you find sequences of tokens (words, punctuation, etc.) in document based on flexible patterns. "Find all phrases that match this shape, structure, or content."

Common token attributes that can be used in matcher:

| **Attribute** | **Description** |
| --- | --- |
| TEXT | Exact text of the token |
| LOWER | Lowercase version of the text |
| LEMMA | Lemma (base form) |
| POS | Part-of-speech tag |
| IS\_PUNCT | Is punctuation |
| IS\_DIGIT | Is a digit |
| IS\_TITLE | Starts with uppercase letter |

multiple condition in one token or Custom attributes like(ENT\_TYPE, DEP, etc.) can be also used.

### Matcher vs Entity Ruler

| **Feature** | **Matcher** | **Entity Ruler** |
| --- | --- | --- |
| Finds token patterns | ✅ Yes | ✅ Yes |
| Assigns entity labels | ❌ No (you do it manually) | ✅ Yes (automatic via label) |
| Updates doc.ents | ❌ No | ✅ Yes |
| Pipeline component | ❌ Not automatically in pipeline | ✅ Part of NLP pipeline (add\_pipe) |
| Best for | Pattern search | Rule-based NER |
| Custom labels | ✅ But manual | ✅ Built-in |

| **Task** | **Use** |
| --- | --- |
| Search patterns in text (e.g., verb + noun) | Matcher |
| Automatically add entities to doc.ents | EntityRuler |
| Add rules for missing or domain-specific NER | EntityRuler |
| Complex token-level matching without labels | Matcher |

## Important Links

<https://www.youtube.com/watch?v=dIUTsFT2MeQ>

<https://spacy.pythonhumanities.com/intro.html>