Day69_NLP_spaCy_vs_NLTK

August 21, 2025

1 spaCy for NLP

Introduction to spaCy

- spaCy is a free, open-source Python library for advanced Natural Language Processing (NLP).
- It is not an API — meaning it won't answer questions out of the box like ChatGPT, but it provides powerful tools to process and analyze text.
- Key idea: Whatever we did using NLTK, we can also do (and often faster) with spaCy.
- spaCy is designed for production use (speed + efficiency) and supports pre-trained language models.

2 Installation & Setup

[ ]:

```
# Install spaCy (only once)
!pip install spacy

# Download the English language model (small)
!python -m spacy download en_core_web_sm

# For larger models:
# en_core_web_md  (medium)
# en_core_web_lg  (large)
```

3 Loading a Model

[2]:

```
import spacy

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

# Example text
text = "Apple is looking at buying U.K. startup for $1 billion"

# Process the text into a spaCy Doc object
doc = nlp(text)
```

3.1 Tokenization

[3]:
```
# Print individual tokens
for token in doc:
    print(token.text)
```

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

3.2 Part of Speech (POS) Tagging

[4]:
```
# Print tokens with their POS tags
for token in doc:
    print(token.text, ":", token.pos_)
```

```
Apple : PROPN
is : AUX
looking : VERB
at : ADP
buying : VERB
U.K. : PROPN
startup : VERB
for : ADP
$ : SYM
1 : NUM
billion : NUM
```

3.3 Lemmatization & Dependency Parsing

[5]:
```
# Tokens with POS, Lemma (base form), and Dependency relation
for token in doc:
    print(token.text, ":", token.pos_, "-->", token.lemma_, "| Dependency:",␣
 ↪token.dep_)
```

```
Apple : PROPN --> Apple | Dependency: nsubj
is : AUX --> be | Dependency: aux
looking : VERB --> look | Dependency: ROOT
at : ADP --> at | Dependency: prep
buying : VERB --> buy | Dependency: pcomp
U.K. : PROPN --> U.K. | Dependency: nsubj
```

```
startup : VERB --> startup | Dependency: ccomp
for : ADP --> for | Dependency: prep
$ : SYM --> $ | Dependency: quantmod
1 : NUM --> 1 | Dependency: compound
billion : NUM --> billion | Dependency: pobj
```

3.4 Extended Token Attributes

[6]:

```python
# All in one: POS, Lemma, Dependency, Shape, Alphabet check, Stop word check
for token in doc:
    print(token.text, ":", token.pos_, "-->", token.lemma_, "|",
          "Dep:", token.dep_, "| Shape:", token.shape_,
          "| Alpha:", token.is_alpha, "| Stopword:", token.is_stop)
```

```
Apple : PROPN --> Apple | Dep: nsubj | Shape: Xxxxx | Alpha: True | Stopword:
False
is : AUX --> be | Dep: aux | Shape: xx | Alpha: True | Stopword: True
looking : VERB --> look | Dep: ROOT | Shape: xxxx | Alpha: True | Stopword:
False
at : ADP --> at | Dep: prep | Shape: xx | Alpha: True | Stopword: True
buying : VERB --> buy | Dep: pcomp | Shape: xxxx | Alpha: True | Stopword: False
U.K. : PROPN --> U.K. | Dep: nsubj | Shape: X.X. | Alpha: False | Stopword:
False
startup : VERB --> startup | Dep: ccomp | Shape: xxxx | Alpha: True | Stopword:
False
for : ADP --> for | Dep: prep | Shape: xxx | Alpha: True | Stopword: True
$ : SYM --> $ | Dep: quantmod | Shape: $ | Alpha: False | Stopword: False
1 : NUM --> 1 | Dep: compound | Shape: d | Alpha: False | Stopword: False
billion : NUM --> billion | Dep: pobj | Shape: xxxx | Alpha: True | Stopword:
False
```

> Note — **spaCy does not do *everything* NLTK does**, but it covers almost all the **practical / production-level NLP tasks**. Let me break this down clearly for your notebook:

4 NLTK vs spaCy – Coverage

4.1 Things you can do with both NLTK & spaCy

- **Tokenization** (word, sentence, paragraph)
- **Part of Speech (POS) tagging**
- **Lemmatization**
- **Dependency Parsing**
- **Named Entity Recognition (NER)**
- **Stopword Removal**
- **Word Similarity (using embeddings)**
- **Text Classification (with training)**

4.2 Things spaCy does better than NLTK

- **Speed** → much faster for large text.
- **Pre-trained models** → spaCy has `en_core_web_sm`, `md`, `lg`.
- **NER & Dependency Parsing** → built-in and more accurate.
- **Integration with deep learning** → easily works with PyTorch, TensorFlow.
- **Pipeline structure** → everything runs in a single `nlp()` call.

4.3 Things NLTK does that spaCy does not (or less focused)

- **Corpora access** (movie reviews, Brown corpus, WordNet, etc.).
- **Linguistic resources** (CFG parsing, grammar trees, etc.).
- **Educational focus** (helps students learn concepts).
- **Rule-based text processing** (regex tokenizer, stemmers).
- **Language coverage** → NLTK supports more small academic datasets.

4.4 Key Points:

- **If you want research, corpora, and learning basics → use NLTK.**
- **If you want production-ready pipelines, speed, and accuracy → use spaCy.**
- In real projects → many people **learn with NLTK, but deploy with spaCy**.

4.5 Quick Example: Same task in NLTK vs spaCy

4.5.1 Tokenization with NLTK

[7]:

```
from nltk.tokenize import word_tokenize
text = "Apple is looking at buying U.K. startup for $1 billion"
print(word_tokenize(text))
```

```
['Apple', 'is', 'looking', 'at', 'buying', 'U.K.', 'startup', 'for', '$', '1',
'billion']
```

4.5.2 Tokenization with spaCy

[8]:

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
print([token.text for token in doc])
```

```
['Apple', 'is', 'looking', 'at', 'buying', 'U.K.', 'startup', 'for', '$', '1',
'billion']
```

4.6 Advantages of spaCy

- Fast & efficient for large-scale NLP tasks.

- Provides state-of-the-art models for POS tagging, NER, dependency parsing.

- Easy to integrate with deep learning frameworks (TensorFlow, PyTorch).

- Production-ready with optimized pipelines.

4.7 Disadvantages of spaCy

- Less suitable for linguistic research (NLTK has more linguistic corpora).
- Requires downloading large models for advanced features.
- Fewer built-in datasets compared to NLTK.

5 Conclusion

- spaCy is a modern alternative to NLTK, better suited for real-world applications.
- It makes text processing pipelines faster and simpler.
- Great for tasks like Tokenization, POS tagging, Lemmatization, Named Entity Recognition (NER), and Dependency Parsing.