4.

spaCy's Pipelines

Dr. W.J.B. Mattingly
Smithsonian Data Science Lab and United States Holocaust Memorial Museum
August 2021

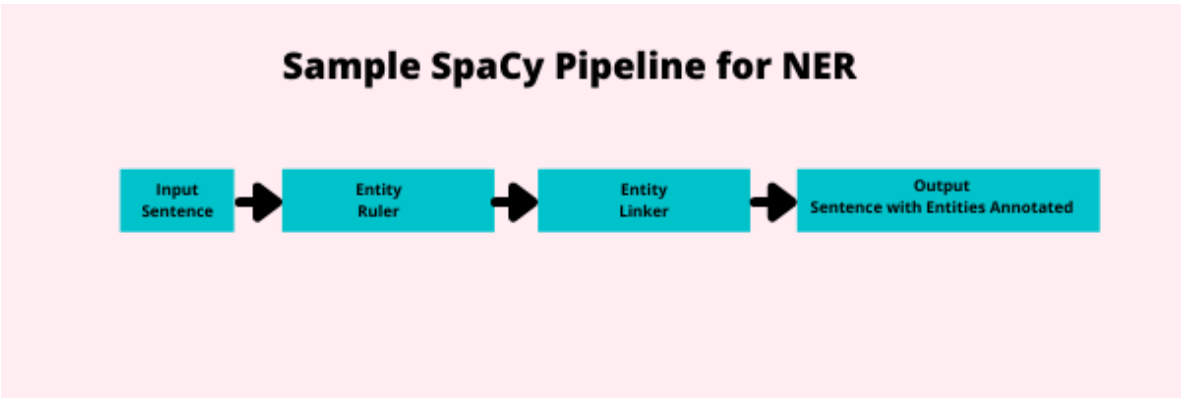
In this notebook, we will be learning about the various pipelines in spaCy. As we have seen, spaCy offers both heuristic (rules-based) and machine learning natural language processing solutions. These solutions are activated by pipes. In this notebook, you will learn about pipes and pipelines generally and the ones offered by spaCy specifically. In a later notebook, we will explore how you can create custom pipes and add them to a spaCy pipeline. Before we jump in, let's import spaCy.

```
import spacy
```

```
INFO:tensorflow:Enabling eager execution
INFO:tensorflow:Enabling v2 tensorshape
INFO:tensorflow:Enabling resource variables
INFO:tensorflow:Enabling tensor equality
INFO:tensorflow:Enabling control flow v2
```


4.1. Standard Pipes (Components and Factories) Available from spaCy

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. 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 see in the image below.



Here, we see an input, in this case a sentence, enter the pipeline from the left. Two pipes are activated on this, a rules-based named entity recognizer known as an EntityRuler which finds entities and an EntityLinker pipe that identifies what entity that is to perform toponym resolution. The sentence is then outputted with the sentence and the entities annotated. At this point, we could use the doc.ents feature to find the entities in our sentence. In spaCy, you will often use pipelines that are more sophisticated than this. You will specifically use a Tok2Vec input layer to vectorize your input sentence. This will allow machine learning pipes to make predictions.

Below is a complete list of the AttributeRuler pipes available to you from spaCy and the Matchers.

4.1.1. Attribute Rulers

- Dependency Parser
- EntityLinker
- EntityRecognizer
- EntityRuler
- Lemmatizer
- Morpholog
- SentenceRecognizer
- Sentencizer
- SpanCategorizer
- Tagger
- TextCategorizer

- Tok2Vec
- Tokenizer
- TrainablePipe
- Transformer

4.1.2. Matchers

- DependencyMatcher
- Matcher
- PhraseMatcher

4.2. How to Add Pipes

In most cases, you will use an off-the-shelf spaCy model. In some cases, however, an off-the-shelf model will not fill your needs or will perform a specific task very slowly. A good example of this is sentence tokenization. Imagine if you had a document that was around 1 million sentences long. Even if you used the small English model, your model would take a long time to process those 1 million sentences and separate them. In this instance, you would want to make a blank English model and simply add the Sentencizer to it. The reason is because each pipe in a pipeline will be activated (unless specified) and that means that each pipe from Dependency Parser to named entity recognition will be performed on your data. This is a serious waste of computational resources and time. The small model may take hours to achieve this task. By creating a blank model and simply adding a Sentencizer to it, you can reduce this time to merely minutes.

To demonstrate this process, let's first create a blank model.

```
nlp = spacy.blank("en")
```

Here, notice that we have used spacy.blank, rather than spacy.load. When we create a blank model, we simply pass the two letter combination for a language, in this case, en for English. Now, let's use the add_pipe() command to add a new pipe to it. We will simply add a sentencizer.

```
nlp.add_pipe("sentencizer")
```

```
<spacy.pipeline.sentencizer.Sentencizer at 0x2167b5468c0>
```

```
import requests
from bs4 import BeautifulSoup
s =
requests.get("https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare
.txt")
soup = BeautifulSoup(s.content).text.replace("-\n", "").replace("\n", " ")
nlp.max_length = 5278439
```

```
%%time
doc = nlp(soup)
print (len(list(doc.sents)))
```

```
94133
Wall time: 7.54 s
```

```
nlp2 = spacy.load("en_core_web_sm")
nlp2.max_length = 5278439
```

```
%%time
doc = nlp2(soup)
print (len(list(doc.sents)))
```

```
112074
Wall time: 47min 15s
```

The difference in time here is remarkable. Our text string was around 5.2 million characters. The blank model with just the Sentencizer completed its task in 7.54 seconds and found around 94k sentences. The small English model, the most efficient one offered by spaCy, did the same task in 46 minutes and 15 seconds and found around 112k sentences. The small English model, in other words, took approximately 380 times longer.

Often times you need to find sentences quickly, not necessarily accurately. In these instances, it makes sense to know tricks like the one above. This notebook concludes part one of this book.

4.3. Examining a Pipeline

In spaCy, we have a few different ways to study a pipeline. If we want to do this in a script, we can do the following command:

```
nlp2.analyze_pipes()
```


```
{'summary': {'tok2vec': {'assigns': ['doc.tensor'],
   'requires': [],
   'scores': [],
   'retokenizes': False},
  'tagger': {'assigns': ['token.tag'],
   'requires': [],
   'scores': ['tag_acc'],
   'retokenizes': False},
  'parser': {'assigns': ['token.dep',
    'token.head',
    'token.is_sent_start',
    'doc.sents'],
   'requires': [],
   'scores': ['dep_uas',
    'dep_las',
    'dep_las_per_type',
    'sents_p',
    'sents_r',
    'sents_f'],
   'retokenizes': False},
  'attribute_ruler': {'assigns': [],
   'requires': [],
   'scores': [],
   'retokenizes': False},
  'lemmatizer': {'assigns': ['token.lemma'],
   'requires': [],
   'scores': ['lemma_acc'],
   'retokenizes': False},
  'ner': {'assigns': ['doc.ents', 'token.ent_iob', 'token.ent_type'],
   'requires': [],
   'scores': ['ents_f', 'ents_p', 'ents_r', 'ents_per_type'],
   'retokenizes': False}},
 'problems': {'tok2vec': [],
  'tagger': [],
  'parser': [],
  'attribute_ruler': [],
  'lemmatizer': [],
  'ner': []},
 'attrs': {'token.lemma': {'assigns': ['lemmatizer'], 'requires': []},
  'doc.sents': {'assigns': ['parser'], 'requires': []},
  'token.is_sent_start': {'assigns': ['parser'], 'requires': []},
  'token.dep': {'assigns': ['parser'], 'requires': []},
  'token.tag': {'assigns': ['tagger'], 'requires': []},
  'doc.ents': {'assigns': ['ner'], 'requires': []},
  'token.ent_iob': {'assigns': ['ner'], 'requires': []},
  'token.head': {'assigns': ['parser'], 'requires': []},
  'doc.tensor': {'assigns': ['tok2vec'], 'requires': []},
  'token.ent_type': {'assigns': ['ner'], 'requires': []}}}
```


Note the dictionary structure. This tells us not only what is inside the pipeline, but its order. Each key after "summary" is a pipe. The value is a dictionary. This dictionary tells us a few different things. All of these value dictionaries state: "assigns" which corresponds to a value of what that particular pipe assigns to the token and doc as it passes through the pipeline. In some cases, there will be a key of "scores" in the dictionary. This indicates how the machine learning model was evaluated. We will learn more about model evaluation in our machine learning section below.

4.4. Conclusion

This notebook concludes part one of this book. It has given you an umbrella overview of spaCy. Over the next few parts of this book, we will deep dive into specific areas and use spaCy to solve general and domain-specific problems from several different areas of industry. Join me as we learn to create custom models and do custom things to leverage the full potential of the spaCy library.

By William Mattingly