5. Using SpaCy's EntityRuler

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

Print to PDF

5.1. Key Concepts in this Notebook

1. pipe
2. factory
3. EntityRuler
4. PhraseMatcher
5. Matcher

5.2. Introduction to Spacy's EntityRuler

The Python library spaCy offers a few different methods for performing rules-based NER. One such method is via its EntityRuler.

The **EntityRuler** is a spaCy factory that allows one to create a set of patterns with corresponding labels. A **factory** in spaCy is a set of classes and functions preloaded in spaCy that perform set tasks. In the case of the EntityRuler, the factory at hand allows the user to create an EntityRuler, give it a set of instructions, and then use this instructions to find and label entities.

Once the user has created the EntityRuler and given it a set of instructions, the user can then add it to the spaCy pipeline as a new pipe. I have spoken in the past notebooks briefly about pipes, but perhaps it is good to address them in more detail here.

A **pipe** is a component of a **pipeline**. A pipeline's purpose is to take input data, perform some sort of operations on that input data, and then output those operations either as a new data or extracted metadata. A pipe is an individual component of a pipeline. In the case of spaCy, there are a few different pipes that perform different tasks. The tokenizer, tokenizes the text into individual tokens; the parser, parses the text, and the NER identifies entities and labels them accordingly. All of this data is stored in the Doc object as we saw in Notebook 01_01 of this series.

It is important to remember that pipelines are sequential. This means that components earlier in a pipeline affect what later components receive. Sometimes this sequence is essential, meaning later pipes depend on earlier pipes. At other times, this sequence is not essential, meaning later pipes can function without earlier pipes. It is important to keep this in mind as you create custom spaCy models (or any pipeline for that matter).

In this notebook, we will be looking closely at the EntityRuler as a component of a spaCy model's pipeline. Off-the-shelf spaCy models come preloaded with an NER model; they do not, however, come with an EntityRuler. In order to incorperate an EntityRuler into a spaCy model, it must be created as a new pipe, given instructions, and then added to the model. Once this is complete, the user can save that new model with the EntityRuler to the disk.

The full documentation of spaCy EntityRuler can be found here: https://spacy.io/api/entityruler .

This notebook with synthesize this documentation for non-specialists and provide some examples of it in action.

5.3. Demonstration of EntityRuler in Action

In the code below, we will introduce a new pipe into spaCy's off-the-shelf small English model. The purpose of this EntityRuler will be to identify small villages in Poland correctly.

```
#Import the requisite library
import spacy

#Build upon the spaCy Small Model
nlp = spacy.load("en_core_web_sm")

#Sample text
text = "The village of Treblinka is in Poland. Treblinka was also an extermination
camp."

#Create the Doc object
doc = nlp(text)

#extract entities
for ent in doc.ents:
    print (ent.text, ent.label_)
```

```
Treblinka GPE
Poland GPE
```

Depending on the version of model you are using, some results may vary.

The output from the code above demonstrates spaCy's small model's to identify Treblinka, which is a small village in Poland. As the sample text indicates, it was also an extermination camp during WWII. In the first sentence, the spaCy model tagged Treblinka as an LOC (location) and in the second it was missed entirely. Both are either imprecise or wrong. I would have accepted ORG for the second sentence, as spaCy's model does not know how to classify an extermination camp, but what these results demonstrate is the model's failure to generalize on data. The reason? There are a few, but I suspect the model never encountered the word Treblinka.

This is a common problem in NLP for specific domains. Often times the domains in which we wish to deploy models, off-the-shelf models will fail because they have not been trained on domain-specific texts. We can resolve this, however, either via spaCy's EntityRuler or via training a new model. As we will see over the next few notebooks, we can use spaCy's EntityRuler to easily achieve both.

For now, let's first remedy the issue by giving the model instructions for correctly identifying Treblinka. For simplicity, we will use spaCy's GPE label. In a later notebook, we will teach a model to correctly identify Treblinka in the latter context as a concentration camp.

```
#Import the requisite library
import spacy

#Build upon the spaCy Small Model
nlp = spacy.load("en_core_web_sm")

#Sample text
text = "The village of Treblinka is in Poland. Treblinka was also an extermination
camp."

#Create the EntityRuler
ruler = nlp.add_pipe("entity_ruler")

#List of Entities and Patterns
patterns = [
                {"label": "GPE", "pattern": "Treblinka"}
            ]

ruler.add_patterns(patterns)


doc = nlp(text)

#extract entities
for ent in doc.ents:
    print (ent.text, ent.label_)
```

```
Treblinka LOC
Poland GPE
Treblinka GPE
```

If you executed the code above and found that you had the same output, then you did everything correctly. This method has failed. Why? The answer comes back to the concept of pipelines. We created and added the EntityRuler to the spaCy model's pipeline, but by default, spaCy add's a new pipe to the end of the pipeline. In order to visualize the pipeline, let's use spaCy's analyze_pipes().

```
nlp.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},
  'entity_ruler': {'assigns': ['doc.ents', 'token.ent_type', 'token.ent_iob'],
   'requires': [],
   'scores': ['ents_f', 'ents_p', 'ents_r', 'ents_per_type'],
   'retokenizes': False}},
 'problems': {'tok2vec': [],
  'tagger': [],
  'parser': [],
  'attribute_ruler': [],
  'lemmatizer': [],
  'ner': [],
  'entity_ruler': []},
 'attrs': {'token.ent_iob': {'assigns': ['ner', 'entity_ruler'],
   'requires': []},
  'token.tag': {'assigns': ['tagger'], 'requires': []},
  'token.dep': {'assigns': ['parser'], 'requires': []},
  'token.is_sent_start': {'assigns': ['parser'], 'requires': []},
  'token.lemma': {'assigns': ['lemmatizer'], 'requires': []},
  'token.ent_type': {'assigns': ['ner', 'entity_ruler'], 'requires': []},
  'doc.sents': {'assigns': ['parser'], 'requires': []},
  'token.head': {'assigns': ['parser'], 'requires': []},
  'doc.tensor': {'assigns': ['tok2vec'], 'requires': []},
  'doc.ents': {'assigns': ['ner', 'entity_ruler'], 'requires': []}}}
```

This can be a bit difficult to read at first, but what it shows us is the order in which our pipes are set up and a few other key pieces of information about each pipe. If we locate "ner", we notice that "entity_ruler" sits behind it.

In order for our EntityRuler to have primacy, we have to assign it to after the "ner" pipe, as the example below shows in this line:

ruler = nlp.add_pipe("entity_ruler", **after="ner"**)

```
#Build upon the spaCy Small Model
nlp = spacy.load("en_core_web_sm")

#Sample text
text = "The village of Treblinka is in Poland. Treblinka was also an extermination
camp."

#Create the EntityRuler
ruler = nlp.add_pipe("entity_ruler", after="ner")

#List of Entities and Patterns
patterns = [
                {"label": "GPE", "pattern": "Treblinka"}
            ]

ruler.add_patterns(patterns)


doc = nlp(text)

#extract entities
for ent in doc.ents:
    print (ent.text, ent.label_)
```

```
Treblinka GPE
Poland GPE
Treblinka GPE
```

Notice now that our EntityRuler is functioning before the "ner" pipe and is, therefore, prefinding entities and labeling them before the NER gets to them. Because it comes earlier in the pipeline, its metadata holds primacy over the later "ner" pipe.

5.4. Introducing Complex Rules and Variance to the EntityRuler (Advanced)

In some instances, labels may have a set type of variance that follow a distinct pattern or sets of patterns. One such example (included in the spaCy documentation) is phone numbers. In the United States, phone numbers have a few forms. The standard formal method is (xxx)-xxx-xxxx, but it is not uncommon to see xxx-xxx-xxxx or xxxxxxxxxx. If the owner of the phone number is giving that same number to someone outside the US, then +1(xxx)-xxx-xxxx.

If you are working within a United States domain, you can pass RegEx formulas to the pattern matcher to grab all of these instances.

The spaCy EntityRuler also allows the user to introduce a variety of complex rules and variances (via, among other things, RegEx) by passing the rules to the pattern. There are many arguments that one can pass to the patterns. For a complete list, see: https://spacy.io/usage/rule-based-matching . To expiremnet with how these work, I recommend using the spaCy Matcher demo: https://explosion.ai/demos/matcher .

In the example below we work with one example from the spaCy documentation in which we extract a phone number from a text. This same task can be done via RegEx as well.

```
#Import the requisite library
import spacy

#Sample text
text = "This is a sample number (555) 555-5555."

#Build upon the spaCy Small Model
nlp = spacy.blank("en")

#Create the Ruler and Add it
ruler = nlp.add_pipe("entity_ruler")

#List of Entities and Patterns (source: https://spacy.io/usage/rule-based-matching)
patterns = [
                {"label": "PHONE_NUMBER", "pattern": [{"ORTH": "("}, {"SHAPE": "ddd"},
{"ORTH": ")"}, {"SHAPE": "ddd"},
                {"ORTH": "-", "OP": "?"}, {"SHAPE": "dddd"}]}
            ]
#add patterns to ruler
ruler.add_patterns(patterns)



#create the doc
doc = nlp(text)

#extract entities
for ent in doc.ents:
    print (ent.text, ent.label_)
```

```
(555) 555-5555 PHONE_NUMBER
```

5.5. Other Rules-Based Matching Techniques in spaCy.

There are two other rules-based methods in spaCy: Matcher and PhraseMatcher. We have already met the Matcher in **01.03: Rules-Based Matching**. We will be meeting other more complex rules-based matching methods in the next few notebooks.

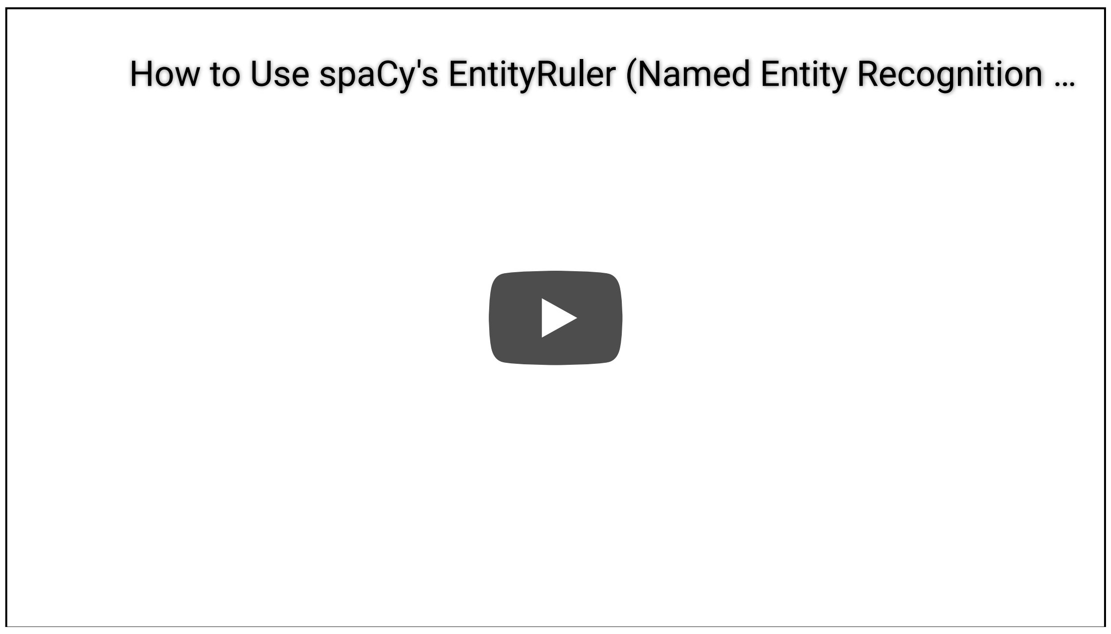
5.6. Exercise

For the exercise for this notebook, try and develop a custom EntityRuler to find a custom entity in your own domain-specific texts.

5.7. Video

Check out the video below in which we explore an entire use-case for spaCy's EntityRuler to find Harry Potter characters in the first book of Harry Potter.

```
%%html
<div align="center">
<iframe width="560" height="315" src="https://www.youtube.com/embed/wpyCzodv03A"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media;
gyroscope; picture-in-picture" allowfullscreen></iframe>
</div>
```


How to Use spaCy's EntityRuler (Named Entity Recognition ...