# 9. Working with Multi-Word Token Entities and RegEx in spaCy 3

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

Contents



## 9.1. Key Concepts in this Notebook

1. Working with Multi-Word Tokens and RegEx in spaCy 3x
2. RegEx's Finditer
3. Spans

## 9.2. Problems with Multi-Word Tokens in spaCy as Entities

As we saw in **01.03: Rules-Based NER**, we can use spaCy's Matcher to grab multi-word tokens, or tokens that span multiple tokens. The main problem with this, however, is that these multi-word tokens are not placed into the doc.ents. This means that we cannot access them the same way we would other entities. In this notebook, we will figure out how to solve that problem with a simple workflow:

1. Extract Multi-Word Tokens with re.finditer()
2. Reconstruct the spans in the spaCy doc
3. Give priority to longer spans (Optional)
4. Inject the Spans into doc.ents

We will cover each of these steps in turn.

## 9.3. Extract Multi-Word Tokens

First, we need to grab the multi-word tokens. In this notebook, we are going to try and grab a multi-word token. In this case, a person whose first name begins with Paul. In the RegEx below, we specify that we are looking for any string that starts with "Paul" and then is followed by a capitalized letter. We then tell it to grab the entire second word until the end of the word.

```
import re

text = "Paul Newman was an American actor, but Paul Hollywood is a British TV Host. The
name Paul is quite common."

pattern = r"Paul [A-Z]\w+"

matches = re.finditer(pattern, text)

for match in matches:
    print (match)
```

```
<re.Match object; span=(0, 11), match='Paul Newman'>
<re.Match object; span=(39, 53), match='Paul Hollywood'>
```

Note that we have not grabbed the final "Paul" which is not followed by a last name. In this case, we are not interested in that Paul. Now that we know how to grab the multi-word tokens, we need to have a way to parse them in spaCy.

## 9.4. Reconstruct Spans

This next stage is a bit more complicated, but works quite well once you understand the process. First, we need to import the libraries we will need. Note that we are also adding Span from spacy.tokens.

```
import re
import spacy
from spacy.tokens import Span
```

```
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
```

We will do the same thing that we did above with our text and our pattern.

```
text = "Paul Newman was an American actor, but Paul Hollywood is a British TV Host. The name Paul is quite common."
pattern = r"Paul [A-Z]\w+"
```

Here, we will create a blank spaCy English model and create the doc object of the text. It will have no entities in it because we are working with a blank model that does not have an “ner” component.

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

Even though this part is unnecessary, it is good to do it here because in other situations you will have entities. If you do, you need to store them as a separate list to which we will append things.

```
original_ents = list(doc.ents)
```

Now, let’s iterate over the results from re.finditer(). In this cell, we are goingg to grab the start and end from each match. we will then create a temporary span that will be equal to where the characters start and end in the doc object. This is important because tokens and characters do not always align correctly. Finally, we append to mwt_ents, the start, end, and text. The text is not necessary but it will help with debugging.

```
mwt_ents = []
for match in re.finditer(pattern, doc.text):
    start, end = match.span()
    span = doc.char_span(start, end)
    if span is not None:
        mwt_ents.append((span.start, span.end, span.text))
```

# 9.5. Inject the Spans into the doc.ents

With that data, we can iterate over each entity and identify where it begins and ends in spaCy. Note, we are using the spaCy Span class. This allows us to create a span object and assign it a custom label. With this data, we can append each Span to original_ents.

```
for ent in mwt_ents:
    start, end, name = ent
    per_ent = Span(doc, start, end, label="PERSON")
    original_ents.append(per_ent)
```

And finally, we set doc.ents equal to original_ents. This effectively loads the spans back into the spaCy doc.ents.

```
doc.ents = original_ents
```

Let’s iterate over the ents as we normally would.

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

```
Paul Newman PERSON
Paul Hollywood PERSON
```

Note that these are now properly identified entities in our doc.ents class.

# 9.6. Give priority to Longer Spans

Sometimes, the situation is not so neat. Sometimes our custom RegEx entities will overlap with spaCy's Entities

```
import re
import spacy

text = "Paul Newman was an American actor, but Paul Hollywood is a British TV Host."
pattern = r"Hollywood"

nlp = spacy.load("en_core_web_sm")

doc = nlp(text)
for ent in doc.ents:
    print (ent.text, ent.label_)
```

```
Paul Newman PERSON
American NORP
Paul Hollywood PERSON
British NORP
```

Let's say that we create a new entity. Maybe words associated with Cinema. So, we want to classify Hollywood as a tag "CINEMA". Now, in the above text, Hollywood is clearly associated with Paul Hollywood, but let's imagine for a moment that it is not. Let's try and run the same code as above. If we do, we notice that we get an error.

```
mwt_ents = []
original_ents = list(doc.ents)
for match in re.finditer(pattern, doc.text):
    print (match)
    start, end = match.span()
    span = doc.char_span(start, end)
    if span is not None:
        mwt_ents.append((span.start, span.end, span.text))
for ent in mwt_ents:
    start, end, name = ent
    per_ent = Span(doc, start, end, label="CINEMA")
    original_ents.append(per_ent)

doc.ents = original_ents
```

```
<re.Match object; span=(44, 53), match='Hollywood'>
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-425d356ded45> in <module>
     12     original_ents.append(per_ent)
     13
---> 14 doc.ents = original_ents

c:\users\wma22\appdata\local\programs\python\python39\lib\site-
packages\spacy\tokens\doc.pyx in spacy.tokens.doc.Doc.ents.__set__()

c:\users\wma22\appdata\local\programs\python\python39\lib\site-
packages\spacy\tokens\doc.pyx in spacy.tokens.doc.Doc.set_ents()

ValueError: [E1010] Unable to set entity information for token 9 which is included in
more than one span in entities, blocked, missing or outside.
```

This error tells us that one of our tokens from the finditer() overlapped with one that our "ner" component found. This is a problem that can be rectified with spaCy's filter_spans. This gives primacy to longer spans. Notice how we have allowed the Paul Hollywood entity to be a PERSON, rather than CINEMA. This is because Hollywood is shorter than Paul Hollywood.



```
from spacy.util import filter_spans
filtered = filter_spans(original_ents)
doc.ents = filtered
for ent in doc.ents:
    print (ent.text, ent.label_)
```

```
Paul Newman PERSON
American NORP
Paul Hollywood PERSON
British NORP
```

# 9.7. Video

Check out the video below on RegEx and Multi-Word Tokens in spaCy 3.

```
%%html
<div align="center">
<iframe width="560" height="315" src="https://www.youtube.com/embed/wpyCzodvO3A"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media;
gyroscope; picture-in-picture" allowfullscreen></iframe>
</div>
```

---

By William Mattingly