Sentence Segmentation in Natural Language Processing

spaCy NLP Series Part 5

Sentence Segmentation or Sentence Tokenization is the process of identifying different sentences among group of words. **spaCy** library designed for Natural Language Processing, perform the sentence segmentation with much higher accuracy. However, lets first talk about, how we as a human identify the start and end of the sentence? Mostly with the help of the punctuations, right? And in most of the cases we say a sentence ends with a dot '.' character. So with this basic idea, we would say that we can split the string based on dot and get the different sentences. Do you think this logic would be enough to get all sentence tokens?

What if there are some abbreviations within the sentences for example consider a sentence "U.K. has two dots in its abbreviation." In this case our brain is trained in such a way that it would not consider dots present in between the abbreviation as the end of the sentence. However it keeps reading until it reaches the dot which actually ends the sentence. Now see in this case, our split logic will fail completely. This is where we need some model training by scratch in text corpus or use some pre-trained models and tweak them based on our specific need. Spacy provides different models for different languages. I will be explaining the concept with respect to English language model. For other language support and installation instructions please refer the documentation available at spacy.io official site.

So, in this post we'll learn how sentence segmentation works, and how to set user defined segmentation rules.

In [30]:
```
# Perform standard imports
import spacy
nlp = spacy.load('en_core_web_sm') #Load English Language Model
```

In [63]:
```
string1 = "This is the first sentence. This is the second sentence. This is the third sentence."
doc = nlp(string1)

for sent in doc.sents:
    print(sent)
```

```
This is the first sentence.
This is the second sentence.
This is the third sentence.
```

Note: Doc.sents is a generator

A generator object cannot produce output/segmented until it is called. For example if there is a list object so you can iterate over it and print item one by

one. Also you can print the output using indexing like list[0], list[1] etc. without calling it explicitly means even without looping over it.

But this is not the case with generator objects. So the Doc is not segmented until doc.sents is called. This means that, where you could print the second Doc token (word token) with print(doc[1]), you can't call the "second Doc sentence" with print(doc.sents[1]):

In [64]:
```
print(doc[1])
print(doc[0])
```

```
is
This
```

Lets try printing the second sentence in the same way and see what happens:

In [65]:
```
print(doc.sents[1])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-65-2bc012eee1da> in <module>
----> 1 print(doc.sents[1])

TypeError: 'generator' object is not subscriptable
```

```
Now see the output, it is clearly telling that **generator object is not subscriptable**
```

In [66]:
```
for sent in doc.sents:
    print(sent)
```

```
This is the first sentence.
This is the second sentence.
This is the third sentence.
```

However, you *can* also build a sentence collection by running doc.sents and saving the result to a list:

In [67]:
```
doc_sents = [sent for sent in doc.sents]
doc_sents
```

Out[67]:
```
[This is the first sentence.,
 This is the second sentence.,
 This is the third sentence.]
```

In [6]:
```
# Now you can access individual sentences:
print(doc_sents[1])
```

```
This is another sentence.
```

NOTE: `list(doc.sents)` also works. We show a list comprehension as it allows you to pass in conditionals.

In [68]:
```
print(list(doc.sents))
print(list(doc.sents)[0])
```

```
[This is the first sentence., This is the second sentence., This is the third sentence.]
This is the first sentence.
```

sents are Spans having the start and end token pointers stored

At first glance it looks like each sent contains text from the original Doc object. In fact they're just Spans with start and end token pointers. By assigning start and end token pointers, spaCy recognizes the sentence tokens.

```
In [69]:
1 type(doc_sents[1])
```

```
Out[69]: spacy.tokens.span.Span
```

```
In [70]:
1 print(doc_sents[1].start, doc_sents[1].end)
```

```
6 12
```

Adding Rules | User defined Start and End of Sentence

spaCy's built-in sentencizer relies on the [dependency parse] and end-of-sentence punctuation to determine segmentation rules. We can add rules of our own, but they have to be added *before* the creation of the Doc object, as that is where the parsing of segment start tokens happens:

```
In [71]:
1 # Start token assignment happens during nlp processing pipline in soacy.
2 doc2 = nlp(u'This is the First sentence. This is the start of the Second Sentence . \
3             This is the start of the third sentence.')
4
5 for token in doc2:
6     print(token.is_sent_start, ' '+token.text)
```

```
True  This
None  is
None  the
None  First
None  sentence
None  .
True  This
None  is
None  the
None  start
None  of
None  the
None  Second
None  Sentence
None  .
None
True  This
None  is
None  the
None  start
None  of
None  the
None  third
None  sentence
None  .
```

Notice we haven't run doc2.sents, and yet token.is_sent_start was set to True on two tokens in the Doc. This means the sentence start and end token initialization happens during nlp pipeline itself.

Let's add a semicolon to our existing segmentation rules. That is, whenever the sentencizer encounters a semicolon, the next token should start a new segment.

In [72]:
```
# SPACY'S DEFAULT BEHAVIOR
doc3 = nlp(u'"Management is doing things right; leadership is doing the right things." -Peter Drucker')

for sent in doc3.sents:
    print(sent)
```

```
"Management is doing things right; leadership is doing the right things."
-Peter
Drucker
```

In [73]:
```
nlp = spacy.load('en_core_web_sm')
```

In [74]:
```
# ADD A NEW RULE TO THE PIPELINE
def set_custom_Sentence_end_points(doc):
    for token in doc[:-1]:
        if token.text == ';':
            doc[token.i+1].is_sent_start = True
    return doc

nlp.add_pipe(set_custom_Sentence_end_points, before='parser')

nlp.pipe_names
```

Out[74]:
```
['tagger', 'set_custom_Sentence_end_points', 'parser', 'ner']
```

The new rule has to run before the document is parsed.

In [75]:
```
# Re-run the Doc object creation:
doc4 = nlp(u'"Management is doing things right; leadership is doing the right things." -Peter Drucker')
for sent in doc4.sents:
    print(sent)
```

```
"Management is doing things right;
leadership is doing the right things."
-Peter
Drucker
```

In [76]:
```
# And yet the new rule doesn't apply to the older Doc object:
for sent in doc3.sents:
    print(sent)
```

```
"Management is doing things right; leadership is doing the right things."
-Peter
Drucker
```

Why not change the token directly?

Why not simply set the `.is_sent_start` value to True on existing tokens?

In [77]:
```
# Find the token we want to change:
doc3[7]
```

Out[77]:
```
leadership
```

In [78]:
```
# Try to change the .is_sent_start attribute:
doc3[7].is_sent_start = True
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-78-bcec3fe6a9a2> in <module>
      1 # Try to change the .is_sent_start attribute:
----> 2 doc3[7].is_sent_start = True

token.pyx in spacy.tokens.token.Token.is_sent_start.__set__()

ValueError: [E043] Refusing to write to token.sent_start if its document is parsed, because this may cause inconsistent state.
```

Note: spaCy refuses to change the tag after the document is parsed to prevent inconsistencies in the data.

Changing the Rules

In some cases we want to *replace* spaCy's default sentencizer with our own set of rules. In this section we'll see how the default sentencizer breaks on periods. We'll then replace this behaviour with a sentencizer that breaks on line breaks.

In [81]:
```
nlp = spacy.load('en_core_web_sm')  # reset to the original
mystring = u"This is a sentence. This is another.\n\nThis is a \nthird sentence."
 #SPACY DEFAULT BEHAVIOR:
doc = nlp(mystring)
for sent in doc.sents:
    print([token.text for token in sent])
```

```
['This', 'is', 'a', 'sentence', '.']
['This', 'is', 'another', '.', '\n\n']
['This', 'is', 'a', '\n', 'third', 'sentence', '.']
```

In [82]:
```
# CHANGING THE RULES
from spacy.pipeline import SentenceSegmenter
def split_on_newlines(doc):
    start = 0
    seen_newline = False
    for word in doc:
        if seen_newline:
            yield doc[start:word.i]
            start = word.i
            seen_newline = False
        elif word.text.startswith('\n'): # handles multiple occurrences
            seen_newline = True
    yield doc[start:]      # handles the last group of tokens
sbd = SentenceSegmenter(nlp.vocab, strategy=split_on_newlines)
nlp.add_pipe(sbd)
```

While the function `split_on_newlines` can be named anything we want, it's important to use the name `sbd` for the SentenceSegmenter.

In [83]:
```
doc = nlp(mystring)
for sent in doc.sents:
    print([token.text for token in sent])
```

```
['This', 'is', 'a', 'sentence', '.', 'This', 'is', 'another', '.', '\n\n']
['This', 'is', 'a', '\n']
['third', 'sentence', '.']
```

Here we see that periods no longer affect segmentation, only line breaks do. This would be appropriate when working with a long list of tweets, for instance.

This is all about Sentence Segmentation using spaCy. Hope you enjoyed the post.

If you like my posts, you are most welcome to connect with me on

LinkedIn: https://www.linkedin.com/in/ashutoshtripathi1/
Instagram: https://www.instagram.com/ashutosh_ai/
Facebook: https://www.facebook.com/enetworkai/

For more post please visit my Website: https://ashutoshtripathi.com/

Thank You!

References:

https://spacy.io/usage/spacy-101

https://www.udemy.com/course/nlp-natural-language-processing-with-python/