01-Intro-to-SpaCy

October 10, 2017

```
In [ ]: !python -m spacy download en
```

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

1 SpaCy: Industrial-Strength NLP

The tradtional NLP library has always been NLTK. While `NLTK` is still very useful for linguistics analysis and exporation, `spacy` has become a nice option for easy and fast implementation of the NLP pipeline. What's the NLP pipeline? It's a number of common steps computational linguists perform to help them (and the computer) better understand textual data. Digital Humanists are often fond of the pipeline because it gives us more things to count! Let's what `spacy` can give us that we can count.

```
In [1]: from datascience import *
        import spacy
```

Let's start out with a short string from our reading and see what happens.

```
In [2]: my_string = '''
        "What are you going to do with yourself this evening, Alfred?" said Mr.
        Royal to his companion, as they issued from his counting-house in New
        Orleans. "Perhaps I ought to apologize for not calling you Mr. King,
        considering the shortness of our acquaintance; but your father and I
        were like brothers in our youth, and you resemble him so much, I can
        hardly realize that you are not he himself, and I still a young man.
        It used to be a joke with us that we must be cousins, since he was a
        King and I was of the Royal family. So excuse me if I say to you, as
        I used to say to him. What are you going to do with yourself, Cousin
        Alfred?"

        "I thank you for the friendly familiarity," rejoined the young man.
        "It is pleasant to know that I remind you so strongly of my good
        father. My most earnest wish is to resemble him in character as much
        as I am said to resemble him in person. I have formed no plans for the
        evening. I was just about to ask you what there was best worth seeing
        or hearing in the Crescent City."'''.replace("\n", " ")
```

We've downloaded the English model, and now we just have to load it. This model will do ***everything*** for us, but we'll only get a little taste today.

```
In [3]: nlp = spacy.load('en', parser=False)  # run this instead if you don't have > 1GB RAM
```

To parse an entire text we just call the model on a string.

```
In [4]: parsed_text = nlp(my_string)
        parsed_text
```

```
Out[4]:  "What are you going to do with yourself this evening, Alfred?" said Mr. Royal to his
```

That was quick! So what happened? We've talked a lot about tokenizing, either in words or sentences.

What about sentences?

```
In [19]: sents_tab = Table()
         sents_tab.append_column(label="Sentence", values=[sentence.text for sentence in parse
         sents_tab.show()
```

```
    ---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call last)

    <ipython-input-19-58082b83b173> in <module>()
      1 sents_tab = Table()
----> 2 sents_tab.append_column(label="Sentence", values=[sentence.text for sentence in pa
      3 sents_tab.show()


    <ipython-input-19-58082b83b173> in <listcomp>(.0)
      1 sents_tab = Table()
----> 2 sents_tab.append_column(label="Sentence", values=[sentence.text for sentence in pa
      3 sents_tab.show()


    /srv/app/venv/lib/python3.6/site-packages/spacy/tokens/doc.pyx in __get__ (spacy/token


    ValueError: Sentence boundary detection requires the dependency parse, which requires
https://spacy.io/docs/usage
```

Words?

```
In [6]: toks_tab = Table()
        toks_tab.append_column(label="Word", values=[word.text for word in parsed_text])
        toks_tab.show()
```

```
<IPython.core.display.HTML object>
```

What about parts of speech?

```
In [7]: toks_tab.append_column(label="POS", values=[word.pos_ for word in parsed_text])
        toks_tab.show()
```

```
<IPython.core.display.HTML object>
```

Lemmata?

```
In [ ]: toks_tab.append_column(label="Lemma", values=[word.lemma_ for word in parsed_text])
        toks_tab.show()
```

What else? Let's just make a function `tablefy` that will make a table of all this information for us:

```
In [ ]: def tablefy(parsed_text):
            toks_tab = Table()
            toks_tab.append_column(label="Word", values=[word.text for word in parsed_text])
            toks_tab.append_column(label="POS", values=[word.pos_ for word in parsed_text])
            toks_tab.append_column(label="Lemma", values=[word.lemma_ for word in parsed_text])
            toks_tab.append_column(label="Stop Word", values=[word.is_stop for word in parsed_t
            toks_tab.append_column(label="Punctuation", values=[word.is_punct for word in parse
            toks_tab.append_column(label="Space", values=[word.is_space for word in parsed_text
            toks_tab.append_column(label="Number", values=[word.like_num for word in parsed_tex
            toks_tab.append_column(label="OOV", values=[word.is_oov for word in parsed_text])
            toks_tab.append_column(label="Dependency", values=[word.dep_ for word in parsed_tex
            return toks_tab
```

```
In [ ]: tablefy(parsed_text).show()
```

1.1 Challenge

What's the most common verb? Noun? What if you only include lemmata? What if you remove "stop words"?

How would lemmatizing or removing "stop words" help us better understand a text over regular tokenizing?

```
In [ ]: tablefy(parsed_text).where('POS',are.equal_to('NOUN')).group('Word').sort('count', desc
```

```
In [ ]: tablefy(parsed_text).where('Stop Word',are.equal_to(False)).group('Lemma').sort('count
```
