Module 8 : Natural Language Processing

Session 8B : Named Entity Recognition

Dr Daniel Chalk

"What's in a name?"

NIHR | Applied Research Collaboration South West Peninsula

@peninsula_ARC #HSMA4

HSMA 4

HSMA 4 Everyone

Let’s recap

A brief recap...

Named Entity Recognition

What do we mean by a “Named Entity”?

A Named Entity is a real world object that can be denoted with a proper name. The entity can have either physical existence (e.g. Dan Chalk, University of Exeter) or abstract existence (Discrete Event Simulation, Woodwork).

Types of Named Entity

There are lots of potential categories for Named Entities. SpaCy is a Python package we will be using today, and has its own set of categories (to which users can add their own) :

PERSON	People (including fictional)
NORP	Nationalities or Religious or Political Groups
FAC	Facilities (Buildings, airports, bridges etc)
ORG	Organisations (Companies, institutions etc)
GPE	Geo-Political Entities (Countries, cities, counties)
LOC	Non-GPE locations (mountain ranges, lakes etc)
PRODUCT	Objects, vehicles, foods etc (not services)
EVENT	Named storms, battles, wars, sports events etc
WORK_OF_ART	Titles of books, songs, films etc
LAW	Named documents made into laws
LANGUAGE	Any named language
DATE	Absolute or relative dates or periods
TIME	Times smaller than a day
PERCENT	Percentages, including the “%” character
MONEY	Monetary values, including the unit (e.g. £2.50)
QUANTITY	Measurements, such as weight or distance
ORDINAL	”First”, “Second” etc
CARDINAL	Numerals that do not fall under another type

Noun Phrases

Noun Phrases are phrases that are either headed by a noun or an indefinite pronoun (more on that shortly), or perform the same grammatical function as a noun.

Put another way, a noun phrase includes a noun (e.g. cat) and the modifiers which identify it (my cat, John’s cat, the cat with white stripes, the cat who ate all the tuna).

Definite Noun Phrases

A **definite noun phrase** is one where the head noun clearly refers to something specific - something previously mentioned or known, something unique, or something being identified by the speaker. The head noun is prefaced by the **“definite article”** (“the”), unless the specificity is contained in the noun instelf (such as with an ordinal number – e.g. “first”). The definite article may potentially followed by adjectives etc – we’ll come back to that.

Examples : I ran over to the dog. I sat next to the very fluffy cat.

Back to Noun Phrases

So why should you care about any of this? (Other than its obvious fun-conferring benefits)

Because Named Entities are found in…

Definite Noun Phrases.

So if we're looking for Named Entities, we need to find Definite Noun Phrases.

Not quite as simple as it may seem...

There are two core problems here :

1) We don’t just refer to nouns by themselves, we may use adjectives, for example to describe the nouns (e.g. we might say “the cat” or “the fluffy cat” - we’d need to pick up both)

2) We need to identify the head noun in a noun phrase, and there may be more than one contender (e.g “The vicar, Brian” “My cat, Bob”)

Parts of Speech - POS

Parts of Speech (or POS) refers to the categories that we assign to words that describe their syntactic functions.

There are 9 Parts of Speech in the English Language. They are (FUN?):

noun	name of a person, place, idea or thing
pronoun	used in place of a noun (eg me, he, she)
verb	an action, or expresses being
adjective	describing words for things–modifies noun or pronoun
adverb	as above, but for modifying verbs
determiner	limits or determines a noun (2 dogs, the rabbit)
preposition	governs a noun or pronoun to express a relation with another word or element in the clause (by, to, at etc)
conjuction	joins words, phrases or clauses (and, but, when etc)
interjection	word used to express emotion (Hey!)

The labelling of words according to their POS category is known as “POS Tagging” in Natural Language Processing.

Parsing

Parsing refers to the process of breaking down a sentence into its component parts and describing the syntactic roles of each component.

For example, through parsing we could identify the noun phrases in some text, and the POS Tags.

Regular Expressions

A Regular Expression (or regex) is a sequence of characters that define a search pattern. They are used in many applications, including search engines, to define the rules for the pattern of text that you want to search for.

When parsing, we can use regular expressions to specify rules for identifying certain syntactic structures in sentences.

Issues with the regex approach

Some potential problems :

1. This is a pain in the bottom. It turns out there are HUGE numbers of potential combinations of POS Tags that could form definite noun phrases. And some of the rules you write may find things that AREN'T definite noun phrases, as well as ones that are.

2. Real world free text is rarely written in perfect (or even good) grammar, but regular expressions assume the text is (or is otherwise 'consistent').

An alternative solution

So if the regex approach to finding definite noun phrases isn’t ideal for our problem of finding named entities, because it is too difficult and too reliant on the text following consistent grammatical rules, what can we do?

Fortunately, there is an alternative : SpaCy – a Python library for Natural Language Processing that includes, amongst many other things, AI-based methods for Named Entity Recognition.

SpaCy

SpaCy is a well regarded and widely used Python library that makes many Natural Language Processing tasks, particularly parsing-based tasks, very easy.

https://spacy.io/

To install SpaCy, just type **pip install spacy** from your command prompt or terminal.

Neural Networks

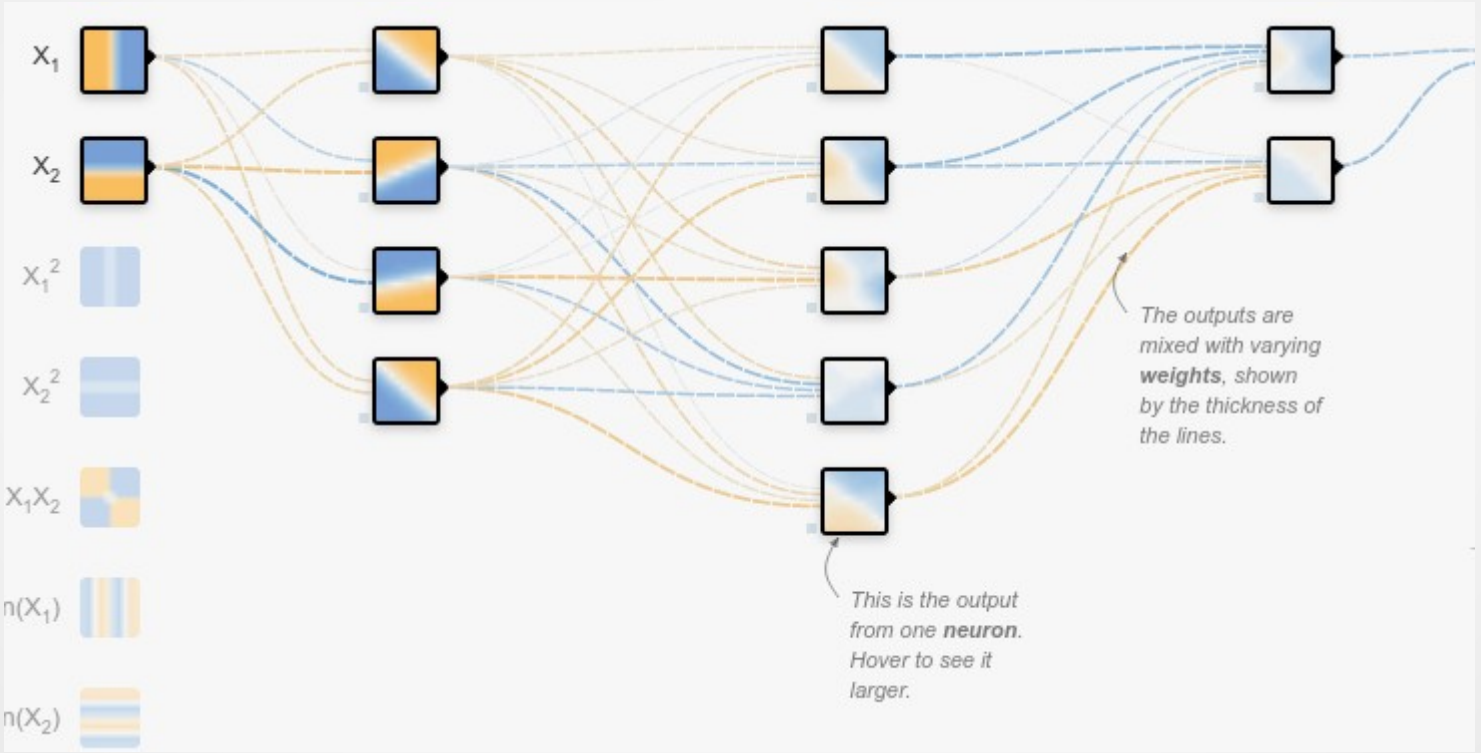
SpaCy learns using something called a **Convolutional Neural Network**. We're not going to go into detail about this at this stage, but we will just give you a very basic understanding of what a Neural Network does.

Neural Networks

Neural Networks are setup to emulate the way in which a brain works – *neurons* connected by *synapses* with *signals* being sent along the synapses to the neurons.

Neural Networks have multiple *layers* of *nodes* (emulating neurons) and *connections* between these nodes on different layers (emulating synapses), with inputs to and outputs from nodes being sent as signals to the other nodes, and a *weight* associated with each signal.

Neural Networks



Neural Networks

In SpaCy, the inputs to the neural network are the POS-tagged groups of words with a corpus, converted to numbers, and it's trying to adjust its weights and internal functions so that when it's told, for example, that word 3 in the group of words is a Named Entity (an output of 1), it is able to come to the same output based on the POS Tags of Word 3, word 2, word 4, word 1 etc.

Key takeaway message :

SpaCy predicts whether a word / words represents a Named Entity based on prior learning from massive amounts of text, where it has learned the kinds of syntactic patterns that surround a Named Entity.

Let’s try SpaCy

So, now we know how it works, let’s try out SpaCy!

Make sure you’ve installed SpaCy before proceeding :

pip install spacy

and you also need to download and install one of the learning models (pre-trained SpaCy model that’s been trained on general text) by going into your Command Prompt or Terminal and typing :

python -m spacy download en_core_web_sm

Let’s try SpaCy

You’ve been provided with a file called “TrumpArticle.txt”, which is text from a New York Times news article about the firing of an FBI agent.

We’re going to use SpaCy to extract the named entities (and their predicted classifications) from this news story.

Let’s open up the file trump_extract.py, which is the code that will extract the named entities from the .txt file. Make sure both the .py file and the .txt file are in the same directory.

Talking through the code

```
# import spacy and the downloaded en_core_web_sm pre-trained model
import spacy
import en_core_web_sm

# Load the pre-trained model as a language into a variable called nlp
nlp = en_core_web_sm.load()

# Read in the document for which we want to extract named entities
with open("TrumpArticle.txt", encoding='utf8') as f:
    # Read in the whole file as a single string but strip out any spaces at
    # the start and end of the file
    raw_read = f.read().strip()

# Apply the pre-trained model to the raw text string to extract named entities
article = nlp(raw_read)

# Store the Named Entity categories (stored in label_ for each entity) in the
# article in a list. The named entities themselves are stored in article.ents
labels = [x.label_ for x in article.ents]

# Print each predicted named entity, along with its predicted category
for i in range(len(article.ents)):
    print (f"{article.ents[i]} : {labels[i]}")
```

Running the code

```
In [3]: runfile('/home/dan/Dropbox/HSMA 3/phase_1_training/11
WASHINGTON : GPE
Peter Strzok : PERSON
F.B.I. : ORG
Trump : PERSON
Hillary Clinton : PERSON
Russia : GPE
Strzok : PERSON
Monday : DATE
Trump : PERSON
2016 : DATE
F.B.I. : ORG
Lisa Page : PERSON
Russia : GPE
Strzok : PERSON
20 years : DATE
F.B.I. : ORG
the early months : DATE
Strzok : PERSON
F.B.I. : ORG
Trump : PERSON
Strzok : PERSON
last summer : DATE
Robert S. Mueller III : PERSON
Strzok : PERSON
Twitter : ORG
Monday : DATE
Trump : PERSON
June : DATE
Strzok : PERSON
F.B.I. : ORG
Hillary Clinton : PERSON
2016 : DATE
Strzok : PERSON
the bureau's Office of Professional Responsibility : ORG
Strzok : PERSON
60 days : DATE
Strzok : PERSON
House : ORG
July : DATE
Strzok : PERSON
F.B.I. : ORG
David Bowdich : PERSON
the Office of Professional Responsibility : ORG
Strzok : PERSON
F.B.I. : ORG
Strzok : PERSON
Trump : PERSON
F.B.I. : ORG
Bowdich : PERSON
F.B.I. : ORG
Christopher A. Wray : PERSON
Aitan Goelman : PERSON
Strzok : PERSON
Special Agent Strzok : ORG
Wray : PERSON
Congress : ORG
F.B.I. : ORG
Goelman : PERSON
Americans : NORP
Goelman : PERSON
Strzok : PERSON
Page : PERSON
```

Some useful extras ahead of the exercise

I'm going to ask you to undertake an exercise in a moment. Here are some extra bits of Python that will make the exercise easier for you…

The count() function

You can call the count() function on an iterable object, such as a list or a dictionary. It counts how many items match the value you provide :

```
list_a = [1,2,3,3,2,2]
number_of_2s = list_a.count(2)
print (number_of_2s)
```

```
3
```

Dictionary Comprehension

This is just like list comprehension, but for dictionaries :

```
list_a = [1,2,3,3,2,2]
frequency_dictionary = {num:list_a.count(num) for num in list_a}
print (frequency_dictionary)
```

```
{1: 1, 2: 3, 3: 2}
```

Collections Counter

The collections library has a module called Counter. This allows us to access useful methods such as retrieving the x most common values as a list of (val, freq) tuples :

```
from collections import Counter
list_a = [1,2,3,3,2,2]
two_most_common = Counter(list_a).most_common(2)
print (two_most_common)
```

Freq

```
[(2, 3), (3, 2)]
```

Value

Exercise 1

I want you to extend the code in trump_extract.py (take a copy first) to do the following :

1. Store the named entities and their frequencies within the article in a dictionary, with keys representing the named entity strings, and the values representing their frequencies. You should first grab the the named entities, convert them to strings and store them in a list. Then use a dictionary comprehension to create the dictionary described above, and print the dictionary.
2. Print the 10 most common named entities in the text.
3. Print the 16th sentence of the article (remember to start counting from 0). The sentences of an article are stored in *article.sents*
4. Create a set which contains only unique named entity labels (types). Remember a Python set only contains unique values, so if you cast something as a set, it'll remove duplicates for you…
5. Print the set of unique labels to the user, and ask them to select one of them. Then, display all of the named entities with that label (so if the user types "ORG", display all the named entities with a label of "ORG"). Hint : use a list comprehension to grab the named entities from article.ents (remember to cast as strings) where their corresponding label meets the condition you need. Then you just need to cast the list as a set to remove duplicates. For extra points, contain all this in an infinite loop so that the user can keep selecting labels, and print a warning message if they type a label that isn't recognised.

You have 40 minutes. You should work in groups.

Updating Training with New Examples

Whilst the pre-trained models in SpaCy are good, they won't recognise everything. There will be words and syntactic patterns that it hasn't seen before in the training corpuses. There are two things that can be done to improve accuracy :

1) Select a pre-trained model that best suits the text from which you are trying to extract information. There are three core models in English on the spacy website that have been trained on increasingly larger corpuses (https://spacy.io/models/en) (sm = small, md = medium, lg = large).

The core models are trained on a range of general texts such as blogs, news and comments. Other models are available in the wild that have been trained in more specialist areas, such as tweets on Twitter or medical papers.

Updating Training with New Examples

2) We can add new examples to a pre-trained model so that it learns to identify new words and syntactic patterns that it hasn't seen in the past to help it improve.

Generally, you need lots of new examples to meaningfully improve things (ideally at least a few hundred).

It can also be quite difficult to update SpaCy training (and they've just changed the process, at the time of writing). But if you want to see how to do it, there are instructions here :

https://spacy.io/usage/training#training-data

Word Clouds

When we're undertaking NLP tasks such as Named Entity Recognition and Sentiment Analysis, it's often useful to get a visualisation of our text to understand some of the frequent things that are being said.

Word Clouds are a really nice visualisation technique that allow us to do exactly that.

In Python, we can use the WordCloud package to generate word clouds easily. To install WordCloud, we use :

pip install wordcloud

Word Clouds

Let’s have a look at how we can use the WordCloud package to generate word clouds. The code I’m going to be demonstrating is wordcloud_example.py, and we’re going to generate a word cloud from a random IMDB movie review, stored in the text file wordcloud_example_text.txt.

En route, I’m going to talk to you about tokenisation, stopwords and translation tables – all important concepts in Natural Language Processing.

Exercise 2

You have been provided with a file called exercise_2_article.txt. This file contains text from a news story featured on our website some time ago about some work that Sean and Kerry did with one of our HSMA alumni at Devon Partnership Trust.

In your groups, I want you to use the techniques I've shown you (Named Entity Recognition, the Counter library, Word Clouds etc) to generate some insights about this text. You should refine your approach as you go (for example, are there named entities that you are picking out that are less relevant and can be excluded? Are there types of words that are commonly being classified in one type of named entity that might be useful to know? What are some of the most common named entities and common words (excluding stopwords)? Can you optimise your word cloud visualisation to be insightful but clear?).

You have 50 minutes. I will ask a selection of groups to present what they've done, along with their findings / insights at the end of the exercise, so be sure to nominate someone to present. Imagine you are presenting the information to someone who has asked for this piece of analytical work to be undertaken.