

Named Entity Recognition

Named Entity Recognition
Dr Daniel Chalk



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”) (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...

...can anyone guess...?

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)
conjunction	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

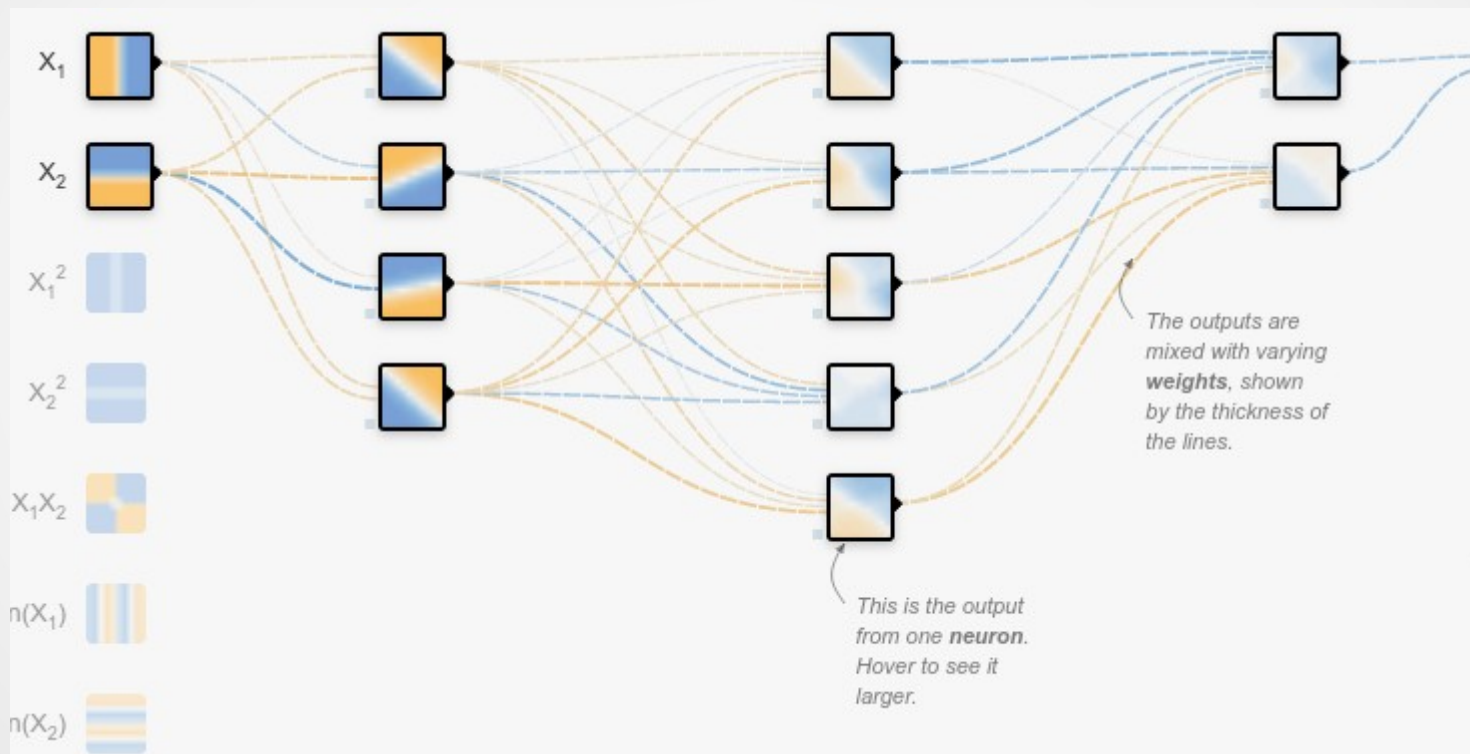
SpaCy learns using something called a **Convolutional Neural Network**. We're not going to go into detail about this at this stage, but we will just give you a very basic understanding of what a Neural Network does.

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 it's 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 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 (article.ents[i], " : ", labels[i], sep="")
```


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
```

Exercise 1

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

1. Print the 10 most common named entities in the text
2. Print sentence number 15 of the article
3. Print a unique list (no duplicates) of the named entity categories found in the article
4. Ask the user to input one of these named entity categories, and then display all of the unique named entities found in that category

Hints :

- The *Counter* module from the *collections* library allows you to easily count occurrences of entities such as list elements. You should first convert the named entities SpaCy has found into a list (remember casting...), and then you can use the *most_common(x)* function of the *Counter* module, passing your list into the *Counter* module, and passing the number of most common entries you want into the *most_common* method (x).
- The individual sentences of the article are stored in *article.sents*. Again, you'll need to convert this to a list first.
- Use for loops to generate lists of unique entries, adding to your new list, and checking each item you encounter isn't a duplicate.

You have 40 minutes. You may wish to work in groups, but either way, use your Peer Support Group.

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.

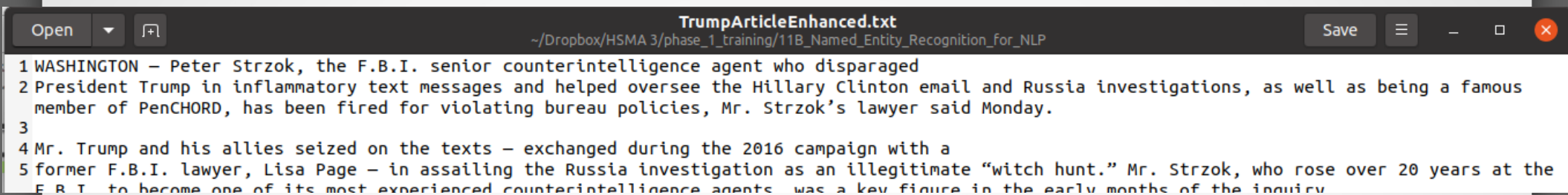
Updating Training with New Examples

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). But let's look at a simple example to see how this works.

Open up `"trump_extract_add_to_training.py"`. And let's have a quick look at the new "enhanced" article we'll be using (`"TrumpArticleEnhanced.txt"`)

Updating Training with New Examples



```
Open [icon] TrumpArticleEnhanced.txt
~/Dropbox/HSMA 3/phase_1_training/11B_Named_Entity_Recognition_for_NLP
Save [icon] [icon] [icon] [icon]
1 WASHINGTON – Peter Strzok, the F.B.I. senior counterintelligence agent who disparaged
2 President Trump in inflammatory text messages and helped oversee the Hillary Clinton email and Russia investigations, as well as being a famous
  member of PenCHORD, has been fired for violating bureau policies, Mr. Strzok’s lawyer said Monday.
3
4 Mr. Trump and his allies seized on the texts – exchanged during the 2016 campaign with a
5 former F.B.I. lawyer, Lisa Page – in assailing the Russia investigation as an illegitimate “witch hunt.” Mr. Strzok, who rose over 20 years at the
  F.B.I. to become one of its most experienced counterintelligence agents, was a key figure in the early months of the inquiry.
```

But if we run the default model on this article, it'll pick up PenCHORD as a Named Entity (Good) but will classify it as a Geo-Political Entity (Bad) :

```
In [6]: runfile('/home/dan/Dr
WASHINGTON : GPE
Peter Strzok : PERSON
F.B.I. : ORG
Trump : PERSON
Hillary Clinton : PERSON
Russia : GPE
PenCHORD : GPE
Strzok : PERSON
Monday : DATE
Trump : PERSON
2016 : DATE
```

Let's look at the new .py file to see how we can teach SpaCy that PenCHORD is an Organisation (ORG).

Exercise 2

You have been provided with a new article that is taken from a recent news story about a PenCHORD project (“exercise_2_article.txt”), and a file named “ex_2_training.csv”, which is a .csv file in which the first entry (column) of each row contains a training sentence, and subsequent columns give the named entities in that sentence (*exactly* as they’re written) along with the SpaCy Named Entity label for each named entity, in pairs of columns (so if there are two named entities in the sentence in column 1, there will be a total of 5 columns – the first column for the sentence, the second and third columns for the named entity text and label (respectively) of the first named entity, and the fourth and fifth columns for the named entity text and label (respectively) of the second named entity).

Your task is to write a program that :

- Asks the user to select one of three options :

- 1) To attempt Named Entity Recognition using the en_core_web_sm model
 - 2) To attempt Named Entity Recognition using an enhanced version of the en_core_web_sm model, which includes additional training
 - 3) To train on the training sentences and named entities given in the ex_2_training.csv file, and save this model as the enhanced model
- If the user selects option 1 or 2, then the appropriate model should be loaded and named entity recognition attempted with the loaded model. Once that’s done, the model should print out a list of the *unique* named entities it found (both the text and label for each entity), and also store them in a .csv file called “predicted_named_entities.csv”
- If the user selects option 3, then the program should read in the data from the .csv file and put use it to train and save an enhanced version of the model. To read in the data in the format you need, you should use a reader from the csv library, and for each row read in by the reader :
- a) take a copy of the row and store it in a temporary list (remember to use .copy())
 - b) remove any empty string values from the temporary list (which will happen if there are fewer named entities on a row) using a list comprehension
 - c) for each pair of named entity and label, create a temporary list of the format [ne_text, 0, 0, ne_label], and add to a temporary list of lists (hint : use `for i in range(1, len(temp_row_list), 2)` to skip the first entry in the row (the sentence) and increment by 2 each time so you can deal with a text, label pair each time
 - d) once you’ve added all named entities for the row, append a list consisting of [sentence, list_of_lists_of_label_data] to a list_of_named_entities (look at the comments in the trump_extract_add_to_training.py we’ve just gone through to remind yourself of the format you need your list_of_named_entities to be in (and therefore why you’re doing it like this)
- Once the data’s read in and stored in your list_of_named_entities, you can proceed as demonstrated in the example. Don’t forget to save the model after training.

Once you’ve written a working version of the program described above, you should test it out with the training sentences provided and compare the results to using the core model. Then, start adding your own training sentences and named entity examples to the ex_2_training.csv file to see if you can improve the named entity recognition still further.

Work in groups. You have 1 hour.