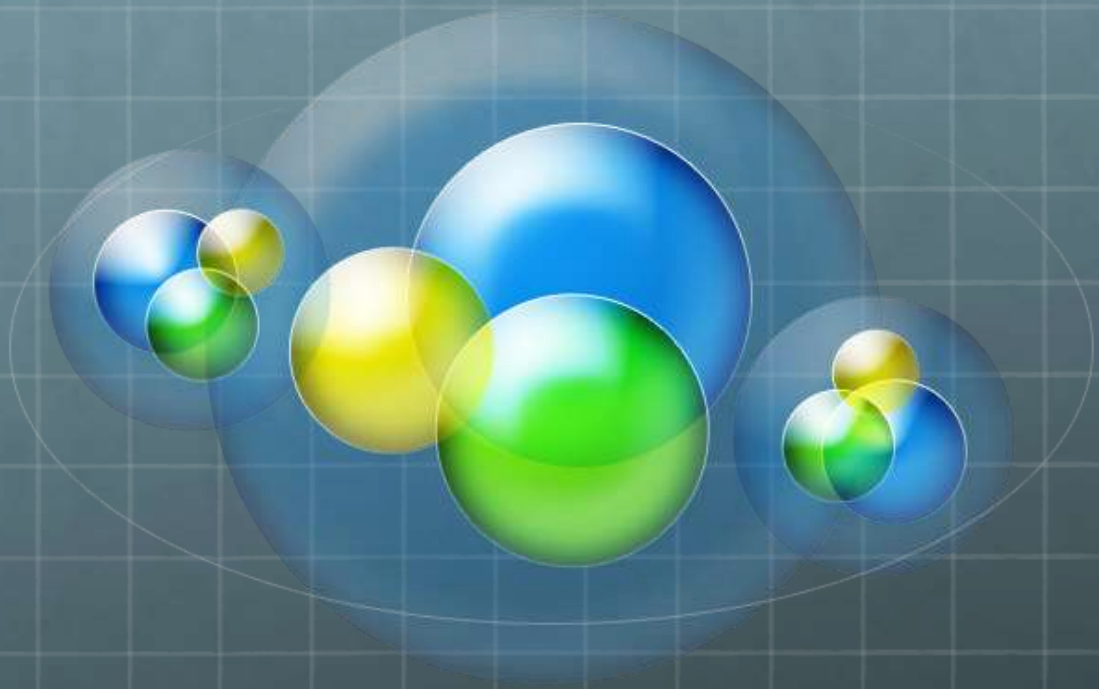


Artificial Intelligence

Dr. O. I. Adelaiye

Natural Language Processing

Lecture 4

Introduction

- Computers are great at working with structured data like spreadsheets and database tables.
- On the other hand, humans usually communicate in words, not in tables.
- That's unfortunate for computers.
- How do computers and humans then communicate

Introduction

- A lot of information in the world is unstructured
- Raw text in English or another human language.
- How can we get a computer to understand unstructured text and extract data from it?

London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium. London's ancient core, the City of London, largely retains its 1.12-square-mile (2.9 km2) medieval boundaries.

What is NLP?

- Natural Language Processing
- Sub-field of AI
- Focused on enabling computers to understand and process human languages
- We will discuss how NLP works and learn how to write programs using Python that can extract information out of raw text and unstructured data

Possible for computers to grab human-language?

- Yes, but not like humans
- Experts have been trying to write programs that understand languages like English. Why?
  - Humans have been writing things down for thousands of years and it would be really helpful if a computer could read and understand all that data.

Possible for computers to grab human-language?

- Computers can't yet truly understand English in the way that humans do
- But they can already do a lot!
- NLP already seems like magic. You might be able to save a lot of time by applying NLP techniques to your own projects.

Possible for computers to grab human-language?

- NLP are easily accessible through open source Python libraries like
  - spaCy,
  - textacy,
  - neuralcoref and more
- What you can do with just a few lines of python is amazing.

Challenge: Getting meaning from text

- Reading and Understanding English is very complex
- English doesn't follow logical and consistent rules.
- What does this mean
  - "Environmental regulators grill business owner over illegal coal fires."
  - Are the regulators questioning a business owner about burning coal illegally?
  - Or are the regulators literally cooking the business owner?
- As you can see, parsing English with a computer is going to be complicated.

Machine Learning approach

- Doing anything complicated in machine learning usually means *building a pipeline*.
- The idea is to break up your problem into very small pieces and then use machine learning to solve each smaller piece separately.
- Then by chaining together several machine learning models that feed into each other, you can do very complicated things.
- And that's exactly the strategy we are going to use for NLP. We'll break down the process of understanding English into small chunks and see how each one works.

Let's build an NLP pipeline

- Taking the text we used earlier
  - *"London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium."*
- Contains several useful facts.
- It would be great if a computer could read this text and understand that London is a city, London is located in England, London was settled by Romans and so on.
- But to get there, we have to first teach our computer the most basic concepts of written language and then move up from there.

Step 1: Segment Sentences

- Break the text apart into separate sentences.
  - “London is the capital and most populous city of England and the United Kingdom.”
  - “Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia.”
  - “It was founded by the Romans, who named it Londinium.”

Step 1: Segment Sentences

- We can assume that each sentence in English is a separate thought or idea. It will be a lot easier to write a program to understand a single sentence than to understand a whole paragraph.
- Coding a Sentence Segmentation model can be as simple as splitting apart sentences whenever you see a punctuation mark. But more complex techniques exist.

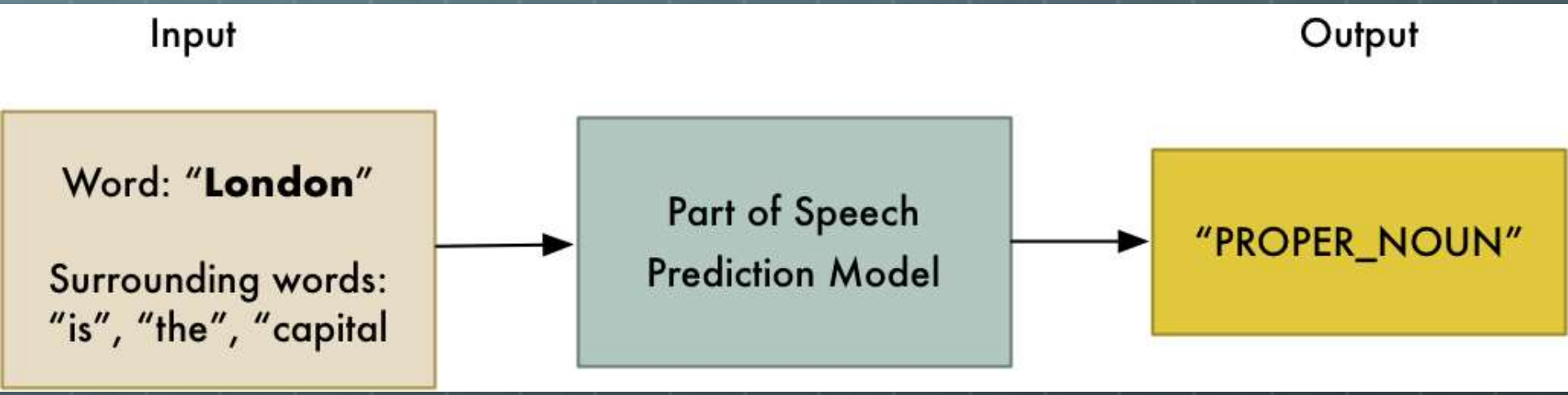
Step 2: Tokenization of words

- One at a time having split them into smaller groups
- “London is the capital and most populous city of England and the United Kingdom.”
- The next step in our pipeline is to break this sentence into separate words or *tokens*. This is called *tokenization*.
- “London”, “is”, “ the”, “capital”, “and”, “most”, “populous”, “city”, “of”, “England”, “and”, “the”, “United”, “Kingdom”, “.”
- Easy to do in English. Just split apart words whenever there’s a space between them. And we’ll also treat punctuation marks as separate tokens since punctuation also has meaning.

Step 3: Predict Part of Speech per Token

- Look at each token and try to guess its part of speech
  - Noun,
  - Verb,
  - Adjective and so on.
- Knowing the role of each word in the sentence will help us start to figure out what the sentence is talking about.
- We can do this by feeding each word (and some extra words around it for context) into a pre-trained part-of-speech classification model:

Some English



Step 3: Predict Part of Speech per Token

- The part-of-speech model was originally trained by feeding it millions of English sentences with each word's part of speech already tagged and having it learn to replicate that behavior. (Labeled dataset)
- Model is completely based on statistics
  - it doesn't actually understand what the words mean in the same way that humans do.
  - Knows how to guess a part of speech based on similar sentences and words it has seen before. (Learning)

Post Process Results

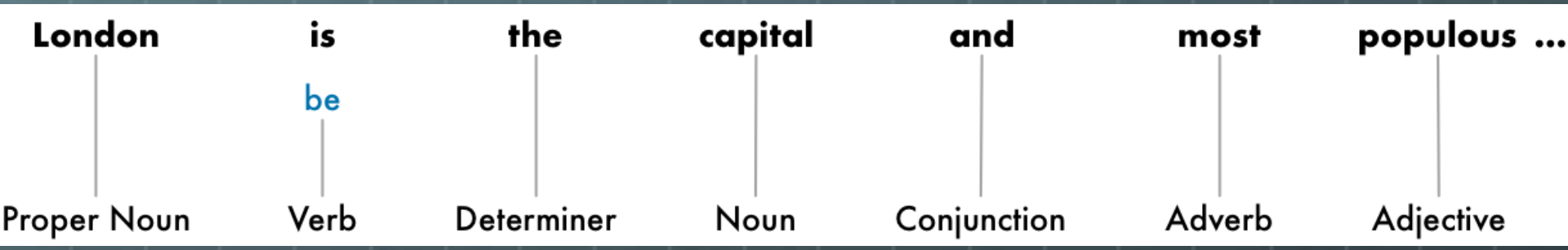
- With this information we can:
- Obtain some very basic meaning.
  - For example, we can see that the nouns in the sentence include “London” and “capital”, so the sentence is probably talking about London.

Step 4: Text Lemmatization

- In English (and most languages), words appear in different forms. Look at these two sentences:
  - I had a pony.
  - I had two ponies.
- Both sentences talk about the noun pony, but they are using different inflections (cases).
- When working with text in a computer, it is helpful to know the base form of each word so that you know that both sentences are talking about the same concept. Otherwise the strings “pony” and “ponies” look like two totally different words to a computer.

Step 4: Text Lemmatization

- In NLP, we call this process *lemmatization*
  - Figuring out the most basic form or *lemma(title, description, caption)* of each word in the sentence.
- The same thing applies to verbs. We can also lemmatize verbs by finding their root, unconjugated (unconnected) form. So “I had two ponies” becomes “I [have] two [pony].”
- Lemmatization is typically done by having a look-up table of the lemma forms of words based on their part of speech and possibly having some custom rules to handle words that you’ve never seen before.



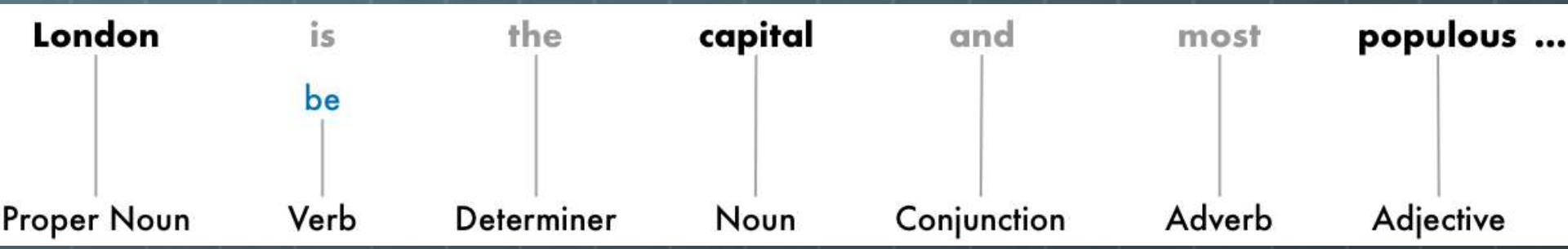
- The only change we made was turning “is” into “be”.

Dealing with Stop Words

- Next, we want to consider the importance of a each word in the sentence. English has a lot of filler words that appear very frequently like “and”, “the”, and “a”.
- When doing statistics on text, these words introduce a lot of noise since they appear way more frequently than other words.
- Some NLP pipelines will flag them as stop words
  - Words that you might want to filter out before doing any statistical analysis.

Dealing with Stop Words

- Here's how our sentence looks with the stop words grayed out

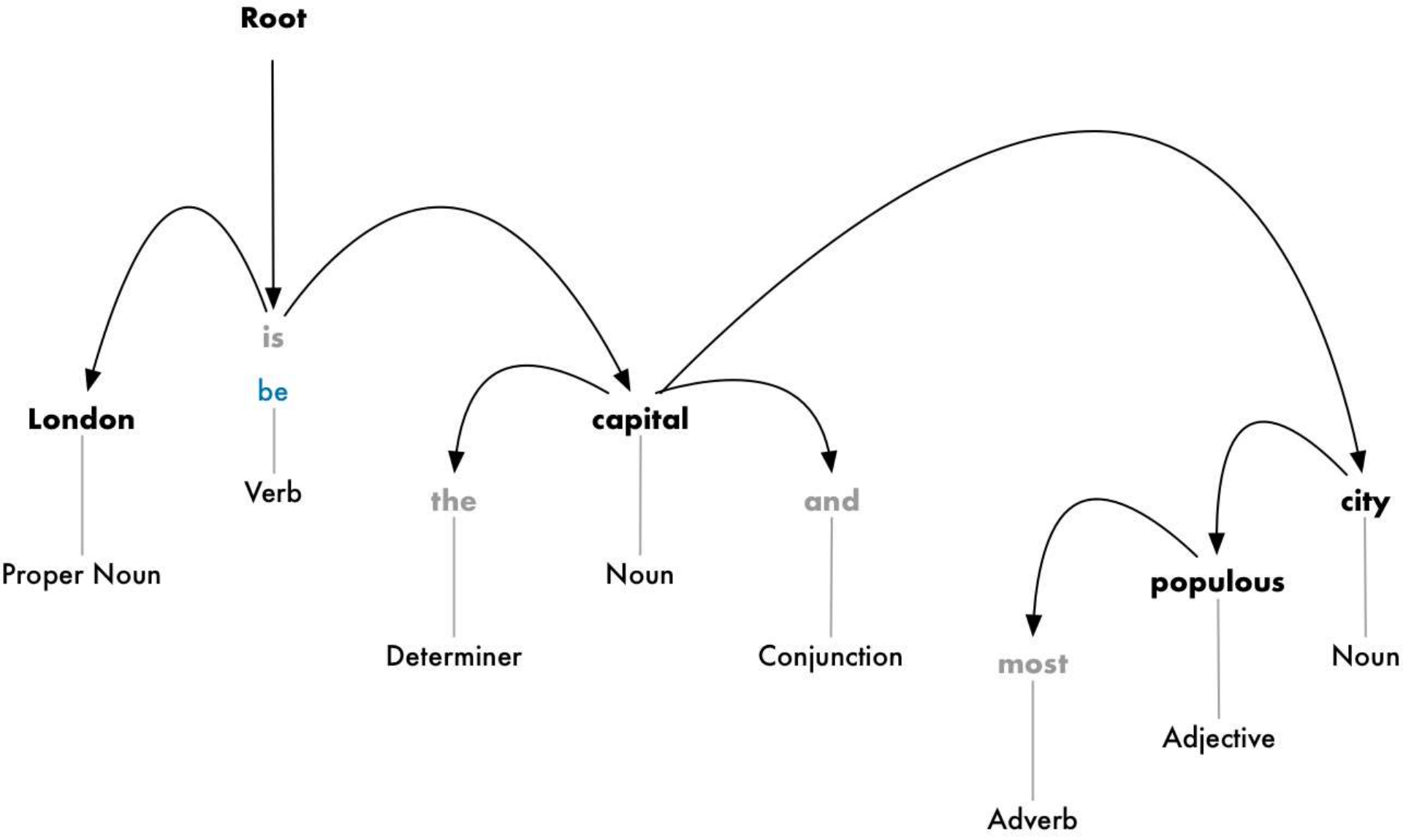


Dealing with Stop Words

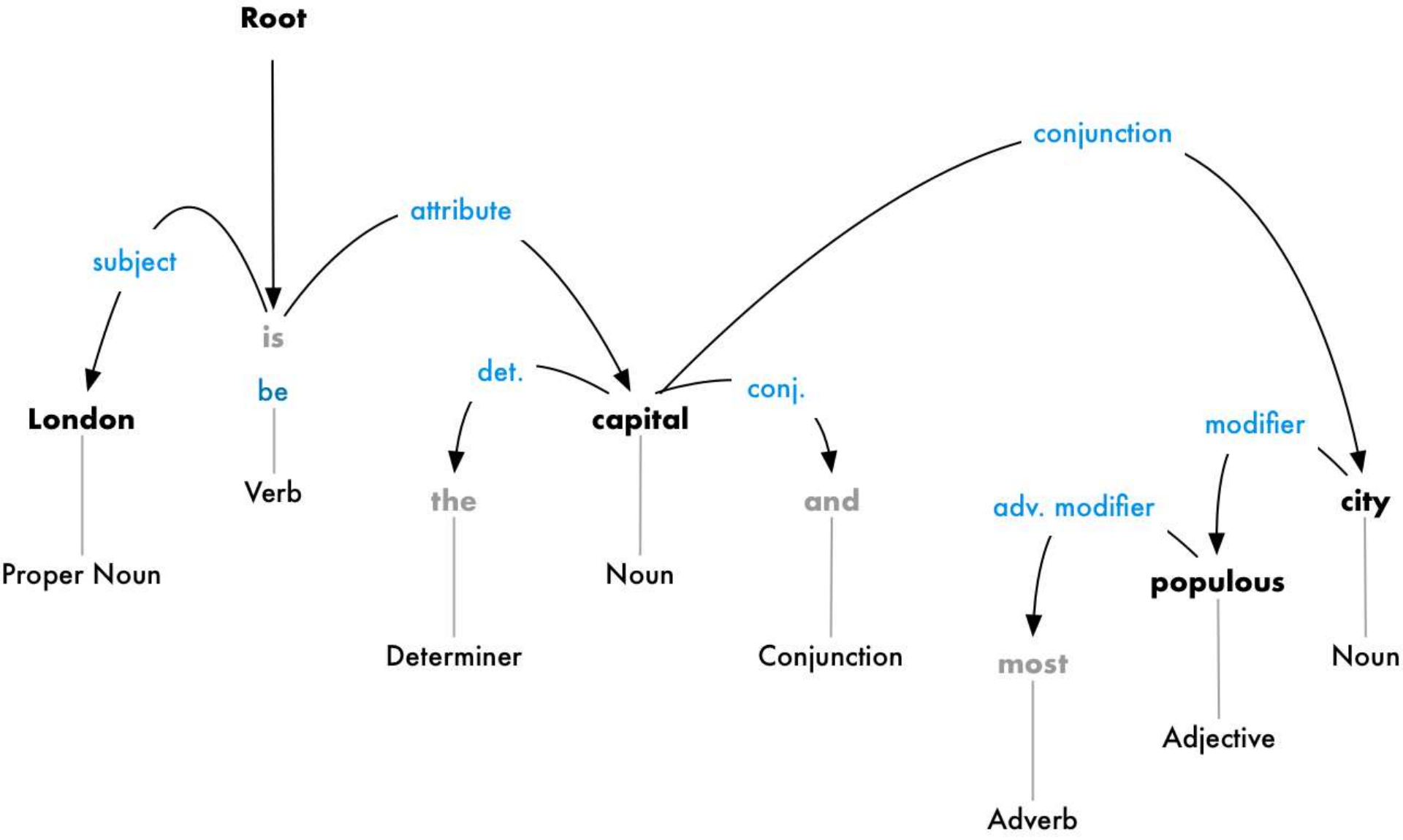
- Stop words are usually identified by just by checking a hardcoded list of known stop words.
- But there's no standard list of stop words that is appropriate for all applications. The list of words to ignore can vary depending on your application.
- For example if you are building a rock band search engine, you want to make sure you don't ignore the word "The". Because not only does the word "The" appear in a lot of band names, there's a famous 1980's rock band called *The The*!

Step 5: Dependency Parsing

- The next step is to figure out how all the words in our sentence relate to each other. This is called *dependency parsing*.
- The goal is to build a tree that assigns a single parent word to each word in the sentence.
- The root of the tree will be the main verb in the sentence.


Root
is
be
Verb
London
Proper Noun
the
Determiner
capital
Noun
and
Conjunction
most
Adverb
populous
Adjective
city
Noun

- But we can go one step further.
- In addition to identifying the parent word of each word, we can also predict the type of relationship that exists between those two words

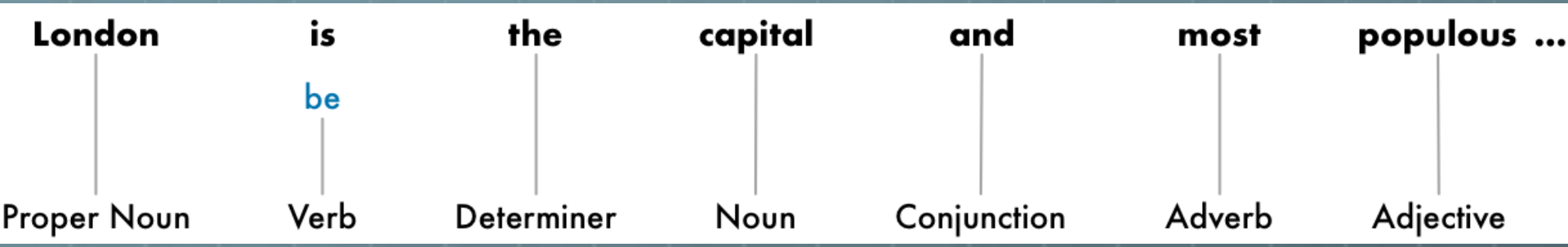

Root
subject
attribute
conjunction
is
be
Verb
London
Proper Noun
det.
the
Determiner
capital
Noun
conj.
and
Conjunction
modifier
adv. modifier
most
Adverb
populous
Adjective
city
Noun

- This parse tree shows us that the subject of the sentence is the noun "*London*" and it has a "*be*" relationship with "*capital*".
- We finally know something useful
  - *London* is a *capital*! And if we followed the complete parse tree for the sentence (beyond what is shown), we would even found out that London is the capital of the *United Kingdom*.
- Just like how we predicted parts of speech earlier using a machine learning model, dependency parsing also works by feeding words into a machine learning model and outputting a result.
- But parsing word dependencies is particularly a complex task and would require an entire article to explain in any detail.

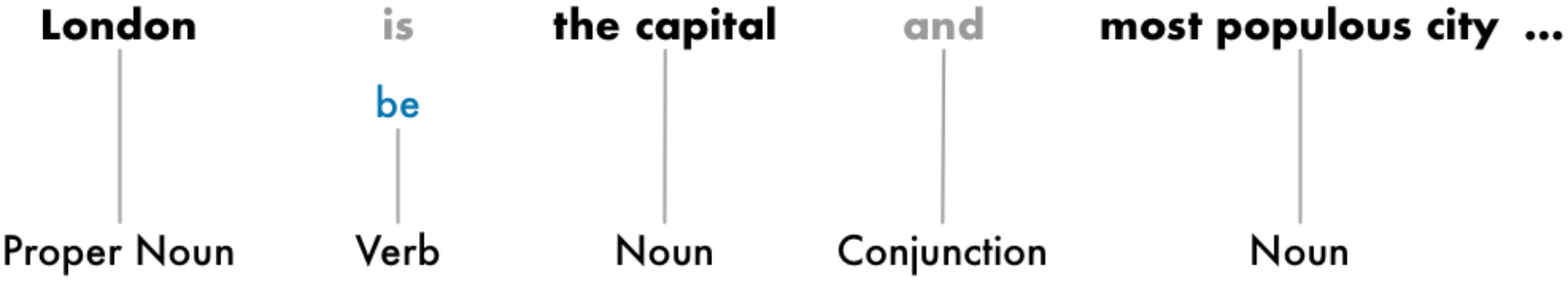
- Parsing techniques are still an active area of research and constantly changing and improving.
- It's also important to remember that many English sentences are ambiguous and just really hard to parse.
- In those cases, the model will make a guess based on what parsed version of the sentence seems most likely but it's not perfect and sometimes the model will be embarrassingly wrong. But over time our NLP models will continue to get better at parsing text in a sensible way.

Just before the next step: lets have a step 6b

- So far, we've treated every word in our sentence as a separate entity.
- But sometimes it makes more sense to group together the words that represent a single idea or thing.
- We can use the information from the dependency parse tree to automatically group together words that are all talking about the same thing.



- We can group the noun phrases to generate this:



- Whether or not we do this step depends on our end goal. But it's often a quick and easy way to simplify the sentence if we don't need extra detail about which words are adjectives and instead care more about extracting complete ideas.

Step 7: Named Entity Recognition

- Now that we've done all that hard work, we can finally start extracting ideas.
- In our sentence, we have the following nouns:

- Some of these nouns present real things in the world.
  - For example, "*London*", "*England*" and "*United Kingdom*" represent physical places on a map. It would be nice to be able to detect that!
  - With that information, we could automatically extract a list of real-world places mentioned in a document using NLP.

Step 7: Named Entity Recognition

- The goal of *Named Entity Recognition* (*NER*) is to detect and label these nouns with the real-world concepts that they represent.
- Here's what our sentence looks like after running each token through our NER tagging model:

Step 7: Named Entity Recognition

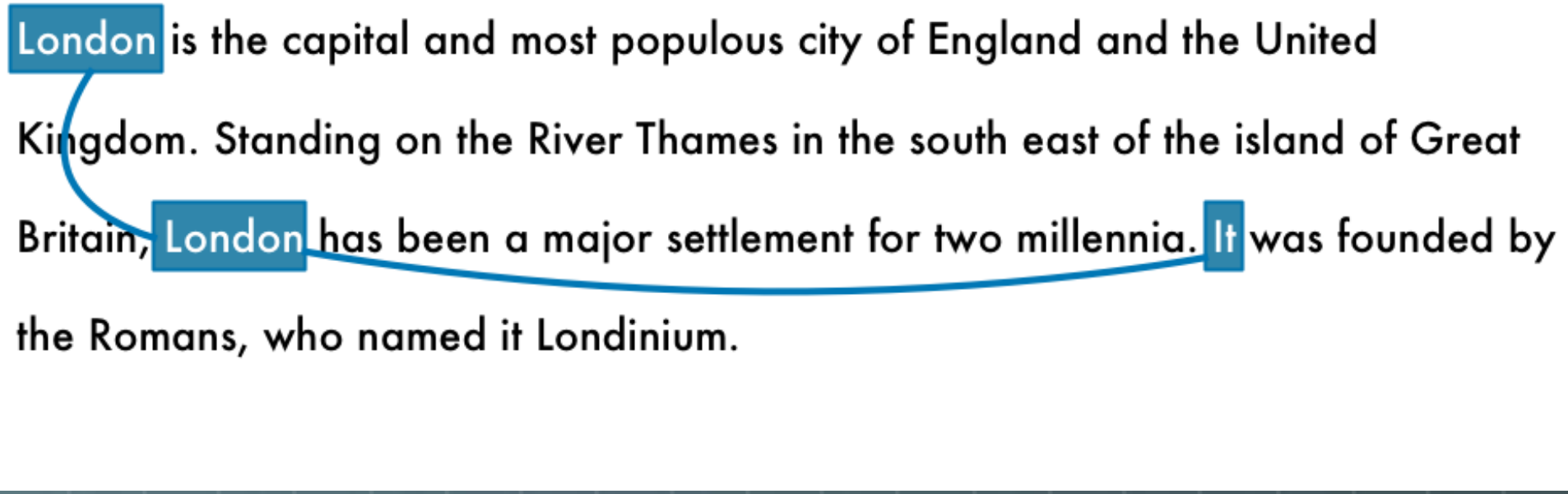
- NER systems aren't just doing a simple dictionary lookup.
- NER uses the context of how a word appears in the sentence and a statistical model to guess which type of noun a word represents. A good NER system can tell the difference between "*Brooklyn Decker*" the person and the place "*Brooklyn*" or Turkey(place) and Turkey(animal) using context clues.
- Here are just some of the kinds of objects that a typical NER system can tag:
  - People's names
  - Company names
  - Geographic locations (Both physical and political)
  - Product names
  - Dates and times
  - Amounts of money
  - Names of events
- NER has tons of uses since it makes it so easy to grab structured data out of text. It's one of the easiest ways to quickly get value out of an NLP pipeline.

Step 8: Coreference Resolution

- At this point, we already have a useful representation of our sentence. We know the parts of speech for each word, how the words relate to each other and which words are talking about named entities.
- One big problem left
- English is full of pronouns
  - words like *he*, *she*, and *it*
  - These are shortcuts that we use instead of writing out names over and over in each sentence.
  - Humans can keep track of what these words represent based on context.
  - NLP model doesn't know what pronouns mean because it only examines one sentence at a time.

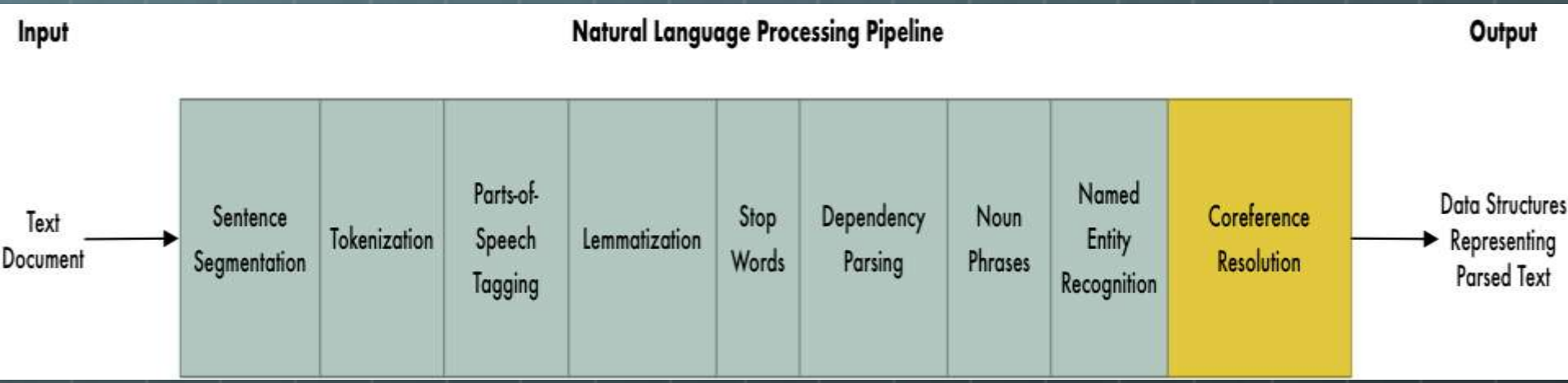
Step 8: Coreference Resolution

- Let's look at the third sentence in our document:
- "It was founded by the Romans, who named it Londinium."
- If we parse this with our NLP pipeline, we'll know that "it" was founded by Romans. But it's a lot more useful to know that "London" was founded by Romans.
- As a human reading this sentence, you can easily figure out that "*it*" means "*London*". The goal of coreference resolution is to figure out this same mapping by tracking pronouns across sentences. We want to figure out all the words that are referring to the same entity.
- Results in next page



- With coreference information combined with the parse tree and named entity information, we should be able to extract a lot of information out of this document
- Coreference resolution is one of the most difficult steps in our pipeline to implement. It's even more difficult than sentence parsing. Recent advances in deep learning have resulted in new approaches that are more accurate, but it isn't perfect yet.

Converting The NLP Pipeline to codes Using Python


Input
Natural Language Processing Pipeline
Output
Text Document
Sentence Segmentation
Tokenization
Parts-of-Speech Tagging
Lemmatization
Stop Words
Dependency Parsing
Noun Phrases
Named Entity Recognition
Coreference Resolution
Data Structures Representing Parsed Text

NPL Pipeline in Python

- *Note: Before we continue, it's worth mentioning that these are the steps in a typical NLP pipeline,*
- *but you will skip steps or re-order steps depending on what you want to do and how your NLP library is implemented.*
- *For example, some libraries like spaCy do sentence segmentation much later in the pipeline using the results of the dependency parse.*
- So how do we code this pipeline?
  - Thanks to amazing python libraries like spaCy, it's already done! The steps are all coded and ready for you to use.

NPL Pipeline in Python

```
# Install spaCy pip3

install -U spacy

# Download the large English model for spaCy

python3 -m spacy download en_core_web_lg

# Install textacy which will also be useful

pip3 install -U textacy
```

Lets code

```
import spacy
# Load the large English NLP model
nlp = spacy.load('en_core_web_lg')
# The text we want to examine
text = """London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium. """
# Parse the text with spaCy. This runs the entire pipeline.
doc = nlp(text)
# 'doc' now contains a parsed version of text. We can use it to do anything we want!
# For example, this will print out all the named entities that were detected:
for entity in doc.ents:
        print(f"{entity.text} ({entity.label_})")
```

Output

London (GPE)
England (GPE)
the United Kingdom (GPE)
the River Thames (FAC)
Great Britain (GPE)
London (GPE)
two millennia (DATE)
Romans (NORP)
Londinium (PERSON)

- Notice that it makes a mistake on “Londinium” and thinks it is the name of a person instead of a place.
- This is probably because there was nothing in the training data set similar to that and it made a best guess.

More SpaCy

- Let's take the idea of detecting entities and twist it around to build a data scrubber
- Let's say you've discovered that you have thousands of documents with personally identifiable information in them like people's names.
- You've been given the task of removing any and all names from your documents.
- Going through thousands of documents and trying to redact all the names by hand could take years.

```python
import spacy

# Load the large English NLP model
nlp = spacy.load('en_core_web_lg')

# Replace a token with "REDACTED" if it is a name
def replace_name_with_placeholder(token):
    if token.ent_iob != 0 and token.ent_type_ == "PERSON":
        return "[REDACTED] "
    else:
        return token.string

# Loop through all the entities in a document and check if they are names
def scrub(text):
    doc = nlp(text)
    for ent in doc.ents:
        ent.merge()
    tokens = map(replace_name_with_placeholder, doc)
    return "".join(tokens)

s = """
In 1950, Alan Turing published his famous article "Computing Machinery and Intelligence". In 195
Syntactic Structures revolutionized Linguistics with 'universal grammar', a rule based system of
"""
```

Output

In 1950, [REDACTED] published his famous article "Computing Machinery and Intelligence". In 1957, [REDACTED]
Syntactic Structures revolutionized Linguistics with 'universal grammar', a rule based system of syntactic structures.

Fact Extraction

- spaCy is a really powerful and amazing library.
- But you can also use the parsed output from spaCy as the input to more complex data extraction algorithms.
- There's a python library called textacy that implements several common data extraction algorithms on top of spaCy.
- One of the algorithms it implements is called Semi-structured Statement Extraction. We can use it to search the parse tree for simple statements where the subject is "London" and the verb is a form of "be"
- That should help us find facts about London.

```python
import spacy
import textacy.extract

# Load the large English NLP model
nlp = spacy.load('en_core_web_lg')

# The text we want to examine
text = """London is the capital and most populous city of England and  the United Kingdom.
Standing on the River Thames in the south east of the island of Great Britain,
London has been a major settlement  for two millennia.  It was founded by the Romans,
who named it Londinium.
"""

# Parse the document with spaCy
doc = nlp(text)

# Extract semi-structured statements
statements = textacy.extract.semistructured_statements(doc, "London")

# Print the results
print("Here are the things I know about London:")

for statement in statements:
    subject, verb, fact = statement
    print(f" - {fact}")
```

Output

Here are the things I know about London:
- the capital and most populous city of England and the United Kingdom.
- a major settlement for two millennia.

- Not too impressive.
- Try it with larger documents instead of just three sentences, you'll get a more impressive result


WHERE? WHEN? WHY? WHO? WHERE? WHAT? HOW? WHAT? HOW? WHY? WHEN? WHY? WHO? WHAT? WHERE? WHEN? HOW? HOW? WHEN? WHERE? WHERE? WHICH? WHOSE? WHEN? WHY? WHOSE? WHERE? WHICH? WHOSE? WHEN? WHY? WHICH? WHOSE? WHEN? WHY? WHOSE? WHEN? WHY? WHAT? HOW? WHY? WHO? WHERE? WHAT? HOW? WHERE? WHICH? WHOSE? WHEN? WHY? WHY? HOW? WHAT? HOW? WHY? WHO? WHERE? WHAT? HOW? WHERE? WHICH? WHOSE? WHEN? WHY? WHY? HOW? WHERE? WHAT? HOW? WHO? WHOSE? WHERE? WHAT? HOW? WHERE? WHICH? WHOSE? WHEN? WHY? WHO? WHERE? WHAT? HOW? WHERE? WHICH? WHOSE? WHEN? WHY? HOW? WHERE? WHY? WHAT? HOW? WHERE? HOW? WHAT? HOW? WHO? WHERE? WHAT? HOW? WHAT? HOW? WHY? WHERE? WHAT? HOW? HOW? WHERE? HOW?