# **Introduction and learning outcomes**

Welcome to this next topic

in which we are going to be talking about

information extraction. To get us started, let's review our

learning objectives for the course and our

topic learning objectives. From a course point of view, there are three objectives that are particularly

relevant to this topic. We're going to be

looking at appropriate statistical language

analysis techniques for a particular

problem which of course is information

extraction. We're going to be

using software tools such as taggers and NLP libraries for building information

extraction pipelines. We're going to do with

thinking a little bit about evaluations, that's objective number

5 there at the bottom. Our topic learning

objectives this time around are to understand

the definition and scope of information extraction. So I'll be looking at some of

the use cases and tasks and the way in which it delivers

value in a business context. I will be applying entity

recognition techniques. So we'll look at a

variety of those. We'll also be creating some practical information

extraction applications. The way this topic is

structured is that in the first half we

think of it as top-down. We'll think about building an overall information

extraction pipeline. In the second half, we'll drill down a little

bit more detail at the individual components and techniques involved in that. So that's it. Let's get started.

# **Introduction to information extraction**

In this segment, we're

going to introduce the topic of

information extraction, and we're going to do it

through a pair of videos. In this first one,

we'll talk more about the tasks involved, and the contexts, and the use cases for

information extraction. In the second video that

accompanies this one, we'll look in detail at some

of the algorithms involved. What is information extraction? Well, essentially,

as we mentioned right at the start

of the course, natural language data

has structure in it, but the structure is implicit. As opposed to, for example, a relational database, which information is organized

into rows and columns, where the relations and the

information is explicit. In some ways, information

extraction is about making that implicit structure

visible or explicit. What that means essentially

in practical terms, is that we can build

applications to extract semantic

content from the text. Typically, that could

be factual information, so the who, what, when, where that you might

see in a new story, so those things would be targets for

information extraction. As I mentioned, really

with the process, you can think of it

as a transformation. You're taking unstructured

information in inputs and you're

trying to output structured information

in some sort. You could think of it also as another transformation from a data modeling

point of view, where you have essentially

strings in the input and you're trying to get to database entries

in the output. Where do we see it? What

examples do we have? Well, we've all had

that experience where we receive an email or a text message and the

phone number's underlined, or the address is underlined, or the date is underlined. That in a sense is a basic form of information

extraction because the algorithms that are doing that markup are

essentially looking for patterns that

represent dates, times, phone numbers, URLs,

and that kind of thing. That offers value because

you can click through from a date to a calendar

entry or for a phone number to the phone

application and so on. That's simplistic, rather formulaic

information extraction, but nonetheless, it's

mainstream and it's out there. Another example, of course,

is business information. There are many news

reports published daily, and to save the time of people reading those,

and absorbing them, and comprehending them we can build systems that facilitate that task and extract the key information for us so that we don't need to do that. Then, of course, there's

a huge amount of scholarly literature

that is published every day on all topics in

environments in life sciences. There's a significant

industry around trying to monitor, manage, and curate that

information in a way that delivers value by

finding funny relations. No, not just in extracting

factual information, but also finding interesting and perhaps unexpected

relations in that scientific literature

that then lead to discoveries and

scientific advances. There's all sorts

of applications for information extraction. In the next couple of slides, we're going to look at

some of the subtasks involved in information

extraction. They're listed here really, and there's five of them. I think of it as the what, the how, and the when. The first part, probably

the most important part is to recognize what we

call named entities, which you can think of those

proper nouns in the text, the things that are

being talked about. Now, things can mean lots

of different things. Typically, it means people and places, perhaps companies, and so on in news reports, but essentially

they're the objects, if you like, of that discussion. We need to identify where they are in the text and

what they refer to. For example, canonical

database entry to resolve them down to some unique

identifier of some sort. That's what named entity

recognition is about, is trying to find the what, what is being talked about. Then there's

relation extraction. So when we've identified

what is being talked about, how do those entities

relate to each other? They're not just all

mentioned as a flat list, there's usually

some narrative that conveys some sense of a

relation between them, and that's what we're

trying to identify with relation extraction. For example, person A is

employed by organization B. Organization B is based in location C, that kind of thing. You can look for employed

by relationships or part-whole

relationships and lots of other different

types of relation. Then there's event extraction. This is trying to identify the specific events or episodes

in which those entities participate so that

you can have a sense of perhaps possibly

a timeline or how these different relations

connect across a time period. Then there's event coreference, which is if you have an

instance of an event that event may be referred

to through use of a pronoun or some other

indirect mention. In-text all the time, we don't always constantly

repeat the entity, we sometimes refer

back to it using other linguistic devices

and the task of event co-reference resolves

those mentions down to a unique event. Then there's temporal event

expression extraction, which is the when trying to find when the above events

happened and there's lots of different

ways that we can convey dates and

times and duration and it's the task of temporal expression

extractions to identify those and

resolve those. Here's an example.

Here's a little fragment of a new story. You can see here, I won't read the whole thing, but you can see there's various entities that are mentioned. United Airlines is

an organization, Friday is time, $6 is some money. American Airlines

is an organization and so undergo

various locations. We might look at

this piece of text, extract the entities from it, and then try and find the

relations, for example, American Airlines is a

unit of AMR cooperation. Tim Wagner works for American Airlines and this

event took place on Thursday, and so on and so forth. It's important to

notice that there isn't one unique way of expressing

these relations and the relationship we're

looking for tend to be chosen on a

case-by-case basis, but one example of what

you might call a taxonomy of relations is from the

ACE evaluation exercise, which is a community-based

evaluation process from a decade or two ago. They conceived this taxonomy of relations where there's essentially six

top-level categories, person's social and the relation beneath that with

business lasting personal, family and physical locations

or general affiliations, part-whole relations, organization affiliation

relations, and artifacts. Seems these are skewed

towards Newswire, the relations that

you might want to extract for a

typical news report. But they're not the

only ones of course. As I said, the relations you're

looking for will often be decided on what is the problem

we're trying to solve. There is not necessarily a universal system that

you would want to adopt, so you choose one according to the task that you're

trying to solve or the problem will remain. In medicine, for example, there is something called UMLS, which is the Unified

Medical Language System. See here relations in

the middle column, but they're quite different

to the ones that we had earlier on the

previous slide. You can see the

entities are different. The entities talk about

typically medical concepts, injuries, bodily

locations, and so on. This is very different

to what we just saw. You got disrupts location

of parts and so on. That just shows you that the system of relations you might want to be interested in

can vary significantly. Then we've got other

systems of relations. DBpedia, you can think of it as the database

version of Wikipedia. Wikipedia as you probably

know has textual content, but it also has things

called infoboxes, which is a little

structured section, usually at the top of

the page that defines some structured information

about that entry. Several attempts have been made to systematize

this knowledge and derive ontologies directly from these Wikipedia infoboxes. That gives us another system of relations defined by DBpedia. Then, of course,

there's WordNet, which we met earlier

in the course. That has systems of relations that are predicated on the is-a, the hyponym relationship

between classes, and also the instance

of which relates particular entities to

being instances of classes. That's another relation.

Then, of course, there are various

bespoke systems of relations that

you might want to create specifically for a task or specifically for a problem. There it is, that's some

information extraction from a problem-centric point of view and in the next video, we'll look at some of

the algorithms involved.

# **Relation extraction**

In this segment, we going to pick up where

we left off in the previous video and start to talk in detail about some

relation extraction algorithms. So there's essentially four main classes

of relation extraction algorithms, handwritten patterns,

supervised machine learning, semi-supervised machine learning,

and unsupervised learning. So let's look at the four

of those in turn. So first up,

we've got a handwritten patterns. And these go back a few decades

actually to I think the early 90s. An intuition behind this approach is

that if you take a sentence like this, agar is a substance prepared for

a mixture of red algae, such as Gelidium, for

laboratory or industrial use. So you may not know what Gelidium is and

even Gelidium and you may not know what red algae is,

or even how to pronounce it. But you actually in some ways you don't

need to because you can just work out or you can infer from the sentence that

Gelidium is a type of red algae. So in other words, we can infer a hyponym relation

from this lexico-syntactic pattern. So we've got something

such as a noun phrase, and then the words such as, and

then another noun phrase followed optionally by further noun phrases,

and the words and or. And then a final noun phrase. Where there's one or more of those, it implies this semantics word

which essentially is written at the bottom here for all NPi,

where i is greater than or equal to 1. There is a hyponym relation

between those NPs and NP0. So this technique generalizes

to other patterns seeing see here we've got

the one that we just saw. And then you've got other ways

to articulate hyponym relations. So temples, treasuries, and other

important civic buildings again, expresses the column, the notion that temples and

treasure is a hyponyms of civic buildings. And then he got such authors as Herrick,

Goldsmith, and Shakespeare. And then the other two examples, you can

see even though they're all articulated in different ways, they're all

expressing a hyponym relationship. So they're all they can all be

harvested essentially as handwritten patterns to trawl the web and

find or trawl any corpus and find examples of hyponym relations and

use that to populate a database. Now we can go a little

bit further than this. We can add other constraints rather than

just looking at any noun phrase we might add an entity type. So we might say that

position of is the pattern. The relation we're looking for. And it's, its arguments are essentially

a person and an organization. So we might find a fragment

of text George Marshall, Secretary of State of the United States,

which matches this expression. So we've got a person on the left. We've got position, and

then we've got the word of and then we've got the organization. So that adds another constraint, extending

the idea to specific entity types. So as I mentioned, this technique

has been around a few decades. And you can build some

very high precision, domain-specific information

extractors using this technique. But recall tends to be low. It tends to miss lots of true positives. And it's also difficult to scale because

somebody has to create these handwritten patterns. So what alternatives are there? Well, that takes us on to our second

approach, which is supervised learning. So like a lot of things

in machine learning, essentially you start off by choosing

entities and relations of interest. Then you annotate a training corpus and

then you can simply train classifiers on that training corpus and the classifier

could be logistic regression, random forests are only of your choosing. And then once you've

trained your classifier, you can run it on your test data or your

corpus to find pairs of named entities. And then you apply that

classifier to each pair. And you may as a as an extra step,

also pre-filter. You'll test data with a kind of binary

classifier that just identifies whether there's any relation at all

existing in the text and then you can focus your original classifier on

identifying the type of relation. And what kind of word, what kind of

features might we be interested in for doing this training? Well, as word features such as what

is the headword of the sentence or create a bag-of-words, which is something we'll talk about

a little bit more in the next topic. Which is essentially just an unstructured

and old set of those terms. And then we could take bigrams as well. So those are all word features. And then you got named entity features. So that you've got the entity type person,

place or organization and also the entity level,

whether it's mentioned directly or whether it's a nominal or

pronomial reference. And then also we've got

syntactic structure. We might want to extract that and

use that as a feature. So that might be the constituent parts

like what we talked about in topic 6. And also possibly dependency paths

if you're using a dependency paths. And you can also use neural

approaches in supervised learning. As you may have realized BERT is

incredibly flexible and scalable. So you could use a pre-trained

encoder such as BERT and then add a linear layer on top of that. Which you would fine tune as a one event classifier to identify

the correct relation. And similarly to the previous technique, we would apply the classifier to

pairs of the entities in the input. And again, we can optimize it by replacing

the entities by their appropriate entity tags to avoid overfitting to

the actual underlying strings. So like a lot of things in machine

learning, you can get very high accuracy. Or supervised machine learning,

you can get very high accuracy and you can make it domain-specific which is

both of suppose a strength and a weakness. But it's expensive creating this

training data takes time and effort. And also they can be a little

bit brittle if the test data or corpus you run it on

doesn't does not match. It's not a good match

with the training data. So that leads us on to the third

techniques and there's two variation, two variations of the third technique. So this is semi-supervised

relation extraction. There's what we call bootstrapping, which is where you start from

a high-precision set of seed patterns. And then you find sentences that

contain those seed patterns. And then you generalize the context

around it to generate new patterns. And that's best really

illustrated with an example. So an example if we wanted to

identify airline and hub pairs, knowing only that Ryanair

has a base at Stansted, what we can do is find mentions

of these terms in our corpus. So we might find the sentence Ryanair,

which uses Stansted as a base, canceled all flights. So that matches our seed pattern

that generates the pattern that you can see here organization,

which uses location as a base. So that then gives us a pattern

which we can use to generate new patterns to generalize this context and

we can find other examples. And therefore,

expand outward from that seed pattern. Now as you've probably noticed,

this is an iterative process. And of course,

it can go a little bit astray, which is what we call semantic drift. We might start picking up

expressions such as p and o ferries, which uses dover as a base, for

example, it matches the pattern, but it's clearly a false positive because

it's not about airline and hub pairs. So what tends to happen in practice

is that confidence values for the matches are determined

based on number of heads. And then those confidence values

are used to limit the the propagation of new generation of new patterns. And that avoids the semantic drift. So that's one type of

semi-supervised relation extraction. The other is what's called

distance supervision. So it's a similar idea we

start from seed patterns. But in this case,

we start from a large database of them and then rather than doing iteratively,

we create lots of noisy patterns in parallel and then we

combine them into a supervised classifier. So for example, if we wanted to

identify place-of-birth relationships, using instances from DBpedia, which we

talked about in the previous video. We might find that Albert Einstein,

his place of birth was Ulm, Edwin Hubble,

place of birth was Marshfield, and so on. What we can then do is run a named

entity tagger on corpus and find sentences with matching tuples. So we might, for example,

find the fragment Einstein, born brackets 1879, Ulm, that matches. Or we might find Hubble's

birthplace in Marshfield matches. So we might find a whole cross section

have lots of different contexts for this, which we call noisy patterns. And then we can use all those as training

data for either of the supervised learning approaches we talked

about earlier the feature based, classical feature based machine

learning or neural approaches. So what this process tends to reduce is

lots of rich training data because we're generating all these context noisy,

but multiple contexts in parallel. So we're using the large numbers in order

to balance out a little bit of the noise because we have many, many training

instances which we can generalize from. So the advantages, this techniques

creates lots of features simultaneously. So rather than the iterative

process of bootstrapping, so it avoids the semantic drift problem. It's supervised by a high-precision,

hand-crafted knowledge, which we can take from variety

of sources such as DBpedia. And ironically, we can actually use

this technique to infer the handcrafted patterns, which we talked

about right at the start. So we can in some ways,

provide vindication I guess, that the original hand patterns

that we thought were valuable, actually are valuable because with this

process can produce them automatically. The disadvantage though is it

can be relatively low precision. And of course, we need an existing

database to start from. So that leads us on to the final

technique, which we call unsupervised relation extraction, where we've

got no label training data at all. And this is sometimes referred to

as open information extraction. And in this context, what we're really

doing is looking for relations. Not so much in a database centric or

schema centric manner, but just finding relations

between strings of words usually focused around a verb in

the sentence or in the corpus. So the way it works is we run out parts

of each tagger & entity chunker over each sentence in our corpus. And then we find the longest phrase. It starts with a verb. And I'm simplifying a little bit here,

but to give you the essence of it. And then we find the nearest noun phrases

to the left and to the right of that verb. And then we assign

a confidence to the relation. And we can do that using logistic

regression trained on our hand annotated sample of say, 1000 sentences that

have been extracted, and so on. And then what we do is we simply say

that relations are stored if they generalize reasonably well I,

or if they occur, enough times, and times where n is typically

a number like 20 or so on. And then we are if the if so

we add them to our database. So obviously, this is a very open technique we can find

all sorts of different relations here. And it's much more schema agnostic than

the other techniques that we talked about. So an example might be Ryanair

has a hub in Stansted, which is the headquarters of UXLabs. So we might extract from that relation 1,

Ryanair, has a hub in, Stansted. So we've got our noun phrase to the left,

our noun phrase to the right. And the longest phrasing that is headed

with the verb, which is has a hub in. And same thing with the other part of

the sentence we've got a noun phrase to the left Stansted a noun phrase to

the right UXLabs and the longest phrase, which is headed by a verb,

is the headquarters of. And of course, in this instance you see

these two relations have been correctly extracted using this simple algorithm

from that fragment of text. So the strength is very flexible,

broad coverage. You can extract all sorts of

relations using this technique, and run it also all sorts of text. But of course, the output is just strings. So it needs to be mapped to some canonical

form prior to database entry unless you just want to keep the strings. But really the whole point of information

extraction is to structure them in some way. And most work on unsupervised relation

extraction tends to focus on verbs. So there's real obviously

relationship could be expressed using nominal constructions or

other types of linguistic phenomena. So finally, before we segue into

a couple of more pragmatic segments, let's have a look how you

might put all this together. So almost regardless of which

technique you use if you want to build an information extraction pipeline, and we'll see some practical

examples of this in a moment. Broadly speaking, the architecture

would look like this on the right, you'll need to take the rotex

splitted into sentences. And then we'll split the sentences

into tokens, all the things that we talked about in textbooks processing,

earlier in the course. We'll tag each token with part of speech

tags which you talked about a little bit, but we'll drill down a little bit

more of that in the next video. And then we identify entities of interest. And we might do a bit of chunking. And again, we'll talk about that

in the second half of this topic. And then finally, we find the relations

between entities and extract those. And of course, the people the entities can

be people, places and organizations or to temporary or numerical expression whatever

is appropriate for the task in hand. So there it is. That's an overview for

relation extraction algorithms. And we've got a quick look at what

an information extraction pipeline might look like from an architectural

point of view. And in the next couple of videos, we'll look in practical terms about

how to start implementing this stuff.

# **Relation extraction practice quiz**

### Question 1

How many named entities would you expect to find in the sentence “Matthew Hickman worked for Lehman Brothers in New York”?

* 0
* 9
* 1
* 3

### Question 2

In a typical information extraction pipeline, in what order would you apply the following processes:

* POS tagging, tokenization, entity recognition
* POS tagging, entity recognition, tokenization
* Tokenization, POS tagging, entity recognition
* Tokenization, entity recognition, POS tagging

### Question 3

Which of the following are common approaches to relation extraction?

* Handwritten patterns
* Supervised machine learning
* Remote learning
* Bootstrapping

### Question 4

Which of the following are strengths of the supervised learning approach?

* Focuses mainly on relations expressed using verbs
* Cheap to build
* Flexible
* High accuracy

# **POS tagging**

In the previous video, we looked at

information extraction from a conceptual point of view, and we saw how you could

build pipeline starting with elementary tags

processing operations such as sentence segmentation and tokenization and building on that to add

part-of-speech tagging, named entity recognition

and relation extraction and using

that pipeline, we could build useful

NLP applications. So in this video, we're going to focus on the first stage, or rather the middle

part of that pipeline, the first stage of the

high-level processing, known as part of speech tagging. Here we are in our

Jupyter notebook and we're going to do a little bit of practical

part of speech tagging, which basically means tagging words with lexical category. That's based on a specific tag set and we'll talk a

little bit about tag sets but briefly they can be

quite small in size, you could have perhaps

just a dozen tag sets, but typically there

will be of the order of 40 or 50 lexical categories and we'll see an example of this as we go through the lesson. To get us started, we'll import NLTK, and then we'll create

a sample sentence, which we've taken here

from the Reuters corpus. Now what we're going to do is, we're going to tokenize

that sentence like we saw in Topic 5. We could look at the

tokenized sentence and it's quite long, but you can see it's split

into the various terms. Now what we're going to do is, we're going to download a part-of-speech

tagger from NLTK. I'm going to apply

that part-of-speech tagger to our

tokenized sentence. Here we have the same

sentences we saw a moment ago, but now you can see in

addition to each of the terms, we've also got the lexical

category over here on the right in the form

of a part of speech tag. Here, for example, with

NNP means proper noun, VBZ is the third person

of part of a verb. VBN looks like the past tense, VBG is what's called the

gerund form, and so on. We've got down the list here. We've got all the various

different parts of speech tags for the sentence that we previously tokenized. Now, the interesting

thing about tagging is that as we saw in

the previous video, language can be ambiguous and syntactic categories

can be ambiguous too. If we take a sentence like this, "They refuse to permit us to

obtain the refuse permit." This is ambiguous because

it contains the words refuse and refuse and

permit and permit. Now, a native English

speaker will know that when you have the

verb form of the word, this word here that

I'm highlighting, refuse, that the emphasis is on the second syllable

when it's a verb. But when it's a

noun, it's refuse. So the emphasis is on

the first syllable. It's the same with this word, to permit, in the verb form puts the emphasis

on the second syllable. But in the noun form, a permit, the emphasis is

on the first syllable. But the point is that we can't

hear how this is spoken. Just purely in the written

form, it's ambiguous. Let's see what happens when we attach parts of

speech tags to this. Indeed, what we find is that refuse comes out

as a verb, permit, rather permit, comes out as a verb and then

refuse, permit, the two noun forms as in "obtain the refuse permit"

are both tagged as nouns. So you can see that

that's a good result. It's an ambiguous sentence, but the part-of-speech

tagger has correctly identified the lexical

categories of these terms, even though they are identical in terms of the way

they're spelled. Now, I mentioned that there are different types of tag sets and they can vary in size

and complexity and detail. What we'll do is, we'll download the ones that come with NLTK and we'll just have a look at one which is the default one, which is the upenn\_tagset and if we call the

help function on that, you can see the tag set. You can see the tags

that we had before, we got PRP, VBP, TO, VB. You can see, for example, that DT was a determiner, I think we had one of those. PRP looks like a

personal pronoun, VBN is the past

participle of a verb, VBP is the present tense

of a verb, and so on. You can see that you've got all the different

lexical categories there, which are applied in the upenn\_tagset by

default using an NLTK. That's it for part of speech tagging, a brief introduction. In the next video, we'll

see how we can use that part-of-speech tagging to do more sophisticated

transformations, in particular, identifying name entities and

extracting relations.

# **POS tagging practice quiz**

### Question 1

What do POS tags represent?

* Tokens
* Lexical categories
* Parse trees
* Named entities

### Question 2

Why might a sentence like ‘They refuse to permit us to obtain the refuse permit’ be difficult to tag accurately?

* The words ‘refuse’ and ‘permit’ appear twice
* The words ‘refuse’ and ‘permit’ can have more than one stem
* The words ‘refuse’ and ‘permit’ can have more than one lexical category
* The words ‘refuse’ and ‘permit’ can have more than one parse tree

# **Information extraction using nltk**

In the previous

video, we looked at part of speech tagging. We left it at the

point where we said this could be useful for

subsequent NLP tasks. In this video, we're

going to pick up where we left off and show how we can use tagged sentences to perform more sophisticated

NLP operations. We're going to

look in particular at information extraction. Now, if you recall a

couple of videos ago, we talked about how information

extraction in principle, could be a very

simple task if it was applied to structured data. As an example here we have

in our Jupyter Notebook a list built up of relations

between different strings. For example here Matthew Hickman works for Lehman Brothers. Lehman Brothers is in New York. We can perform simple

look-up operations on this simple structure. We could ask who is

based in New York? We could run a little list

comprehension looking for the relation with the second

entity being New York, or we could ask who

works for Lehman? Doing a similar

list comprehension, but this time looking for

the relation works for, and the second entity

being Lehman Brothers. Sure enough, we got

the correct answers. The correct answer to

the first question, what is based in New York? Is Lehman Brothers.

Who works for Lehman? Is Matthew Hickman. I will extend this

idea a little bit further on applying

the same technique, but this time to data that

is in its unstructured form. To illustrate the example, let's set up a few

little bits and pieces that we're going

to use further on. Here's the sentence

that we saw earlier. The sentence from

the Reuters corpus. There we have it. We can see just in its basic string form. What we might want to

do at this point is pre-package or package up all our pre-processing operations

into a single function. We're going to define

ie-preprocess, which takes a document, does sentence tokenization and there's the word tokenization, and then there's the part

of speech tagging and returns the tagged sentences. For example, we could call that function on our

little sentence, and then we could print

a pre-process sentence. You can see as expected, we've got all the words in that sentence plus all

the parts of speech tags, and it's been tokenized

and segmented. Now what we might want

to do at this point is some named

entity recognition. We mentioned in the

video that there's lots of different entity

types and it really does depend on your particular

application which ones you might be interested

in but typically, most named entity applications

will look for things like organizations,

people's names, locations, maybe

dates and times, or money or currency

or percentage. Also things like facilities

or geopolitical entities. They're examples of these

things but as I said, it does vary according

to the task. Now, how does this

work in practice? Well in NLTK, what we might want to do is

start with the default maxent named entity chunker which

comes by default with NLTK. Then add all the other bits

and pieces that it needs. Now what we can do is we can apply that named

entity chunker to the pre-processed

sentence that we created earlier to create

some name entity results. If we open that up and

have a look at that, we can see it's

what's called a tree structure but the

interesting thing is that inside that

you can see you got a geopolitical entity of Mexico, you've got an organization

of Lehman Brothers, you've got a person that

is Matthew Hickman, and you've got another

geopolitical entity that is New York. At this stage, we might

want to pretty print that, and you can see

that as expected, we've got the person, the organization, and both

geopolitical entities. We can also draw

it on the screen, which would open up a separate

window allowing you to see how that is

constructed graphically. Finally, what we might

want to do is show how we can do that same task

of relation extraction, but this time on the

unstructured form of that data. The way we do this is we can

call another method within NLTK called extract roles and we give it the types of entity

that we're looking for. For example, if we wanted to

know who works for Lehman, we would say a person,

or organization, give me the object that

we want to search, which is any results

that we created earlier. A corpus type, which basically means that the patterns to look for or rather how that

corpus is constructed. Also, the pattern which is a regular expression that defines what you're looking for. Now if we're looking

for the word of, that'll be just a simple

regular expression like this. But there are more complex

regular expressions that might look for

variations on the word, in, but we're just

going to look for of, as in person A, of

organization B. If you run that bit

of code, sure enough, it looks in our entity results

which we created earlier, and it finds that Matthew Hickman works

for Lehman Brothers. We managed to do the

relation extraction or rather information extraction that we talked about earlier, but this time applied to the unstructured

form of that data. Now we could scale this up. We can do it on a more

industrial scale. Let's look at a slightly

more sophisticated example. In this one, we're looking

for the construction that in connection between objects. We might say, give us

all the instances where an organization is in a specific location and

to look in a corpus, which comes with NLTK

called the IEER corpus, which is already marked

up with named entities. What this little fragment

of code does is it goes through all the

documents within that corpus, it finds all the

relationships of this type running

exactly the same line of code as we saw up here, but this time iterating overall as documents

in that corpus. Then it prints out all

the tuples that it finds for organizations

and locations. If we run that now, you can see it iterates over that corpus and it

finds examples. Freedom Forum is

based in Arlington, Open Text is based in Waterloo, the Bastille Opera

is based in Paris, and so on. There we have it. That's a complete NLP

pipeline all the way from sentence segmentation up

to relation extraction. Very simple, very relatively

easy to do in NLTK. There's all the pieces

that you need to build your own NLP pipeline.

# **Information extraction using nltk practice quiz**

### Question 1

What of the following entity types would you expect to find in the sentence “Fred worked for Microsoft in London last year”?

* PERSON
* ORGANISATION
* MONEY
* TIME

### Question 2

Which of the following corpora is marked up with a variety of named entities?

* Gutenberg
* IEER
* Brown
* Reuters

# **Understanding information extraction discussion prompt**

Consider the following questions, then post your comments in the forum: What are the two main subtasks of named entity recognition (NER)? What is a gazetteer, and how might it be useful in NER? What is the main shortcoming of using a gazetteer-based approach to NER? What approach might you adopt to deal with ambiguous entities such as ‘June’ and ‘West’? What approach might you adopt to deal with nested entities such as ‘Professor Stuart Hall Building’ and ‘Liverpool John Lennon Airport’? Once you’ve posted your comments in the forum, take a look at those of other learners and comment on the differences.

# **Chunking**

In this segment,

we're going to have a little look at how to do some practical entity

recognition using NLTK. We're going to look a little

bit more detail at one of the key sub-tasks in

information extraction, which is a process

called chunking. Which is the identification of meaningful chunks within

sentences in our data. Meaningful chunks can mean

lots of different things, but typically and for the

purposes of this example, think of chunks as

being noun phrases. To get us started, we'll import NLTK and we'll

create a little sentence, it's just been parts-of-speech

tagged already. The little yellow dog

barked at the cat, you can see the speech tag is

already attached to those. Then we can define a pattern. Now, this is where

it starts to get interesting because

we're essentially using a regular expression to define the pattern

for noun phrases. We're saying that noun phrase, unless we adopt this syntax

here with the curly braces, consists of an

optional determiner, followed by zero or

more adjectives, followed by a noun. So we can define our

grammar as that. Then we can create what's

called a chunk parser called by calling nltk.RegexpParser

and giving it the grammar. Then calling that parser

to parser a sentence, the sentence that

we defined earlier. What we can do, we can

run that and sure enough, we get a sentence

at the top level. It's identified the noun phrase

based on this pattern of a determiner followed by zero or more adjectives

followed by a noun. Then we've got some stuff

in the middle that doesn't match and then we've got

another round phrase, which is a determiner, followed by a noun. We can actually call it a

draw method and you can just, you probably can't

see that actually, but it spawns another little

window with a tree in it. If you run that locally, you'll be able to

see that window. But not all sentences

are that simple. Let's take another one here. Any new policy measures? What we'd hope to

do here is identify new policy measures

as the noun phrase, so we could call that

with our parser, but you can see it's to

identify the noun phrases, any new policy, and it's

dropped the word measures. We could take another

example, earliest stages, that's clearly a noun phrase, but nothing is detected because our grammar

is incomplete. So we have to modify

the grammar to accommodate more complex

examples of noun phrases. We might say, for example, a noun phrase, unlike

what we had earlier, in this instance is an

optional determiner followed by zero or

more adjectives, which can be of different types. You've got JJR, its

comparative adjective, and then followed by

one or more nouns, of which can be of

different type here. So we've got noun plural, NNS. If we modify our grammar to

accommodate those structures, we can call the parser,

and sure enough, earlier stages is now

recognized as a noun phrase. If we switched this sentence

to the previous one, that is now correctly

or recognize any policy measures

as a noun phrase. What we can do now, is we can start to think about this process of

chunking and looking in a little bit more detail about defining chunks using

multiple regular expressions. You can see here we've

got an example where the grammar extends

over two lines. Of course, you could extend

this over many lines. In the practical, you got an example to try this

out for yourself. If we call our RegexpParser

on this sentence, Rapunzel let down her

long golden hair, and we print the parser

of the sentence. We can see we've got the noun Rapunzel

recognized correctly, but her long golden

hair has been split, the word hair has dropped off, and it's just come out

'long golden hair'. So we might want to

go back and define a slightly more

extensive grammar. Here, determiner or possessive, which is optional, followed by zero or more adjectives

followed by a noun. Also, we can chunk sequences

of proper nouns as well. If we apply that grammar, and then apply that

to our sentence, you can see now, her

long golden hair, the possessive has been included in the noun phrase because we added

it to our pattern. Now you may be thinking, well like a lot of things in

regular expressions, what happens if we get

more than one match? Here, for example, a grammar

where noun phrase is a noun followed by a noun,

two consecutive nouns. What happens if we

get something like money market fund where

we've actually got three. What happens is, it will take the

first match it finds. So it's essentially

greedy matching. You may have heard that term used in context of

regular expressions. It matches on the first pair

of nouns that it finds. Finally, let's look at

something called chinking, which is the opposite

of chunking. So these are the bits

in between chunks. The way that works, is

it's defined here, again, we've got a similar

construction where we've got regular expressions

over multiple lines, but the chinks are, well the curly braces are flip

the other way round. So, in this little grammar we're saying a noun phrase is,

we chunk everything, but then we, instead of trying

to find the things we do, we remove the things

we don't want. So we chink the sequences of the past tense

and the preposition, so represented here by this

second line in the grammar. We can apply that to

the little yellow dog barked at the cat, and pass the sentence. Sure enough, it does as we expected and as

we saw earlier, it finds the first noun phrase, it finds the second noun phrase. But this time it's doing

a negative matching by chunking everything and then chinking the bits in

between that we don't want, and that's an

alternative approach. Finally, you may have heard of IOB tags or BIO tags

as sometimes called, use commonly in natural

language processing to identify sequences or rather

labeled sequences and B stands for beginning, I stands for inside and

O stands for outside. They're commonly used

as markup in chunking, but also in named entity

recognition in a lot of tasks. If we have a fragment tags

like we saw the yellow dog, might start with the

part of speech tags, but look also at the B

and the O and the I. So we've got a beginning

of a noun phrase. Then we got the word

saw, which is outside. Then we've got the yellow dogs. So we've got beginning

of a noun phrase, inside a noun phrase, and inside the noun phrase. That is how we mark up in text where chunks begin and end and also have a

middle part as well. That has also very flexible

notation which allows us to represent more than

one chunk type as long as they do not overlap. That's some chunking with NLTK, practical example and you can see for yourself

how you can build some quite complex

grammars to do some interesting

information extraction using these techniques.in

# **Extracting noun chunks with spaCy**

So earlier on we looked at

doing information extraction, in particular noun phrase

chunking using NLTK. In this video we're going to have a brief

look at a different API in particular, we're going to look at

something called spacy. Now you may have heard of spacy it's an

open source library that allows you to do production quality,

natural language processing. So we're going to exercise a little bit

of its API here doing a bit of noun phrase chunking. So to get us started, let's import spacy. We'll define a little helper

function to read a text file. We're going to read one of my blog posts,

which you can see here, it's just the text. Just as an example. Basically, you could take your

own news story if you wanted and we're going to load a model Into spacy. Now spacy comes ship for

the variety of models. Here's one that's for English triangle,

the web and SM main small, there's also a medium and a larger sizes as well,

but in the interests of simplicity. I'm just loading the smallest model here. So then what we do as we saw early on,

we briefly looked at spaces we can call the model on our text and

that produces something called a doc, which has, which is the the output

of that which has got all sorts of an LP annotation and

structured information inside it. So let's see if we can get some of

that structured information out. >> So we will simply iterate through

the noun chunks in the noun chunks member of the doc objects and

print out the noun chunk text. So you can see we've found

an undergraduate module, natural language processing and various other bits the field many

years recent times of transformation. So you can see it's a reasonable job

of pulling out noun phrases from my blog post. So we could try a slightly harder

sentence with some bracketing in it and some various other constructions. Again, apply the same process. And we can see we've got a reasonable set,

an undergraduate module and we've got natural language processing

being on level being pulled out. So just to walk through a few other

aspects of the API, what we can do is we also identify when now a chunk

starts and end, so it can do that hand. That's what these indices represents. The I starts at zero and ends at one, work starts at a position four and

is five characters long. And we've also got

an undergraduate module, on natural language processing as well. So, we've got various other elements of

the API though with the start in the end. Well, we can also show the sentences

associated with each start point. So, noun chunk.sent seeing see

that's wrapped brown there, but it's pulled out the various bits of the

sentence, or we could show the root text. So the root l is l, the root work is work. The root of one of the graduate

module is module, and root natural language

processing is processing. On final thing we can look

at is a similarity metric. So if we create another sentence

computation linguistics, and we perform our analysis on that,

what we can do is work out the similarity between our original

text and our second document text. So it rates through that and you can see

here, it's decided that computational linguistics is most similar to natural

language processing with a score 0.37, as opposed to 0.31 or

the other scores we have here. And notice the little warning,

we use the smallest model and it points out to us that actually

we would have got possibly a better result if we used a larger

model which should got some word vectors IE dense embeddings

as part of the model. So you can feel free to try that for

yourself, load a bigger model perhaps to medium

sized one and you'll see that almost certainly you'll get a more accurate

result as a result using that model. So there it is a quick look at spacy for

extracting noun chunks. Very useful, very flexible API. So feel free to have

an experiment with that.

# **NER with spaCy**

So, in this segment we're going to

take another quick look at spacey and we're going to have a look at how we might

apply it to the task of named entity recognition. So, here we are in our Jupyter Notebook

to get us started will impose spacey and as before, we will load the small

models for English trained on the web. And we'll define an article

which is a simple news article. You can see it's got some interesting

entities in it, though it looks like it's talking about the business and tech around

the iPhone launch of the iPhone 12. And as always with spacey

we can call the model on the article which gives

us a marked up document. And then we can iterate through

the entities in the document entities and we can print the NC text the start

character, the character and the label. Notice this will go off

the edge of the screen but you can see some interesting stuff here. It's got the digits 12 identified as

a cardinal starting position seven ending at nine. Apple is an organization. Lots of cardinals and ordinals for

the numbers US is a geopolitical entity. Apple is an organization. Got dates, more cardinals. And you can see here,

quite a lot of interesting markup, and reasonably accurate. Tim Cook person, Dan Ives person,

Wedbush Securities an organization. iPhones is an organization that's wrong,

but most of the time it's doing pretty

well many years a person is wrong, but you can see a lot of these others,

recognized correctly. So, you can feel free to

experiment with the API and try it with your own content as well. Just as a final note,

if you're interested in what is the set of entities from which

this sample is drawn. You can find that again in

the spacey documentation. But briefly, they look like this. You've got named entities

divided into two categories. You've got the actual names or

named entities. So, you got people of nationalities. Political groups, facilities,

organizations, geopolitical entities, other locations, products, events,

work of arts, law and languages. And then the things that aren't

sort of named entities, but are still often recognized as

part of the named entity task. And it's the other things that we talked

about earlier the dates, and times and percentages, and money and

other numerical quantities. So there it is. That's a quick look at doing

named entity recognition using an alternative API and

in particular, using spacey.

# **Information extraction practice quiz**

### Question 1

What is a “chink”?

* A failure in the pattern matching process
* A pattern that we want to exclude from a chunk
* A chunk consisting of one term

### Question 2

What happens if a tag pattern matches in multiple places?

* The rightmost match takes precedence
* The leftmost match takes precedence
* All matching sequences are represented

# **Experimenting with noun phrase chunking lab**

In this exercise you will extend the tag patterns shown in the video.

1. Write a tag pattern to match noun phrases containing plural head nouns, e.g. "many/JJ students/NNS", "three/CD days/NNS", "both/DT old/JJ positions/NNS". Try to do this by generalizing the tag pattern that handled singular noun phrases
2. Write a tag pattern to cover noun phrases that contain gerunds, e.g. "the/DT receiving/VBG end/NN", "assistant/NN managing/VBG director/NN".
3. Write one or more tag patterns to handle coordinated noun phrases, e.g. “April/NNP and/CC May/NNP", "all/DT your/PRP$ students/NNS and/CC researchers/NNS", "company/NN officers/NNS and/CC directors/NNS"

# **Understanding noun phrase chunking discussion prompt**

In the previous exercise you extended the tag patterns shown in the video to accommodate more complex constructions. Write a summary for each of these bullet points and post it in the discussion forum. What kinds of pattern did you add to accommodate these extensions? To what extent did adding new patterns break any previous ones? How do you think chunkers would be evaluated in practice? The IOB format categorises tagged tokens as I, O and B. Why are three tags necessary? What problem would be caused if we used I and O tags exclusively Once you’ve posted your comments in the forum, take a look at those of other learners and comment on the differences.

# **Information extraction summary**

So in the final video this topic,

we're going to briefly recap on some of the things that we have studied in

our look at the information extraction. So as you may remember,

right at the start, we said, we'd understand the definition of

scope of information extraction with suddenly looked at lots of used

as examples of how word supplied. We said we'd apply entity

recognition techniques. And we've certainly looked at

that from both a theoretical and a pragmatic viewpoint. And we said we create practical

information extraction applications and we saw how we could create

a simple pipeline for doing information extraction

task earlier on in the topic. So just to briefly recap the two

parts really we looked at relation extraction from

an algorithmic point of view, and you may recall that there were

four different types of algorithm. There was the handwritten patterns the

sort of original technique if you like, and then of course,

there's supervised machine learning. And we looked at two different types of

semi supervised with a bootstrapping and distant learning. And we also looked at unsupervised

learning or open extraction on the web. And finally we looked at

how to build pipelines. We saw how from an architectural point

of view you could think of an extract information extraction as building

a pipeline starting from raw text, which is split into sentences. And then you split the sentences,

the tokens and then you tag each token with

the part of speech tags. Then you apply chunking or entity recognition to identify

the chunks within that. And then there's all sorts of methods

exactly as we talked about a moment ago, finding the relationships

between those entities. So that's it for information extraction. Hope you find it useful. See you in the next video.