Introduction

Today, text is one of the most abundant sources of information, and an integral part of our lives. On an average day, we read about 9000 words. That includes emails, text messages, the news, blog posts, reports, tweets, and things like street names and advertisements. Over the course of a lifetime of reading, this gets us to about 200,000,000 words. It sounds impressive, and it is, but yet, this amount of information can be stored in less than half a gigabyte. We could carry all our life's worth of reading around on a USB stick. At the time of writing this, the Internet contains an estimated minimum of over 1200 TB of text, or 2.5 million lives worth of reading. A large part of that text is now it in the form social media: microblogs, tweets, Facebook statuses, Instagram posts, online reviews, LinkedIn profiles, YouTube comments, and many others. However, text is abundant even offline, in quarterly earnings reports, patent filings, questionnaire responses, written correspondence, song lyrics, poems, diaries, novels, parliamentary proceedings, meeting minutes, and thousands of other forms that can be (and are) used in social science research and data mining.

Text is such a great source of information, not just because of its scale and availability. It is (relatively) permanent, and most importantly, it encodes language, the one human property that reflects (indirectly and sometimes even directly) a wide range of socio-cultural and psychological constructs: trust, power, beliefs, fears. Words can be used to measure the attitudes of society towards those target words over time, see (Kulkarni et al., 2015; Hamilton et al., 2016; Garg et al., 2018). The age, gender, origin, and many other demographic factors of the author are encoded in the language they use (Labov, 1972; Trudgill, 2000; Pennebaker, 2011). Language has similarly been used to measure constructs such as trust (Niculae et al., 2015) and power (Prabhakaran et al., 2012).

unstructured data

However, this avalanche of great data can become overwhelming, and dealing with it can feel daunting. Text is often called "**unstructured data**", meaning it does not come neatly ordered into categories, has varying lengths, and can not readily be fed into your favorite statistical analysis tool. Really, it is a bit of a misnomer, though: text is by no means without any structure - it follows very regular structures, governed by syntactic rules. If you know about these, it becomes a lot easier to make sense of text.

There are even simpler regularities in text that we can explore to get information. From simple counts to probabilities, we can do a lot to weed out the noise and gather some insights into the data. And then there are the heavy-lifters: machine learning tools that can crunch almost limitless amounts of data and extract precisely the information you want. Well, as precise as these tools can get.

This book is aimed at the working social scientist who wants to use text in their analysis. It is therefore first and foremost a practical book, giving concrete examples. Each chapter contains Python 3.6 code using the latest available Python packages to execute the examples being discussed. This book assumes enough familiarity with Python to follow those examples. It does not serve as an introduction to programming.

Using off-the-shelf libraries, rather than coding the algorithms from first principles, strips away some of the complexity of the task, which makes it a lot more approachable. It also allows programming novices to use efficient code to try out a research idea, rather than spending days to make the code scale to the problem and debug it. At the same time, using packages hides some of the inner workings of the algorithms. However, I would argue that we do not need to implement the algorithms from scratch in order to use these tools to the full potential. It is important to know how these tools work, though, and what is going on under the hood, in order to make the right choices and to understand the possibilities and limitations. For each example, we will therefore look at both the general idea behind the algorithm, as well as the mathematical underpinnings. If you would like to dive deeper into a certain technique, or need to modify it to suit your needs, I will provide pointers to detailed tutorials and background reading.

There is a growing number of overview articles and tutorials on NLP and text analysis in several social sciences. They cover the most relevant terms, models, and use cases for the specific fields, and it makes sense to consult them to see what people have been using. Some of the more recent examples include Grimmer and Stewart (2013) for political science, Evans and Aceves (2016) for sociology, Humphreys and Wang (2017) and Hartmann et al. (2018) for marketing, and Gentzkow et al. (2017) for economics.

To dive into NLP in all its glory, the best starting point is the textbook by Jurafsky and Martin (2014), which is constantly updated (important for such a dynamic field). The book by Manning and Schütze (1999), despite its age and lack of some of the latest machine learning approaches, is still a good complement, especially on the fundamentals. In order to explore machine learning and deep learning with Python, the book by Marsland (2015) and Chollet (2017) respectively, are very practically oriented and beginner-friendly. For a very readable general overview of machine learning, see Murphy (2012).

This book has three parts: in the first part (see page 6), we will look at some of the basic properties of text and language – the levels of linguistic analysis, grammatical and semantic components, how to describe them – and discuss what to remove and what to keep for our analyses, and how to compute simple useful statistics.

In the second part (see page **??**), we will look at exploration, the discovery of latent structure in the data. We will go from simple statistics to more complex machine learning methods, such as topic models, word embeddings, and dimensionality reduction.

In the third part (see page **??**), we will look at prediction, the use of patterns and regularities in the data in order to impute missing values, classify documents, and uncovering their linguistic structure.

Part I

Background

CHAPTER I.1

What's in a Word

We have collected a large amount of companies' self-description, and want to start find out whether there are any systematic differences, what the big themes are, and how the companies typically act. The question is: How do we do all that? Does it matter that firms *buy*, *bought*, and *are buying* – or do we only want to know that there is any buying event, no matter when? Do we care about prepositions (*in*, *off*, *over*, *about*, etc.), or are they just distracting us from what we really want to know? How do we tell the program that "Facebook acquires Whatsapp" and "Whatsapp acquired by Facebook" mean the same thing, but "Whatsapp acquires Facebook" does not (even though it uses the same words)?

Before we dive into the applications, let's take a look at the basic element we are working with in text: language. In order to choose the right method for any text-related research question we have, it makes sense to think about what language is and is not, how it is structured, and how it "works". This does not replace an introduction to linguistics, but it gives us a starting point.

Language often encodes information redundantly, i.e., we say the same thing in several ways: through the meaning of the words, their position, the context, and many other cues. Words themselves are made up of different components, which are the focus of different linguistic disciplines: their meaning (**semantics**), their function in a sentence (**syntax**), the prefixes and endings (**morphology**). Not all words have all of this information. And when we work with textual data, we might not be interested in all of this information. In fact, it can be beneficial to remove some of the information we do not need. In this chapter, we will look at the basic terminology to describe text, words, and their properties. If you are interested in reading more on this, there are many excellent introductory textbooks to linguistics and its diverse subfields. One of the most entertaining ones is Fromkin et al. (2018). For a more historic overview of English, see Crystal (2003).

semantics
syntax
morphology

When we work with text, the unit we are interested in depends strongly on the problem we investigate. Traditionally, this was a report or article, but when we work with social media, it can also refer to a user's entire posting history, to a single message, or even to an individual sentence. Throughout this book, we will refer to all these units of text as **documents**. It should be clear from context what size the document has. One document represents always one observation in our data. To refer to the entire collection

documents

corpus

of documents, we use the word **corpus** (plural *corpora*). Unless specified differently, we assume a document to be a list of words.

vocabulary
type
token

The set of all the unique words in our data is called the **vocabulary** (V). Each word in this set is called a **type**. Each *occurrence* of a type in the data is called a **token**. So the sentence "A good sentence is a sentence that has good words" has 10 tokens, but only 7 types.

1. Word Descriptors

1.1. Tokens and Splitting. Imagine we are looking at reports and want to filter out short sentences, because they don't contain anything of interest. The easiest way to do so is by defining a cutoff for the number of words. However, it can be surprisingly tricky

word

to define what a **word** is. How many words are there in "She went to Berlin" and in "She went to San Luis Obispo"? The most general definition used in many languages is any string of characters delimited by white space (note that Chinese, for example, does not use white space between words). Unfortunately, not all words are surrounded by white space: words at the beginning of a line have no white space beforehand, and words at the end of a phrase or sentence might have a punctuation symbol (commas, full stops, exclamation or question marks, etc.) directly attached to them. Quotation marks and brackets complicate things even more.

tokenization
sentence splitting

Of course we can separate out these symbols by introducing extra white space. This process is called **tokenization** (because we make each word and punctuation mark a separate token). There is a separate process, called **sentence splitting**, which separates a document into sentences and works in a similar way. The problem there is to decide when a full stop is part of a word (as in titles like *Mr.* or abbreviations like *abbr.*), or the end of a sentence. Both tokenization and sentence splitting are prediction tasks. We will discuss prediction in the last part of the book, so we will not discuss these basic linguistic analysis task in detail here, but instead rely on the available Python implementations in the `spacy` library.

```
1 import spacy
2 nlp = spacy.load('en')
3
4 documents = "I've been 2 times to New York in 2011, but did not have
    the constitution for it. It DIDN'T appeal to me. I preferred Los
    Angeles."
5 tokens = [[token.text for token in sentence] for sentence in nlp(
    documents).sents]
```

CODE 1. Applying sentence sentence splitting and tokenization to a set of document.

Executing the example in Code 1 gives us the following results for `tokens`:

```
[['I', "'ve", 'been', '2', 'times', 'to', 'New', 'York', 'in', '2011'
   , ',', 'but', 'did', 'not', 'have', 'the', 'constitution', 'for',
   'it', '.'],
 ['It', "DIDN'T", 'appeal', 'to', 'me', '.'],
 ['I', 'preferred', 'Los', 'Angeles', '.']]
```

1.2. Lemmatization. Suppose we have a large corpus of articles from the business section of various newspapers. We are interested in how often and which companies acquire each other. Since we are dealing with newspaper articles, they might be referring to recent events, to future plans, and they might quote people. Words come in different forms, depending on the tense and aspect, so we might have to look for "acquire", "acquires", "acquired", and "acquiring". We could try to formalize this by just looking for the endings *–e*, *–es*, *–ed*, and *–ing*, but that only works for **regular verbs** ending in *–re*. If we are interested in **irregular verbs**, we might see forms like "go", "goes", "went", "gone", or "going". And it does not stop there: the firms we are looking for might acquire just one "subsidi*ary*", or several "subsidi*aries*", one "comp*any*" or several "comp*anies*". Clearly, we need a way to deal with this variation.

regular verbs

irregular verbs

When we look up a word in a dictionary, we usually just look for the base form (in the previous example, that would be the infinitive, "go"). This dictionary base form is called the **lemma**. All the other forms don't change the meaning of this lemma, but are often required by the context (syntax). When we work with text and are more interested in the meaning, rather than the syntax, it can be useful to replace each word with its lemma. This reduces the amount of variation in the data, and makes it easier to collect meaningful statistics: rather than getting a single count for each of "go", "goes", "went", "gone", and "going", we would record having seen the form "go" five times in our data. In many cases (such as "to glue", "to walk", etc.), lemmatization can be done by undoing certain patterns (e.g., remove "–ed" from the end of a verb), but for the exceptions (for example "to go"), we have to have a lookup table.

lemma

Luckily, **lemmatization** is already built into `spacy`, so we can make use of it. Applying lemmatization to our example sentences from above:

lemmatization

```
lemmas = [[token.lemma_ for token in nlp(sentence)] for sentence in
   nlp(documents).sents]
```

CODE 2. Applying lemmatization to a set of documents.

we get

```
[['-PRON-', 'have', 'be', '2', 'time', 'to', 'new', 'york', 'in', '
   2011', ',', 'but', 'do', 'not', 'have', 'the', 'constitution', '
   for', '-PRON-', '.'],
 ['-PRON-', "didn't", 'appeal', 'to', '-PRON-', '.'],
 ['-PRON-', 'prefer', 'los', 'angeles', '.']]
```

Note that as part of the process, all words are converted to lower case, while pronouns are all conflated as a special token `-PRON-` (see also next section, 2).

1.3. Stemming. Maybe we are searching legal texts about political decisions. We are generally interested in anything that has to do with the constitution, so the words "constitution", "constitutions", "constitutional", "constitutionality", and "constitutionalism" are all of interest, despite having different parts of speech. Lemmatization will not help us here, so we need another way to group them across word classes.

stemming

stem

suffix

An even more radical way to reduce variation is **stemming**. Rather than reducing a word to the lemma, we strip away all parts except for the irreducible core meaning part (the **stem**). The most famous and commonly used stemming tool is based on the algorithm developed by Porter (1980). For each language, it defines a number of **suffix**es (i.e., word endings), and the order in which they should be removed or replaced. By repeatedly applying these actions, we reduce all words to their stem. In our example, all words derive from the stem "constitut–", by attaching different endings.

Again, a version of the Porter stemmer is already available in Python, in the `nltk` library (Loper and Bird, 2002), but we have to specify the language:

```
from nltk import SnowballStemmer
stemmer = SnowballStemmer('english')
stems = [[stemmer.stem(token) for token in sentence] for sentence in
    tokens]
```

CODE 3. Applying stemming to a set of documents.

This gives us

```
[['i', 've', 'been', '2', 'time', 'to', 'new', 'york', 'in', '2011',
    ',', 'but', 'did', 'not', 'have', 'the', 'constitut', 'for', 'it',
    '.'],
 ['it', "didn't", 'appeal', 'to', 'me', '.'],
 ['i', 'prefer', 'los', 'angel', '.']]
```

While this is extremely effective in reducing variation in the data, it can make the results harder to interpret. One way to address this is to keep track of all the original words that share a stem, and how many times each of them occurred. We can then replace the stem with the most common derived form (in our example, this would most likely be "constitution"). Note that this can conflate word classes, say nouns and verbs, so it needs to be decided depending on the context.

1.4. n-Grams. Looking at words individually can be a good start, but often, we want to look also at the direct context: in our example, we have two concepts that span two words, "New York" and "Los Angeles", and we do not capture them by looking at each word in turn.

Instead of looking at each word in turn, we can use a sliding window of n words to examine the text. This window is called an n**-gram**, where n can have any size. Individual words are also called **unigrams**, whereas combinations of two or three words are called **bigrams** and **trigrams**. For larger n, we typically simple write the number, e.g. "7-grams".

n**-gram**
unigrams
bigrams
trigrams

We can extract n-grams with a function from `nltk`:

```
from nltk import ngrams
bigrams = [gram for gram in ngrams(tokens[0], 2)]
```

CODE 4. Extracting bigrams from a sentence.

This gives us

```
[('I', "'ve"),
 ("'ve", 'been'),
 ('been', '2'),
 ('2', 'times'),
 ('times', 'to'),
 ('to', 'New'),
 ('New', 'York'),
 ('York', 'in'),
 ('in', '2011'),
 ('2011', ','),
 (',', 'but'),
 ('but', 'did'),
 ('did', 'not'),
 ('not', 'have'),
 ('have', 'the'),
 ('the', 'constitution'),
 ('constitution', 'for'),
 ('for', 'it'),
 ('it', '.')]
```

We will see later how we can combine bigram expressions that are part of the same "word" (see chapter **??**).

2. Parts of Speech

Let's say we have collected a large corpus of restaurant menus, including both fine dining and fast-food joints. We want to know how each of them describe the food. Is it simply "grilled", or is it "juicy", "fresh", or even "artisanal"? All of these food descriptors are **adjectives**, and by extracting them and correlating them with the type of restaurant who use them, we can learn something about the restaurants' self-representation –

adjectives

and about the correlation with price.[1]

parts of speech
content words
open-class words

At a very high level, words denote things, actions, and qualities in the world. These categories correspond to **parts of speech**, e.g., nouns, verbs, and adjectives (most languages have more categories than just these three). They are jointly called **content words** or **open-class words**, because we can add new words to each of these categories (for example new nouns, like "Tweet", or verbs like "twerking"). There are other words, including determiners, prepositions, and some others (see below). They don't "mean" anything (i.e., they are not referring to a concept), but help to structure a sentence and to make it grammatical, and so are usually referred to jointly as **function words** or **closed-class words** (it is very unlikely that anybody comes up with a new preposition any time soon). Partially because function words are so short and ubiquitous, they are often overlooked. And while we have a rough idea how many times we have seen (or used) the noun "class" in the last few sentences, it is very hard to consciously notice how often we have seen or used, say, "in'.[2]

function words
closed-class words

Because languages differ in the way they structure sentence, there was little agreement about the number and applicability of these grammatical categories, beyond the big three of content words (and even those are not always sure). However, recently, there have been efforts to come up with a small set of 15 categories that can be applied to a wide range of languages (Petrov et al., 2011). It is called the Universal Part-of-Speech tag set (see `https://universaldependencies.org/u/pos/`). In this book, we will use this set of parts of speech.

Open-class words:

- ADJ: adjectives. They modify nouns to specify their properties. Examples: *awesome*, *red*, *boring*
- ADV: adverbs. They modify verbs, but also serve as question markers. Examples: *quietly*, *where*, *never*
- INTJ: interjections. Exclamations of some sort. Examples: *ouch*, *shhh*, *oi*
- NOUN: nouns. Entities in the world. Examples: *book*, *war*, *shark*
- PROPN: proper nouns. Names of entities, a subclass of nouns. Examples: *Rosa*, *Twitter*, *CNN*
- VERB: full verbs. Events in the world. Examples: *codes*, *submitted*, *succeed*

Closed-class words:

[1] This example is based on the book "The Language of Food: A Linguist Reads the Menu", by Dan Jurafsky. He did exactly this analysis, and found that the kinds of adjectives used are a good indicator of the price the restaurant charges (though maybe not necessarily of the quality).

[2] The book "The Secret Life of Pronouns: What Our Words Say About Us" by James Pennebaker examines precisely this quality of function words, which makes them particularly interesting for psychological analysis and forensic linguistics. Function words can be used to identify kidnappers, predict depression, or assess the stability of a marriage.

- ADP: adpositions. Prepositions or postpositions, markers of time, place, beneficiary, etc. Examples: *over*, *before*, *(get) down*
- AUX: auxiliary and modal verbs. Used to change time or modality. Examples: *have (been)*, *could (do)*, *will (change)*
- CCONJ: coordinating conjunctions. Link together parts of sentences with equal importance. Examples: *and*, *or*, *but*
- DET: determiners. Articles and quantifiers. Examples: *a*, *they*, *which*
- NUM: numbers. Exactly what you would think it is...
- PART: particles. Possessives and grammatical markers. Examples: *'s*
- PRON: pronouns. Substitutions for nouns. Examples: *you*, *her*, *his*, *myself*
- SCONJ: subordinating conjunctions. Link together parts of sentences with one part being more important. Examples: *since*, *if*, *that*

Other

- PUNCT: punctuation marks. Examples: *!*, *?*, –
- SYM: symbols. Word-like entities, often special characters, including emojis. Examples: *%*, *$*, *:)*
- X: other. Anything that does not fit into any of the above. Examples: *pfffffrt*

We will see later (see Structured Prediction, **??**) how we can automatically determine the parts of speech and syntax of a sentence. For now, we can again use the **POS-tagger** in `spacy`: **POS-tagger**

```
pos = [[token.pos_ for token in sentence] for sentence in nlp(
    documents).sents]
```

CODE 5. POS-tagging a set of documents.

This gives us

```
[['PRON', 'VERB', 'VERB', 'NUM', 'NOUN', 'ADP', 'PROPN', 'PROPN', '
    ADP', 'NUM', 'PUNCT', 'CCONJ', 'VERB', 'ADV', 'VERB', 'DET', 'NOUN
    ', 'ADP', 'PRON', 'PUNCT'],
 ['PRON', 'PUNCT', 'VERB', 'ADP', 'PRON', 'PUNCT'],
 ['PRON', 'VERB', 'PROPN', 'PROPN', 'PUNCT']]
```

POS tagging was one of the first successful NLP applications, and a lot of research has gone into it. By now, POS taggers are more accurate, more consistent, and definitely much faster than even the best-trained linguists.

3. Stopwords

If we want to assess the general topics in product reviews, do we care whether a review refers to "the price" or "a price"? If we are looking at the position of parties in their manifestos, we know who they are, so do we need their names in the document? In both cases, it can be beneficial to remove these words, as they occur often, but do not contribute much to our task. The set of these ignorable words is called **stopwords**. **stopwords**

As we have seen, many words in a text belong to the set of function words. In fact, most of the most-frequent words in any language are function words. The ten most common words in English (according to the Oxford English Corpus), with their parts of speech: *the* (DET), *be* (VERB/AUX), *to* (ADP/PART), *of* (ADP), *and* (CCONJ), *a* (DET), *in* (ADP), *that* (CCONJ), *have* (VERB/AUX), *I* (PRON). While these are all very useful words when building a sentence, they do not really mean much on their own, i.e., without context. In fact, for most applications (e.g., topic models, see Boyd-Graber et al. (2014)), it is better to ignore them altogether.[3]

We can either exclude stopwords based on their part of speech (see above), or by using a list. The former is more general, but risks throwing out some unintended candidates (if we exclude prepositions, we might losing the noun "round" if it gets incorrectly labeled). The latter is more task-specific (we can supply a list of political party names), but risks leaving words we were not aware of when compiling the list. Often times, it can therefore be beneficial to use a combination of both.

For our running example, if we exclude common stopwords, and filter out non-content words (all parts of speech except NOUN, VERB, PROPN, ADJ, and ADV):

```
content = [[token.text for token in sentence
        if token.pos_ in {'NOUN', 'VERB', 'PROPN', 'ADJ', 'ADV'}
        and not token.is_stop]
        for sentence in nlp(documents).sents]
```

CODE 6. Selecting content words based on POS.

This gives us

```
[["'ve", 'times', 'New', 'York', 'constitution'],
 ['appeal'],
 ['preferred', 'Los', 'Angeles']]
```

4. Named Entities

Say we are interested in the changing patterns of tourism, and want to find the most popular destinations over time in a corpus of travel blogs. We would like a way to identify the names of countries, cities, and landmarks. Using POS tagging, we can easily identify all proper names, but that still does not tell us what kind of entity we are dealing with. These semantic categories of words are referred to as **named entities**.

named entities

Proper names refer to entities in the real world, such as companies, places, persons, or something else: e.g., *Apple*, *Italy*, *George Bluth*, *LAX* or *Mercedes*. While implementing a **named entity recognizer** (NER) is outside the scope of this work, it generally works by using a list of known entities that are unlikely to change (for example country

named entity recognizer

[3] See the exception for profiling above, though.

names or first names). However, names are probably the most open-ended and innovative class of words, and there are plenty of entities that are not listed anywhere. Luckily, we can often determine what kind of entity we have by observing the context in which they occur: company names might be followed by "Inc.", "LLC", etc., and syntax can give us more cues—if an entity *says* something, *eats*, *sleeps*, or is associated with other verbs that denote human activities, we can likely label it as a person (this gets trickier when we have metaphoric language, e.g., "parliament says...").

Luckily, `spacy` provides a named entity recognizer that can help us identify a wide range of types: PERSON, NORP (Nationality OR Religious or Political group), FAC (facility), ORG (organization), GPE (GeoPolitical Entity), LOC (locations, such as seas or mountains), PRODUCT, EVENT (in sports, politics, history, etc.), WORK_OF_ART, LAW, LANGUAGE, DATE, TIME, PERCENT, MONEY, QUANTITY, ORDINAL (ordinal numbers such as "third", etc.), and CARDINAL (regular numbers).

For our running example, we can use the following code to extract the words and their types:

```
entities = [[(entity.text, entity.label_)
             for entity in nlp(sentence.text).ents]
             for sentence in nlp(documents).sents]
```

CODE 7. Named Entity Recognition.

This gives us

```
[[('2', 'CARDINAL'), ('New York', 'GPE'), ('2011', 'DATE')],
 [],
 [('Los Angeles', 'GPE')]]
```

5. Preprocessing

All of the last few section are examples of **preprocessing**, to get rid of noise and unnecessary variation, and to increase the amount of signal in the data. **preprocessing**

All of these steps can be put together, so we get a much more homogenous, less noisy version of our input corpus. However, there is no general rule of what steps to exclude an which to keep. For topic models (see chapter **??**), removing stopwords is crucial. However, for other tasks, such as author attribute prediction, the number and choice of stopwords, such as pronouns, is very important. We might not need numbers for sentiment analysis and can either remove them or replace them with a generic number token, but we would definitely want to keep the exact number if we are interested in tracking earnings.

Ultimately, which set of preprocessing steps to apply is highly dependent on the task and the goals. It makes sense to try out different combinations on a special held-out development set (see page **??**), and observe which one gives the best results.

6. Syntax

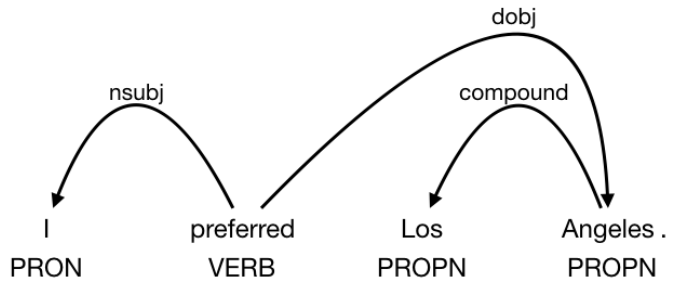

FIGURE 1. Example of a dependency parse.

Let's say we are interested in firm acquisitions. Imagine we want to filter out all the business transactions (who bought whom) from a large corpus of newswire text. We define a list of verbs that express acquisitions, and want to search for all the nouns associated with these verbs, i.e., doing the acquiring (the subjects), or being acquired (the objects). We want to end up with NOUN-VERB-NOUN triples that give us the names of the firms and other entities involved in the acquisitions, as well as the verb.

We have already seen that words can have different grammatical forms, i.e., parts of speech (nouns, verbs, etc). However, a sentence can have many words with the same POS, and what a specific word means in a sentence depends in part on its **grammatical function**. Sentence structure, or **syntax** is an important field of study in linguistics, and explains how words in a sentence hang together.

grammatical function
syntax

One of the most important cues to a word's function in English is its position in the sentence. By changing the word order, we change the grammatical function of the words (in English, grammatical function is mainly determined by word order, other languages use markers on the words): see the difference between "Swimmer eats fish" and "Fish eats swimmer". The **subject** (the entity doing the eating) and the **object** (the entity being eaten) are now switched. We have already seen verbs, subjects, and objects as examples of syntactic functions, but there are a number of other possible **dependency relations** that can hold between words. The idea in dependency grammar is that the sentence "hangs" off the main verb like a mobile, and that the connections between words describe how the words are connected.

subject
object
dependency relations

Naturally, these connections can vary quite a bit between languages, but similar to the Universal POS tags, there have been efforts to unify and limit the set of possibilities (McDonald et al., 2013; Nivre et al., 2015, 2016, inter alia). Here is a list of the ones

used in the Universal Dependencies project (again, see `https://universaldependencies.org`):

- `acl`, clausal modifier of a noun (adjectival clause)
- `advcl`, adverbial clause modifier
- `advmod`, adverbial modifier
- `amod`, adjectival modifier
- `appos`, appositional modifier
- `aux`, auxiliary verb
- `case`, case marker
- `cc`, coordinating conjunction
- `ccomp`, clausal complement
- `clf`, classifier
- `compound`, compound
- `conj`, conjunction
- `cop`, copula
- `csubj`, clausal subject
- `dep`, unspecified dependency
- `det`, determiner
- `discourse`, discourse element
- `dislocated`, dislocated elements
- `dobj`, direct object
- `expl`, expletive
- `fixed`, fixed multiword expression
- `flat`, flat multiword expression
- `goeswith`, goes with
- `iobj`, indirect object
- `list`, list
- `mark`, marker
- `nmod`, nominal modifier
- `nsubj`, nominal subject
- `nummod`, numeric modifier
- `obj`, object
- `obl`, oblique nominal
- `orphan`, orphan
- `parataxis`, parataxis
- `pobj`, prepositional object
- `punct`, punctuation
- `reparandum`, overridden disfluency
- `root`, the root of the sentence, usually a verb
- `vocative`, vocative
- `xcomp`, open clausal complement

Many of these might never or rarely occur in the language we study, and for many applications, we might only want to look at a small handful of relations, e.g. the core arguments, `nsubj`, `obj`, `iobj`, `pobj`, or `root`. We can see an example of a dependency parse in Figure 1. Why is this relevant? Knowing where a word stands in the sentence, and how many other words are related to it in which functions can therefore help us make sense of a sentence.

In Python, we can use the parser from the `spacy` library to get

```
[[(c.text, c.head.text, c.dep_) for c in nlp(sentence.text)]
 for sentence in nlp(documents).sents]
```

CODE 8. Parsing.

For the first sentence, this gives us

```
[('I', 'been', 'nsubj'),
  ("'ve", 'been', 'aux'),
  ('been', 'been', 'ROOT'),
  ('2', 'been', 'npadvmod'),
  ('times', '2', 'quantmod'),
  ('to', '2', 'prep'),
  ('New', 'York', 'compound'),
  ('York', 'to', 'pobj'),
  ('in', 'been', 'prep'),
  ('2011', 'in', 'pobj'),
  (',', 'been', 'punct'),
  ('but', 'been', 'cc'),
  ('did', 'have', 'aux'),
  ('not', 'have', 'neg'),
  ('have', 'been', 'conj'),
  ('the', 'constitution', 'det'),
  ('constitution', 'have', 'dobj'),
  ('for', 'have', 'prep'),
  ('it', 'for', 'pobj'),
  ('.', 'been', 'punct')]
```

We can see how the verb "been" is the root node that everything else hangs off of. For our example with the firm acquisitions, we would want to extract anything labeled with `nsubj`, `dobj`, or `pobj`.

7. Caveats – What if it's not English?

Say you have a whole load of Italian data that you want to work with, doing some of the things we have done in the previous sections. What are your options?

`spacy` comes with support for a number of other languages, including German (`de`),

Spanish (`es`), French (`fr`), Italian (`it`), Dutch (`nl`), and Portuguese (`pt`). All you have to do is load the correct library:

```
import spacy
nlp = spacy.load('it')
```

However, you need to download these language models separately, and make sure they are in the right path. They mostly offer all the same functionalities (lemmatization, tagging, parsing), but the performance can vary: because NLP has been so focused on English, there is often much less training data for other languages, and consequently, the statistical models often much are less precise.

NLP was developed predominantly in the English-speaking world, by people working on English texts. However, English is not like many other languages: due to its history (it is a pidgin language between some Celtic, old Germanic languages, Franco-Latin, and Norse), and at some point lost most of its inflections (the word endings that we learn in declension tables). However, word endings allow you to move the word order around: you still know what is the subject and the object is, no matter where they occur. In order to still know subject from object, English had to adopt a fixed word order. Because of this historic development, n-gram methods work well for English: we see the same order many times can can get good statistics over word combinations. In highly inflected languages (Finnish has 15 cases, i.e., many more possible word endings) or languages with variable word order (German orders subordinate clauses differently from main clauses), we might not even see the same n-gram twice, no matter how large our corpus is. For those languages, lemmatization becomes even more important, and we need to rely on syntactic n-grams rather than sequences.

Alternatively, `nltk` offers support to stem text in Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish and Swedish. This is of course much coarser than lemmatization, but it is a good way to reduce the noise in the data.

There are other, non-Python packages out there to handle tagging and parsing for more languages, most notably the TreeTagger algorithm, which can be trained on data from the Universal POS project, but it involves some manual installation and command-line running that go beyond the scope of this book.

If you are working with other languages than English, you can in many cases still lemmatize the data, but you might have fewer or no options for POS tagging, parsing, and NER. For many of the applications in the following chapters, having good preprocessing is often more important than being able to perform all types of linguistic analyses. However, if your analysis relies on these tools, you can take a look at the NLP literature on low-resource language processing, a thankfully growing subfield in NLP.

Bibliography

Jordan Boyd-Graber, David Mimno, and David Newman. 2014. *Care and Feeding of Topic Models: Problems, Diagnostics, and Improvements*, CRC Handbooks of Modern Statistical Methods. CRC Press, Boca Raton, Florida.

Francois Chollet. 2017. *Deep learning with python*. Manning Publications Co.

David Crystal. 2003. *The Cambridge Encyclopedia of the English Language*, 3rd edition. Cambridge University Press.

James A Evans and Pedro Aceves. 2016. Machine translation: mining text for social theory. *Annual Review of Sociology*, 42:21–50.

Victoria Fromkin, Robert Rodman, and Nina Hyams. 2018. *An introduction to language*. Cengage Learning.

Nikhil Garg, Londa Schiebinger, Dan Jurafsky, and James Zou. 2018. Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 115(16):E3635–E3644.

Matthew Gentzkow, Bryan T Kelly, and Matt Taddy. 2017. Text as data. Technical report, National Bureau of Economic Research.

Justin Grimmer and Brandon M Stewart. 2013. Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political analysis*, 21(3):267–297.

William L Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. In *Proceedings of the 54th Meeting of the Association for Computational Linguistics*, pages 1489–1501. Association for Computational Linguistics.

Jochen Hartmann, Juliana Huppertz, Christina Schamp, and Mark Heitmann. 2018. Comparing automated text classification methods. *International Journal of Research in Marketing*.

Ashlee Humphreys and Rebecca Jen-Hui Wang. 2017. Automated text analysis for consumer research. *Journal of Consumer Research*, 44(6):1274–1306.

Dan Jurafsky. 2014. *The language of food: A linguist reads the menu*. WW Norton & Company.

Dan Jurafsky and James H Martin. 2014. *Speech and language processing*, 3rd edition. Pearson London.

Vivek Kulkarni, Rami Al-Rfou, Bryan Perozzi, and Steven Skiena. 2015. Statistically significant detection of linguistic change. In *Proceedings of the 24th International Conference on World Wide Web*, pages 625–635. International World Wide Web Conferences Steering Committee.

William Labov. 1972. *Sociolinguistic patterns*. University of Pennsylvania Press.

Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*.

Christopher D Manning and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. MIT press.

Stephen Marsland. 2015. *Machine learning: an algorithmic perspective*, 2nd edition. Chapman and Hall/CRC.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 92–97.

Kevin P Murphy. 2012. *Machine learning: a probabilistic perspective*. MIT Press.

Vlad Niculae, Srijan Kumar, Jordan Boyd-Graber, and Cristian Danescu-Niculescu-Mizil. 2015. Linguistic Harbingers of Betrayal: A Case Study on an Online Strategy Game. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1650–1659.

Joakim Nivre, Željko Agić, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Cristina Bosco, et al. 2015. Universal dependencies 1.2.

Joakim Nivre, Marie-Catherine De Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan T McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *LREC*.

James W. Pennebaker. 2011. *The Secret Life of Pronouns: What Our Words Say About Us*. Broadway: Bloomsbury Press.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2011. A universal part-of-speech tagset. In *Proceedings of LREC*.

Martin F Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130–137.

Vinodkumar Prabhakaran, Owen Rambow, and Mona Diab. 2012. Predicting overt display of power in written dialogs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 518–522. Association for Computational Linguistics.

Peter Trudgill. 2000. *Sociolinguistics: An introduction to language and society*. Penguin UK.