Chapter 1

Introduction

1.1 The Big Picture

1.1.1 Applications

The pursuit of making computers do things with language is *natural language processing* (NLP). As I see it, there are three major applications of NLP.

- They can communicate with humans using a natural-language interface (*dialogue systems*) which we are beginning to see in systems like Apple Siri.
- If a computer can use two languages, it can enable two humans who don't speak the same language to communicate with each other (*machine translation*).
- They can take in a large amount of language data much faster than humans can, and then answer questions about it, summarize it, extract relevant pieces of information from it, etc.

Historically, translation was the oldest application, though not for the sake of enabling two-way communication, but rather for helping US intelligence to read Russian communications (and presumably vice versa). Later, natural language interfaces seemed to be the holy grail of NLP – for example, Bill Gates was a major advocate, saying things like "Most of [our research] now is focused on what we call the natural interface – the computer being able to listen and talk and recognize handwriting.... Now we're betting the company on these natural interface technologies." [1] But in these days of "big data," the third type of application is taking center stage.

1.1.2 Language

In the 1960s, the government appointed a committee to see how research in machine translation was going, and the committee wasn't happy. But, it did prescribe more basic research into *computational linguistics*, the use of computational methods for the scientific study of language. The hope was, and is, that by understanding better how human language works, we will do a better job programming computers to imitate it.

[1] Remarks at Gartner Symposium, 1997/10/06, Orlando, FL.

Of course, we don't know, in the end, whether computers will process language in the same way that humans do, or if it is even possible at all. We got the idea for building airplanes from watching birds, and airplanes and birds both have two wings, but the similarity ends there. NLP is like building airplanes: we take our inspiration from the human language faculty, but ultimately we do whatever works best.

1.1.3 Learning

In the 1990s, there was a second major shift in the way natural language processing was done. Instead of just building systems that simulate human use of language, we began trying to simulate a second human behavior: *learning* language. In other words, we used to program the rules of language directly into the computer, but now we program computers to learn the rules, and their weights, automatically from *data*. So the goal of modern, statistical NLP is to build computer systems that learn from data how to use human language.

Initially, people who used linguistics and the people who used statistics were at odds with each other. The reason was simple: linguistics is primarily interested in structures and representations that exist in the mind and cannot be directly observed, whereas statistics are based on observable quantities. So for a time, it was assumed that if you were using linguistics, you did not believe in statistics, and if you were using statistics, you did not believe in linguistics.

1.1.4 The synthesis

Over time, however, a synthesis emerged, as people started to build datasets annotated with linguistic structures (for example, the Penn Treebank) and to experiment with models that can learn unobserved things. The result is that nearly all work that appears at NLP conferences today is statistical, and a great deal of it also operates on linguistic structures – not necessarily tied to a particular linguistic theory, but very much drawing on the collective insights of linguistics about how language works.

In my view, the key to this synthesis was the formal models of language – finite automata and context-free grammars – proposed by and quickly discarded by Chomsky. These models make it possible to represent linguistically-reasonable structures in a way that machine learning algorithms can be very effectively adapted to.

We will look at three such formalisms, and these form the backbone of the course.

1. *Bags* of words. We don't care about the relationships between words, only the number of times they occur. (This part of the course has significant overlap with CSE 4/60497.)

2. *Strings* of words (or characters or sounds). We take into account the linear ordering of words, using finite-state automata.

3. *Trees* of words. We take into account the recursive structure of language, using context-free grammars.

1.2 Language

If you're interested in learning more about linguistics as it relates to NLP, see the survey by Bender (2013).

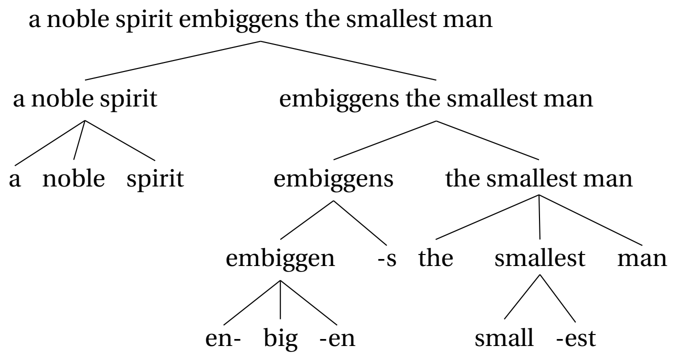

Figure 1.1: Levels of structure.

1.2.1 Speech and text

Primarily, language is *speech*; some languages have *signs* as their primary form. Both have complex structures that we don't have time to get into, but you'll probably hear me use the term *phoneme*, which is the basic abstract unit of speech sound. *Automatic speech recognition* and its inverse, *speech synthesis*, are a huge area of research, distinct from NLP. Recognition and generation of sign language are also an active area of research.

Secondarily, language is *text*, which represents speech (or signs) in written, printed, or digital form. Writing systems vary widely as to what information they capture. The symbols (*graphemes*) may correspond to sounds (Spanish), or just the consonant sounds (Arabic), or whole syllables (Japanese). The symbols may or may not capture meaning distinctions (Chinese). The writing system may or may not have word and sentence boundaries.

Optical character recognition and *handwriting recognition* are big areas of research, distinct from NLP. With the rise of mobile devices and their tiny keyboards, *adaptive text input* has become an interesting NLP problem.

There are other secondary forms of language like whistling and drums that are not the focus of any serious NLP research that I'm aware of.

1.2.2 Structure

Now we move to higher levels of structure, which are shared by, and independent of, speech and writing (Figure 1.1).

Words In English, the writing system groups characters into words. In such languages, automatically determining word boundaries is a relatively easy task. You may be surprised to hear that it's not totally trivial, and there are even cases where it's unclear what a word *is*; for example, in "Notre Dame's campus," is "Dame's" a word? Such cases notwithstanding, English tokenization is usually thought of as routine preprocessing rather than a research topic. By contrast, in Chinese writing, there are no spaces between words. What a "word" is is controversial, and even according to a given standard, the word boundaries are not trivial to detect, so Chinese word segmentation is a more serious problem.

Sentences The task of *sentence boundary detection* is, like word segmentation, fairly easy in languages that delimit sentences with punctuation. But it is also not quite as simple as splitting on periods, because they not only mark ends of sentences but are also used in abbreviations and numbers.

```
John lives on First St. Mary lives on Second St.
```

Nevertheless, automatic *sentence boundary detection* is usually accomplished using fairly simple rules.

Syntax Between words and sentences is *syntactic* structure. Suppose you want to translate this sentence into Latin:

(1.1)	spiritus	nobile	minimum	virum	auget
	spirit	noble	smallest	man	embiggens

But in order to do this right, your system has to learn that in Latin, verbs (auget = embiggen) usually come after their objects (minimum virum = the smallest man). These elements belong to syntactic structure, which is not explicit in our data (punctuation in text and intonation in speech give hints, but not very much).

The task of assigning a syntactic structure (or several possible syntactic structures) to a sentence is *parsing*, and there are various slices of this task that people have worked on, like *part-of-speech tagging* and *chunking*.

Morphemes Finally, words themselves have an internal structure. The word "embiggens" is not even a standard English word, but you know what it means and how to use it. It has a third-person singular suffix *-s*, so you know that "I embiggens" is ungrammatical. The prefix *en-* and the suffix *-en* both mean "to cause to be," so you know that "embiggen" means "to cause to be big." The smallest meaning-bearing subparts of a word are called *morphemes*.

Morphological analysis is an important NLP task (but under-researched, since NLP is dominated by English, which doesn't have very complex morphology), as is morphological generation.

1.2.3 Meaning

Each of the units discussed in the previous section (1.2.2) has a *meaning*. A letter/sound *r* doesn't have meaning, but a morpheme *re-* has meaning. Two important ideas relate structure to meaning: the principle of compositionality, and the problem of ambiguity.

The principle of *compositionality*, which originates in the philosophy of language, says that the meaning of an expression is a function of the meanings of its subexpressions. A word's meaning is a function of its morphemes' meanings; a phrase's meaning is a function of its words' meanings, and so on.

All human languages are *ambiguous*, that is, a given expression can have more than one meaning. And by the principle of compositionality, this means that an expression can have more than one structure, each of which yields a different meaning.

Words The study of word meanings is *lexical semantics*, and a lot of work has been done building electronic resources for lexical semantics and developing models for lexical semantics. The most common task in this area is *word sense disambiguation*: in the sentence "John ran to the *bank*," is a *bank* a place where you keep money, or the side of a river?

Syntax The task of converting from a syntactic structure to a meaning representation is currently known as *semantic parsing*. The meaning representation could be first-order logic, SQL queries, etc., anything that a computer can somehow execute. There are many less-ambitious slices of this problem, like *semantic role labeling* (is this person the one doing the action, or is the action done to them?), *coreference resolution* (who is "he" in this sentence?), and so on. The inverse is *generation*.

1.3 Learning

1.3.1 Models

In this course, we'll look at four kinds of models or machine learning methods, which can roughly be classified as follows.

- Do the variables in the model have *categorical distributions* (like rolling an n-sided die) or more general (*exponential-family*) distributions? The former are easy to understand and easy to learn, whereas the latter offer a lot more flexibility.

- Does the model have *hidden variables*? That is, does it learn about things that are not directly observed? For example, given two sentences in French and English that mean the same thing, the word-to-word correspondence between the two sentences is a hidden variable that the model can learn. Models with no hidden variables are easier to learn, whereas hidden variables are interesting because what the model discovers from the data is often useful in its own right.

This is indeed a rough classification, but good enough to distinguish the four methods we'll look at.

This year we're going to break from tradition and use the same basic technique for all of our models. All of our models will be probability distributions, and the goal in training is to adjust the parameters of the model to maximize the likelihood (probability) of the training data. That maximum will sometimes have a closed-form solution, but otherwise we will apply a simple numerical method, *stochastic gradient descent* (SGD), using a neural network toolkit. The reason this is a big break from tradition is that there's a class of models (the generative, unsupervised ones) that is normally trained by a method called *expectation-maximization*. We're going to skip that, and instead just use SGD. (I am told that this is how they do it in computational biology.)

1.3.2 Probability

Below is a very brief review of basic probability theory. The notation used for probabilities in NLP is a little sloppy, but hopefully this is good enough. For a proper treatment, see the textbook by Bertsekas and Tsitsiklis (2008).

A random variable is a variable with a different random value in each "experiment". For example, if our experiments are coin flips, we could define a random variable $C \in \{\text{heads}, \text{tails}\}$ for the result of the flip. Or, if our experiments are the words of a speech, we could define a random variable $W \in \{\text{a}, \text{aa}, \text{ab}, \ldots\}$ for the words spoken. If X is a random variable with values in $\mathscr{X}$, we call $P(X)$ the distribution of X. If $x \in \mathscr{X}$, we write $P(X = x)$ for the probability that X has value x. We must have

$$\sum_{x \in \mathscr{X}} P(X = x) = 1.$$

For example, if $P(W)$ is a distribution over English words, we might have

$$\begin{aligned} P(W = \text{the}) &= 0.1 \\ P(W = \text{syzygy}) &= 10^{-10} \\ &\vdots \end{aligned}$$

Things get more interesting when we deal with more than one random variable. For example, suppose our experiments are words spoken during a debate, and W is again the words spoken, while $S \in \{\text{Clinton}, \text{Trump}, \ldots\}$ is the person speaking. We can talk about the *joint distribution* of S and W, written $P(S,W)$, which should satisfy

$$\sum_{s,w} P(S = s, W = w) = 1.$$

Let's make up some numbers:

$$\begin{aligned} P(S = \text{Trump}, W = \text{bigly}) &= 0.2 \\ P(S = \text{Trump}, W = \text{huge}) &= 0.4 \\ P(S = \text{Clinton}, W = \text{people}) &= 0.3 \\ P(S = \text{Clinton}, W = \text{think}) &= 0.1. \end{aligned}$$

We also have to have

$$\begin{aligned} P(S = s) &= \sum_{w} P(S = s, W = w) \\ P(W = w) &= \sum_{s} P(S = s, W = w). \end{aligned}$$

Using our made-up numbers, we have

$$\begin{aligned} P(S = \text{Trump}) &= 0.2 + 0.4 = 0.6 \\ P(S = \text{Clinton}) &= 0.3 + 0.1 = 0.4 \end{aligned}$$

and

$$\begin{aligned} P(W = \text{bigly}) &= 0.2 \\ P(W = \text{huge}) &= 0.4 \\ P(W = \text{people}) &= 0.3 \\ P(W = \text{think}) &= 0.1. \end{aligned}$$

It's extremely common to write $P(w)$ as shorthand for $P(W = w)$. This leads to some sloppiness, because the symbol P is now "overloaded" to mean several things and you're supposed to know which one. To be precise, we should distinguish the distributions (using $P(S = s)$ or $P_S(s)$). But in NLP, we deal with some fairly complicated structures, and it becomes messy to keep this up. In practice, it's rarely a problem to use the sloppier notation.

We also define the *conditional distributions*

$$\begin{aligned} P(s \mid w) &= \frac{P(s,w)}{P(w)} \\ P(w \mid s) &= \frac{P(s,w)}{P(s)}. \end{aligned}$$

Note that

$$\sum_s P(s \mid w) = 1$$
$$\sum_w P(w \mid s) = 1.$$

You should know this already, but it should be second nature, and in particular, be sure never to get $p(s \mid w)$ and $p(w \mid s)$ confused! Using our made-up numbers:

$$P(\text{Trump} \mid \text{bigly}) = 0.2/0.2 = 1$$
$$P(\text{bigly} \mid \text{Trump}) = 0.2/0.6 \approx 0.333.$$

Finally, if a random variable has numeric values, we can talk about its average or expected value. For example, let $c_\text{e}(w)$ be the number of occurrences of the letter e in w. The *expectation* of c_e is

$$E[c_\text{e}] = \sum_w P(W = w) c_\text{e}(w),$$

and using our made-up numbers, this is

$$E[c_\text{e}] = 0.2 \cdot 0 + 0.4 \cdot 1 + 0.3 \cdot 2 + 0.1 \cdot 0 = 1.$$

Bibliography

Bender, Emily M. (2013). *Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax*. Morgan and Claypool.

Bertsekas, Dimitri P. and John N. Tsitsiklis (2008). *Introduction to Probability*. 2nd. Athena Scientific.