# 4 *Corpus-Based Work*

THIS CHAPTER begins with some brief advice on getting set up to do corpus-based work. The main requirements for Statistical NLP work are computers, corpora, and software. Many of the details of computers and corpora are subject to rapid change, and so it does not make sense to dwell on these. Moreover, in many cases, one will have to make do with the computers and corpora at one s local establishment, even if they are not in all respects ideal. Regarding software, this book does not attempt to teach programming skills as it goes, but assumes that a reader interested in implementing any of the algorithms described herein can already program in some programming language. Nevertheless, we provide in this section a few pointers to languages and tools that may be generally useful.

After that the chapter covers a number of interesting issues concerning the formats and problems one encounters when dealing with raw data – plain text in some electronic form. A very important, if often neglected, issue is the low-level processing which is done to the text before the real work of the research project begins. As we will see, there are a number of difficult issues in determining what is a word and what is a sentence. In practice these decisions are generally made by imperfect heuristic methods, and it is thus important to remember that the inaccuracies of these methods affect all subsequent results.

Finally the chapter turns to marked up data, where some process – often a human being – has added explicit markup to the text to indicate something of the structure and semantics of the document. This is often helpful, but raises its own questions about the kind and content of the markup used. We introduce the rudiments of SGML markup (and thus

also XML) and then turn to substantive issues such as the choice of tag sets used in corpora marked up for part of speech.

## 4.1 Getting Set Up

### 4.1.1 Computers

Text corpora are usually big. It takes quite a lot of computational resources to deal with large amounts of text. In the early days of computing, this was the major limitation on the use of corpora. For example in the earliest years of work on constructing the Brown corpus (the 1960s), just sorting all the words in the corpus to produce a word list would take 17 hours of (dedicated) processing time. This was because the computer (an IBM 7070) had the equivalent of only about 40 kilobytes of memory, and so the sort algorithm had to store the data being sorted on tape drives. Today one can sort this amount of data within minutes on even a modest computer.

As well as needing plenty of space to store corpora, Statistical NLP methods often consist of a step of collecting a large number of counts from corpora, which one would like to access speedily. This means that one wants a computer with lots of hard disk space, and lots of memory. In a rapidly changing world, it does not make much sense to be more precise than this about the hardware one needs. Fortunately, all the change is in a good direction, and often all that one will need is a decent personal computer with its RAM cheaply expanded (whereas even a few years ago, a substantial sum of money was needed to get a suitably fast computer with sufficient memory and hard disk space).

### 4.1.2 Corpora

A selection of some of the main organizations that distribute text corpora for linguistic purposes are shown in table 4.1. Most of these organizations charge moderate sums of money for corpora.[1] If your budget does not extend to this, there are now numerous sources of free text, ranging from email and web pages, to the many books and (maga)zines

1. Prices vary enormously, but are normally in the range of US$100–2000 per CD for academic and nonprofit organizations, and reflect the considerable cost of collecting and processing material.

| | |
|---|---|
| Linguistic Data Consortium (LDC) | http://www.ldc.upenn.edu |
| European Language Resources Association (ELRA) | http://www.icp.grenet.fr/ELRA/ |
| International Computer Archive of Modern English (ICAME) | http://nora.hd.uib.no/icame.html |
| Oxford Text Archive (OTA) | http://ota.ahds.ac.uk/ |
| Child Language Data Exchange System (CHILDES) | http://childes.psy.cmu.edu/ |

**Table** 4.1 Major suppliers of electronic corpora with contact URLs.

that are available free on the web. Such free sources will not bring you linguistically-marked-up corpora, but often there are tools that can do the task of adding markup automatically reasonably well, and at any rate, working out how to deal with raw text brings its own challenges. Further resources for online text can be found on the website.

When working with a corpus, we have to be careful about the validity of estimates or other results of statistical analysis that we produce. A corpus is a special collection of textual material collected according to a certain set of criteria. For example, the Brown corpus was designed as a representative sample of written American English as used in 1961 (Francis and Kučera 1982: S-6). Some of the criteria employed in its construction were to include particular texts in amounts proportional to actual publication and to exclude verse because it presents special linguistic problems (p. 5).

As a result, estimates obtained from the Brown corpus do not necessarily hold for British English or spoken American English. For example, the estimates of the entropy of English in section 2.2.7 depend heavily on the corpus that is used for estimation. One would expect the entropy of poetry to be higher than that of other written text since poetry can flout semantic expectations and even grammar. So the entropy of the Brown corpus will not help much in assessing the entropy of poetry. A more mundane example is text categorization (see chapter 16) where the performance of a system can deteriorate significantly over time because a sample drawn for training at one point can lose its representativeness after a year or two.

REPRESENTATIVE SAMPLE

The general issue is whether the corpus is a ***representative sample*** of the population of interest. A sample is representative if what we find for the sample also holds for the general population. We will not discuss methods for determining representativeness here since this issue is dealt with at length in the corpus linguistics literature. We also refer

BALANCEDCORPUS

the reader to this literature for creating *balanced* corpora, which are put together so as to give each subtype of text a share of the corpus that is proportional to some predetermined criterion of importance. In Statistical NLP, one commonly receives as a corpus a certain amount of data from a certain domain of interest, without having any say in how it is constructed. In such cases, having more training text is normally more useful than any concerns of balance, and one should simply use all the text that is available.

In summary, there is no easy way of determining whether a corpus is representative, but it is an important issue to keep in mind when doing Statistical NLP work. The minimal questions we should attempt to answer when we select a corpus or report results are what type of text the corpus is representative of and whether the results obtained will transfer to the domain of interest.

▼ The effect of corpus variability on the accuracy of part-of-speech tagging is discussed in section 10.3.2.

### 4.1.3 Software

There are many programs available for looking at text corpora and analyzing the data that you see. In general, however, we assume that readers will be writing their own software, and so all the software that is really needed is a plain text editor, and a compiler or interpreter for a language of choice. However, certain other tools, such as ones for searching through text corpora can often be of use. We briefly describe some such tools later.

#### Text editors

**You** will want a plain text editor that shows fairly literally what is actually in the file. Fairly standard and cheap choices are **Emacs** for Unix (or Windows), TextPad for Windows, and BBEdit for Macintosh.

#### Regular expressions

In many places and in many programs, editors, etc., one wishes to find certain patterns in text, that are often more complex than a simple match against a sequence of characters. The most general widespread notation for such matches are regular *expressions* which can describe patterns

REGULAR EXPRESSIONS

REGULAR LANGUAGE

that are a regular *language*, the kind that can be recognized by a finite state machine. If you are not already familiar with regular expressions, you will want to become familiar with them. Regular expressions can be used in many plain text editors (Emacs, TextPad, Nisus, BBEdit, ...), with many tools (such as grep and sed), and as built-ins or libraries in many programming languages (such as Pet-l, C, ...). Introductions to regular expressions can be found in (Hopcroft and Ullman 1979; Sipser 1996; Fried1 1997).

### Programming languages

Most Statistical NLP work is currently done in C/C++. The need to deal with large amounts of data collection and processing from large texts means that the efficiency gains of coding in a language like C/C++ are generally worth it. But for a lot of the ancillary processing of text, there are many other languages which may be more economical with human labor. Many people use Perl for general text preparation and reformatting. Its integration of regular expressions into the language syntax is particularly powerful. In general, interpreted languages are faster for these kinds of tasks than writing everything in C. Old timers might still use awk rather than Perl – even though what you can do with it is rather more limited. Another choice, better liked by programming purists is Python, but using regular expressions in Python just is not as easy as Perl. One of the authors still makes considerable use of Prolog. The built-in database facilities and easy handling of complicated data structures makes Prolog excel for some tasks, but again, it lacks the easy access to regular expressions available in Perl. There are other languages such as SNOBOL/SPITBOL or Icon developed for text computing, and which are liked by some in the humanities computing world, but their use does not seem to have permeated into the Statistical NLP community. In the last few years there has been increasing uptake of Java. While not as fast as C, Java has many other appealing features, such as being object-oriented, providing automatic memory management, and having many useful libraries.

### Programming techniques

This section is not meant as a substitute for a general knowledge of computer algorithms, but we briefly mention a couple of useful tips.

Coding words. Normally Statistical NLP systems deal with a large number of words, and programming languages like C(++) provide only quite limited facilities for dealing with words. A method that is commonly used in Statistical NLP and Information Retrieval is to map words to numbers on input (and only back to words when needed for output). This gives a lot of advantages because things like equality can be checked more easily and quickly on numbers. It also maps all tokens of a word to its type, which has a single number. There are various ways to do this. One good way is to maintain a large hash table (a hash function maps a set of objects into a specificed range of integers, for example, $[0, \ldots, 127]$). A hash table allows one to see efficiently whether a word has been seen before, and if so return its number, or else add it and assign a new number. The numbers used might be indices into an array of words (especially effective if one limits the application to 65,000 or fewer words, so they can be stored as 16 bit numbers) or they might just be the address of the canonical form of the string as stored in the hashtable. This is especially convenient on output, as then no conversion back to a word has to be done: the string can just be printed.

There are other useful data structures such as various kinds of trees. See a book on algorithms such as (Cormen et al. 1990) or (Frakes and Baeza-Yates 1992).

Collecting count data. For a lot of Statistical NLP work, there is a first step of collecting counts of various observations, as a basis for estimating probabilities. The seemingly obvious way to do that is to build a big data structure (arrays or whatever) in which one counts each event of interest. But this can often work badly in practice since this model requires a huge memory address space which is being roughly randomly accessed. Unless your computer has enough memory for all those tables, the program will end up swapping a lot and will run very slowly. Often a better approach is for the data collecting program to simply emit a token representing each observation, and then for a follow on program to sort and then count these tokens. Indeed, these latter steps can often be done by existing system utilities (such as sort and **uniq** on Unix systems). Among other places, such a strategy is very successfully used in the CMU-Cambridge Statistical Language Modeling toolkit which can be obtained from the web (see website).

## 4.2 Looking at Text

MARKUP

Text will usually come in either a raw format, or marked up in some way. *Markup* is a term that is used for putting codes of some sort into a computer file, that are not actually part of the text in the file, but explain something of the structure or formatting of that text. Nearly all computer systems for dealing with text use mark-up of some sort. Commercial word processing software uses markup, but hides it from the user by employing WYSIWYG (What You See Is What You Get) display. Normally, when dealing with corpora in Statistical NLP, we will want explicit markup that we can see. This is part of why the first tool in a corpus linguist s toolbox is a plain text editor.

There are a number of features of text in human languages that can make them difficult to process automatically, even at a low level. Here we discuss some of the basic problems that one should be aware of. The discussion is dominated by, but not exclusively concerned with, the most fundamental problems in *English* text.

### 4.2.1 Low-level formatting issues

#### Junk formatting/content

OCR

Depending on the source of the corpus, there may be various formatting and content that one cannot deal with, and is just junk that needs to be filtered out. This may include: document headers and separators, typesetter codes, tables and diagrams, garbled data in the computer file, etc. If the data comes from *OCR* (Optical Character Recognition), the OCR process may have introduced problems such as headers, footers and floating material (tables, figures, and footnotes) breaking up the paragraphs of the text. There will also usually be OCR errors where words have been misrecognized. If your program is meant to deal with only connected English text, then other kinds of content such as tables and pictures need to be regarded as junk. Often one needs a filter to remove junk content before any further processing begins.

#### Uppercase and lowercase

The original Brown corpus was all capitals (a $*$ before a letter was used to indicate a capital letter in the original source text). All uppercase text is

rarely seen these days, but even with modern texts, there are questions of how to treat capitalization. In particular, if we have two tokens that are identical except that one has certain letters in uppercase, should we treat them as the same? For many purposes we would like to treat the, The, and THE as the same, for example if we just want to do a study of the usage of definite articles, or noun phrase structure. This is easily done by converting all words to upper- or lowercase, but the problem is that at the same time we would normally like to keep the two types of Brown in *Richard* Brown and brown *paint* distinct. In many circumstances it is easy to distinguish *proper* names and hence to keep this distinction, but sometimes it is not. A simple heuristic is to change to lowercase letters capital letters at the start of a sentence (where English regularly capitalizes all words) and in things like headings and titles when there is a series of words that are all in capitals, while other words with capital letters are assumed to be names and their uppercase letters are preserved. This heuristic works quite well, but naturally, there are problems. The first problem is that one has to be able to correctly identify the ends of sentences, which is not always easy, as we discuss later. In certain genres (such as *Winnie the Pooh),* words may be capitalized just to stress that they are making a Very Important Point, without them indicating a proper name. At any rate, the heuristic will wrongly lowercase names that appear sentence initially or in all uppercase sequences. Often this source of error can be tolerated (because regular words are usually more common than proper names), but sometimes this would badly bias estimates. One can attempt to do better by keeping lists of proper names (perhaps with further information on whether they name a person, place, or company), but in general there is not an easy solution to the problem of accurate proper name detection.

PROPER NAMES

### 4.2.2 Tokenization: What is a word?

Normally, an early step of processing is to divide the input text into units called tokens where each is either a *word* or something else like a number or a punctuation mark. This process is referred to as *tokenization.* The treatment of punctuation varies. While normally people want to keep sentence boundaries (see section 4.2.4 below), often sentence-internal punctuation has just been stripped out. This is probably unwise. Recent work has emphasized the information contained in all punctuation. No matter how imperfect a representation, punctuation marks like commas and

TOKENS
WORD
TOKENIZATION

dashes give some clues about the macro structure of the text and what is likely to modify what.

The question of what counts as a word is a vexed one in linguistics, and often linguists end up suggesting that there are words at various levels, such as phonological words versus syntactic words, which need not all be the same. What is a humble computational linguist meant to do?

GRAPHIC WORD

Kučera and Francis (1967) suggested the practical notion of a graphic word which they define as a string of contiguous alphanumeric characters with space on either side; may include hyphens and apostrophes, but no other punctuation marks. But, unfortunately, life is not that simple, even if one is just looking for a practical, workable definition. Kučera and Francis seem in practice to use intuition, since they regard as words numbers and monetary amounts like $22.50 which do not strictly seem to obey the definition above. And things get considerably worse. Especially if using online material such as newsgroups and web pages for data, but even if sticking to newswires, one finds all sorts of oddities that should presumably be counted as words, such as references to *Micro$oft* or the web company *C*| net, or the various forms of smilies made out of punctuation marks, such as `:-)`. Even putting aside such creatures, working out word tokens is a quite difficult affair. The main clue used in English is

WHITESPACE

the occurrence of *whitespace* – a space or tab or the beginning of a new line between words – but even this signal is not necessarily reliable. What are the main problems?

### Periods

Words are not always surrounded by white space. Often punctuation marks attach to words, such as commas, semicolons, and periods (full stops). It at first seems easy to remove punctuation marks from word tokens, but this is problematic for the case of periods. While most periods are end of sentence punctuation marks, others mark an abbreviation such as in etc. or *Calif.* These abbreviation periods presumably should remain as part of the word, and in some cases keeping them might be important so that we can distinguish Wash., an abbreviation for the state of Washington, from the capitalized form of the verb *wash*. Note especially that when an abbreviation like etc. appears at the end of the sentence, then only one period occurs, but it serves both functions of the period, simultaneously! An example occurred with *Calif.* earlier in this paragraph. Within morphology, this phenomenon is referred to as *haplology*.

HAPLOLOGY

The issue of working out which punctuation marks do indicate the end of a sentence is discussed further in section 42.4.

### Single apostrophes

It is a difficult question to know how to regard English contractions such as *I'll* or *isn't*. These count as one graphic word according to the definition above, but many people have a strong intuition that we really have two words here as these are contractions for I *will* and *is not*. Thus some processors (and some corpora, such as the Penn Treebank) split such contractions into two words, while others do not. Note the impact that not splitting them has. The traditional first syntax rule:

$$S \longrightarrow NP\ VP$$

stops being obviously true of sentences involving contractions such as I *right*. On the other hand, if one does split, there are then funny words like *'s* and *n't* in your data.

Phrases such as the dog s and *the child s*, when not abbreviations for the dog *is* or *the dog has*, are commonly seen as containing dog s as the genitive or possessive case of dog. But as we mentioned in section 3.1.1, this is not actually correct for English where *'s* is a *clitic* which can attach to other elements in a noun phrase, such as in The house *I rented yesterday s garden is really big*. Thus it is again unclear whether to regard *dog s* as one word or two, and again the Penn Treebank opts for the latter. Orthographic-word-final single quotations are an especially tricky case. Normally they represent the end of a quotation – and so should not be part of a word, but when following an *s*, they may represent an (unpronounced) indicator of a plural possessive, as in *the boys toys* – and then should be treated as part of the word, if other possessives are being so treated. There is no easy way for a tokenizer to determine which function is intended in many such cases.

CLITIC

### Hyphenation: Different forms representing the same word

Perhaps one of the most difficult areas is dealing with hyphens in the input. Do sequences of letters with a hyphen in between count as one word or two? Again, the intuitive answer seems to be sometimes one, sometimes two. This reflects the many sources of hyphens in texts.

One source is typographical. Words have traditionally been broken and hyphens inserted to improve justification of text. These line-breaking hyphens may be present in data if it comes from what was actually typeset. It would seem to be an easy problem to just look for hyphens at the end of a line, remove them and join the part words at the end of one line and the beginning of the next. But again, there is the problem of haplology. If there is a hyphen from some other source, then after that hyphen is regarded as a legitimate place to break the text, and only one hyphen appears not two. So it is not always correct to delete hyphens at the end of a line, and it is difficult in general to detect which hyphens were line-breaking hyphens and which were not.

Even if such line-breaking hyphens are not present (and they usually are not in truly electronic texts), difficult problems remain. Some things with hyphens are clearly best treated as a single word, such as e-mail or *co-operate* or *A-l-plus* (as in *A-l -plus commercial paper,* a financial rating). Other cases are much more arguable, although we usually want to regard them as a single word, for example, *non-lawyer, pro-Arab,* and *so-culled.* The hyphens here might be termed lexical hyphens. They are commonly inserted before or after small word formatives, sometimes for the purpose of splitting up vowel sequences.

The third class of hyphens is ones inserted to help indicate the correct grouping of words. A common copy-editing practice is to hyphenate compound pre-modifiers, as in the example earlier in this sentence or in examples like these:

*(4.1)* a. the once-quiet study of superconductivity

b. a tough regime of business-conduct rules

c. the aluminum-export ban

d. a text-based medium

And hyphens occur in other places, where a phrase is seen as in some sense quotative or as expressing a quantity or rate:

(4.2) a. the idea of a child-as-required-yuppie-possession must be motivating them

b. a final take-it-or-leave-it offer

c. the 90-cent-an-hour raise

d. the 26-year-old

In these cases, we would probably want to treat the things joined by hyphens as separate words. In many corpora this type of hyphenation is very common, and it would greatly increase the size of the word vocabulary (mainly with items outside a dictionary) and obscure the syntactic structure of the text if such things were not split apart into separate words.[2]

A particular problem in this area is that the use of hyphens in many such cases is extremely inconsistent. Some texts and authorities use *cooperate,* while others use *co-operate.* As another example, in the Dow Jones newswire, one can find all **of** *database*, **data-base and** *data base* (the first and third are commonest, with the former appearing to dominate in software contexts, and the third in discussions of company assets, but without there being any clear semantic distinction in usage). Closer to home, look back at the beginning of this section. When we initially drafted this chapter, we (quite accidentally) used all of markup, *murk-up* and *mark(ed) up.* A careful copy editor would catch this and demand consistency, but a lot of the text we use has never been past a careful copy editor, and at any rate, we will commonly use texts from different sources which often adopt different conventions in just such matters. Note that this means that we will often have multiple forms, perhaps some treated as one word and others as two, for what is best thought of as a single

LEXEME

*lexeme* (a single dictionary entry with a single meaning).

Finally, while British typographic conventions put spaces between dashes and surrounding words, American typographic conventions normally have a long dash butting straight up against the words-like this. While sometimes this dash will be rendered as a special character or as multiple dashes in a computer file, the limitations of traditional computer character sets means that it can sometimes be rendered just as a hyphen, which just further compounds the difficulties noted above.

### The same form representing multiple 'words'

In the main we have been collapsing distinctions and suggesting that one may wish to regard variant sequences of characters as really the

2. One possibility is to split things apart, but to add markup, as discussed later in this chapter, which records that the original was hyphenated. In this way no information is lost.

same word. It is important to also observe the opposite problem, where one might wish to treat the identical sequence of characters as different words. This happens with ***homographs,*** where two lexemes have overlapping forms, such as saw as a noun for a tool, or as the past tense of the verb see. In such cases we might wish to assign occurrences of saw to two different lexemes.

HOMOGRAPHS

▼ Methods of doing this automatically are discussed in chapter 7.

### Word segmentation in other languages

Many languages do not put spaces in between words at all, and so the basic word division algorithm of breaking on whitespace is of no use at all. Such languages include major East-Asian languages/scripts, such as Chinese, Japanese, and Thai. Ancient Greek was also written by Ancient Greeks without word spaces. Spaces were introduced (together with accent marks, etc.) by those who came afterwards. In such languages, ***word segmentation*** is a much more major and challenging task.

WORD SEGMENTATION

While maintaining most word spaces, in German compound nouns are written as a single word, for example *Lebensversicherungsgesellschaftsangestellter* life insurance company employee. In many ways this makes linguistic sense, as compounds *are* a single word, at least phonologically. But for processing purposes one may wish to divide such a compound, or at least to be aware of the internal structure of the word, and this becomes a limited word segmentation task. While not the rule, joining of compounds sometimes also happens in English, especially when they are common and have a specialized meaning. We noted above that one finds both ***data base*** and ***database.*** As another example, while ***hard disk*** is more common, one sometimes finds *harddisk* in the computer press.

### Whitespace not indicating a word break

Until now, the problems we have dealt with have mainly involved splitting apart sequences of characters where the word divisions are not shown by whitespace. But the opposite problem of wanting to lump things together also occurs. Here, things are separated by whitespace but we may wish to regard them as a single word. One possible case is the reverse of the German compound problem. If one decides to treat ***database*** as one word, one may wish to treat it as one word even when it is written as ***data base.*** More common cases are things such as phone numbers, where we

may wish to regard 9365 *1873* as a single word, or in the cases of multi-part names such as New York or San *Francisco. An* especially difficult case is when this problem interacts with hyphenation as in a phrase like this one:

*(4.3)* the New York-New Haven railroad

Here the hyphen does not express grouping of just the immediately adjacent graphic words – treating York-New as a semantic unit would be a big mistake.

Other cases are of more linguistic interest. For many purposes, one would want to regard phrasal verbs (make up, work out) as a single lexeme (section 3.1.4), but this case is especially tricky since in many cases the particle is separable from the verb (*I couldn t work the answer out),* and so in general identification of possible phrasal verbs will have to be left to subsequent processing. One might also want to treat as a single lexeme certain other fixed phrases, such as *in spite of, in order to,* and because *of,* but typically a tokenizer will regard them as separate words. A partial implementation of this approach occurs in the LOB corpus where certain pairs of words such as because of are tagged with a single part of speech, here preposition, by means of using so-called *ditto tags.*

DITTO TAGS

**Variant coding of information of a certain semantic type**

Many readers may have felt that the example of a phone number in the previous section was not very recognizable or convincing because *their* phone numbers are written as 812-4374, or whatever. However, even if one is not dealing with multilingual text, any application dealing with text from different countries or written according to different stylistic conventions has to be prepared to deal with typographical differences. In particular, some items such as phone numbers are clearly of one semantic sort, but can appear in many formats. A selection of formats for phone numbers with their countries, all culled from advertisements in one issue of the magazine The *Economist,* is shown in table 4.2. Phone numbers variously use spaces, periods, hyphens, brackets, and even slashes to group digits in various ways, often not consistently even within one country. Additionally, phone numbers may include international or national long distance codes, or attempt to show both (as in the first three UK entries in the table), or just show a local number, and there may or may not be explicit indication of this via other marks such as brackets and plus

| Phone number | Country | Phone number | Country |
|---|---|---|---|
| 01713780647 | UK | +45 43 48 60 60 | Denmark |
| (44.171) 830 1007 | UK | 95-51-279648 | Pakistan |
| +44 (0) 1225 753678 | UK | +411/284 3797 | Switzerland |
| 01256468551 | UK | (94-l) 866854 | Sri Lanka |
| (202) 522-2230 | USA | +49 69 136-2 98 05 | Germany |
| l-925-225-3000 | USA | 3 3 1 3 4 4 3 3 2 2 6 | France |
| 212. 995.5402 | USA | ++31-20-5200161 | The Netherlands |

Table 4.2 Different formats for telephone numbers appearing in an issue of *The Economist.*

INFORMATION EXTRACTION

signs. Trying to deal with myriad formats like this is a standard problem in information *extraction.* It has most commonly been dealt with by building carefully handcrafted regular expressions to match formats, but given the brittleness of such an approach, there is considerable interest in automatic means for learning the formatting of semantic types.

▼ We do not cover information extraction extensively in this book, but there is a little further discussion in section 10.6.2.

**Speech corpora**

Our discussion has concentrated on written text, but the transcripts of speech corpora provide their own additional challenges. Speech corpora normally have more contractions, various sorts of more phonetic representations, show pronunciation variants, contain many sentence fragments, and include fillers like *er* and *um.* Example (4.4) – from the Switchboard corpus available from the LDC – shows a typical extract from a speech transcript:

(4.4) Also I [cough] not convinced that the, at least the kind of people that I work with, I m not convinced that that s really, uh, doing much for the progr-, for the, uh, drug problem.

### 4.2.3 Morphology

Another question is whether one wants to keep word forms like *sit, sits* and *sat* separate or to collapse them. The issues here are similar to those in the discussion of capitalization, but have traditionally been regarded

as more linguistically interesting. At first, grouping such forms together and working in terms of lexemes feels as if it is the right thing to do. Doing this is usually referred to in the literature as *stemming* in reference to a process that strips off affixes and leaves you with a stem. Alternatively, the process may be referred to as *lemmatization* where one is attempting to find the lemma or *lexeme* of which one is looking at an inflected form. These latter terms imply disambiguation at the level of lexemes, such as whether a use of *lying* represents the verb lie-lay to prostrate oneself or *lie-lied* to fib.'

STEMMING

LEMMATIZATION

LEMMA

Extensive empirical research within the Information Retrieval (IR) community has shown that doing stemming does not help the performance of classic IR systems when performance is measured as an average over queries (Salton 1989; Hull 1996). There are always some queries for which stemming helps a lot. But there are others where performance goes down. This is a somewhat surprising result, especially from the viewpoint of linguistic intuition, and so it is important to understand why that is. There are three main reasons for this.

One is that while grouping the various forms of a stem seems a good thing to do, it often costs you a lot of information. For instance, while *operating* can be used in a periphrastic tense form as in *Bill is operating a tractor* (section 3.1.3), it is usually used in noun- and adjective-like uses such as *operating systems* or *operating costs.* It is not hard to see why a search for *operating systems* will perform better if it is done on inflected words than if one instead searches for all paragraphs that contain *operat-* and *system.* Or to consider another example, if someone enters *business* and the stemmer then causes retrieval of documents with busy in them, the results are unlikely to be beneficial.

Secondly, morphological analysis splits one token into several. However, often it is worthwhile to group closely related information into chunks, notwithstanding the blowout in the vocabulary that this causes. Indeed, in various Statistical NLP domains, people have been able to improve system performance by regarding frequent multiword units as a single distinctive token. Often inflected words are a useful and effective chunk size.

Thirdly, most information retrieval studies have been done on English – although recently there has been increasing multilingual work. English has very little morphology, and so the need for dealing intelligently with morphology is not acute. Many other languages have far richer systems of inflection and derivation, and then there is a pressing need for mor-

FULL-FORM LEXICON

phological analysis. A *full-form* lexicon for such languages, one that separately lists all inflected forms of all words, would simply be too large. For instance, Bantu languages (spoken in central and southern Africa) display rich verbal morphology. Here is a form from KiHaya (Tanzania). Note the prefixes for subject and object agreement, and tense:

(4.5) akabimúha
a-ka-b&mu-ha
1SG-PAST-3PL-3SG-give
'I gave them to him.'

For historical reasons, some Bantu language orthographies write many of these morphemes with whitespace in between them, but in the languages with conjunctive orthographies, morphological analysis is badly needed. There is an extensive system of pronoun and tense markers appearing before the verb root, and quite a few other morphemes that can appear after the root, yielding a large system of combinatoric possibilities. Finnish is another language famous for millions of inflected forms for each verb.

One might be tempted to conclude from the paragraphs above that, in languages with rich morphology, one would gain by stripping inflectional morphology but not derivational morphology. But this hypothesis remains to be carefully tested in languages where there is sufficient inflectional morphology for the question to be interesting.

It is important to realize that this result from IR need not apply to any or all Statistical NLP applications. It need not even apply to all of IR. Morphological analysis might be much more useful in other applications. Stemming does not help in the non-interactive evaluation of IR systems, where a query is presented and processed without further input, and the results are evaluated in terms of the appropriateness of the set of documents returned. However, principled morphological analysis is valuable in IR in an interactive context, the context in which IR should really be evaluated. A computer does not care about weird stems like busy from business, but people do. They do not understand what is going on when *business* is stemmed to *busy* and a document with *busy* in it is returned.

It is also the case that nobody has systematically studied the possibility of letting people interactively influence the stemming. We believe that this could be very profitable, for cases like saw (where you want to stem for the sense see, but not for the sense cutting implement), or derivational cases where in some cases you want the stems (*arbitrary*

from *arbitrariness*), but in some you do not (busy from *business*). But the suggestion that human input may be needed does show the difficulties of doing automatic stemming in a knowledge-poor environment of the sort that has often been assumed in Statistical NLP work (for both ideological and practical reasons).

▼ Stemming and IR in general are further discussed in chapter 15.

### 4.2.4 Sentences

**What is a sentence?**

The first answer to what is a sentence is something ending with a '.', '?' or '!'." We have already mentioned the problem that only some periods mark the end of a sentence: others are used to show an abbreviation, or for both these functions at once. Nevertheless, this basic heuristic gets one a long way: in general about 90% of periods are sentence boundary indicators (Riley 1989). There are a few other pitfalls to be aware of. Sometimes other punctuation marks split up what one might want to regard as a sentence. Often what is on one or the other or even both sides of the punctuation marks colon, semicolon, and dash (':', ';', and '—') might best be thought of as a sentence by itself, as ':' in this example:

(4.6) The scene is written with a combination of unbridled passion and sure-handed control: In the exchanges of the three characters and the rise and fall of emotions, Mr. Weller has captured the heartbreaking inexorability of separation.

Related to this is the fact that sometimes sentences do not nicely follow in sequence, but seem to nest in awkward ways. While normally nested things are not seen as sentences by themselves, but clauses, this classification can be strained for cases such as the quoting of direct speech, where we get subsentences:

(4.7) "You remind me," she remarked, "of your mother."

A second problem with such indirect speech is that it is standard typesetting practice (particularly in North America) to place quotation marks after sentence final punctuation. Therefore, the end of the sentence is not after the period in the example above, but after the close quotation mark that follows the period.

The above remarks suggest that the essence of a heuristic sentence division algorithm is roughly as in figure 4.1. In practice most systems

- Place putative sentence boundaries after all occurrences of .?! (and maybe ;:—)
- Move the boundary after following quotation marks, if any.
- Disqualify a period boundary in the following circumstances:
  - If it is preceded by a known abbreviation of a sort that does not normally occur word finally, but is commonly followed by a capitalized proper name, such as Prof. or vs.
  - If it is preceded by a known abbreviation and not followed by an uppercase word. This will deal correctly with most usages of abbreviations like *etc.* or *Jr.* which can occur sentence medially or finally.
- Disqualify a boundary with a ? or ! if:
  - It is followed by a lowercase letter (or a known name).
- Regard other putative sentence boundaries as sentence boundaries.

Figure 4.1 Heuristic sentence boundary detection algorithm.

have used heuristic algorithms of this sort. With enough effort in their development, they can work very well, at least within the textual domain for which they were built. But any such solution suffers from the same problems of heuristic processes in other parts of the tokenization process. They require a lot of hand-coding and domain knowledge on the part of the person constructing the tokenizer, and tend to be brittle and domain-specific.

There has been increasing research recently on more principled methods of sentence boundary detection. Riley (1989) used statistical classification trees to determine sentence boundaries. The features for the classification trees include the case and length of the words preceding and following a period, and the a priori probability of different words to occur before and after a sentence boundary (the computation of which requires a large quantity of labeled training data). Palmer and Hearst (1994; 1997) avoid the need for acquiring such data by simply using the part of speech distribution of the preceding and following words, and using a neural network to predict sentence boundaries. This yields a