

Chapter 2: Tokenisation and Sentence Segmentation

David D. Palmer
The MITRE Corporation

1. Introduction

In linguistic analysis of a natural language text, it is necessary to clearly define what constitutes a word and a sentence. Defining these units presents different challenges depending on the language being processed, but neither task is trivial, especially when considering the variety of human languages and writing systems. Natural languages contain inherent ambiguities, and much of the challenge of Natural Language Processing (NLP) involves resolving these ambiguities.

In this chapter we will discuss the challenges posed by **text segmentation**, the task of dividing a text into linguistically-meaningful units - at the lowest level characters representing the individual graphemes in a language's written system, words consisting of one or more characters, and sentences consisting of one or more words. Text segmentation is a frequently overlooked yet essential part of any NLP system, since the words and sentences identified at this stage are the fundamental units passed to further processing stages such as morphological analyzers, part-of-speech taggers, parsers, and information retrieval systems.

Tokenisation is the process of breaking up the sequence of characters in a text by locating the **word boundaries**, the points where one word ends and another begins. For computational linguistics purposes, the words thus identified are frequently referred to as **tokens**. In written languages where no word boundaries are explicitly marked in the writing system, tokenisation is also known as **word segmentation**, and this term is frequently used synonymously with tokenisation.

Sentence segmentation is the process of determining the longer processing units consisting of one or more words. This task involves identifying **sentence boundaries** between words in different sentences. Since most written languages have punctuation marks which occur at sentence boundaries, sentence segmentation is frequently referred to as **sentence boundary detection**, **sentence boundary disambiguation**, or **sentence boundary recognition**. All these terms refer to the same task: determining how a text should be divided into sentences for further processing.

In practice, sentence and word segmentation cannot be performed successfully independent from one another. For example, an essential subtask in both word and sentence segmentation for English is identifying abbreviations, because a period can be used in English to mark an abbreviation as well as to mark the end of a sentence. In the case of a period marking an abbreviation, the period is usually considered a part of the abbreviation token, whereas a period at the end of a sentence is usually considered a token in and of itself. Tokenising abbreviations is complicated further when an abbreviation occurs at the end of a sentence, and the period marks *both* the abbreviation and the sentence boundary.

This chapter will provide an introduction to word and sentence segmentation in a

variety of languages. We will begin in Section 2 with a discussion of the challenges posed by text segmentation, and emphasize the issues which must be considered before implementing a tokenisation or sentence segmentation algorithm. The section will describe the dependency of text segmentation algorithms on the language being processed and the character set in which the language is encoded. It will also discuss the dependency on the application that uses the output of the segmentation and the dependency on the characteristics of the specific corpus being processed.

In Section 3 we will introduce some common techniques currently used for tokenisation. The first part of the section will focus on issues that arise in tokenising languages in which words are separated by whitespace. The second part of the section will discuss tokenisation techniques in languages where no such whitespace word boundaries exist. In Section 4 we will discuss the problem of sentence segmentation and introduce some common techniques currently used to identify sentences boundaries in texts.

2. Why is text segmentation challenging?

There are many issues that arise in tokenisation and sentence segmentation which need to be addressed when designing NLP systems. In our discussion of tokenisation and sentence segmentation, we will emphasize the main types of dependencies that must be addressed in developing algorithms for text segmentation: **language dependence** (Section 2.1), **character-set dependence** (Section 2.2), **application dependence** (Section 2.3), and **corpus dependence** (Section 2.4). In Section 2.5, we will briefly discuss the evaluation of the text segmentation algorithms.

2.1 Language dependence

We focus here on written language rather than transcriptions of spoken language. This is an important distinction since it limits the discussion to those languages that have established writing systems. Since every human society uses language, there are thousands of distinct languages and dialects; yet only a small minority of the languages and dialects currently have a system of visual symbols to represent the elements of the language.

Just as the spoken languages of the world contain a multitude of different features, the written adaptations of languages have a diverse set of features. Writing systems can be **logographic**, where a large number (often thousands) of individual symbols represent words. In contrast, writing systems can be **syllabic**, in which individual symbols represent syllables, or **alphabetic**, in which individual symbols (more or less) represent sounds; unlike logographic systems, syllabic and alphabetic systems typically have less than 100 symbols. In practice, no modern writing system employs symbols of only one kind, so no natural language writing system can be classified as purely logographic, syllabic, or alphabetic. Even English, with its relatively simple writing system based on the Roman alphabet, utilizes logographic symbols including Arabic numerals (0 - 9), currency symbols (\$, £), and other symbols (% , & , #). English is nevertheless predominately alphabetic, and most other writing systems are comprised of symbols which are mainly of one type.

In addition to the variety of symbol types used in writing systems, there is a range of orthographic conventions used in written languages to denote the boundaries between linguistic units such as syllables, words or sentences. In many written Amharic texts, for example, both word and sentence boundaries are explicitly marked, while in written Thai texts neither is marked. In the latter case where no boundaries are explicitly indicated in the written language, written Thai is similar to spoken language, where there are no explicit boundaries and few cues to indicate segments at any level. Between the two extremes are languages that mark boundaries to different degrees. English employs

whitespace between most words and punctuation marks at sentence boundaries, but neither feature is sufficient to segment the text completely and unambiguously. Tibetan and Korean both explicitly mark syllable boundaries, either through layout or by punctuation, but neither marks word boundaries. Written Chinese and Japanese have adopted punctuation marks for sentence boundaries, but neither denotes word boundaries. For a very thorough description of the various writing systems employed to represent natural languages, including detailed examples of all languages and features discussed in this chapter, we recommend (Daniels and Bright, 1996).

The wide range of writing systems used by the languages of the world result in language-specific as well as orthography-specific features that must be taken into account for successful text segmentation. The first essential step in text segmentation is thus to understand the writing system for the language being processed. In this chapter we will provide general techniques applicable to a variety of different writing systems. Since many segmentation issues are language-specific, we will also highlight the challenges faced by robust, broad-coverage tokenisation efforts.

2.2 Character-set dependence

At its lowest level, a computer-based text or document is merely a sequence of digital bits stored in a computer's memory. The first essential task is to interpret these bits as characters of a writing system of a natural language. Historically, this was trivial, since nearly all texts were encoded in the 7-bit ASCII character set, which allowed only 128 characters and included only the roman alphabet and essential characters for writing English. This limitation required the “asciification” or “romanization” of many texts, in which ASCII equivalents were defined for characters not defined in the character set. An example of this asciification is the adaptation of many European languages containing umlauts and accents, in which the umlauts are replaced by a double quotation mark or the letter ‘e’ and accents are denoted by a single quotation mark or even a number code. In this system, the German word *über* would be written as *u"ber* or *ueber*, and the French word *déjà* would be written *de'ja* or *de1ja2*. Languages less similar to English, such as Russian and Hebrew, required much more elaborate romanization systems, in order to allow ASCII processing. In fact, such adaptations are still common since many current email systems are limited to this 7-bit encoding.

Extended character sets have been defined to allow 8-bit encodings, but an 8-bit computer byte can still encode just 256 distinct characters, and there are tens of thousands of distinct characters in all the writing systems of the world. For this reason, characters in different languages are currently encoded in a large number of overlapping character sets. Most alphabetic and syllabic writing systems can be encoded in a single byte, but larger character sets, such as those of written Chinese and Japanese which have several thousand distinct characters, require a two-byte system where a single character is represented by a pair of 8-bit bytes. It is further complicated by the fact that multiple encodings currently exist for the same character set; for example, the Chinese character set is encoded in two widely-used variants: GB and Big-5.¹ The fact that the same range of numeric values represents different characters in different encodings can be a problem for tokenisation, because tokenisers are usually targeted to a specific language in a specific encoding. For example, a tokeniser for a text in English or Spanish, which are normally stored in the common encoding Latin-1 (or ISO 8859-1), would need to be

¹ The Unicode standard (Consortium, 1996) specifies a single 2-byte encoding system that includes 38,885 distinct coded characters derived from 25 supported scripts. However, until Unicode becomes universally accepted and implemented, the many different character sets in use will continue to be a problem.

aware that bytes in the (decimal) range 161-191 in Latin-1 represent punctuation marks and other symbols (such as 'ı', 'ç', '£', and '©'); tokenisation rules would be required to handle each symbol (and thus its byte code) appropriately for that language. However, the same byte range 161-191 in the common Thai alphabet encoding TIS620 corresponds to a set of Thai consonants, which would naturally be treated differently from punctuation or other symbols. A completely different set of tokenisation rules would be necessary to successfully tokenise a Thai text in TIS620 due to the use of overlapping character ranges.²

Determining characters in two-byte encodings involves locating pairs of bytes representing a single character. This process can be complicated by the tokenisation equivalent of code-switching, in which characters from many different writing systems occur within the same text. It is very common in texts to encounter multiple writing systems and thus multiple encodings. In Chinese and Japanese newswire texts, which are usually encoded in a two-byte system, one byte (usually Latin-1) letters, spaces, punctuation marks (e.g., periods, quotation marks, and parentheses), and Arabic numerals are commonly interspersed with the Chinese and Japanese characters. Such texts also frequently contain Latin-1 SGML headers.

2.3 Application dependence

Although word and sentence segmentation are necessary, in reality, there is no absolute definition for what constitutes a word or a sentence. Both are relatively arbitrary distinctions that vary greatly across written languages. However, for the purposes of computational linguistics we need to define exactly what we need for further processing; in most cases, the language and task at hand determine the necessary conventions. For example, the English words *I am* are frequently contracted to *I'm*, and a tokeniser frequently expands the contraction to recover the essential grammatical features of the pronoun and verb. A tokeniser that does not expand this contraction to the component words would pass the single token *I'm* to later processing stages. Unless these processors, which may include morphological analysers, part-of-speech taggers, lexical lookup routines, or parsers, were aware of both the contracted and uncontracted forms, the token may be treated as an unknown word.

Another example of the dependence of tokenisation output on later processing stages is the treatment of the English possessive *'s* in various tagged corpora.³ In the Brown corpus (Francis and Kucera, 1982), the word *governor's* is considered one token and is tagged as a possessive noun. In the Susanne corpus (Sampson, 1995), on the other hand, the same word is treated as two tokens, *governor* and *'s*, tagged singular noun and possessive, respectively.

Because of this essential dependence, the tasks of word and sentence segmentation overlap with the techniques discussed in other chapters in this Handbook: *Lexical Analysis* in Chapter 3, *Parsing Techniques* in Chapter 4, and *Semantic Analysis* in Chapter 5, as well as the practical applications discussed in later chapters.

2.4 Corpus dependence

Until recently, the problem of robustness was rarely addressed by NLP systems, which normally could process only well-formed input conforming to their hand-built grammars. The increasing availability of large corpora in multiple languages that encompass a wide

² Actually, due to the nature of written Thai, there are more serious issues than character set encoding which prevent the use of an English tokeniser on Thai texts, and we will discuss these issues in Section 3.2.

³ This example is taken from (Grefenstette and Tapanainen, 1994).

range of data types (e.g. newswire texts, email messages, closed captioning data, optical character recognition (OCR) data, multimedia documents) has required the development of robust NLP approaches, as these corpora frequently contain misspellings, erratic punctuation and spacing, and other irregular features. It has become increasingly clear that algorithms which rely on input texts to be well-formed are much less successful on these different types of texts.

Similarly, algorithms that expect a corpus to follow a set of conventions for a written language are frequently not robust enough to handle a variety of corpora. It is notoriously difficult to prescribe rules governing the use of a written language; it is even more difficult to get people to “follow the rules.” This is in large part due to the nature of written language, in which the conventions are determined by publishing houses and national academies and are arbitrarily subject to change.⁴ So while punctuation roughly corresponds to the use of suprasegmental features in spoken language, reliance on well-formed sentences delimited by predictable punctuation can be very problematic. In many corpora, traditional prescriptive rules are commonly ignored. This fact is particularly important to our discussion of both word and sentence segmentation, which to a large degree depend on the regularity of spacing and punctuation. Most existing segmentation algorithms for natural languages are both language-specific and corpus-dependent, developed to handle the predictable ambiguities in a well-formed text. Depending on the origin and purpose of a text, capitalization and punctuation rules may be followed very closely (as in most works of literature), erratically (as in various newspaper texts), or not at all (as in email messages or closed captioning). For example, an area of current research is summarization, filtering, and sorting techniques for email messages and usenet news articles. Corpora containing email and usenet articles can be ill-formed, such as Example (1), an actual posting to a usenet newsgroup, which shows the erratic use of capitalization and punctuation, “creative” spelling, and domain-specific terminology inherent in such texts.

(1) *ive just loaded pcl onto my akcl. when i do an 'in- package' to load pcl, ill get the prompt but im not able to use functions like defclass, etc... is there womething basic im missing or am i just left hanging, twisting in the breeze?*

Many online corpora, such as those from OCR or handwriting recognition, contain substitutions, insertions, and deletions of characters and words, which affect both tokenisation and sentence segmentation. Example (2) shows a line taken from a corpus of OCR data in which large portions of several sentences (including the punctuation) were clearly elided by the OCR algorithm. In this extreme case, even accurate sentence segmentation would not allow for full processing of the partial sentences that remain.

(2) *newsprint. Furthermore, shoe presses have Using rock for granite roll Two years ago we reported on advances in*

Robust text segmentation algorithms designed for use with such corpora must therefore have the capability to handle the range of irregularities which distinguish these texts from well-formed newswire documents frequently processed.

2.5 Evaluation of text segmentation algorithms

Because of the dependencies discussed above, evaluation and comparison of text segmentation algorithms is very difficult. Due to the variety of corpora for which the algorithms

⁴ This is evidenced by the numerous spelling “reforms” which have taken place in various languages, most recently in the German language, or the attempts by governments (such as the French) to “purify” the language or to outlaw foreign words.

are developed, an algorithm that performs very well on a specific corpus may not be successful on another corpus. Certainly, an algorithm fine-tuned for a particular language will most likely be completely inadequate for processing another language. Nevertheless, evaluating algorithms provides important information about their efficacy, and there are common measures of performance for both tokenisation and sentence segmentation.

The performance of word segmentation algorithms is usually measured using *recall* and *precision*, where recall is defined as the percent of words in the manually segmented text identified by the segmentation algorithm, and precision is defined as the percentage of words returned by the algorithm that also occurred in the hand segmented text in the same position. For a language like English, where a great deal of the initial tokenisation can be done by relying on whitespace, tokenisation is rarely scored. However, for unsegmented languages (see Section 3.2), evaluation of word segmentation is critical for improving all aspects of NLP systems.

A sentence segmentation algorithm’s performance is usually reported in terms of a single score equal to the number of punctuation marks correctly classified divided by the total number of punctuation marks. It is also possible to evaluate sentence segmentation algorithms using recall and precision, given the numbers of **false positives**, punctuation marks erroneously labeled as a sentence boundary, and **false negatives**, actual sentence boundaries not labeled as such.

3. Tokenisation

Section 2 discussed the many challenges inherent in segmenting freely occurring text. In this section we will focus on the specific technical issues that arise in tokenisation.

Tokenisation is well-established and well-understood for artificial languages such as programming languages.⁵ However, such artificial languages can be strictly defined to eliminate lexical and structural ambiguities; we do not have this luxury with natural languages, in which the same character can serve many different purposes and in which the syntax is not strictly defined. Many factors can affect the difficulty of tokenising a particular natural language. One fundamental difference exists between tokenisation approaches for **space-delimited** languages and approaches for **unsegmented** languages. In space-delimited languages, such as most European languages, some word boundaries are indicated by the insertion of whitespace. The character sequences delimited are not necessarily the tokens required for further processing, however, so there are still many issues to resolve in tokenisation. In unsegmented languages, such as Chinese and Thai, words are written in succession with no indication of word boundaries. Tokenisation of unsegmented languages therefore requires additional lexical and morphological information.

In both unsegmented and space-delimited languages, the specific challenges posed by tokenisation are largely dependent on both the writing system (logographic, syllabic, or alphabetic, as discussed in Section 2.1) and the typographical structure of the words. There are three main categories into which word structures can be placed,⁶ and each category exists in both unsegmented and space-delimited writing systems. The morphology of words in a language can be **isolating**, where words do not divide into smaller units; **agglutinating** (or **agglutinative**), where words divide into smaller units (morphemes) with clear boundaries between the morphemes; or **inflectional**, where the boundaries between morphemes are not clear and where the component morphemes can express

⁵ For a thorough introduction to the basic techniques of tokenisation in programming languages, see (Aho, Sethi, and Ullman, 1986).

⁶ This classification comes from (Comrie, Matthews, and Polinsky, 1996) and (Crystal, 1987).

more than one grammatical meaning. While individual languages show tendencies toward one specific type (e.g., Mandarin Chinese is predominantly isolating, Japanese is strongly agglutinative, and Latin is largely inflectional), most languages exhibit traces of all three.⁷

Since the techniques used in tokenising space-delimited languages are very different from those used in tokenising unsegmented languages, we will discuss the techniques separately, in Sections 3.1 and 3.2, respectively.

3.1 Tokenisation in space-delimited languages

In many alphabetic writing systems, including those that use the Latin alphabet, words are separated by whitespace. Yet even in a well-formed corpus of sentences, there are many issues to resolve in tokenisation. Most tokenisation ambiguity exists among uses of punctuation marks, such as periods, commas, quotation marks, apostrophes, and hyphens, since the same punctuation mark can serve many different functions in a single sentence, let alone a single text. Consider example sentence (3) from the *Wall Street Journal* (1988).

(3) *Clairson International Corp. said it expects to report a net loss for its second quarter ended March 26 and doesn't expect to meet analysts' profit estimates of \$3.9 to \$4 million, or 76 cents a share to 79 cents a share, for its year ending Sept. 24.*

This sentence has several items of interest that are common for Latinate, alphabetic, space-delimited languages. First, it uses periods in three different ways - within numbers as a decimal point (*\$3.9*), to mark abbreviations (*Corp.* and *Sept.*), and to mark the end of the sentence, in which case the period following the number *24* is not a decimal point. The sentence uses apostrophes in two ways - to mark the genitive case (where the apostrophe denotes possession) in *analysts'* and to show contractions (places where letters have been left out of words) in *doesn't*. The tokeniser must thus be aware of the uses of punctuation marks and be able to determine when a punctuation mark is part of another token and when it is a separate token.

In addition to resolving these cases, we must make tokenisation decisions about a phrase such as *76 cents a share*, which on the surface consists of four tokens. However, when used adjectivally such as in the phrase *a 76-cents-a-share dividend*, it is normally hyphenated and appears as one. The semantic content is the same despite the orthographic differences, so it makes sense to treat the two identically, as the same number of tokens. Similarly, we must decide whether to treat the phrase *\$3.9 to \$4 million* differently than if it had been written *3.9 to 4 million dollars* or *\$3,900,000 to \$4,000,000*. We will discuss these ambiguities and other issues in the following sections.

A logical initial tokenisation of a space-delimited language would be to consider as a separate token any sequence of characters preceded and followed by space. This successfully tokenises words that are a sequence of alphabetic characters, but does not take into account punctuation characters. In many cases, characters such as commas, semicolons, and periods should be treated as separate tokens, although they are not preceded by whitespace (such as the case with the comma after *\$4 million* in Example (3)). Additionally, many texts contain certain classes of character sequences which should be filtered out before actual tokenisation; these include existing markup and headers (including SGML markup), extra whitespace, and extraneous control characters.

⁷ A fourth typological classification frequently studied by linguists, **polysynthetic**, can be considered an extreme case of agglutinative, where several morphemes are put together to form complex words which can function as a whole sentence. Chukchi and Inuit are examples of polysynthetic languages.

3.1.1 Tokenising punctuation. While punctuation characters are usually treated as separate tokens, there are many cases when they should be “attached” to another token. The specific cases vary from one language to the next, and the specific treatment of the punctuation characters need to be enumerated within the tokeniser for each language. In this section we will give examples of English tokenisation.

Abbreviations are used in written language to denote the shortened form of a word. In many cases abbreviations are written as a sequence of characters terminated with a period. For this reason, recognizing abbreviations is essential for both tokenisation and sentence segmentation, since abbreviations can occur at the end of a sentence, in which case the period serves two purposes. Compiling a list of abbreviations can help in recognizing them, but abbreviations are productive, and it is not possible to compile an exhaustive list of all abbreviations in any language. Additionally, many abbreviations can also occur as words elsewhere in a text (e.g., the word *Mass* is also the abbreviation for *Massachusetts*). An abbreviation can also represent several different words, as is the case for *St.* which can stand for *Saint*, *Street*, or *State*. However, as *Saint* it is less likely to occur at a sentence boundary than as *Street*, or *State*. Examples (4) and (5) from the *Wall Street Journal* (1991 and 1987 respectively) demonstrate the difficulties produced by such ambiguous cases, where the same abbreviation can represent different words and can occur both within and at the end of a sentence.

(4) *The contemporary viewer may simply ogle the vast wooded vistas rising up from the Saguenay River and Lac St. Jean, standing in for the St. Lawrence River.*

(5) *The firm said it plans to sublease its current headquarters at 55 Water St. A spokesman declined to elaborate.*

Recognizing an abbreviation is thus not sufficient for complete tokenisation, and we will discuss abbreviations at sentence boundaries fully in Section 4.2.

Quotation marks and apostrophes (“ ” ‘ ’) are a major source of tokenisation ambiguity. In most cases, single and double quotes indicate a quoted passage, and the extent of the tokenisation decision is to determine whether they open or close the passage. In many character sets, single quote and apostrophe are the same character, and it is therefore not always possible to immediately determine if the single quotation mark closes a quoted passage, or serves another purpose as an apostrophe. In addition, as discussed in Section 2.2, quotation marks are also commonly used when “romanizing” writing systems, in which umlauts are replaced by a double quotation mark and accents are denoted by a single quotation mark or apostrophe.

The apostrophe is a very ambiguous character. In English, the main uses of apostrophes are to mark the genitive form of a noun, to mark contractions, and to mark certain plural forms. In the genitive case, some applications require a separate token while some require a single token, as discussed in Section 2.3. How to treat the genitive case is important, since in other languages, the possessive form of a word is not marked with an apostrophe and cannot be as readily recognized. In German, for example, the possessive form of a noun is usually formed by adding the letter *s* to the word, without an apostrophe, as in *Peters Kopf* (*Peter’s head*). However, in modern (informal) usage in German, *Peter’s Kopf* would also be common; the apostrophe is also frequently omitted in modern (informal) English such that *Peters head* is a possible construction. Furthermore, in English, *’s* also serves as a contraction for the verb *is*, as in *he’s*, *it’s*, and *she’s*, as well as the plural form of some words, such as *I.D.’s* or *1980’s* (although the apostrophe is also frequently omitted from such plurals). The tokenisation decision in these cases is context dependent and is closely tied to syntactic analysis.

In the case of apostrophe as contraction, tokenisation may require the expansion of the word to eliminate the apostrophe, but the cases where this is necessary are very language-dependent. The English contraction *I'm* could be tokenised as the two words *I am*, and *we've* could become *we have*. And depending on the application, *wouldn't* probably would expand to *would not*, although it has been argued by Zwicky and Pullum (1981) that the ending *n't* is an inflectional marker rather than a clitic. Written French contains a completely different set of contractions, including contracted articles (*l'homme*, *c'était*), as well as contracted pronouns (*j'ai*, *je l'ai*) and other forms such as *n'y*, *qu'ils*, *d'ailleurs*, and *aujourd'hui*. Clearly, recognizing the contractions to expand requires knowledge of the language, and the specific contractions to expand, as well as the expanded forms, must be enumerated. All other word-internal apostrophes are treated as a part of the token and not expanded, which allows the proper tokenisation of multiply-contracted words such as *fo'c's'le* and *Pudd'n'head* as single words. In addition, since contractions are not always demarcated with apostrophes, as in the French *du*, which is a contraction of *de la*, or the Spanish *del*, contraction of *de el*, other words to expand must also be listed in the tokeniser.

3.1.2 Multi-part words. To different degrees, many written languages contain space-delimited words composed of multiple units, each expressing a particular grammatical meaning. For example, the single Turkish word *çöplüklerimizdekilerdenmiydi* means “was it from those that were in our garbage cans?”.⁸ This type of construction is particularly common in strongly agglutinative languages such as Swahili, Quechua, and most Altaic languages. It is also common in languages such as German, where noun-noun (*Lebensversicherung*, life insurance), adverb-noun (*Nichtraucher*, non-smoker), and preposition-noun (*Nachkriegszeit*, post-war period) compounding are all possible. In fact, though it is not an agglutinative language, German compounding can be quite complex, as in *Feuerundlebensversicherung* (fire and life insurance) or *Kundenzufriedenheitsabfragen* (customer satisfaction survey).

To some extent, agglutinating constructions are present in nearly all languages, though this compounding can be marked by hyphenation, in which the use of hyphens can create a single word with multiple grammatical parts. In English it is commonly used to create single-token words like *end-of-line* as well as multi-token words like *Boston-based*. As with the apostrophe, the use of the hyphen is not uniform; for example, hyphen usage varies greatly between British and American English, as well as between different languages. However, as with the case of apostrophes as contractions, many common language-specific uses of hyphens can be enumerated in the tokeniser.

Many languages use the hyphen to create essential grammatical structures. In French, for example, hyphenated compounds such as *va-t-il*, *c'est-à-dire* and *celui-ci* need to be expanded during tokenisation, in order to recover necessary grammatical features of the sentence. In these cases, the tokeniser needs to contain an enumerated list of structures to be expanded, as with the contractions discussed above.

Another tokenisation difficulty involving hyphens stems from the practice, common in traditional typesetting, of using hyphens at the ends of lines to break a word too long to include on one line. Such end-of-line hyphens can thus occur within words that are not normally hyphenated. Removing these hyphens is necessary during tokenisation, yet it is difficult to distinguish between such incidental hyphenation and cases where naturally-hyphenated words happen to occur at a line break. In an attempt to dehyphenate the artificial cases, it is possible to incorrectly remove necessary hyphens. Grefenstette and

⁸ This example is from (Hankamer, 1986).

Tapanainen (1994) found that nearly 5% of the end-of-line hyphens in an English corpus were word-internal hyphens which happened to also occur as end-of-line hyphens.

In tokenising multi-part words, such as hyphenated or agglutinative words, whitespace does not provide much useful information to further processing stages. In such cases, the problem of tokenisation is very closely related both to tokenisation in unsegmented languages, discussed in Section 3.2 of this chapter, and to lexical analysis, discussed in Chapter 3 of this Handbook.

3.1.3 Multi-word expressions. Spacing conventions in written languages do not always correspond to the desired tokenisation. For example, the three-word English expression *in spite of* is, for all intents and purposes, equivalent to the single word *despite*, and both could be treated as a single token. Similarly, many common English expressions, such as *au pair*, *de facto*, and *joie de vivre*, consist of foreign loan words that can be treated as a single token.

Multi-word numerical expressions are also commonly identified in the tokenisation stage. Numbers are ubiquitous in all types of texts in every language. For most applications, sequences of digits and certain types of numerical expressions, such as dates and times, money expressions, and percents, can be treated as a single token. Several examples of such phrases can be seen in Example (3) above: *March 26*, *\$3.9 to \$4 million*, and *Sept. 24* could each be treated as a single token. Similarly, phrases such as *76 cents a share* and *\$3-a-share* convey roughly the same meaning, despite the difference in hyphenation, and the tokeniser should normalize the two phrases to the same number of tokens (either one or four). Tokenising numeric expressions requires knowledge of the syntax of such expressions, since numerical expressions are written differently in different languages. Even within a language or in languages as similar as English and French, major differences exist in the syntax of numeric expressions, in addition to the obvious vocabulary differences. For example, the English date *November 18, 1989* could alternately appear in English texts as any number of variations, such as *Nov. 18, 1989* or *18 November 1989* or *11/18/89*.

Closely related to hyphenation, treatment of multi-word expressions is highly language-dependent and application-dependent, but can easily be handled in the tokenisation stage if necessary. We need to be careful, however, when combining words into a single token. The phrase *no one*, along with *noone* and *no-one*, is a commonly encountered English equivalent for *nobody*, and should normally be treated as a single token. However, in a context such as *No one man can do it alone*, it needs to be treated as two words. The same is true of the two-word phrase *can not*, which is not always equivalent to the single word *cannot* or the contraction *can't*.⁹ In such cases, it is safer to allow a later process (such as a parser) to make the decision.

3.1.4 A flex tokeniser. Tokenisation in space-delimited languages can usually be accomplished with a standard lexical analyzer, such as *lex* (Lesk and Schmidt, 1975) or *flex* (Nicol, 1993), or the UNIX tools *awk*, *sed*, or *perl*. Figure 1 shows an example of a basic English tokeniser written in *flex*, a lexical analyzer with a simple syntax that allows the user to write rules in a regular grammar. This flex example contains implementations of several issues discussed above.

The first part of the tokeniser in Figure 1 contains a series of character category definitions. For example, `NUMBER_CHARACTER` contains characters which may occur within a number, `NUMBER_PREFIX` contains characters which may occur at the

⁹ For example, consider the following sentence: “Why is my soda can not where I left it?”

beginning of a numeric expression, `DIACRITIC_LETTER` contains letters with diacritics (such as ñ,) which are common in many European languages and may occur in some English words. Note that some category definitions contain octal byte codes (such as in `NUMBER_PREFIX`, where `\243` and `\245` represent the currency symbols for English pounds and Japanese yen) as discussed in Section 2.2.

The character categories defined in the first part of the tokeniser are combined in the second part in a series of rules explicitly defining the tokens to be output. The first rule, for example, states that if the tokeniser encounters an apostrophe followed by the letters 'v' and 'e', it should expand the contraction and output the word "have". The next rule states that any other sequence of letters following an apostrophe should (with the apostrophe) be tokenised separately and not expanded. Other rules in Figure 1 show examples of tokenising multi-word expressions (*au pair*), dates (*Jan. 23rd*), numeric expressions (*\$4,347.12*), punctuation characters, abbreviations, and hyphenated words. Note that this basic tokeniser is by no means a complete tokeniser for English, but rather is intended to show some simple examples of flex implementations of the issues discussed in previous sections.

3.2 Tokenisation in unsegmented languages

The nature of the tokenisation task in unsegmented languages like Chinese, Japanese, and Thai is essentially different from tokenisation in space-delimited languages like English. The lack of any spaces between words necessitates a more informed approach than simple lexical analysis, and tools like flex are not as successful. However, the specific approach to word segmentation for a particular unsegmented language is further limited by the writing system and orthography of the language, and a single general approach has not been developed. In Section 3.2.1, we will describe some general algorithms which have been applied to the problem to obtain an initial approximation for a variety of languages. In Section 3.2.2, we will give details of some successful approaches to Chinese segmentation, and in Section 3.2.3, we will describe some approaches which have been applied to other languages.

3.2.1 Common approaches. An extensive word list combined with an informed segmentation algorithm can help to achieve a certain degree of accuracy in word segmentation, but the greatest barrier to accurate word segmentation is in recognizing unknown words, words not in the lexicon of the segmenter. This problem is dependent both on the source of the lexicon as well as the correspondence (in vocabulary) between the text in question and the lexicon. Wu and Fung (1994) reported that segmentation accuracy is significantly higher when the lexicon is constructed using the same type of corpus as the corpus on which it is tested.

Another obstacle to high-accuracy word segmentation is the fact that there are no widely-accepted guidelines as to what constitutes a word, and there is therefore no agreement on how to "correctly" segment a text in an unsegmented language. Native speakers of a language do not always agree about the "correct" segmentation, and the same text could be segmented into several very different (and equally correct) sets of words by different native speakers. A simple example from English would be the hyphenated phrase *Boston-based*. If asked to "segment" this phrase into words, some native English speakers might say *Boston-based* is a single word and some might say *Boston* and *based* are two separate words; in this latter case there might also be disagreement about whether the hyphen "belongs" to one of the two words (and to which one) or whether it is a "word" by itself. Disagreement by native speakers of Chinese is much more prevalent; in fact, Sproat et. al. (1996) give empirical results showing that native speakers of Chinese frequently agree on the correct segmentation in less than 70% of the cases. Such ambiguity

```

DIGIT                [0-9]
NUMBER_CHARACTER     [0-9\.\,]
NUMBER_PREFIX        [\$\243\245]
NUMBER_SUFFIX        (\%|\242|th|st|rd)
ROMAN_LETTER         [a-zA-Z]
DIACRITIC_LETTER     [\300-\377]
INTERNAL_CHAR        [\-\\/]
WORD_CHAR            ({ROMAN_LETTER}|{DIACRITIC_LETTER})
ABBREVIATION         (mr|dr)
WHITESPACE           [\040\t\n]
APOSTROPHE           [\']
UNDEFINED            [\200-\237]
PERIOD               [\.]
SINGLE_CHARACTER      [\.,\,;\!\"?\":\\"251]
MONTH                (jan(uary)?|feb(ruary)?)

%%
{APOSTROPHE}(ve)      { printf("have\n", yytext); } ;
{APOSTROPHE}{ROMAN_LETTER}+ { printf("%s\n", yytext); } ;
(au){WHITESPACE}+(pair) { printf("au pair\n", yytext); } ;

{MONTH}{PERIOD}*{WHITESPACE}+{DIGIT}+{NUMBER_SUFFIX}* |
{MONTH}{PERIOD}*{WHITESPACE}+{DIGIT}+[\,]{WHITESPACE}+{DIGIT}+ |
{NUMBER_PREFIX}*{NUMBER_CHARACTER}+{NUMBER_SUFFIX}* |
{SINGLE_CHARACTER} | {ABBREVIATION}{PERIOD} |
{WORD_CHAR}+({INTERNAL_CHAR}{WORD_CHAR}+)* { printf("%s\n",yytext); } ;

{WHITESPACE}+|{UNDEFINED}+ ;

%%
yywrap() {
    printf("\n");
    return(1);
}

```

Figure 1

An (incomplete) basic flex tokeniser for English.

in the definition of what constitutes a word makes it difficult to evaluate segmentation algorithms which follow different conventions, since it is nearly impossible to construct a “gold standard” against which to directly compare results.

A simple word segmentation algorithm is to consider each character a distinct word. This is practical for Chinese because the average word length is very short (usually between one and two characters, depending on the corpus¹⁰) and actual words can be recognized with this algorithm. Although it does not assist in tasks such as parsing, part-of-speech tagging, or text-to-speech systems (see (Sproat et al., 1996)), the character-

¹⁰ As many as 95% of Chinese words consist of one or two characters, according to Fung and Wu (1994).

as-word segmentation algorithm has been used to obtain good performance in Chinese information retrieval (Buckley, Singhal, and Mitra, 1996), a task in which the words in a text play a major role in indexing.

A very common approach to word segmentation is to use a variation of the *maximum matching algorithm*, frequently referred to as the *greedy algorithm*. The greedy algorithm starts at the first character in a text and, using a word list for the language being segmented, attempts to find the longest word in the list starting with that character. If a word is found, the maximum-matching algorithm marks a boundary at the end of the longest word, then begins the same longest match search starting at the character following the match. If no match is found in the word list, the greedy algorithm simply segments that character as a word (as in the character-as-word algorithm above) and begins the search starting at the next character. A variation of the greedy algorithm segments a sequence of unmatched characters as a single word; this variant is more likely to be successful in writing systems with longer average word lengths. In this manner, an initial segmentation can be obtained that is more informed than a simple character-as-word approach. The success of this algorithm is largely dependent on the word list.

As a demonstration of the application of the character-as-word and greedy algorithms, consider an example of “desegmented” English, in which all the white space has been removed: the desegmented version of the phrase *the table down there* would thus be *thetabledownthere*. Applying the character-as-word algorithm would result in the useless sequence of tokens *t h e t a b l e d o w n t h e r e*, which is why this algorithm only makes sense for languages such as Chinese. Applying the greedy algorithm with a “perfect” word list containing all known English words would first identify the word *theta*, since that is the longest sequence of letters starting at the initial *t* which forms an actual word. Starting at the *b* following *theta*, the algorithm would then identify *bled* as the maximum match. Continuing in this manner, *thetabledownthere* would be segmented by the greedy algorithm as *theta bled own there*.

A variant of the maximum matching algorithm is the *reverse maximum matching algorithm*, in which the matching proceeds from the end of the string of characters, rather than the beginning. In the example above, *thetabledownthere* would be correctly segmented as *the table down there* by the reverse maximum matching algorithm. Greedy matching from the beginning and the end of the string of characters enables an algorithm such as *forward-backward matching*, in which the results are compared and the segmentation optimized based on the two results. In addition to simple greedy matching, it is possible to encode language-specific heuristics to refine the matching as it progress.

3.2.2 Chinese segmentation. The Chinese writing system is frequently described as logographic, though it is not entirely logographic since each character (known as *Hanzi*) does not always represent a single word. It has also been classified as morphosyllabic (DeFrancis, 1984), in that each Hanzi represents both a single lexical (and semantic) morpheme as well as a single phonological syllable. Regardless of its classification, the Chinese writing system has been the focus of a great deal of computational linguistics research in recent years.

Most previous work¹¹ in Chinese segmentation falls into one of three categories: statistical approaches, lexical rule-based approaches, and hybrid approaches that use both statistical and lexical information. Statistical approaches use data such as the mutual information between characters, compiled from a training corpus, to determine which

¹¹ A detailed treatment of Chinese word segmentation is beyond the scope of this chapter. Much of our summary is taken from (Sproat et al., 1996). For a comprehensive recent summary of work in Chinese segmentation, we also recommend (Wu and Tseng, 1993).

characters are most likely to form words. Lexical approaches use manually-encoded features about the language, such as syntactic and semantic information, common phrasal structures, and morphological rules, in order to refine the segmentation. The hybrid approaches combine information from both statistical and lexical sources. Sproat et al. (1996) describe such an approach which uses a weighted finite-state transducer to identify both dictionary entries as well as unknown words derived by productive lexical processes. Palmer (1997) also describes a hybrid statistical-lexical approach in which the segmentation is incrementally improved by a trainable sequence of transformation rules.

3.2.3 Other segmentation algorithms. According to Comrie, Matthews, and Polinsky (1996), the majority of all written languages use an alphabetic or syllabic system. Common unsegmented alphabetic and syllabic languages are Thai, Balinese, Javanese, and Khmer. While such writing systems have fewer characters, they also have longer words. Localized optimization is thus not as practical as in Chinese segmentation. The richer morphology of such languages often allows initial segmentations based on lists of words, names, and affixes, usually using some variation of the maximum matching algorithm. Successful high-accuracy segmentation requires a thorough knowledge of the lexical and morphological features of the language. Very little research has been published on this type of word segmentation, but a recent discussion can be found in (Kawtrakul et al., 1996), which describes a robust Thai segmenter and morphological analyzer.

Languages like Japanese and Korean have writing systems that incorporate alphabetic, syllabic and logographic symbols. Modern Japanese texts, for example, frequently consist of many different writing systems: Kanji (Chinese symbols), hiragana (a syllabary for grammatical markers and for words of Japanese origin), katakana (a syllabary for words of foreign origin), romanji (words written in the Roman alphabet), Arabic numerals, and various punctuation symbols. In some ways, the multiple character sets make tokenisation easier, as transitions between character sets give valuable information about word boundaries. However, character set transitions are not enough, since a single word may contain characters from multiple character sets, such as inflected verbs, which can contain a Kanji base and katakana inflectional ending. Company names also frequently contain a mix of Kanji and romanji. To some extent, Japanese can be segmented using the same techniques developed for Chinese. For example, Nagata (1994) describes an algorithm for Japanese segmentation similar to that used for Chinese segmentation by (Sproat et al., 1996).

4. Sentence Segmentation

Sentences in most written languages are delimited by punctuation marks, yet the specific usage rules for punctuation are not always coherently defined. Even when a strict set of rules exists, the adherence to the rules can vary dramatically based on the origin of the text source and the type of text. Additionally, in different languages, sentences and sub-sentences are frequently delimited by different punctuation marks. Successful sentence segmentation for a given language thus requires an understanding of the various uses of punctuation characters in that language. In most languages, the problem of sentence segmentation reduces to disambiguating all instances of punctuation characters that may delimit sentences. The scope of this problem varies greatly by language, as does the number of different punctuation marks that need to be considered.

Written languages that do not use many punctuation marks present a very difficult challenge in recognizing sentence boundaries. Thai, for one, does not use a period (or any other punctuation mark) to mark sentence boundaries. A space is sometimes used at sentence breaks, but very often the space is indistinguishable from the carriage return,

or there is no separation between sentences. Spaces are sometimes also used to separate phrases or clauses, where commas would be used in English, but this is also unreliable. In cases such as written Thai where punctuation gives no reliable information about sentence boundaries, locating sentence boundaries is best treated as a special class of locating word boundaries.

Even languages with relatively rich punctuation systems like English present surprising problems. Recognizing boundaries in such a written language involves determining the roles of all punctuation marks which can denote sentence boundaries: periods, question marks, exclamation points, and sometimes semicolons, colons, dashes, and commas. In large document collections, each of these punctuation marks can serve several different purposes in addition to marking sentence boundaries. A period, for example, can denote a decimal point or a thousands marker, an abbreviation, the end of a sentence, or even an abbreviation at the end of a sentence. Ellipsis (a series of periods (...)), can occur both within sentences and at sentence boundaries. Exclamation points and question marks can occur at the end of a sentence, but also within quotation marks or parentheses (really!) or even (albeit infrequently) within a word, such as in the band name *Therapy?* and the language name *!Xǃ*. However, conventions for the use of these two punctuation marks also vary by language; in Spanish, both can be unambiguously recognized as sentence delimiters by the presence of '¡' or '¿' at the start of the sentence. In this section we will introduce the challenges posed by the range of corpora available and the variety of techniques that have been successfully applied to this problem and discuss their advantages and disadvantages.

4.1 Sentence boundary punctuation

Just as the definition of what constitutes a sentence is rather arbitrary, the use of certain punctuation marks to separate sentences depends largely on an author's adherence to changeable and frequently ignored conventions. In most NLP applications, the only sentence boundary punctuation marks considered are the period, question mark, and exclamation point, and the definition of what constitutes a sentence is limited to the *text-sentence*, as defined by (Nunberg, 1990). However, grammatical sentences can be delimited by many other punctuation marks, and restricting sentence boundary punctuation to these three can cause an application to overlook many meaningful sentences or unnecessarily complicate processing by allowing only longer, complex sentences. Consider Examples (6) and (7), two English sentences that convey exactly the same meaning; yet, by the traditional definitions, the first would be classified as two sentences, the second as just one. The semicolon in Example (7) could likewise be replaced by a comma or a dash, retain the same meaning, but still be considered a single sentence. Replacing the semicolon with a colon is also possible, though the resulting meaning would be slightly different.

(6) *Here is a sentence. Here is another.*

(7) *Here is a sentence; here is another.*

The distinction is particularly important for an application like part-of-speech tagging. Many taggers, such as the one described in (Cutting et al., 1991), seek to optimize a tag sequence for a sentence, with the locations of sentence boundaries being provided to the tagger at the outset. The optimal sequence will usually be different depending on the definition of sentence boundary and how the tagger treats "sentence-internal" punctuation.

For an even more striking example of the problem of restricting sentence boundary punctuation, consider Example (8), from Lewis Carroll's *Alice in Wonderland*, in which *!/?* are completely inadequate for segmenting the meaningful units of the passage:

(8) *There was nothing so VERY remarkable in that; nor did Alice think it so VERY much out of the way to hear the Rabbit say to itself, ‘Oh dear! Oh dear! I shall be late!’ (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually TOOK A WATCH OUT OF ITS WAISTCOAT-POCKET, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.*

This example contains a single period at the end and three exclamation points within a quoted passage. However, if the semicolon and comma were allowed to end sentences, the example could be decomposed into as many as ten grammatical sentences. This decomposition could greatly assist in nearly all NLP tasks, since long sentences are more likely to produce (and compound) errors of analysis. For example, parsers consistently have difficulty with sentences longer than 15-25 words, and it is highly unlikely that any parser could ever successfully analyze this example in its entirety.

In addition to determining which punctuation marks delimit sentences, the sentence in parentheses as well as the quoted sentences *‘Oh dear! Oh dear! I shall be late!’* suggest the possibility of a further decomposition of the sentence boundary problem into types of sentence boundaries, one of which would be “embedded sentence boundary.” Treating embedded sentences and their punctuation differently could assist in the processing of the entire text-sentence. Of course, multiple levels of embedding would be possible, as in Example (9), taken from (Adams, 1972). In this example, the main sentence contains an embedded sentence (delimited by dashes), and this embedded sentence also contains an embedded quoted sentence.

(9) *The holes certainly were rough - “Just right for a lot of vagabonds like us,” said Bigwig - but the exhausted and those who wander in strange country are not particular about their quarters.*

It should be clear from these examples that true sentence segmentation, including treatment of embedded sentences, can only be achieved through an approach which integrates segmentation with parsing. Unfortunately, there has been little research in integrating the two; in fact, little research in computational linguistics has focused on the role of punctuation in written language.¹² With the availability of a wide range of corpora and the resulting need for robust approaches to Natural Language Processing, the problem of sentence segmentation has recently received a lot of attention. Unfortunately, nearly all published research in this area has focused on the problem of sentence boundary detection in English, and all this work has focused exclusively on disambiguating the occurrences of period, exclamation point, and question mark. A promising development in this area is the recent focus on trainable approaches to sentence segmentation, which we will discuss in Section 4.4. These new methods, which can be adapted to different languages and different text genres, should make a tighter coupling of sentence segmentation and parsing possible. While the remainder of this chapter will focus on published work that deals with the segmentation of a text into text-sentences, the above discussion of sentence punctuation indicates the application of trainable techniques to broader problems may be possible.

¹² A notable exception is (Nunberg, 1990).

4.2 The importance of context

In any attempt to disambiguate the various uses of punctuation marks, whether in text-sentences or embedded sentences, some amount of the context in which the punctuation occurs is essential. In many cases, the essential context can be limited to the character immediately following the punctuation mark. When analyzing well-formed English documents, for example, it is tempting to believe that sentence boundary detection is simply a matter of finding a period followed by one or more spaces followed by a word beginning with a capital letter, perhaps also with quotation marks before or after the space. A single rule¹³ to represent this pattern would be:

```
IF (right context = period + space + capital letter
    OR period + quote + space + capital letter
    OR period + space + quote + capital letter)
THEN sentence boundary
```

Indeed, in some corpora (e.g. literary texts) this single pattern accounts for almost all sentence boundaries. In *The Call of the Wild* by Jack London, for example, which has 1640 periods as sentence boundaries, this single rule will correctly identify 1608 boundaries (98%). However, the results are different in journalistic texts such as the *Wall Street Journal* (*WSJ*). In a small corpus of the *WSJ* from 1989 which has 16466 periods as sentence boundaries, this simple rule would detect only 14562 (88.4%) while producing 2900 **false positives**, placing a boundary where one does not exist.

Most of the errors resulting from this simple rule are cases where the period occurs immediately after an abbreviation. Expanding the context to consider the word preceding the period is thus a logical step. An improved rule would be:

```
IF ((right context = period + space + capital letter
    OR period + quote + space + capital letter
    OR period + space + quote + capital letter)
    AND (left context != abbreviation))
THEN sentence boundary
```

This can produce mixed results, since the use of abbreviations in a text depends on the particular text and text genre. The new rule improves performance on *The Call of the Wild* to 98.4% by eliminating 5 false positives (previously introduced by the phrase “St. Bernard” within a sentence). On the *WSJ* corpus, this new rule also eliminates all but 283 of the false positives introduced by the first rule. However, this rule also introduces 713 **false negatives**, erasing boundaries where they were previously correctly placed, yet still improving the overall score. Recognizing an abbreviation is therefore not sufficient to disambiguate the period, because we also must determine if the abbreviation occurs at the end of a sentence.

The difficulty of disambiguating abbreviation-periods can vary depending on the corpus. Liberman and Church (1992) report that 47% of the periods in a *Wall Street Journal* corpus denote abbreviations, compared to only 10% in the Brown corpus (Francis and Kucera, 1982), as reported by Riley (1989). In contrast, Müller (1980) reports abbreviation-period statistics ranging from 54.7% to 92.8% within a corpus of English scientific abstracts. Such a range of figures suggests the need for a more informed treatment of the context that considers more than just the word preceding or following the punctuation mark. In difficult cases, such as an abbreviation which can occur at the end of a sentence, three or more words preceding and following must be considered. This

¹³ Parts of this discussion originally appeared in (Bayer et al., 1998).

is the case in the following examples of “garden path sentence boundaries”, the first consisting of a single sentence, the other of two sentences.

(10) *Two high-ranking positions were filled Friday by Penn St. University President Graham Spanier.*

(11) *Two high-ranking positions were filled Friday at Penn St. University President Graham Spanier announced the appointments.*

Many contextual factors have been shown to assist sentence segmentation in difficult cases. These contextual factors include:

- **case distinctions** - In languages and corpora where both upper-case and lower-case letters are consistently used, whether a word is capitalized provides information about sentence boundaries.
- **part of speech** - Palmer and Hearst (1997) showed that the parts of speech of the words within three tokens of the punctuation mark can assist in sentence segmentation. Their results indicate that even an estimate of the *possible* parts of speech can produce good results.
- **word length** - Riley (1989) used the length of the words before and after a period as one contextual feature.
- **lexical endings** - Müller, Amerl, and Natalis (1980) used morphological analysis to recognize suffixes and thereby filter out words which weren't likely to be abbreviations. The analysis made it possible to identify words that were not otherwise present in the extensive word lists used to identify abbreviations.
- **prefixes and suffixes** - Reynar and Ratnaparkhi (1997) used both prefixes and suffixes of the words surrounding the punctuation mark as one contextual feature.
- **abbreviation classes** - Riley (1989) and Reynar and Ratnaparkhi (1997) further divided abbreviations into categories such as titles (which are not likely to occur at a sentence boundary) and corporate designators (which are more likely to occur at a boundary).

4.3 Traditional rule-based approaches

The success of the few simple rules described in the previous section is a major reason sentence segmentation has been frequently overlooked or idealized away. In well-behaved corpora, simple rules relying on regular punctuation, spacing, and capitalization can be quickly written, and are usually quite successful. Traditionally, the method widely used for determining sentence boundaries is a regular grammar, usually with limited lookahead. More elaborate implementations include extensive word lists and exception lists to attempt to recognize abbreviations and proper nouns. Such systems are usually developed specifically for a text corpus in a single language and rely on special language-specific word lists; as a result they are not portable to other natural languages without repeating the effort of compiling extensive lists and rewriting rules. Although the regular grammar approach can be successful, it requires a large manual effort to compile the individual rules used to recognize the sentence boundaries. Nevertheless, since rule-based sentence segmentation algorithms can be very successful when an application does deal with well-behaved corpora, we provide a description of these techniques.

An example of a very successful regular-expression-based sentence segmentation algorithm is the text segmentation stage of the Alembic information extraction system (Aberdeen et al., 1995), which was created using the lexical scanner generator flex (Nicol, 1993). The Alembic system uses flex in a preprocess pipeline to perform tokenisation and sentence segmentation at the same time. Various modules in the pipeline attempt to classify all instances of punctuation marks by identifying periods in numbers, date and time expressions, and abbreviations. The preprocess utilizes a list of 75 abbreviations and a series of over 100 hand-crafted rules and was developed over the course of more than 6 staff months. The Alembic system alone achieved a very high accuracy rate (99.1%) on a large *Wall Street Journal* corpus. However, the performance was improved when integrated with the trainable system Satz, described in (Palmer and Hearst, 1997) and summarized later in this chapter. In this hybrid system, the rule-based Alembic system was used to disambiguate the relatively unambiguous cases, while Satz was used to disambiguate difficult cases such as the five abbreviations *Co.*, *Corp.*, *Ltd.*, *Inc.*, and *U.S.*, which frequently occur in English texts both within sentences and at sentence boundaries. The hybrid system achieved an accuracy of 99.5%, higher than either of the two component systems alone.

4.4 Robustness and Trainability

Throughout this chapter we have emphasized the need for robustness in NLP systems, and sentence segmentation is no exception. The traditional rule-based systems, which rely on features such as spacing and capitalization, will not be as successful when processing texts where these features are not present, such as in Example (1) above. Similarly, some important kinds of text consist solely of upper-case letters; closed captioning (CC) data is an example of such a corpus. In addition to being upper-case-only, CC data also has erratic spelling and punctuation, as can be seen from the following example of CC data from CNN:

(12) *THIS IS A DESPERATE ATTEMPT BY THE REPUBLICANS TO
SPIN THEIR STORY THAT NOTHING SEAR WHYOUS – SERIOUS HAS
BEEN DONE AND TRY TO SAVE THE SPEAKER’S SPEAKERSHIP AND
THIS HAS BEEN A SERIOUS PROBLEM FOR THE SPEAKER, HE DID
NOT TELL THE TRUTH TO THE COMMITTEE, NUMBER ONE.*

The limitations of manually-crafted rule-based approaches suggest the need for trainable approaches to sentence segmentation, in order to allow for variations between languages, applications, and genres. Trainable methods provide a means for addressing the problem of embedded sentence boundaries discussed earlier, as well as the capability of processing a range of corpora and the problems they present, such as erratic spacing, spelling errors, single-case, OCR errors.

For each punctuation mark to be disambiguated, a typical trainable sentence segmentation algorithm will automatically encode the context using some or all of the features described above. A set of training data, in which the sentence boundaries have been manually labeled, is then used to train a machine learning algorithm to recognize the salient features in the context. As we describe below, machine learning algorithms which have been used in trainable sentence segmentation systems have included neural networks, decision trees, and maximum entropy calculation.

4.5 Trainable algorithms

One of the first published works describing a trainable sentence segmentation algorithm was (Riley, 1989). The method described used regression trees (Breiman et al., 1984) to classify periods according to contextual features describing the single word preceding

and following the period. These contextual features included word length, punctuation after the period, abbreviation class, case of the word, and the probability of the word occurring at beginning or end of a sentence. Riley’s method was trained using 25 million words from the AP newswire, and he reported an accuracy of 99.8% when tested on the Brown corpus.

Reynar and Ratnaparkhi (1997) described a trainable approach to identifying English sentence boundaries (.!?) which used a statistical maximum entropy model. The system used a system of contextual templates which encoded one word of context preceding and following the punctuation mark, using such features as prefixes, suffixes, and abbreviation class. They also reported success in inducing an abbreviation list from the training data for use in the disambiguation. The algorithm, trained in less than 30 minutes on 40,000 manually-annotated sentences, achieved a high accuracy rate (98%+) on the same test corpus used by Palmer and Hearst (1997), without requiring specific lexical information, word lists, or any domain-specific information. Though they only reported results on English, they indicated the ease of trainability should allow the algorithm to be used with other Roman-alphabet languages, given adequate training data.

Palmer and Hearst (1997) developed a sentence segmentation system called Satz, which used a machine learning algorithm to disambiguate all occurrences of periods, exclamation points, and question marks. The system defined a contextual feature array for three words preceding and three words following the punctuation mark; the feature array encoded the context as the parts of speech which can be attributed to each word in the context. Using the lexical feature arrays, both a neural network and a decision tree were trained to disambiguate the punctuation marks, and achieved a high accuracy rate (98-99%) on a large corpus from the *Wall Street Journal*. They also demonstrated the algorithm, which was trainable in as little as one minute and required less than one thousand sentences of training data, to be rapidly ported to new languages. They adapted the system to French and German, in each case achieving a very high accuracy. Additionally, they demonstrated the trainable method to be extremely robust, as it was able to successfully disambiguate single-case texts and OCR data.

Trainable sentence segmentation algorithms such as these are clearly a step in the right direction, toward enabling robust processing of a variety of texts and languages. Algorithms that offer rapid training while requiring small amounts of training data will allow systems to be retargeted in hours or minutes to new text genres and languages.

5. Conclusion

Until recently, the problem of text segmentation was overlooked or idealized away in most NLP systems; tokenisation and sentence segmentation were frequently dismissed as uninteresting “preprocessing” steps. This was possible because most systems were designed to process small texts in a single language. When processing texts in a single language with predictable orthographic conventions, it was possible to create and maintain hand-built algorithms to perform tokenisation and sentence segmentation. However, the recent explosion in availability of large unrestricted corpora in many different languages, and the resultant demand for tools to process such corpora, has forced researchers to examine the many challenges posed by processing unrestricted texts. The result has been a move toward developing robust algorithms which do not depend on the well-formedness of the texts being processed. Many of the hand-built techniques have been replaced by trainable corpus-based approaches which use machine learning to improve their performance.

While the move toward trainable robust segmentation systems has been very promising, there is still room for improvement of tokenisation and sentence segmentation algorithms. Since errors at the text segmentation stage directly affect all later processing

stages, it is essential to completely understand and address the issues involved in tokenisation and sentence segmentation and how they impact further processing. Many of these issues are language-dependent: the complexity of tokenisation and sentence segmentation and the specific implementation decisions depend largely on the language being processed and the characteristics of its writing system. For a corpus in a particular language, the corpus characteristics and the application requirements also affect the design and implementation of tokenisation and sentence segmentation algorithms. In most cases, since text segmentation is not the primary objective of NLP systems, it cannot be thought of as simply an independent “preprocessing” step, but rather must be tightly integrated with the design and implementation of all other stages of the system.

References

- Aberdeen, John, John Burger, David Day, Lynette Hirschman, Patricia Robinson, and Marc Vilain. 1995. MITRE: Description of the Alembic system used for MUC-6. In *The Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, Maryland, November 1995.
- Adams, Richard. 1972. *Watership Down*. Macmillan Publishing Company, Inc., New York.
- Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. 1986. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley Publishing Company, Reading, MA.
- Bayer, Samuel, John Aberdeen, John Burger, Lynette Hirschman, David Palmer, and Marc Vilain. 1998. Theoretical and computational linguistics: Toward a mutual understanding. in (Lawler and Dry, 1998).
- Breiman, Leo, Jerome H. Friedman, Richard Olshen, and Charles J. Stone. 1984. *Classification and regression trees*. Wadsworth International Group, Belmont, CA.
- Buckley, Chris, Amit Singhal, and Mandar Mitra. 1996. Using query zoning and correlation within SMART: TREC 5. in (Harman, 1996).
- Comrie, Bernard, Stephen Matthews, and Maria Polinsky. 1996. *The Atlas of Languages*. Quarto Inc., London.
- Consortium, Unicode. 1996. *The Unicode Standard, Version 2.0*. Addison-Wesley.
- Crystal, David. 1987. *The Cambridge Encyclopedia of Language*. The Cambridge University Press, Cambridge.
- Cutting, Doug, Julian Kupiec, Jan Pedersen, and Penelope Sibun. 1991. A practical part-of-speech tagger. In *The 3rd Conference on Applied Natural Language Processing*, Trento, Italy.

- Daniels, Peter T. and William Bright. 1996. *The World's Writing Systems*. Oxford University Press, Inc., New York.
- DeFrancis, John. 1984. *The Chinese Language*. The University of Hawaii Press, Honolulu.
- Francis, W. Nelson and Henry Kucera. 1982. *Frequency Analysis of English Usage*. Houghton Mifflin Co., New York.
- Fung, Pascale and Dekai Wu. 1994. Statistical augmentation of a chinese machine-readable dictionary. In *Proceedings of Second Workshop on Very Large Corpora (WVLC-94)*.
- Grefenstette, Gregory and Pasi Tapanainen. 1994. What is a word, What is a sentence? Problems of Tokenization. In *The 3rd International Conference on Computational Lexicography (COMPLEX 1994)*.
- Hankamer, Jorge. 1986. Finite state morphology and left to right phonology. In *Proceedings of the Fifth West Coast Conference on Formal Linguistics*.
- Harman, Donna K. 1996. Proceedings of the fifth text retrieval conference (TREC-5). Gaithersburg, MD, November 20-22.
- Kawtrakul, Asanee, Chalutip Thumkanon, Thitima Jamjanya, Parinee Muangyunnan, Kritsada Poolwan, and Yasuyoshi Inagaki. 1996. A gradual refinement model for a robust thai morphological analyzer. In *Proceedings of COLING96*, Copenhagen, Denmark.
- Lawler, John and Helen Aristar Dry. 1998. *Using Computers in Linguistics*. Routledge, London.
- Lesk, M. E. and E. Schmidt. 1975. Lex - a lexical analyzer generator. Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, N.J.
- Lieberman, Mark Y. and Kenneth W. Church. 1992. Text analysis and word pronunciation in text-to-speech synthesis. In Sadaoki Furui and Man Mohan Sondhi, editors, *Advances in Speech Signal Processing*. Marcel Dekker, Inc., pages 791–831.
- Müller, Hans, V. Amerl, and G. Natalis. 1980. Worterkennungsvorgang als Grundlage einer Universalmethode zur automatischen Segmentierung von Texten in Sätze. Ein Verfahren zur maschinellen Satzgrenzenbestimmung im Englischen. *Sprache und Datenverarbeitung*, 1.
- Nagata, Masaaki. 1994. A stochastic japanese morphological analyzer using a forward-dp backward a* n-best search algorithm. In *Proceedings of COLING94*.

- Nicol, G. T. 1993. *Flex - The Lexical Scanner Generator*. The Free Software Foundation, Cambridge, MA.
- Nunberg, Geoffrey. 1990. *The Linguistics of Punctuation*. C.S.L.I. Lecture Notes, Number 18. Center for the Study of Language and Information, Stanford, CA.
- Palmer, David D. 1997. A trainable rule-based algorithm for word segmentation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL97)*, Madrid.
- Palmer, David D. and Marti A. Hearst. 1997. Adaptive multilingual sentence boundary disambiguation. *Computational Linguistics*, 23(2):241–67.
- Reynar, Jeffrey C. and Adwait Ratnaparkhi. 1997. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth ACL Conference on Applied Natural Language Processing*, Washington, D.C.
- Riley, Michael D. 1989. Some applications of tree-based modelling to speech and language indexing. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 339–352. Morgan Kaufmann.
- Sampson, Geoffrey R. 1995. *English for the Computer*. Oxford University Press.
- Sproat, Richard W., Chilin Shih, William Gale, and Nancy Chang. 1996. A stochastic finite-state word-segmentation algorithm for chinese. *Computational Linguistics*, 22(3):377–404.
- Wu, Dekai and Pascale Fung. 1994. Improving chinese tokenization with linguistic filters on statistical lexical acquisition. In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing*, Stuttgart, Germany.
- Wu, Zimin and Gwyneth Tseng. 1993. Chinese text segmentation for text retrieval: Achievements and problems. *Journal of the American Society for Information Science*, 44(9):532–542.
- Zwicky, Arnold M. and Geoffrey K. Pullum. 1981. Cliticization vs. inflection: English n't. In *1981 Annual Meeting of the Linguistics Society of America*.