

Chapter 1

Tokenizing

1.1 Introduction

Most text processing applications like POS taggers, parsers, stemmers, key word extractors, search engines etc. operate on words and sentences. Texts in their raw form, however, are just sequences of characters without explicit information about word and sentence boundaries. Before any further processing can be done, a text needs to be segmented into words and sentences. This process is called *tokenization*. Tokenization divides the character sequence into sentences and the sentences into *tokens*. Not only words are considered as tokens, but also numbers, punctuation marks, parentheses and quotation marks.

There is a fundamental difference in the tokenization of alphabetic languages and ideographic languages like Chinese. Alphabetic languages usually separate words by blanks, and a tokenizer which simply replaces whitespace with word boundaries and cuts off punctuation marks, parentheses, and quotation marks at both ends of a word, is already quite accurate. The only major problem is the disambiguation of periods which are ambiguous between abbreviation periods and sentence markers. Ideographic languages, on the other hand, provide no comparable information about sentence boundaries which makes tokenization a much harder task. The tokenization of alphabetic and ideographic languages are actually two rather different tasks which require different methods.

The rest of this chapter is organized as follows. The next section gives an overview over the problems encountered in tokenization. The following sections focus on subproblems. Section 1.3 describes the preprocessing required to transform idiosyncratic text formats into a standard form on which the tokenizer can operate. Section 1.4 describes the restoration of words which have been split up at line boundaries during typesetting. Section 1.5 addresses the period disambiguation problem, and section 1.6 presents methods for Chinese tokenization. Finally, section 1.6.1 suggests further reading.

1.2 Tokenization Problems

In alphabetic languages, words are usually surrounded by whitespace and optionally preceded and followed by punctuation marks, parentheses, or quotes. A simple tokenization rule can therefore be stated as follows: Split the character sequence at whitespace positions and cut off punctuation marks, parentheses, and quotes at both ends of the fragments to obtain the sequence of tokens. This simple rule is quite accurate because whitespace and punctuation are fairly reliable indicators of word boundaries. Yet, there are some problems which need to be solved.

Periods First of all, not all periods are punctuation. They also serve as markers for abbreviations (like “etc.” or “U.S.A.”) and ordinal numbers (as in the German date expression “1. Oktober 2005”). Only periods acting as sentence markers should be cut off to form separate tokens. The distinction between abbreviations and sentence-final punctuation is often difficult. Abbreviation lists help to recognize frequent abbreviations, but many abbreviations are rare or created ad hoc and cannot be listed exhaustively. There are also abbreviations like “fig.” and “no.” which are truly ambiguous because they could also be regular words which are followed by a full stop. Therefore, the disambiguation of periods requires contextual information. If the following token is a lower-case word, the period probably belongs to an abbreviation. If the following word is a capitalized word like “The” which would normally written in lower-case, the period is probably a full stop, but it could also be part of an abbreviation in sentence-final position.

Other Punctuation The sentence markers “!” or “?” are mostly unambiguous with a few exceptions like the company name “Yahoo!”. More problematic are the symbols “.” and “;”. Semicolons often separate two sentences as in “Our business is beer; we don’t know much about things like racing and concerts.”, but they are also used in enumerations, in particular when the elements of the enumeration contain commas as in the following fragment from the Wall Street Journal: “Also mentioned: Cleveland; Akron, Ohio; Jerusalem; and Tacoma, Wash.” Similarly, colons sometimes separate two sentences (Mr. Atwood believes more weapons and funds for the military also are necessary: “If there is going to be a single source of instability, it will be the loyalty of the military.”), and sometimes they are rather sentence-internal punctuation (One more remains: the North-Poindexter hurdle.) Distinguishing the different uses of colons and semicolons is very hard without analyzing the whole sentence.

Ordinal Numbers In German and other languages, ordinal numbers are written with a trailing period after the number. So, “17th” is written “17.” These ordinal numbers pose the same problem as abbreviations: A number which is followed by a period is either an ordinal number, a cardinal number in sentence-final position, or an ordinal number in sentence-final position. The problem is actually harder than the disambiguation of potential abbreviations because the number itself provides little information, although

e.g. numbers between 1 and 12 are more likely to be ordinal numbers than numbers between 1900 and 2010 which often designate years. The disambiguation is impossible without contextual information. If a candidate for an ordinal number is preceded by a sentence boundary, for instance, it is probably indeed an ordinal number. A candidate number in the range 1900-2010 which is preceded by the word *April* is likely to be a cardinal number in sentence-final position.

Multi Word Expressions So far, it was implicitly assumed that tokens do not contain whitespace. However, for some applications it is advantageous to treat certain multi word expressions as single tokens. Such expressions are complex prepositions like “because of”, conjunctions like “so that”, and adverbs like “at all”, foreign phrases like “et cetera” and “en vogue”, date expressions like “Feb. 1, 2004”, time expressions like “3:30 pm”, and proper names like “Daimler Chrysler AG”. Recognizers for such expressions are often implemented by regular expression matching with tools like “Lex” or “Perl” which are available on most Unix systems. The following regular expression, for example, recognizes date expressions like “Dec. 31, 1999”.

```
(Jan—Feb—Mar—Apr—Jun—Jul—Aug—Sep—Oct—Nov—Dec)[.] [123]?[0-9], (19—20)[0-9][0-9]
```

The recognition of arbitrary multi word names like “Abdul Aziz bin Abdul Rahman Al Saud” is very difficult because they can neither be listed exhaustively nor be exactly described with regular expressions. The recognition and classification of proper names developed into a separate field, called “Named Entity Recognition”, which is outside of the scope of this chapter.

Clitics Multi word expressions combine several words into a single token. The opposite is often done with clitic expressions like “isn’t” and “we’ll” which are split into two tokens. English clitics, as well as German clitics (“Stimmt’s?” - Is it correct?) and French clitics (“Permettez-vous?” - Would you allow?) are easily identified with suffix matching if exceptions like “rendez-vous” in French are taken care of. French has also determiner clitics (“l’Europe” - Europe), which can be recognized in the same way. The recognition of pronominal clitics in Spanish (“garantizarles” - to guarantee them) and Italian (“applicarlo” - to apply it) is harder because there is no separator, and requires a morphological analysis.

Dehyphenation Another problem for tokenization arises when the text was broken into lines during typesetting. Words which were split in order to fit the length of a line to the width of the column need to be rejoined. This process is called *dehyphenation*. It is not trivial because hyphens serve different purposes. Three cases have to be distinguished when a line ends with a hyphen, e.g. with the string “pre-”:

1. A word like “preprocessing” was split into “pre-” and “processing”. The hyphen and the newline have to be deleted.

2. A word like “pre-processing” was split into “pre-” and “processing”. Only the newline symbol has to be deleted.
3. The original text contained a word sequence like “pre- and postprocessing”. Here, it is necessary to replace the newline symbol with a word boundary.

Missing Whitespace Sometimes there is no blank after a punctuation mark, resulting in strings like “*hours.The*” or “*however,that*”, which should be split up into three separate tokens. Given word frequency information from a corpus, it is possible to statistically disambiguate such strings. The following rule e.g. works quite well for the disambiguation of strings which contain periods:

“If a potential token is of the form $s.r$, and the overall frequency of s times the frequency of r in sentence-initial position divided by the size of the training corpus is larger than the frequency of $s.r$, then split up $s.r$ into three tokens.”

Similar rules can be used for other punctuation marks.

Sentence Boundary Detection The detection of word boundaries is one task in tokenization, the other is the identification of sentence boundaries. Again, it is possible to correctly annotate most sentence boundaries with a simple rule, namely by putting a sentence boundary marker after sentence-final punctuation like “.”, “?” and “!”. But, as mentioned before, periods are ambiguous. They can also be part of an abbreviation or an ordinal number. Even worse, they can be sentence markers and part of an abbreviation at the same time when an abbreviation happens to occur at the end of a sentence.

Sentence boundary detection is complicated by quoted sentences appearing inside of a matrix sentence. The following sentence from the British National Corpus is an example:

“Why not bite the bullet?” said a spokesman.

The insertion of a sentence marker before the word “said” would create the ungrammatical sentence “said a spokesman.” In this example, the lower-case word after the quotation indicates that the quoted sentence is part of a matrix clause. However, such a hint is not always available, as the following sentence from the British National Corpus shows:

“You still do that?” Abbey said.

Ideographic Languages In contrast to alphabetic languages, ideographic languages like Chinese, Korean or Japanese do not mark the word boundaries with blanks. Tokenization is therefore much more difficult. Most Chinese characters are words by themselves, but appear also in multi-character words. It is difficult to decide where a word ends and where the next word begins. All tokenization methods for ideographic languages use a dictionary, either explicitly or implicitly. A simple tokenizer can be

implemented by a longest-match strategy: the tokenizer determines the longest character sequence which starts at the current position and is listed in the dictionary. It prints the recognized token, moves the position pointer behind the token, and starts to scan the next word. This method works quite well because long words are more likely to be correct than short words, but it has two problems. (1) It is too greedy: Given a dictionary with the strings “AB”, “ABC”, “CDEE”, “D”, and “E”, it would always tokenize the string “ABCDEE” as “ABC/D/E/E” rather than “AB/CDEE”, although the latter is more likely because it contains fewer tokens. (2) Words which are not in the dictionary, cannot be recognized. Dealing with unknown words is actually the hardest problem in the tokenization of ideographic languages.

1.3 Preprocessing

Machine-readable texts are stored in a wide variety of idiosyncratic document formats, character sets and typing conventions, which cannot be directly processed by a general-purpose tokenizer. The first processing step is therefore the normalization of the text document to a standard format, namely to a sequence of characters from a standard character set like Unicode. The formatting information is lost in this step. If it needs to be preserved, because it is relevant for applications, it should be encoded as SGML or XML markup (see also chapter ?? of this volume).

Formatting information may be relevant for tokenization. Dehyphenation, for instance, is trivial if the original file format uses different encodings for hyphens which were inserted during line breaking, and for other hyphens. Formatting information can also be useful for sentence boundary detection. Headers, e.g., are usually not terminated by a sentence marker. Once the formatting information has been removed, it is hard to determine automatically where the header ends and where the first sentence of an article begins. Paragraph boundaries are useful for sentence boundary detection because sentences are unlikely to cross paragraph boundaries. Information about headers and paragraph boundaries should therefore be preserved.

1.4 Dehyphenation

If a text was broken into lines, and words have been split at the boundaries of the lines, it is necessary to restore the words in their original form. Given a line ending with some string “x” and a hyphen, and followed by a line starting with some string “y”, the tokenizer needs to disambiguate between three possible output strings, namely (1) the string “xy” (a “regular” word like “preprocessing”), (2) the string “x-y” (a hyphenated word like “pre-processing”), and (3) the string “x- y” (a truncated word and a regular word as in “pre- and postprocessing”).

The first alternative is the most likely one. The third alternative is often rare enough to be ignored without degrading the accuracy noticeably. Grefenstette and Tapanainen

(1994) report on an experiment in which they ran the BNC through the typesetting program “nroff”. 12 % of the lines of the formatted text ended in a letter plus hyphen. Simply joining these lines and deleting the hyphen (corresponding to output string (1) above) correctly restored 95.1 % of the original words.

Grefenstette and Tapanainen (1994) suggested that more informed decisions could be based on dictionary information. Mikheev (2003) implemented a dictionary-based method and reports a reduction of the error rate from 4.9 % to 0.9 %. He used the following disambiguation rule: If the unhyphenated word form “xy” appears in the dictionary, print “xy”. Otherwise, if both “x” and “y” are listed in the dictionary as separate words, print the hyphenated form “x-y”. Otherwise, the unhyphenated form “xy” is printed.

If no dictionary is available, a wordlist with frequencies from a tokenized corpus can be used instead. Such a “corpus dictionary” also contains frequent hyphenated word forms like “long-term” or “full-time”. Furthermore it provides frequency information. The word “makeup” e.g. occurs 175 times in the BNC, but the word “make-up” is much more frequent with 1267 occurrences. A corpus-derived dictionary can be used in combination with the following disambiguation rule: If the hyphenated word form “x-y” is more frequent than “xy”, print “x-y”, Otherwise, if the hyphenated word form “x-y” is less frequent than “xy”, print “xy”. Otherwise, if both “x” and “y” are listed in the dictionary as separate words, print “x-y”. Otherwise, print “xy”.

1.5 Period Disambiguation

The most difficult problem for the tokenization of alphabetic languages is the disambiguation of periods. Periods either indicate (1) the end of a sentence, or (2) an abbreviation, or (3) an ordinal number, or (4) an abbreviation at the end of a sentence, or (5) an ordinal number at the end of a sentence.

If the ambiguity is ignored and all periods are treated as sentence markers, about 93.2 % of the sentence boundaries are correctly recognized in the Brown corpus according to Grefenstette and Tapanainen (1994). If all periods which are not followed by whitespace (like the period in the numeric expression 1,234.56 or the first period of the abbreviation “Ph.D.”) are classified as not sentence-final, the accuracy raises to 93.8 %.

Disambiguation Heuristics Many abbreviations consist of a sequence of letter-period pairs like “U.S.A.” or “e.g.”, or of a capitalized letter followed by a sequence of consonants like “Mrs.”, “St.” or “Eds.”. Such character sequences are mostly abbreviations, although there are a few exceptions like “Ash.” which is rather a name followed by a period. If strings of this form are always classified as sentence-internal abbreviations, the accuracy of the sentence boundary detection is raised to 97.66 %.

Potential abbreviations are often disambiguated by the context. When a period is followed by punctuation such as “,” “?” “!” “:” or “;”, or by a lower-case word, it is unlikely to be a sentence marker and the tokenizer should treat the preceding

string as an abbreviation. Such unambiguous occurrences of abbreviations allow the extraction of abbreviation lists from corpora. The results are not perfect, however. Some German newspapers e.g. write certain organization name like “amnesty international” and “adidas” always in lower-case, even at the beginning of a sentence. The last word of the preceding sentence is then incorrectly identified as an abbreviation. In order to cope with this and similar problems, the abbreviation list needs to be filtered e.g. by deleting items which are listed in a dictionary without the period. Unfortunately, the filtering prevents the recognition of abbreviations like “fig.” and “no.” because “fig” and “no” are words. With automatically extracted abbreviation lists, Grefenstette and Tapanainen (1994) increased the accuracy to 98.35 %.

Capitalized words which follow a potential abbreviation also provide valuable information. When a period e.g. is followed by “Some”, it is unlikely to be an abbreviation because the capitalized word “Some” normally appears only at the beginning of a sentence. Using information about the frequency of lower-case word forms and capitalized word forms, the tokenizer is able to classify a period as sentence-final if it is followed by a word whose lower-case version is more frequent than its capitalized version. This heuristic fails when an abbreviation is followed by a proper name which is also a regular word as in “Lts. Black and Henley”. In order to correctly tokenize such sentences, “Black” needs to be recognized as a proper name. A detailed discussion of this problem is found in (Mikheev, 2002).

The heuristic for the extraction of abbreviations from corpora fails to find abbreviations which are typically followed by upper-case words or by numbers. Examples are title abbreviations like “Prof.” or “Mrs.” which are followed by names, and abbreviations like “fig.” or “sec.” which are usually followed by numbers. In order to extract such abbreviations, it is necessary to consider ambiguous occurrences of abbreviations, as well. If the string “Mr.” was followed 1000 times by an upper-case word and only three times by a lower-case word, it is probably an abbreviation unless it is a word which appears mostly at the end of a sentence. To exclude this case, it is useful to count how often the potential abbreviation is followed by a capitalized word like “The” which is normally written in lower-case. If the relative frequency of such words after the potential abbreviation is low, it is probably an abbreviation.

Many abbreviations end with complex consonant clusters like “qns” in “eqns.” which are not found in regular words. The endings therefore provide valuable information for the distinction between abbreviations and regular words. Müller et al. (1980) proposed a sentence boundary detection algorithm which is based on word suffix statistics. Suffix-based recognition of abbreviations is particularly useful in German with its complex morphology. Abbreviations of English street names like “Main St.” are easy to recognize by looking up the string “St.” in an abbreviation dictionary. This is much harder for German where the whole street name is usually a single word resulting in abbreviations like “Hauptstr.” or “Kerschensteinerstr.”. Suffix analysis is a way to deal with these complex abbreviations.

The different period disambiguation heuristics discussed so far need to be combined in order to achieve optimal accuracy. This can be done in several ways: One option is to

put the heuristics into rules of the form “If condition C is satisfied then disambiguate the period as y.” These rules are ordered according to their reliability and the first matching rule determines the result.

Classification Approaches Period disambiguation can also be treated as a classification problem. The contextual information available for disambiguation is turned into a feature vector, and a classifier is trained on manually disambiguated training data. The classifier learns to assign the correct class (abbreviation, full stop or abbreviation+full stop) based on the feature vector. Riley (1989) implemented a classifier with decision trees (Breiman et al., 1984; Quinlan, 1983). His features included (1) the probability that the word preceding the period occurs in sentence-final position, (2) the probability that the word following the period occurs in sentence-initial position, (3+4) the length of the preceding and the following word, (5+6) whether the preceding/following word is lower-case, upper-case, capitalized or a number, (7) whether the period is followed by punctuation (8) whether the preceding word with the period is an abbreviation and the type of the abbreviation. The system was trained on 25 million words from AP newswire and achieved 99.8 % accuracy on the Brown corpus.

Palmer and Hearst (1997) implemented period disambiguators based on neural networks as well as decision trees. The feature vectors included information about capitalization, punctuation and the part-of-speech distributions of the six words around the period. They report an accuracy of 99.0 % for Wall Street Journal Data. They also evaluated their systems on German and French data with error rates between 1.9 and 0.4 %.

Mikheev (2000) went one step further than Palmer and Hearst (1997) by integrating period disambiguation into part-of-speech tagging. In the input of the POS tagger, the periods were always treated as separate tokens, and the POS tagger disambiguated the periods by assigning one of three possible tags which indicated whether the period is (1) an abbreviation (2) a full stop or (3) an abbreviation and full stop at the same time. He reports an accuracy of 99.8 % on the Brown corpus and 99.69 % on the WSJ corpus for a hybrid system which also used unsupervised methods.

Reynar and Ratnaparkhi (1997) presented a Maximum-Entropy approach to sentence boundary detection. Their system uses only information which was extracted from the manually tokenized training corpus. They report 98.8 % accuracy on the Wall Street Journal corpus and 97.9 % on the Brown corpus. Mikheev (1998) describes a similar system with an accuracy of 99.3 % on WSJ data and 98.7 % on the Brown corpus.

Unsupervised Methods The disadvantage of classification-based approaches is the need for manually annotated training data whose creation is expensive and time-consuming. The accuracy of these systems is often impressive on held-out data from the corpus they were trained on, but usually degrades on other corpora. One reason is that other corpora often contain different abbreviations, such that abbreviation lists extracted from one corpus are less useful on other corpora. In order to obtain optimal results, classification-based systems need to be retrained for new text types. Unsupervised methods (Grefen-

stette and Tapanainen, 1994; Mikheev, 2002; Schmid, 2000) avoid this problem by extracting the required information from raw text. They can even be trained on the corpus which is to be tokenized (unless online processing is required). Their performance is similar to the performance of systems which are trained on annotated data. The system described in (Schmid, 2000), for instance, extracts information about likely abbreviations (such as “Calif.”), typical abbreviation suffixes (like “str.” in German), lower-case words appearing capitalized after sentence boundaries (like “Fortunately”), lower-case words appearing in lower-case after sentence boundaries, proper names, and about abbreviations occurring frequently before numbers (like “Oct.”, “p.”, “Fig.” or “No.”). The system also extracts statistical information for the disambiguation of potential ordinal numbers. The decision rule which combines the different sources of information is statistically motivated.

1.6 Tokenization of Ideographic Languages

The tokenization of texts from languages with ideographic writing systems (Chinese, Japanese, Korean) is more difficult because there are no blanks to indicate the word boundaries. A Chinese text consists of a sequence of characters which is only interrupted by punctuation. Most characters are words by themselves, but can also be part of multi-character words. A few characters are affixes appearing exclusively in multi-character words.

There are four major tokenization approaches for these languages: rule-based methods, statistical methods based on word n-grams, tagging approaches, and systems which integrate tokenization with POS tagging or parsing.

Rule-based systems (e.g. Ma and Chen (2003)) first identify potential words which are listed in a dictionary. Overlapping candidate words represent ambiguities which are resolved by rules. The disambiguation rules are developed by hand and make use of linguistic principles, statistical information, and heuristics. In a second stage, unknown words are recognized with another set of rules.

Statistical systems define tokenization as the task of finding the most probable word sequence $\hat{w}_1^k = \hat{w}_1 \hat{w}_2 \dots \hat{w}_k$ which yields the character sequence $c_1^m = c_1 c_2 \dots c_m$. In mathematical terms, this is stated as follows:

$$\hat{w}_1^k = \arg \max_{w_1^n : \text{yield}(w_1^n) = c_1^m} p(w_1^n)$$

The joint probability $p(w_1^n)$ is decomposed into a product of conditional probabilities:

$$p(w_1^n) = \prod_{i=1}^n p(w_i | w_1, \dots, w_{i-1})$$

The conditional probabilities are too specific to be estimated reliably from training corpora. Therefore the model is simplified by assuming that the k preceding words are

sufficient to predict the next word and that words further away provide no additional information. This assumption is a crude approximation, but it works quite well in practice. By setting $k = 1$, a *bigram model* (or first-order Markov model) is obtained:

$$p(w_1^n) = \prod_{i=1}^n p(w_i|w_{i-1}) \quad (1.1)$$

The parameters $p(w|w')$ are estimated from manually tokenized corpora by dividing the frequency of the word pair $w'w$ by the frequency of the first word w' . This is the so-called *maximum likelihood estimate* (MLE):

$$p(w|w') = \frac{f(w'w)}{f(w')}$$

Due to the well-known Zipfian distribution of most language events, the training corpora are never large enough to contain all possible word bigrams. The probability of a bigram which is not observed in the training data, is 0 according to the ML estimate. Therefore, the tokenizer will never return a word sequence containing an unseen bigram unless there is no other alternative. In order to avoid this behavior, a small amount of the probability mass is redistributed from the observed bigrams to the unseen bigrams in order to guarantee that any bigram has a non-zero probability and can be recognized by the tokenizer. This process is called “parameter smoothing”. (See Chen and Goodman (1996) for a comparison of different smoothing techniques.)

Given a bigram model, the most likely word sequence for a given character sequence is efficiently computed by the Viterbi algorithm (Viterbi, 1967; Manning and Schütze, 1999). The main problem for statistical tokenizers is the recognition of unknown words. The simple bigram approach is unable to recognize words which are not listed in the dictionary. Many unknown words are names. Chinese names consist of the family name with one or two characters and the given name with also one or two characters. The number of family names is small. It is possible to list them exhaustively. Names from Western countries are transliterated into a sequence of Chinese characters whose pronunciation is similar to that of the foreign name. Each character corresponds to one syllable and each syllable is usually consistently transliterated in the same way. Therefore there is a small set of characters which tend to be used again and again. The structure of location and organization names is less predictable, but they often end with a typical character.

The simple bigram tokenizer described above usually splits unknown words into smaller units. Many systems recognize unknown words by recombining these fragments in a second step, e.g. by means of support vector machines (Asahara et al., 2003). Others integrate the recognition of unknown words into the statistical model. Sun et al. (2002) and Wu et al. (2003) replace the bigram model by a Hidden Markov model with sub-models for the recognition of names. They annotate each word w_i with a tag t_i . Known words are tagged with the words themselves. Unknown words are either tagged as Chinese names (${}_i\text{CN}_i$), as transliterated names (${}_i\text{TN}_i$), as location names (${}_i\text{LN}_i$), or as

organization names (iON_i). Overall there are $N+4$ tags, if N is the number of words. The tokenizer computes the best tag sequence for the given character sequence:

$$\hat{t}_1^k = \arg \max_{t_1^n} p(t_1^n | c_1^m)$$

After applying Bayes' theorem, the following formula is obtained:

$$\hat{t}_1^k = \arg \max_{t_1^n} \frac{p(c_1^m | t_1^n) p(t_1^n)}{c_1^m}$$

The prior probability of the character sequence $p(c_1^m)$ is positive and independent of the tag sequence. Hence it has no influence on the ranking of the different tag sequences and can be ignored.

$$\hat{t}_1^k = \arg \max_{t_1^n} p(t_1^n) p(c_1^m | t_1^n)$$

Assuming that each tag depends only on the preceding tag and that the characters depend only on the words they are a part of, we get:

$$\hat{t}_1^k = \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | t_{i-1}) p(c_{s_i}^{e_i} | t_i) \quad (1.2)$$

Here, s_i is the start position of the word tagged with t_i , e_i is the end position, and $c_{s_i}^{e_i}$ is the character sequence of the word. If all tags are identical to the words (i.e. there are no unknown words), then equation 1.2 simplifies to equation 1.1.

In the model of Wu et al. (2003), the probabilities of the character sequences are defined as follows:

$$\begin{aligned} p(c_1^j | w) &= 1 \text{ if } w \text{ is a word and } yield(w) = c_1^j \\ p(c_1^4 | iCN_i) &= p_{fam}(c_1 c_2) p_{mid}(c_3) p_{end}(c_4) \\ p(c_1^3 | iCN_i) &= p_{fam}(c_1 c_2) p_{end}(c_3) + p_{fam}(c_1) p_{mid}(c_2) p_{end}(c_3) \\ p(c_1^2 | iCN_i) &= p_{fam}(c_1) p_{end}(c_3) \\ p(c_1^j | iTN_i) &= \prod_{i=1}^j p_T(c_i) \\ p(c_1^j | iLN_i) &= p_{EL}(c_j) \prod_{i=1}^{j-1} p_L(c_i) \\ p(c_1^j | iON_i) &= p_{EO}(c_j) \prod_{i=1}^{j-1} p_O(c_i) \end{aligned}$$

Any other character sequence probability is 0. p_{fam} is the probability distribution over family names with one or two characters. $p_{mid}(c)$ is the probability that character c is the first character of a two-character given name, and $p_{end}(c)$ is the probability that c

is the last character of a given name. $p_{EL}(c)$ ($p_{EO}(c)$) is the probability that c appears at the end of a location (organization) name, and $p_L(c)$ ($p_O(c)$) is the probability that c appears in non-final position of a location (organization) name. $p_T(c)$ is the probability of c in transliterated names.

The sub-models for the named entity recognition in (Sun et al., 2002) are more complex. They use a character-based trigram model for names, a word-based trigram model for locations, and a recursive trigram model for organization names which allows location names to be embedded into organization names.

Zhang et al. (2003) extended the tagset with three more tags for numeric expressions, dates, and strings of alphabetic characters.

Tagging-based tokenizers do not necessarily use a dictionary. They annotate each character with a tag which indicates whether the character starts a new word or not. (Note that the models discussed before tagged complete words rather than single characters.) From a statistical point of view, the task of the tagger is to find the most likely tag sequence \hat{t}_1^n for a given character sequence c_1^n :

$$\hat{t}_1^n = \arg \max_{t_1^n} p(t_1^n | c_1^n)$$

Assuming that a tag depends only on the current character and the preceding character, the simple word juncture model proposed in (Fu and Luke, 2003) is obtained:

$$\hat{t}_1^n = \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | c_{i-1}, c_i) \quad (1.3)$$

The tagset of Fu and Luke (2003) has only two elements indicating the presence or absence of a word boundary. Xue and Shen (2003), in contrast, use four different tags indicating whether the character is the first (LM), the last (MR), or an inner (MM) character of a multi-character word, or a single character word (LR). They decompose the probability of the tag sequence given the word sequence into a product of the probabilities of each tag given the k preceding tags and characters, the current character and the k following characters:

$$\begin{aligned} \hat{t}_1^n &= \arg \max_{t_1^n} p(t_1^n | c_1^n) \\ &= \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | t_1^{i-1} c_1^n) \\ &\approx \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | t_{i-k}^{i-1} c_{i-k}^{i+k}) \end{aligned}$$

The conditional probabilities are estimated with a maximum-entropy model. The system is very similar to the Maximum-Entropy part-of-speech tagger of Ratnaparkhi (1996).

Tagging-based tokenizers are not necessarily statistical systems. Goh et al. (2004) use support vector machines (Vapnik, 1995) to annotate characters with the tags B (beginning of word), E (end of word), I (word internal), and S (single character word). As

features they use the surrounding characters, the word boundaries assigned to the surrounding characters by a simple forward-matching longest-match segmentation method as well as a backward-matching segmentation method, and the tags assigned to the preceding characters by the SVM itself.

All tokenizers for ideographic languages presented so far, consider segmentation as a separate processing step which is carried out prior to any syntactic analysis. However, it is also possible to integrate tokenization into syntactic analysis. The main advantage of this strategy is that higher-level information becomes available for word segmentation. Jiang et al. (2004) describe a system which integrates tokenization and part-of-speech (POS) tagging. Using a Hidden Markov Model, they compute the best sequence of words and POS tags as follows:

$$\begin{aligned} (\hat{t}_1^k, \hat{w}_1^k) &= \arg \max_{t_1^n, w_1^n: \text{yield}(t_1^n) = c_1^n} p(t_1^n, w_1^n) \\ &= \arg \max_{t_1^n, w_1^n: \text{yield}(t_1^n) = c_1^n} p(t_1^n) p(w_1^n | t_1^n) \\ &\approx \arg \max_{t_1^n, w_1^n: \text{yield}(t_1^n) = c_1^n} \prod_{i=1}^n p(t_i | t_{i-2}, t_{i-1}) p(w_i | t_i) \end{aligned}$$

This POS tagger and tokenizer is very similar to the standard HMM POS taggers for tokenized input. The main difference is that the start and end positions of the words are not fixed. Another peculiarity of the system is the modeling of speech rhythm. Chinese speakers seem to have a preference for words which have the same number of syllables as their neighbors. So, whenever there is a choice between two equivalent words, the speakers tend to prefer the word with the same number of syllables as the neighbor. The formula which they actually use for tagging includes the word length l_i which is assumed to depend only on the length of the preceding words:

$$(\hat{t}_1^k, \hat{l}_1^k, \hat{w}_1^k) = \arg \max_{t_1^n, w_1^n: \text{yield}(t_1^n) = c_1^n} \prod_{i=1}^n p(t_i | t_{i-2}, t_{i-1}) p(l_i | l_{i-2}, l_{i-1}) p(w_i | t_i, l_i)$$

Zhang et al. (2003) describe a cascaded Hidden Markov model where the integration of tokenization and POS tagging is less tight. The HMM used for word segmentation resembles the extended bigram model of Sun et al. (2002). Instead of fully disambiguating the word sequence they compute the n most likely word sequences and leave it to a HMM POS tagger to select the best word sequence.

Wu (2003) integrated tokenization with parsing. Similar to parsers which process the ambiguous output of speech recognizers, their system operates on word lattices which represent all possible word sequences. The word boundary ambiguities are eliminated as a side effect of syntactic disambiguation.

1.6.1 Further Reading

Grefenstette and Tapanainen (1994) provide a good overview over the problems en-

countered in tokenization. German tokenization problems are discussed at length in (Klatt, 2005). Riley (1989), Palmer and Hearst (1997), and Reynar and Ratnaparkhi (1997) describe period disambiguation methods which are trained on manually annotated text. Unsupervised period disambiguation methods have been presented in Grefenstette and Tapanainen (1994), Schmid (2000), and Mikheev (2002). The Proceedings of the SIGHAN Workshops on Chinese Language Processing are a good starting point to learn more about word segmentation methods for Chinese and other ideographic writing systems.

Bibliography

- Asahara, M., Goh, C.-L., Wang, X., and Matsumoto, Y. (2003). Combining segmenter and chunker for chinese word segmentation. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing (ACL 2003)*, pages 144–147, Sapporo, Japan.
- Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Pacific Grove CA.
- Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In Joshi, A. and Palmer, M., editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318, San Francisco. Morgan Kaufmann Publishers.
- Fu, G. and Luke, K.-K. (2003). A two-stage statistical word segmentation system for chinese. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing (ACL 2003)*, pages 156–159, Sapporo, Japan.
- Goh, C.-L., Asahara, M., and Matsumoto, Y. (2004). Chinese word segmentation by classification of characters. In *Proceedings of the Third SIGHAN Workshop on Chinese Language Processing (ACL 2004)*, pages 57–64, Barcelona, Spain.
- Grefenstette, G. and Tapanainen, P. (1994). What is a word, what is a sentence? problems of tokenization. In *Proceedings of the 3rd Conference on Computational Lexicography and Text Research (COMPLEX'94)*, Budapest.
- Jiang, F., Liu, H., Chen, Y., and Lu, R. (2004). An enhanced model for chinese word segmentation and part-of-speech tagging. In *Proceedings of the Third SIGHAN Workshop on Chinese Language Processing (ACL 2004)*, pages 28–32, Barcelona, Spain.
- Klatt, S. (2005). *Kombinierbare Textanalyseverfahren für die Korpusannotation und Informationsextraktion*. PhD thesis, Universität Stuttgart.
- Ma, W.-Y. and Chen, K.-J. (2003). Introduction to ckip chinese word segmentation system for the first international chinese word segmentation bakeoff. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing (ACL 2003)*, pages 168–171, Sapporo, Japan.

- Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. The MIT Press, Cambridge, Ma.
- Mikheev, A. (1998). Feature lattices for maximum entropy modelling. In *Proceedings of ACL-COLING'98*, pages 845–848, Montreal, Canada.
- Mikheev, A. (2000). Tagging sentence boundaries. In *Proceedings of the 1st Meeting of the North American Chapter of the Association of Computational Linguistics (NAACL 2000)*, pages 264–271, Seattle.
- Mikheev, A. (2002). Periods, capitalized words, etc. *Computational Linguistics*, 28(3):289–318.
- Mikheev, A. (2003). *The Oxford Handbook of Computational Linguistics*, chapter Text Segmentation, pages 201–218. Oxford Handbooks in Linguistics. Oxford University Press.
- Müller, H., V.Amerl, and Natalis, G. (1980). Worterkennungsverfahren als Grundlage einer Universalmethode zur automatischen Segmentierung von Texten in Sätze. Ein Verfahren zur maschinellen Satzgrenzenbestimmung im Englischen. *Sprache und Datenverarbeitung*, (1):46–63.
- Palmer, D. D. and Hearst, M. A. (1997). Adaptive multilingual sentence boundary disambiguation. *Computational Linguistics*, 23(2):241–269.
- Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An artificial intelligence approach*, pages 463–482. Morgan Kaufmann, San Mateo, CA.
- Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*. University of Pennsylvania.
- Reynar, J. C. and Ratnaparkhi, A. (1997). A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, D.C.
- Riley, M. 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.
- Schmid, H. (2000). *LoPar: Design and Implementation*. Number 149 in Arbeitspapiere des Sonderforschungsbereiches 340. Institute for Computational Linguistics, University of Stuttgart.
- Sun, J., Gao, J., Zhang, L., Zhou, M., and Huang, C. (2002). Chinese named entity identification using class-based language model. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 967–973, Taipei, Taiwan.

- Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer.
- Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269.
- Wu, A. (2003). Chinese word segmentation in MSR-NLP. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing (ACL 2003)*, pages 172–175, Sapporo, Japan.
- Wu, Y., Zhao, J., and Xu, B. (2003). Chinese named entity recognition combining a statistical model with human knowledge. In *Proceedings of the Workshop on Multilingual and Mixed-language Named Entity Recognition: Combining Statistical and Symbolic Models (ACL 2003)*, pages 65–72, Sapporo, Japan.
- Xue, N. and Shen, L. (2003). Chinese word segmentation as lmr tagging. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing (ACL 2003)*, pages 176–179, Sapporo, Japan.
- Zhang, H.-P., Liu, Q., Cheng, X.-Q., Zhang, H., and Yu, H.-K. (2003). Chinese lexical analysis using hierarchical hidden Markov model. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing (ACL 2003)*, pages 63–70, Sapporo, Japan.