3 Tokens and Terms

Tokens provide the link between queries and documents. During indexing, the IR system divides each document into a sequence of tokens and inserts these tokens into an inverted index for searching. At query time, a corresponding tokenization is applied to the query. The resulting query terms are then matched against the inverted index to effect retrieval and ranking.

Chapter 1 introduced simple rules for tokenizing raw text: Tokens are sequences of alphanumeric characters separated by nonalphanumeric characters. The tokens are *case normalized* by converting uppercase letters to lowercase. In addition — ignoring the complexities of full XML — we treat simple XML tags of the form <*name*> and </*name*> as tokens.

Although these simple rules are sufficient for the purposes of the experiments reported in this book, more complex tokenization rules may be required in practice. For example, under our simple rules, contractions such as "didn't" are represented by the pair of tokens "didn t", which would not match a search for the phrase "did not". The number "10,000,000" would be represented by the three tokens "10 000 000", the first of which would incorrectly match the query term "10".

Moreover, in applying these simple rules we explicitly assume that the raw text is written in the English language, perhaps encoded as ASCII values. Under these rules the definition of "alphanumeric" does not include the letter β or the number 八 because these characters cannot be represented in ASCII. Even the English word "naïve" cannot be properly represented using ASCII values.

In most practical environments, IR systems must provide appropriate support for languages other than English. Supporting a broad range of human languages requires us to take into account specific properties of the individual languages, including their character sets and rules for tokenization. Crudely speaking, tokenization splits documents into *words*. Although the concept of a word is essentially universal in human language (Trask, 2004), the characteristics of words vary greatly from language to language. Not every word makes a good token; not every token is a word.

In this chapter we revisit the problem of tokenization and term matching for English, and extend our discussion to other languages. We focus on the content of documents, leaving the complexities of XML and other document structure for Chapters 5 and 16. Relatively minor details of tokenization can have a substantial impact on retrieval effectiveness for specific queries. Getting it right requires careful consideration of these specific queries, as well as measuring overall retrieval effectiveness.

Section 3.1 covers English, including a discussion of two traditional IR topics closely associated with tokenization: *stemming* and *stopping*. Stemming provides a way for the query term "orienteering" to match an occurrence of "orienteers" by reducing both terms to their common root. Stopping reflects the observation that many common words, such as "the" and "I", may have little value for retrieval purposes. Ignoring these *stopwords* may improve retrieval efficiency and reduce the size of the inverted index.

Section 3.2 discusses character encodings, covering the basics of Unicode and UTF-8. Section 3.3 describes character n-gram indexing, which can provide a baseline for supporting new languages. Character n-gram indexing can also serve as a method for tokenizing noisy text, such as that produced by OCR (optical character recognition) systems. The final sections of the chapter discuss tokenization for a number of European and Asian languages, providing a sketch of the difficulties associated with them.

3.1 English

English provides a good starting point for understanding tokenization and term matching. The language is, by necessity, familiar to you as a reader of this book. It is one of the most widely spoken languages, with hundreds of millions of native speakers. Hundreds of millions more learn it as a second or third language. Content in the English language continues to comprise a disproportionate share of the Web. As a result, the ability to provide effective support for English is a fundamental requirement for many IR systems.

3.1.1 Punctuation and Capitalization

English contains a number of features that create minor tokenization problems. Many of them are related to punctuation and capitalization. In English a particular punctuation mark may be used for multiple unrelated purposes. The use of the apostrophe in contractions such as "I'll" and "it's" has already been mentioned, and it might be reasonable to tokenize these as "I will" and "it is" rather than "I ll" and "it s". However, the apostrophe is also used in constructions such as "o'clock" and "Bill's". The first might best be treated as a single token rather than two, because a match against "clock" may not be appropriate. On the other hand, if a match against "bill" is desirable, the second must be indexed as two tokens.

Periods end sentences. However, the same character appears in acronyms, numbers, initials, Internet addresses, and other contexts. Acronyms such as "I.B.M." and "U.S." may alternatively be written as "IBM" and "US". An IR system might recognize and tokenize such acronyms consistently, so that a query for "IBM" matches both forms. Punctuation marks may also appear as an integral part of the names of companies and organizations, such as "Yahoo!" and the band "Panic! At the Disco". For a small set of terms, the improper treatment of punctuation characters may render the search results completely useless. Examples include "C++", "C#", and "\index" (a LaTeX command). For the vast majority of queries, however, punctuation may safely be ignored.

Case normalization (i.e., converting all characters to lowercase in the index and in each query), on the other hand, affects almost every query. In many languages the first character of the word that starts a given sentence is always capitalized, and we want to be able to find those words even if the user types her query in lowercase (as most users do). Unfortunately, case normalization has the potential to cause great harm because capitalization is essential to preserve the meaning of many terms. Only capitalization distinguishes the acronym "US" from the pronoun "us", or the acronym "THE"[1] from the English definite article "the".

A simple heuristic to handle acronyms and similar tokenization problems is to *double index* these terms. For example, the acronym "US" would be indexed twice at each position at which it occurs. The inverted index would include postings lists for both "us" and "US". An occurrence of the acronym would appear in both lists, whereas an occurrence of the pronoun would appear only in the first list. A query containing the term "us" would be processed with the first list; a query containing the term "US" (and possibly "U.S.") would be processed with the second. Double indexing may also provide support for proper names, allowing the personal name "Bill" to be distinguished from an itemized invoice.

We must be cautious when adjusting our tokenization procedure to account for capitalization and punctuation. Adjustments to a tokenization procedure can negatively impact some queries while having a positive impact on others. Fans of the movie *The Terminator* (1984) will immediately recognize the quote "I'll be back", and a match against "I will be back" would not be appropriate. Occasionally, text is WRITTEN ENTIRELY IN CAPITAL LETTERS, perhaps accidentally or as a method for communicating the emotional state of the writer. Such text should not be treated as a long series of acronyms and abbreviations. The term "IN" in this sentence does not refer to the US state of Indiana, and this document does not discuss its capital. As with any information retrieval technique, tokenization must be evaluated with respect to its overall impact on effectiveness and efficiency.

3.1.2 Stemming

Stemming allows a query term such as "orienteering" to match an occurrence of "orienteers", or "runs" to match "running". Stemming considers the *morphology*, or internal structure, of terms, reducing each term to a *root form* for comparison. For example, "orienteering" and "orienteers" might reduce to the root form "orienteer"; "runs" and "running" might reduce to "run".

In an IR system a *stemmer* may be applied at both indexing time and query time. During indexing each token is passed through the stemmer and the resulting root form is indexed. At query time, the query terms are passed through the same stemmer and matched against the index terms. Thus the query term "runs" would match an occurrence of "running" by way of their common root form "run".

[1] The THE operating system, a pioneering multiprogramming system, was created at the Technische Universiteit Eindhoven by Turing award winner Edsger Dijkstra and colleagues in the mid-1960's.

Original
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to, 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
To sleep: perchance to dream: ay, there's the rub;

Normalized	**Normalized and stemmed**
to be or not to be that is the question	to be or not to be that is the question
whether tis nobler in the mind to suffer	whether ti nobler in the mind to suffer
the slings and arrows of outrageous fortune	the sling and arrow of outrag fortun
or to take arms against a sea of troubles	or to take arm against a sea of troubl
and by opposing end them to die to sleep	and by oppos end them to die to sleep
no more and by a sleep to say we end	no more and by a sleep to sai we end
the heart ache and the thousand natural shocks	the heart ach and the thousand natur shock
that flesh is heir to tis a consummation	that flesh is heir to ti a consumm
devoutly to be wish d to die to sleep	devoutli to be wish d to die to sleep
to sleep perchance to dream ay there s the rub	to sleep perchanc to dream ay there s the rub

Figure 3.1 The opening lines of Hamlet's soliloquy. At the top is the original text. The bottom-left version has been case normalized and stripped of punctuation. The bottom-right version is stemmed with the Porter stemmer.

Stemming is related to the concept of *lemmatization* from linguistics. Lemmatization reduces a term to a *lexeme*, which roughly corresponds to a word in the sense of a dictionary entry. For each lexeme a particular form of the word, or *lemma*, is chosen to represent it. The lemma is the form of a word you look up in a dictionary. For example, the form "run" is conventionally chosen to represent the group that includes "runs", "running", and "ran".

Stemming and lemmatization are sometimes equated, but this view is misleading. Stemming is strictly an operational process. When a stemmer transforms a term into its root form, we are not directly concerned with the linguistic validity of this transformation, but only with its measurable impact on retrieval effectiveness for specific queries.

The Porter stemmer, developed by Martin Porter in the late 1970s, is one of the best-known stemmers for the English language (Porter, 1980). Figure 3.1 shows the opening lines of Hamlet's famous soliloquy before and after application of the Porter stemmer. The text was stripped of punctuation and case normalized before applying the stemmer. As you can see in the figure, the stemmer may produce root forms that are not English words. For example, "troubles" stems down to "troubl". This behavior does not cause problems in practice because these root forms are never seen by a user. The term "troubling" in a query would also stem down to "troubl", providing a correct match.

Table 3.1 Impact of stemming. The table lists effectiveness measures for selected retrieval methods discussed in this book. The results in this table may be compared with those in Table 2.5 on page 72.

	TREC45				**GOV2**			
	1998		**1999**		**2004**		**2005**	
Method	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**
Proximity (2.2.2)	0.418	0.139	0.430	0.184	0.453	0.207	0.576	0.283
BM25 (Ch. 8)	0.440	0.199	0.464	0.247	0.500	0.266	0.600	0.334
LMD (Ch. 9)	0.464	0.204	0.434	0.262	0.492	0.270	0.600	0.343
DFR (Ch. 9)	0.448	0.204	0.458	0.253	0.471	0.252	0.584	0.319

The Porter stemmer may sometimes appear to be overly aggressive. Both “orienteering” and “orienteers” stem down to “orient” rather than “orienteer”. The term “oriental” also stems down to “orient”, incorrectly conflating these terms. Moreover, the stemmer does not handle irregular verbs and pluralizations correctly. Both “runs” and “running” stem down to “run”, but the irregular form “ran” stems (unchanged) to “ran”. Similarly, “mouse” stems to “mous”, but “mice” stems to “mice”. In addition, the stemmer handles only suffixes. Prefixes such as “un” and “re” are not removed.

The stemmer operates by applying lists of rewrite rules organized into a sequence of steps. For example, the first list of rewrite rules (constituting step 1a) handles plural forms as follows:

sses $\rightarrow$ ss
ies $\rightarrow$ i
ss $\rightarrow$ ss
s $\rightarrow$

To apply a rule, the pattern on the left-hand side of the rule is matched against the current suffix of the term. If the suffix matches, the suffix is rewritten by removing it and replacing it with the pattern on the right-hand side. Thus, the first of these rules rewrites “caresses” into “caress”, and the second rewrites “ponies” into “poni” (not “pony”). Only the first matching rule in a list is applied. The third rule, which appears to do nothing, exists to prevent the “ss” suffix on words such as “caress” from being rewritten by the fourth rule. This last rule simply removes a final “s”, rewriting “cats” to “cat”. Overall, the algorithm has five main steps, some of which have sub-steps. In total there are nine lists of rules, the longest of which contains twenty rules.

The application of a stemmer may improve recall by allowing more documents to be retrieved but may also harm precision by allowing inappropriate matches, assigning high scores to non-relevant documents. Nonetheless, when averaged over the topics in a typical TREC experiment, stemming often produces a noticable improvement. Table 3.1 shows the impact of applying the Porter stemmer to the test collections discussed in Chapter 1. The values in this table may be directly compared with those in Table 2.5 on page 72. For all four data sets, stemming has a positive impact on retrieval effectiveness. These results suggest that the application of a stemmer is an excellent idea if you are participating in a TREC-style experiment.

When stemming goes wrong, however, it can have a strongly negative impact. For example, the title of TREC topic 314, “marine vegetation”, stems down to “marin veget” under the Porter stemmer. Unfortunately, “marinated vegetables” stems down to the same root form. A user issuing this query may be surprised and confused if the IR system retrieves recipes and restaurant reviews along with (or instead of) articles on aquatic botany. A user may have no idea why such documents would be retrieved by her query. To her, the system may appear broken or buggy. Even if she understands the cause of the problem, she may not know how to solve it unless a method for explicitly disabling the stemmer is provided.

In research contexts, stemming may often be appropriate. For participants in an experimental effort such as TREC, stemming may substantially improve performance with little additional effort on the part of the researchers. However, in more realistic settings, any tokenization method that has the potential to produce inexplicably erroneous results should be treated with caution. Before integrating a stemmer into an operational IR system, the downside should be considered carefully.

3.1.3 Stopping

Function words are words that have no well-defined meanings in and of themselves; rather, they modify other words or indicate grammatical relationships. In English, function words include prepositions, articles, pronouns and articles, and conjunctions. Function words are usually among the most frequently occurring words in any language. All of the words (as opposed to XML tags) appearing in Figure 1.1 on page 15 are function words.

When documents are viewed as unstructured “bags of words”, as they are under the vector space model, the inclusion of function words in a query may appear to be unnecessary. Even under a proximity model, the close proximity of a particular function word to other query terms is hardly surprising, given the frequency of all function words in ordinary text. As a result IR systems traditionally define a list of *stopwords*, which usually include the function words. At query time these stopwords are stripped from the query, and retrieval takes place on the basis of the remaining terms alone.

Consider the appearance of the term “the” in a query. Roughly 6% of tokens in English text will match this term. Similarly, roughly 2% of tokens will match the term “of”. Nearly every English-language document contains both terms. If a user is searching for information on “the marriage of William Shakespeare”, reducing this information need to the query ⟨“marriage”, “william”, “shakespeare”⟩ — treating “of” and “the” as stopwords — should have no negative impact on retrieval effectiveness.

In addition to function words, a list of stopwords might include single letters, digits, and other common terms, such as the state-of-being verbs. In the specific environment of the Web, the stopword list might include terms such as “www”, “com”, and “http”, which are essentially meaningless in this context. Depending on the IR system and its environment, the stopword list might range from less than a dozen terms to many hundreds of terms. After the removal of typical stopwords, the text of Figure 3.1 (page 87) reduces to:

question
ti nobler mind suffer
sling arrow outrag fortun
take arm sea troubl
oppos end die sleep
sleep sai end
heart ach thousand natur shock
flesh heir ti consumm
devoutli wish die sleep
sleep perchanc dream ay rub

Because stopwords are usually frequent terms with relatively long postings lists, their elimination may substantially reduce query execution times by avoiding the processing of these longer lists. Furthermore, if stopwords are consistently stripped from all queries, they do not need to be included in the index at all. In early IR systems, this elimination of stopwords provided the benefit of reducing index sizes, an important consideration when disks and memory were small and expensive.

Unfortunately, a small number of queries are adversely affected by stopword elimination, usually because stopwords form an essential part of a phrase. The start of Hamlet's soliloquy, "to be or not to be that is the", provides a well-known example. Although we may immediately recognize this quote, it is composed entirely of terms that are traditionally stopwords. The name of the band "The The" provides another example — as well as further demonstrating the importance of capitalization. Although you might view these examples as extreme cases — and they are — they still should be handled in a reasonable way by a search engine.

In order to handle cases such as these, we recommend the inclusion of all stopwords in the index. Depending on the retrieval algorithm, it may be possible to selectively eliminate them from queries when the positive impact of their presence is expected to be minimal. With stopwords present in the index, the IR system can make a decision on a query-by-query basis. Ranking methods in a modern commercial search engine will incorporate many ranking features, including features based on term frequency and proximity. For features that do not consider proximity between query terms, stopwords may be elminated. For features that do consider proximity between query terms, particularly to match their occurrence in phrases, it may be appropriate to retain stopwords.

3.2 Characters

Tokenizing raw text requires an understanding of the characters it encodes. So far in the book, we have tacitly assumed an encoding of the text as 7-bit ASCII values. ASCII is sufficient to encode most characters in English text, including uppercase and lowercase letters, digits, and many punctuation characters.

Although ASCII encoding is acceptable for English, it is wholly inadequate for most other languages. ASCII was standardized in 1963, at a time when memory was at a premium, networking was a dream, and English-speaking countries represented the major market for computing equipment. Support for other languages was undertaken as needed, on a country-by-country or region-by-region basis. Often, these specialized character encodings provided incomplete support for their target languages. For some languages, incompatible encodings were developed for different countries or regions, or by different hardware manufacturers. Even for English, IBM's Extended Binary Coded Decimal Interchange Code (EBCDIC) provided a widely implemented competitor to ASCII.

Progress came slowly, but by the late 1980s efforts were under way to create a single unified character encoding that encompassed all living languages (and eventually many extinct ones). This Unicode[2] standard now provides encodings for the characters from a large and growing set of languages. Except for simple experimental IR systems, support for Unicode is essential. Although many documents still use other encodings, these can be handled by conversion to Unicode. Providing native support for Unicode allows an IR system to grow and accommodate new languages as required.

Unicode assigns a unique value, called a *codepoint*, to each character but does not specify how these values are represented as raw text. A codepoint is written in the form U+*nnnn*, where *nnnn* indicates the value of the codepoint in hexadecimal. For example, the character β is represented by the codepoint U+03B2. Unicode currently supports more than 100,000 characters, with codepoints ranging beyond U+2F800.

An associated standard, UTF-8, provides a convenient method for representing these codepoints as raw text. Although there are competing methods for representing Unicode, UTF-8 has several advantages, including backward compatibility with ASCII. UTF-8 represents each codepoint with one to four bytes. Each character appearing in the ASCII character set is encoded as a single byte having the same value as the corresponding ASCII character. Thus, raw text in ASCII is automatically raw text in UTF-8.

Although UTF-8 uses a variable-length encoding for a character, the interpretation of this encoding is straightforward, with the high-order bits of the first byte indicating the length of the encoding. If the most significant bit is 0 — so that the byte has the form 0xxxxxxx — the length of the encoding is one byte. This byte represents the codepoint having the value indicated

[2] `www.unicode.org`

by the lower seven bits, which represents the character having the same seven-bit ASCII value. For example, the UTF-8 byte 01100101 represents the character "e" (U+0065), just as it does in ASCII.

If the most significant bit of the first byte is 1, the length of the encoding is two to four bytes, with the length encoded in unary by the bits that follow. Thus, the first byte of a two-byte encoding has the form 110xxxxx, the first byte of three-byte encoding has the form 1110xxxx, and the first byte of a four-byte encoding has the form 11110xxx. The second and subsequent bytes of a multibyte encoding all have the form 10xxxxxx. By examining the two most significant bits of any byte, we can determine whether it starts an encoding.

The value of the codepoint for a two-to-four-byte encoding is taken from the remaining unspecified bits, with the most significant bit being taken from the first byte. A two-byte encoding can represent codepoints in the range U+0080 to U+07FF, a three-byte encoding can represent codepoints in the range U+0800 to U+FFFF, and a four-byte encoding can represent codepoints with values U+10000 and above. For example, a three-byte encoding has the overall form

1110xxxx 10yyyyyy 10zzzzzz

and represents the 16-bit value xxxxyyyyyyzzzzzz. The character 八 is assigned the codepoint U+516B, which is 01010001 01101011 in binary. In UTF-8, the character would be encoded by the three bytes 11100101 10000101 10101011.

3.3 Character N-Grams

The difficulty of translating a sequence of characters into a sequence of tokens for indexing depends on the details of the underlying language. The process of recognizing and stemming words differs greatly from language to language. Character n-grams represent an alternative to complex language-specific tokenization. In this section we illustrate the technique with English. In later sections we apply it to other languages.

Using this technique, we simply treat overlapping sequences of n characters as tokens. For example, if $n = 5$, the word "orienteering" would be split into the following 5-grams:

orie orien rient iente entee nteer teeri eerin ering ring

The "_" character indicates the beginning or end of a word. Indexing then proceeds with each distinct n-gram given its own postings list. At query time the query is similarly split into n-grams for retrieval. Under this technique a three-character word such as "the" would be indexed with the 5-gram "_the_". For two-character and one-character words, we cheat a little, indexing "of" as "_of_" and "a" as "_a_".

In principle, n-gram indexing need not account for punctuation, capitalization, white space, or other language characteristics. Although the optimal value for n varies from language to

Table 3.2 Impact of 5-gram indexing. The table lists effectiveness measures for selected retrieval methods discussed in this book. The results in this table may be compared with those in Tables 2.5 and 3.1.

	TREC45				**GOV2**			
	1998		**1999**		**2004**		**2005**	
Method	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**
Proximity (2.2.2)	0.392	0.125	0.388	0.149	0.431	0.171	0.552	0.233
BM25 (Ch. 8)	0.410	0.177	0.446	0.214	0.463	0.226	0.522	0.296
LMD (Ch. 9)	0.416	0.186	0.438	0.222	0.404	0.188	0.502	0.276
DFR (Ch. 9)	0.440	0.203	0.444	0.230	0.478	0.243	0.540	0.284

language, the approach is otherwise language-independent. We just split documents into n-grams, construct an index, and issue queries. In practice, n-gram indexing typically takes into account basic language characteristics. For English, we might strip punctuation and apply case normalization.

N-grams can substitute for a stemmer when no stemmer is available. For many languages, morphology is reflected in the number of n-grams shared by two terms. More of the 5-grams in "orienteers" are matched by 5-grams from "orienteering" than by 5-grams from "oriental".

For English, n-gram indexing has no strong impact. Table 3.2 shows the effectiveness results for 5-gram indexing over our standard experimental collections. The results in the table may be compared with those in Tables 2.5 and 3.1. Compared with those in Table 2.5, the results are mixed. Some are a little better; some are a little worse. As we show in the next section, the outcome may be different for other languages.

N-gram indexing comes at the price of increased index size and decreased efficiency. The compressed indices for the collections in Table 3.2 require up to six times as much storage as the corresponding indices for word-based tokens (in a typical corpus of English text, an average word consists of six characters). Query execution time may be expected to increase by a factor of thirty or more because we increase the number of terms in each query (after tokenization) as well as the lengths of the postings lists.

N-gram indexing can be further extended. Even when words are delimited by white space and punctuation, as they are in English, n-grams may be allowed to cross word boundaries, thus potentially allowing phrasal relationships to be captured. For example, the fragment "...perchance to dream..." would be split into the following 5-grams:

> _perc perch ercha rchan chanc hance ance_ nce_t ce_to e_to_ _to_d to_dr o_dre _drea dream ream_

where the "_" character indicates an interword space. Although this technique generally has a slightly negative impact on the retrieval methods and English-language collections used in our experiments, it may be of benefit for other methods and languages.

3.4 European Languages

In this section we broaden our discussion of tokenization from English to other European languages. Into this category we place a broad range of languages belonging to several distinct language families. For example, French and Italian are Romance languages; Dutch and Swedish are Germanic languages; Russian and Polish are Slavic languages. The Romance, Germanic, and Slavic languages are related to each other, but Finnish and Hungarian belong to a fourth, unrelated family. Irish and Scottish Gaelic belong to a fifth family. Basque, spoken by more than a million people in Spain and France, is an *isolate*, related to no other living language.

Although the category “European language” has little or no meaning from a linguistics perspective, from an IR perspective we can consider these languages as a group. Due to the shared history of the continent, these languages use similar conventions in their *orthography*, the rules by which they are written. Each uses an alphabet of a few dozen letters with uppercase and lowercase forms. Punctuation provides structure. Most usefully, letters are divided into words by white space and punctuation.

There are a number of tokenization issues common to many European languages that are not shared by English. Accents and other diacritical marks are almost nonexistent in English, with “naïve” being a rare exception. In many other European languages, diacritics are essential for pronunciation and may be required to distinguish one word from another. In Spanish, “cuna” is a cradle but “cuña” is a wedge. In Portuguese, “nó” is a knot but “no” is a function word meaning “in the” or “at the”. Although carefully edited text will have correct diacritics, these marks may be missing in queries and more casual writing, thus preventing a match.

One solution is to remove the accent marks during tokenization, so that “cuna” and “cuña” are treated as the same term. Another solution is to double index such terms, with and without diacritics. During query processing, the IR system would determine when diacritics are essential and when they are not. Specialized retrieval methods might consider both forms, allowing a match either way. The best solution depends on details of the specific language and the IR environment.

Other tokenization issues are specific to individual languages or language families. In many Germanic languages compound nouns are written as single words. For example, the Dutch equivalent of “bicycle wheel” is “fietswiel”. During tokenization this compound noun might be *broken* or *segmented* into two terms, allowing the query term “fiets” to match. Similarly, the German word “Versicherungsbetrug” is a compound of “Versicherung” (“insurance”) and “Betrug” (“fraud”). Note that in this case, it is not sufficient simply to split the word into two parts; we also need to take care of the “s” character that connects the two components. Other problems occur when IR systems contain documents in a mixture of languages. Only the acute accent distinguishes the French “thé” (“tea”) from the English definite article.

Table 3.3 Comparison of tokenization techniques for several European languages (based on McNamee, 2008). The table lists mean average precision values for tokenizations based on unstemmed words, the Snowball stemmer, character 4-grams, and character 5-grams.

Language	Words	Stemmer	4-grams	5-grams
Dutch	0.416	0.427	0.438	0.444
English	0.483	0.501	0.441	0.461
Finnish	0.319	0.417	0.483	0.496
French	0.427	0.456	0.444	0.440
German	0.349	0.384	0.428	0.432
Italian	0.395	0.435	0.393	0.422
Spanish	0.427	0.467	0.447	0.438
Swedish	0.339	0.376	0.424	0.427

Stemmers are available for most European languages. Of particular note is the Snowball stemmer, which provides support for more than a dozen European languages, including Turkish.[3] Snowball is the creation of Martin Porter, the developer of the Porter stemmer for English, and it shares the algorithmic approach of his older stemmer. *N*-gram character tokenization also works well for European languages, with values of $n = 4$ or $n = 5$ being optimal.

Table 3.3 compares tokenization techniques for eight European languages, using TREC-style test collections. For all languages the Snowball stemmer outperforms unstemmed words. For Dutch, Finnish, German, and Swedish, character 5-grams give the best overall performance.

3.5 CJK Languages

Chinese, Japanese, and Korean form what are called the *CJK languages*. Like the European languages, the written forms of these languages are linked by shared orthographic conventions deriving from their common history, although they are not members of a single language family. Compared with the European languages, their character sets are enormous. A typical Chinese-language newspaper contains thousands of distinct characters. Moreover, Chinese and Japanese words are not separated by spaces. As a result, segmentation is an essential part of tokenization for these languages.

The complexity of the character sets poses substantial problems. Japanese uses three main scripts. In two of these, each character represents a syllable in the spoken language. The third script is derived from Chinese characters and shares Unicode codepoints with them. In principle,

[3] snowball.tartarus.org

a word may be written in any of the three scripts. A query term written in one script may be expected to match against a document term written in another.

Many Chinese characters have a traditional form and a simplified form. For historical reasons, different forms dominate in different regions of the Chinese-speaking world. For example, traditional characters are common in Hong Kong and Macau, whereas simplified characters are standard elsewhere in China. It may be appropriate to return a document in one form in response to a query in the other form. Many people can read text written in either form. Software tools such as browser plugins are also available to automatically translate from one form to the other.

The Chinese language is unusual in that it assigns a meaning to each character, but this property does not imply that each character may be treated as a word. Moreover, the meaning of a word is not consistently linked to the meanings of its characters; a word is not just the sum of its parts. For instance, the Chinese word for "crisis" (危機) is often described as being composed of the character for "danger" followed by the character for "opportunity" (Gore, 2006). However, in this context the second character means something closer to "crucial point", and it often means "machine" in other contexts. The same character appears as the first character in "airport" (機場) in which the second character means "field". Connecting the parts correctly is essential to preserve meaning. An airport is not a "machine field". Freedom (自由) is not just another word for "self cause".

Although some Chinese words, including many function words, consist of a single character, most words are longer. A large proportion of these words are bigrams: two characters in length. Segmenting a sequence of characters into words is a difficult process. Consider the following English sentence written without spaces: "Wegotogethertogether.". The correct segmentation ("We go to get her together.") requires a knowledge of English grammar as well as English vocabulary. In Chinese a character may form a word with both the character that proceeds it and the character that follows it. The correct segmentation depends on the context. Nonetheless, word segmentation is not absolutely essential for Chinese-language information retrieval. Perhaps because of the predominance of bigrams, 2-gram character indexing works well for Chinese, providing a simple baseline for evaluating automatic word segmentation.

The CJK languages all have standardized conventions for their transliteration into the Roman alphabet. For Chinese the modern transliteration standard is *Hanyu Pinyin* (or just *Pinyin*). In Pinyin, 危機 becomes "wēijī" and 機場 becomes "jīchǎng". The diacritical marks indicate *tonality*, the changes in pitch used when pronouncing the syllable.

Pinyin provides a convenient alternative for expressing Chinese-language queries. Unfortunately, the use of Pinyin introduces a certain level of ambiguity. For example, about six common characters are transliterated as "jī". Moreover, Pinyin queries are commonly entered without diacritics, giving rise to additional ambiguity. With tone marks omitted, some thirty common characters are transliterated as "yi", and even more are transliterated as "shi". The query term "shishi" could mean "current event" (時事), "implementation" (實施), and several other things. To address this ambiguity, a search engine might present alternative interpretations to users, allowing them to choose the correct one.

3.6 Further Reading

Simplistically, tokenization attempts to split a document into words. Trask (2004) provides a readable discussion of the concept of "a word" from a linguistics standpoint. Additional discussion of tokenization and lemmatization from an NLP perspective is given in Chapter 4 of Manning and Schütze (1999).

Most programming languages and operating systems now provide support for Unicode and UTF-8. Details of the Unicode standard are given on its official Web site.[4] The UTF-8 encoding for Unicode codepoints was invented by Rob Pike and Ken Thompson as part of their Plan-9 Operating System effort (Pike and Thompson, 1993). Since the time of Pike and Thompson's invention, UTF-8 has been modified to accommodate changes and extensions to Unicode. It is defined as an Internet standard by RFC 3639.[5]

In addition to the Porter stemmer (Porter, 1980), early stemmers for English include those by Lovins (1968), Frakes (1984), and Paice (1990). Harman (1991) describes a simple *S stemmer*, designed to reduce plural words to their singular forms. She compared the performance of this S stemmer with those of the Porter and Lovins stemmers but found no significant improvement in retrieval performance over unstemmed words for any of the stemmers. Through a more extensive evaluation using larger test collections, Hull (1996) demonstrates that stemming can significantly improve average effectiveness measures. However, he cautions that aggressive stemming may degrade the performance of many queries while still improving average effectiveness by greatly improving the performance of a few queries.

The Snowball[6] stemmer provides an algorithmic framework for creating stemmers. Nonetheless, the creation and validation of a stemmer for a new language remains a labor-intensive process. As a result, several attempts have been made to automatically create stemmers by using large corpora to provide language examples. Creutz and Lagus (2002) describe a method for segmenting words by minimizing a cost function and compare their method with existing stemmers for English and Finnish. Majumder et al. (2007) describe a method that clusters words into equivalence classes based on a morphological analysis. They evaluate their method over several languages, including Bengali.

McNamee and Mayfield (2004) provide an overview of character n-gram tokenization for information retrieval. McNamee et al. (2008) and McNamee (2008) compare n-gram indexing with the Snowball stemmer and with the method of Creutz and Lagus (2002). In addition, character n-grams provide one method for coping with errors introduced by optical character recognition (OCR). Beitzel et al. (2002) provide a short survey of this area.

[4] `www.unicode.org`

[5] `tools.ietf.org/html/rfc3629`

[6] `snowball.tartarus.org`

Spanish IR was the subject of an experimental track at TREC from 1994 to 1996; Chinese was the subject of a track in 1996 and 1997 (Voorhees and Harman, 2005). A multilingual track that considered Arabic along with other languages ran from 1997 until 2002 (Gey and Oard, 2001). Starting in 2000 the Cross-Language Evaluation Forum (CLEF[7]) conducted multilingual and crosslingual experiments on European languages. The results in Table 3.3 are derived from CLEF test collections. Since 2001 the Japanese National Institute of Informatics Test Collection for IR Systems project (NTCIR[8]) has provided a similar experimental forum for Asian languages. The Indian Forum for Information Retrieval Evaluation (FIRE[9]) conducts experiments on languages of the Indian subcontinent.

Computer processing for Asian languages presents difficult and complex problems, and it remains the subject of substantial ongoing research. A major journal, *ACM Transactions on Asian Language Information Processing* is devoted entirely to the topic. Luk and Kwok (2002) provide an extensive overview of word segmentation and tokenization techniques for Chinese information retrieval. Peng et al. (2002) explore the importance of word segmentation for Chinese information retrieval, evaluating and comparing a number of competing techniques. Braschler and Ripplinger (2004) consider word segmentation (also called *decompounding*) for German. Kraaij and Pohlmann (1996) consider decompounding for Dutch. A special issue of the *Information Retrieval Journal* is devoted to non-English Web retrieval (Lazarinis et al., 2009).

Tokenization for a new language requires careful consideration of both its orthography and its morphology. Fujii and Croft (1993) cover the basics of tokenization for Japanese. Asian et al. (2005) discuss Indonesian. Larkey et al. (2002) discuss Arabic. Nwesri et al. (2005) present and evaluate an algorithm for removing affixes representing conjunctions and prepositions from Arabic words.

Spelling correction is closely related to tokenization. Because tokens provide the link between queries and documents, matching becomes harder when misspellings are present. Kukich (1992) provides a survey of foundational work on the detection and correction of spelling errors. Brill and Moore (2000) present an error model for spelling correction. Ruch (2002) examines the impact of spelling errors on a basic adhoc retrieval task. Both Cucerzan and Brill (2004) and Li et al. (2006) describe spell checkers created by identifying spelling errors and their corrections from the query logs of a commercial Web search engine. Jain et al. (2007) describe related methods for expanding acronyms using query logs.

[7] `www.clef-campaign.org`

[8] `research.nii.ac.jp/ntcir`

[9] `www.isical.ac.in/~clia`

3.7 Exercises

Exercise 3.1 Try the following queries on at least two commercial search engines:

(a) to be or not to be
(b) U.S.
(c) US
(d) THE
(e) The The
(f) I'll be back
(g) R-E-S-P-E-C-T

Each query has a peculiar meaning that may be lost through tokenization. Judge the top five results. Which are relevant?

Exercise 3.2 In the content of the Web certain terms appear so frequently that they may viewed as stopwords. Try the following queries on at least two commercial search engines:

(a) www
(b) com
(c) http

Judge the top five results. Which are relevant?

Exercise 3.3 Unicode assigns the character β a codepoint of U+03B2. What is its binary UTF-8 representation?

Exercise 3.4 (project exercise) Tokenize the collection you created in Exercise 1.9. For this exercise, keep only the tokens consisting entirely of characters from the English alphabet. Ignore tags and tokens containing non-ASCII characters. Generate character 5-grams from these tokens, following the procedure of Section 3.3. Generate a log-log plot of frequency vs. rank order, similar to the plot in Figure 1.5 or the plot generated for Exercise 1.12. Do the 5-grams follow Zipf's law? If so, what is an approximate value for α?

Exercise 3.5 (project exercise) Repeat Exercise 3.4 using Chinese-language text from Wikipedia, tokenized as character bigrams.

Exercise 3.6 (project exercise) Obtain a copy of the Porter stemmer or another stemmer for English. Tokenize the collection you created in Exercise 1.9. As you did in Exercise 3.4, keep only the tokens consisting entirely of characters from the English alphabet. Eliminate duplicate tokens to create a vocabulary for the collection. Execute the stemmer over the terms in the vocabulary to create sets of equivalent terms, all of which stem to the same root form. Which set (or sets) is the largest? Identify at least three sets containing terms that are inappropriately conflated by the stemmer.

Exercise 3.7 (project exercise) If you are familiar with a language other than English for which a stemmer is available, repeat Exercise 3.6 for that language. You may use Wikipedia as a source of text.

3.8 Bibliography

Asian, J., Williams, H. E., and Tahaghoghi, S. M. M. (2005). Stemming Indonesian. In *Proceedings of the 28th Australasian Computer Science Conference*, pages 307–314. Newcastle, Australia.

Beitzel, S., Jensen, E., and Grossman, D. (2002). Retrieving OCR text: A survey of current approaches. In *Proceedings of the SIGIR 2002 Workshop on Information Retrieval and OCR: From Converting Content to Grasping Meaning*. Tampere, Finland.

Braschler, M., and Ripplinger, B. (2004). How effective is stemming and decompounding for German text retrieval? *Information Retrieval*, 7(3-4):291–316.

Brill, E., and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293. Hong Kong, China.

Creutz, M., and Lagus, K. (2002). Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pages 21–30.

Cucerzan, S., and Brill, E. (2004). Spelling correction as an iterative process that exploits the collective knowledge of Web users. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 293–300.

Frakes, W. B. (1984). Term conflation for information retrieval. In *Proceedings of the 7th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 383–389. Cambridge, England.

Fujii, H., and Croft, W. B. (1993). A comparison of indexing techniques for Japanese text retrieval. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 237–246. Pittsburgh, Pennsylvania.

Gey, F. C., and Oard, D. W. (2001). The TREC-2001 cross-language information retrieval track: Searching Arabic using English, French or Arabic queries. In *Proceedings of the 10th Text REtrieval Conference*, pages 16–25. Gaithersburg, Maryland.

Gore, A. (2006). *An Inconvenient Truth*. Emmaus, Pennsylvania: Rodale.

Harman, D. (1991). How effective is suffixing? *Journal of the American Society for Information Science*, 42(1):7–15.

Hull, D. A. (1996). Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society for Information Science*, 47(1):70–84.

Jain, A., Cucerzan, S., and Azzam, S. (2007). Acronym-expansion recognition and ranking on the Web. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*, pages 209–214. Las Vegas, Nevada.

Kraaij, W., and Pohlmann, R. (1996). *Using Linguistic Knowledge in Information Retrieval.* Technical Report OTS-WP-CL-96-001. Research Institute for Language and Speech, Utrecht University.

Kukich, K. (1992). Technique for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439.

Larkey, L. S., Ballesteros, L., and Connell, M. E. (2002). Improving stemming for Arabic information retrieval: Light stemming and co-occurrence analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–282. Tampere, Finland.

Lazarinis, F., Vilares, J., Tait, J., and Efthimiadis, E. N. (2009). Introduction to the special issue on non-English Web retrival. *Information Retrieval*, 12(3).

Li, M., Zhu, M., Zhang, Y., and Zhou, M. (2006). Exploring distributional similarity based models for query spelling correction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 1025–1032. Sydney, Australia.

Lovins, J. B. (1968). Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1–2):22–31.

Luk, R. W. P., and Kwok, K. L. (2002). A comparison of Chinese document indexing strategies and retrieval models. *ACM Transactions on Asian Language Information Processing*, 1(3):225–268.

Majumder, P., Mitra, M., Parui, S. K., Kole, G., Mitra, P., and Datta, K. (2007). YASS: Yet another suffix stripper. *ACM Transactions on Information Systems*, 25(4):article 18.

Manning, C. D., and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* Cambridge, Massachusetts: MIT Press.

McNamee, P. (2008). Retrieval experiments at Morpho Challenge 2008. In *Cross-Language Evaluation Forum.* Aarhus, Denmark.

McNamee, P., and Mayfield, J. (2004). Character n-gram tokenization for European language text retrieval. *Information Retrieval*, 7(1-2):73–97.

McNamee, P., Nicholas, C., and Mayfield, J. (2008). Don't have a stemmer?: Be un+concern+ed. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 813–814. Singapore.

Nwesri, A. F. A., Tahaghoghi, S. M. M., and Scholer, F. (2005). Stemming Arabic conjunctions and prepositions. In *Proceedings of the 12th International Conference on String Processing and Information Retrieval*, pages 206–217. Buenos Aires, Agentina.

Paice, C. D. (1990). Another stemmer. *ACM SIGIR Forum*, 24(3):56–61.

Peng, F., Huang, X., Schuurmans, D., and Cercone, N. (2002). Investigating the relationship between word segmentation performance and retrieval performance in Chinese IR. In *Proceedings of the 19th International Conference on Computational Linguistics*. Taipei, Taiwan.

Pike, R., and Thompson, K. (1993). Hello world. In *Proceedings of the Winter 1993 USENIX Conference*, pages 43–50. San Diego, California.

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

Ruch, P. (2002). Information retrieval and spelling correction: An inquiry into lexical disambiguation. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 699–703. Madrid, Spain.

Trask, L. (2004). *What is a Word?* Technical Report LxWP11/04. Department of Linguistics and English Language, University of Sussex, United Kingdom.

Voorhees, E. M., and Harman, D. K. (2005). The Text REtrieval Conference. In Voorhees, E. M., and Harman, D. K., editors, *TREC — Experiment and Evaluation in Information Retrieval*, chapter 1, pages 3–20. Cambridge, Massachusetts: MIT Press.