Tokenizing

Lingüística Computacional

- Most text processing applications like POS taggers, parsers, stemmers, key word extractors, search engines etc. operate on words and sentences.

- 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.

- 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*.

- 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.

- 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.

- 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.

- 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.

- *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.

- In alphabetic languages, words are usually surrounded by whitespace and optionally preceded and followed by punctuation marks, parentheses, or quotes.

- 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:

- 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

- 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
  - cut off punctuation marks, parentheses, and quotes at both ends of the fragments

- 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
  - cut off punctuation marks, parentheses, and quotes at both ends of the fragments
- This simple rule is quite accurate because whitespace and punctuation are fairly reliable indicators of word boundaries.

- 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
  - cut off punctuation marks, parentheses, and quotes at both ends of the fragments
- 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.

- Not all periods are punctuation

- Not all periods are punctuation
- They also serve as markers for

- Not all periods are punctuation
- They also serve as markers for
  - abbreviations (like “etc.” or “U.S.A.”)

- Not all periods are punctuation
- They also serve as markers for
  - abbreviations (like “etc.” or “U.S.A.”)
  - ordinal numbers (as in the German date expression “1. Oktober 2005”)

- Not all periods are punctuation
- They also serve as markers for
  - abbreviations (like “etc.” or “U.S.A.”)
  - 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

- 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.

- 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.

- 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 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,

- 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.

- The sentence markers “!” or “?” are mostly unambiguous with a few exceptions like the company name “Yahoo!”

- The sentence markers “!” or “?” are mostly unambiguous with a few exceptions like the company name “Yahoo!”
- More problematic are the symbols “:” and “;”.

- 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

- 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
- but they are also used in enumerations, in particular when the elements of the enumeration contain commas

- 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
- but they are also used in enumerations, in particular when the elements of the enumeration contain commas
- Similarly, colons sometimes separate two sentences

- 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
- but they are also used in enumerations, in particular when the elements of the enumeration contain commas
- Similarly, colons sometimes separate two sentences
- and sometimes they are rather sentence-internal 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
- but they are also used in enumerations, in particular when the elements of the enumeration contain commas
- Similarly, colons sometimes separate two sentences
- and sometimes they are rather sentence-internal punctuation
- Distinguishing the different uses of colons and semicolons is very hard without analyzing the whole sentence.

- In German and other languages, ordinal numbers are written with a trailing period after the number. So, “17th” is written “17.”

- 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.

- 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

- 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
- The disambiguation is impossible without contextual information

- It was implicitly assumed that tokens do not contain whitespace.

- 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.

- 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

- 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”,

- 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”,

- 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”,
  - adverbs like “at all”,

- 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”,
  - adverbs like “at all”,
  - foreign phrases like “et cetera” and “en vogue”,

- 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”,
  - adverbs like “at all”,
  - foreign phrases like “et cetera” and “en vogue”,
  - date expressions like “Feb. 1, 2004”,

- 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”,
  - adverbs like “at all”,
  - foreign phrases like “et cetera” and “en vogue”,
  - date expressions like “Feb. 1, 2004”,
  - time expressions like “3:30 pm”,

- 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”,
  - adverbs like “at all”,
  - foreign phrases like “et cetera” and “en vogue”,
  - date expressions like “Feb. 1, 2004”,
  - time expressions like “3:30 pm”,
  - proper names like “Daimler Chrysler AG”

- Recognizers for such expressions are often implemented by regular expression matching with tools like “Lex” or “Flex” which are available on most Unix systems.

- Recognizers for such expressions are often implemented by regular expression matching with tools like “Lex” or “Flex” which are available on most Unix systems.
- 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.

- Recognizers for such expressions are often implemented by regular expression matching with tools like “Lex” or “Flex” which are available on most Unix systems.
- 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*

- 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.

- 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.

- 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.

- 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.

- Another problem for tokenization arises when the text was broken into lines during typesetting.

- 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*

- 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-”:

- 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.

- 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.

- 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.

- 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.

- 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 works quite well for the disambiguation of strings which contain periods:

- The following rule works quite well for the disambiguation of strings which contain periods:
  - If a potential token is of the form $s.r$,

- The following rule 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*,

- The following rule 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.

- The following rule 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.

- The detection of word boundaries is one task in tokenization, the other is the identification of sentence boundaries.

- 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 “!”.

- 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.

- 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.

- 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

- 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.”

- 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.

- 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

- In contrast to alphabetic languages, ideographic languages like Chinese, Korean or Japanese do not mark the word boundaries with blanks.

- 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.

- 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.

- 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.

- 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:

- A simple tokenizer can be implemented by a longest-match strategy:
  1. the tokenizer determines the longest character sequence which starts at the current position and is listed in the dictionary.

- A simple tokenizer can be implemented by a longest-match strategy:
  1. the tokenizer determines the longest character sequence which starts at the current position and is listed in the dictionary.
  2. 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

- 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.

- 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.

- 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.

- 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.

- 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.

- 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

- Formatting information may be relevant for tokenization.

- 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 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.

- 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.

- 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.

- 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.

- 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.

- 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,

- 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,
  1. the string “xy” (a “regular” word like “preprocessing”),

- 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,
  1. the string “xy” (a “regular” word like “preprocessing”),
  2. the string “x-y” (a hyphenated word like “pre-processing”), and

- 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,
  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.

- 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.

- 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.

- 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 %.

- 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:

- 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”.

- 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”.

- 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.

- 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.

- 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”,

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”.

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”.

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”

- The most difficult problem for the tokenization of alphabetic languages is the disambiguation of periods.

- 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.

- 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).

- 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 %.

- Many abbreviations consist of a sequence of letter-period pairs like “U.S.A.”, or of a capitalized letter followed by a sequence of consonants like “Mrs.”

- Many abbreviations consist of a sequence of letter-period pairs like “U.S.A.”, or of a capitalized letter followed by a sequence of consonants like “Mrs.”
- If strings of this form are always classified as sentence-internal abbreviations, the accuracy of the sentence boundary detection is raised to 97.66 %.

- Many abbreviations consist of a sequence of letter-period pairs like “U.S.A.”, or of a capitalized letter followed by a sequence of consonants like “Mrs.”
- 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.

- Many abbreviations consist of a sequence of letter-period pairs like “U.S.A.”, or of a capitalized letter followed by a sequence of consonants like “Mrs.”
- 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.

- Many abbreviations consist of a sequence of letter-period pairs like “U.S.A.”, or of a capitalized letter followed by a sequence of consonants like “Mrs.”
- 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.

- 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 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.

- 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.

- 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 “no.” because “no” is a word.

- 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 “no.” because “no” is a word.
- 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.

- 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.

- 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.

- 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”.

- 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.

- The heuristic for the extraction of abbreviations from corpora fails to find abbreviations which are typically followed by upper-case words or by numbers.

- 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.

- 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.

- 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 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.

- 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.

- 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.

- 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.
- Suffixbased recognition of abbreviations is particularly useful in German with its complex morphology.

- The different period disambiguation heuristics discussed so far need to be combined in order to achieve optimal accuracy.

- 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

- 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.

- 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.

- Period disambiguation can also be treated as a classification problem.

- 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.

- 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

- 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,

- 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,

- 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. the length of the preceding and the following word,

- 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. the length of the preceding and the following word,
  4. whether the preceding/following word is lower-case, upper-case, capitalized or a number,

- 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. the length of the preceding and the following word,
  4. whether the preceding/following word is lower-case, upper-case, capitalized or a number,
  5. whether the period is followed by punctuation

- 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. the length of the preceding and the following word,
  4. whether the preceding/following word is lower-case, upper-case, capitalized or a number,
  5. whether the period is followed by punctuation
  6. whether the preceding word with the period is an abbreviation and the type of the abbreviation.

- 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. the length of the preceding and the following word,
  4. whether the preceding/following word is lower-case, upper-case, capitalized or a number,
  5. whether the period is followed by punctuation
  6. 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- The disadvantage of classification-based approaches is the need for manually annotated training data whose creation is expensive and time-consuming.

- 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.

- 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, classificationbased systems need to be retrained for new text types.

- In order to obtain optimal results, classificationbased systems need to be retrained for new text types.
- Unsupervised methods avoid this problem by extracting the required information from raw text.

- In order to obtain optimal results, classificationbased systems need to be retrained for new text types.
- Unsupervised methods 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).

- In order to obtain optimal results, classificationbased systems need to be retrained for new text types.
- Unsupervised methods 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 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 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.