Chapter NLP:III

III. Document Models

- Layers of Text Representation
- Text Preprocessing
- Tokenization
- Text Representation
- Text Similarity
- Sequence Modeling

Layers of Text Representation

Overview

A document model is a representation of natural language text in a computer.

Layer	Unit of Meaning	Model	Standards
Application	Request, task, problem	Retrieval, analysis, synthesis	
Knowledge	Entity relation	Graph, ontology, knowledge frame	RDF, OWL
Pragmatics	Utterance, speech/dialogue act	Information state, dialogue state, task frame	DiAML, DR-core, DRS, RST, QUD
Semantics	Vector, λ-expression	BOW, embedding, λ-calculus, semantic frame	CIFF, NumPy arrays, Parquet
Syntax	Morpheme, token, phrase	Sequence, Tree, PCFG, Dependency G.	TEI, ISO LAF, WADM, CoNLL tab
Presentation	Document	Document Object Model (DOM)	
Document structure	Content object	Hierarchy	HTML/CSS, PDF, DOCx
Character sequence	Character	String, buffer, rope, suffix array,	C/Posix
Character encoding	Byte sequence		ASCII, ISO-8859-n, Unicode, etc.
Bit stream	Bits, Symbol	File, message	File systems, network protocols

- Document models enable semantic text analysis for NLP applications.
- Semantics are captured in layered abstractions over documents.

Layers of Text Representation

Overview

A document model is a representation of natural language text in a computer.

Layer	Unit of Meaning	Model	Standards
Application	Request, task, problem	Retrieval, analysis, synthesis	
Knowledge	Entity relation	Graph, ontology, knowledge frame	RDF, OWL
Pragmatics	Utterance, speech/dialogue act	Information state, dialogue state, task frame	DiAML, DR-core, DRS, RST, QUD
Semantics	Vector, λ-expression	BOW, embedding, λ-calculus, semantic frame	CIFF, NumPy arrays, Parquet
Syntax	Morpheme, token, phrase	Sequence, Tree, PCFG, Dependency G.	TEI, ISO LAF, WADM, CoNLL tab
Presentation	Document	Document Object Model (DOM)	
Document structure	Content object	Hierarchy	HTML/CSS, PDF, DOCx
Character sequence	Character	String, buffer, rope, suffix array,	C/Posix
Character encoding	Byte sequence		ASCII, ISO-8859-n, Unicode, etc.
Bit stream	Bits, Symbol	File, message	File systems, network protocols

- Text Extraction. Conversion of documents to plain text (e.g., String), and cleansing.
- Tokenization. Dividing plain text into its constituent parts: tokens.

Remarks:

- The layers in the lower group build on top of each other to facilitate human-readable text in a computer.
- The layers in the upper group may build on top of any of the lower layers to facilitate natural language processing in a computer.
- Presuming advanced multimodal neural networks, natural language processing can be built directly on top of human-readable document.
- However, current NLP approaches typically rely on plain text documents and tailored algorithms for language processing to enable solving NLP tasks that require higher layers of representation.

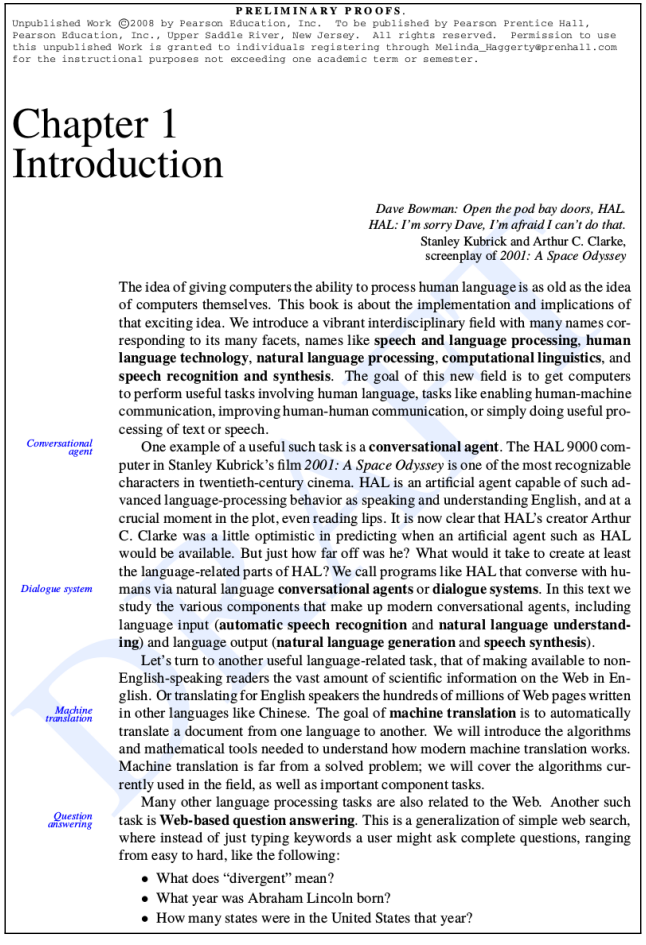
Chapter NLP:III

III. Document Models

- Layers of Text Representation
- Text Preprocessing
- Tokenization
- Text Representation
- Text Similarity
- Sequence Modeling

Text Preprocessing

Text Extraction



PRELIMINARY PROOFS.

Chapter 1
Introduction

Dave Bowman: Open the pod bay doors, HAL.
HAL: I'm sorry Dave, I'm afraid I can't do that.
Stanley Kubrick and Arthur C. Clarke,
screenplay of *2001: A Space Odyssey*

The idea of giving computers the ability to process human language is as old as the idea of computers themselves. This book is about the implementation and implications of that exciting idea. We introduce a vibrant interdisciplinary field with many names corresponding to its many facets, names like **speech and language processing**, **human language technology**, **natural language processing**, **computational linguistics**, and **speech recognition and synthesis**. The goal of this new field is to get computers to perform useful tasks involving human language, tasks like enabling human-machine communication, improving human-human communication, or simply doing useful processing of text or speech.

Conversational agent

One example of a useful such task is a **conversational agent**. The HAL 9000 computer in Stanley Kubrick's film *2001: A Space Odyssey* is one of the most recognizable characters in twentieth-century cinema. HAL is an artificial agent capable of such advanced language-processing behavior as speaking and understanding English, and at a crucial moment in the plot, even reading lips. It is now clear that HAL's creator Arthur C. Clarke was a little optimistic in predicting when an artificial agent such as HAL would be available. But just how far off was he? What would it take to create at least the language-related parts of HAL? We call programs like HAL that converse with humans via natural language **conversational agents** or **dialogue systems**. In this text we study the various components that make up modern conversational agents, including language input (**automatic speech recognition** and **natural language understanding**) and language output (**natural language generation** and **speech synthesis**).

Dialogue system

Machine translation

Let's turn to another useful language-related task, that of making available to non-English-speaking readers the vast amount of scientific information on the Web in English. Or translating for English speakers the hundreds of millions of Web pages written in other languages like Chinese. The goal of **machine translation** is to automatically translate a document from one language to another. We will introduce the algorithms and mathematical tools needed to understand how modern machine translation works. Machine translation is far from a solved problem; we will cover the algorithms currently used in the field, as well as important component tasks.

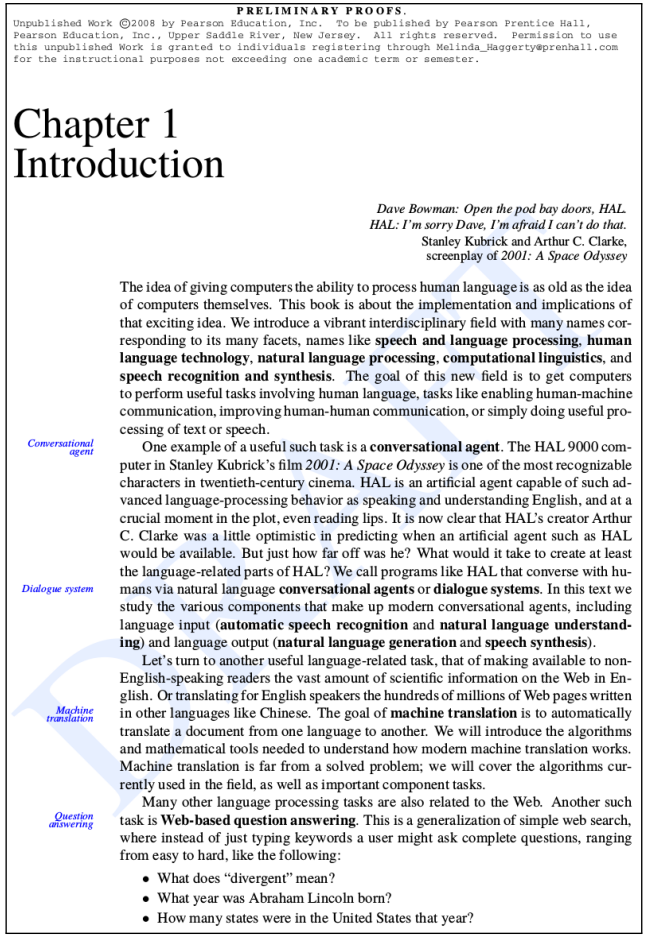
Question answering

Many other language processing tasks are also related to the Web. Another such task is **Web-based question answering**. This is a generalization of simple web search, where instead of just typing keywords a user might ask complete questions, ranging from easy to hard, like the following:

- What does "divergent" mean?
- What year was Abraham Lincoln born?
- How many states were in the United States that year?

Text Preprocessing

Text Extraction



PRELIMINARY PROOFS.

Chapter 1 Introduction

Dave Bowman: Open the pod bay doors, HAL.
HAL: I'm sorry Dave, I'm afraid I can't do that.
Stanley Kubrick and Arthur C. Clarke,
screenplay of *2001: A Space Odyssey*

The idea of giving computers the ability to process human language is as old as the idea of computers themselves. This book is about the implementation and implications of that exciting idea. We introduce a vibrant interdisciplinary field with many names corresponding to its many facets, names like **speech and language processing**, **human language technology**, **natural language processing**, **computational linguistics**, and **speech recognition and synthesis**. The goal of this new field is to get computers to perform useful tasks involving human language, tasks like enabling human-machine communication, improving human-human communication, or simply doing useful processing of text or speech.

Conversational agent

One example of a useful such task is a **conversational agent**. The HAL 9000 computer in Stanley Kubrick's film *2001: A Space Odyssey* is one of the most recognizable characters in twentieth-century cinema. HAL is an artificial agent capable of such advanced language-processing behavior as speaking and understanding English, and at a crucial moment in the plot, even reading lips. It is now clear that HAL's creator Arthur C. Clarke was a little optimistic in predicting when an artificial agent such as HAL would be available. But just how far off was he? What would it take to create at least the language-related parts of HAL? We call programs like HAL that converse with humans via natural language **conversational agents** or **dialogue systems**. In this text we study the various components that make up modern conversational agents, including language input (**automatic speech recognition** and **natural language understanding**) and language output (**natural language generation** and **speech synthesis**).

Dialogue system

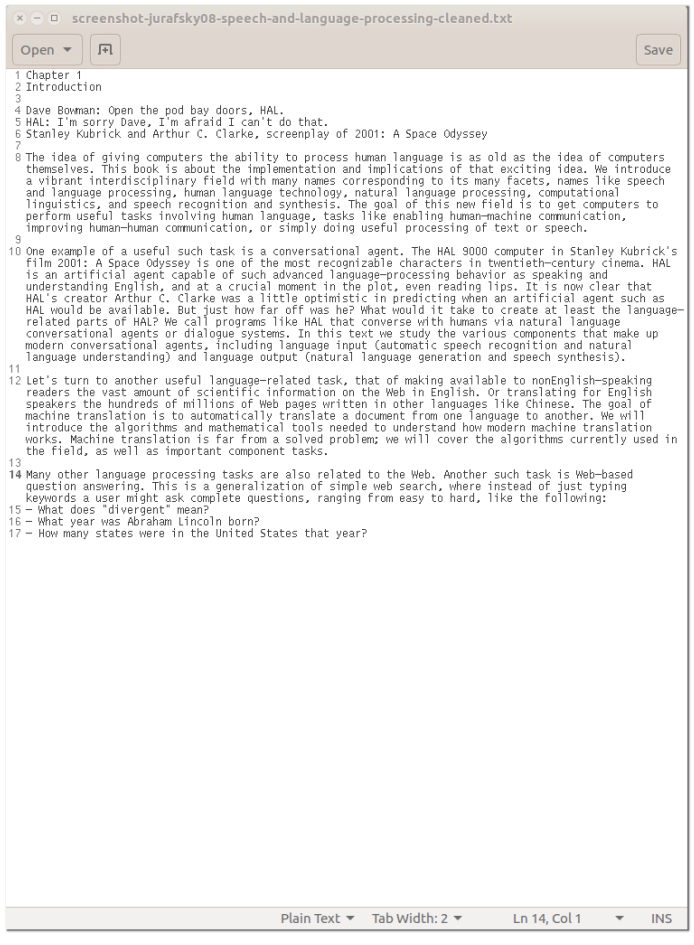
Machine translation

Let's turn to another useful language-related task, that of making available to non-English-speaking readers the vast amount of scientific information on the Web in English. Or translating for English speakers the hundreds of millions of Web pages written in other languages like Chinese. The goal of **machine translation** is to automatically translate a document from one language to another. We will introduce the algorithms and mathematical tools needed to understand how modern machine translation works. Machine translation is far from a solved problem; we will cover the algorithms currently used in the field, as well as important component tasks.

Question answering

Many other language processing tasks are also related to the Web. Another such task is **Web-based question answering**. This is a generalization of simple web search, where instead of just typing keywords a user might ask complete questions, ranging from easy to hard, like the following:

- What does "divergent" mean?
- What year was Abraham Lincoln born?
- How many states were in the United States that year?



Remarks:

- Example: Conversion of a PDF to text using the command line tool pdftotext.
- The Portable Document Format (PDF) is based on the PostScript language, representing the layout of a document and its pages independent of application software, hardware, and operating system. Text is stored as text elements, which basically specifies that characters should be drawn at certain positions.
- The order in which text elements appear on a PDF page does not necessarily correspond to semantic units visible on a page. This is why plain text extracted from PDF documents may mix up things like watermarks, header and footer, and side notes with the “main text”.

Text Preprocessing

Text Extraction

The goal of preprocessing is to convert documents into a canonical plain text format.

Degrees of document structure maintenance:

- Raw text stream. Character sequence of concatenated text documents.
- Raw documents. One line in a TXT file = one document. No document structure.
- Formatted text. One TXT file = one document. Line breaks signify some document structure (e.g., paragraphs vs. headings). Example: Project Gutenberg.
- Markup text. Usage of a markup language to encode document structure and maintain layout information. Example: Web search engines use (simplified) HTML as internal format.
- Meta information. Maintenance of provenance, structure, layout, and annotation information in (separate) markup documents or databases.

Remarks: (text extraction tools)

- Typical text extraction tools include converters, encoding tools, main content extractors, and OCR tools.
- Conversion tools.
  Pandoc, LibreOffice headless (Python wrapper: unoconv), pdftotext, catdoc, antiword, docx2txt, pptx3txt, xlsx2csv, lynx -dump, w3m -dump, html2text, ebook-convert, Apache Tika.
- Encoding detectors.
  file –mime-encoding, enca and enconv (Central-/Eastern-European encodings), nkf (Japanese encodings), chardet, uchardet.

  Encoding converters.
  iconv -f src-enconding -t dst-encoding, uconv -f src-enconding -t dst-encoding, recode.

  File name encodings.
  convmv.
- Main content extraction tools.
  jusText, Resiliparse.
- OCR tools.
  Tesseract OCR, Kraken, Calamari, OCRmyPDF, PaddleOCR, doctr, mmocr, TrOCR, Donut, olmOCR.

Chapter NLP:III

III. Document Models

- Layers of Text Representation
- Text Preprocessing
- Tokenization
- Text Representation
- Text Similarity
- Sequence Modeling

Tokenization

Definition 1 (Tokenization, Token)

Given a text d in the form of a character sequence, the task of tokenization is to split d into meaningful units called tokens.

Example:

“Friends, Romans, Countrymen, lend me your ears!”

“	Friends	,	Romans	,	Countrymen	,	lend	me	your	ears	!	”

Terminology: (simplified)

- A token is typically a word-like character sequence, or punctuation.
- A type relates to a token as a class relates to an object.

Tokenization

Definition 1 (Tokenization, Token)

Given a text d in the form of a character sequence, the task of tokenization is to split d into meaningful units called tokens.

Why is tokenization necessary? Why not use characters, sentences, or entire texts as units?

- Alphabetic and syllabic languages encode sounds as sets of characters or short character sequences, reused to express an infinite amount of meaningful statements.
- Logographic and continuous script languages represent meaning as characters or short character sequences, but have no word boundaries, such as spaces.
- Corpus linguistics and statistical language processing requires sufficiently large unit (token) populations. Long units are too infrequent, and different units may have the same meaning.
- A tokenized text encodes knowledge of language in terms of how it is composed of basic units of meaning.

Tokenization

Definition 1 (Tokenization, Token)

Given a text d in the form of a character sequence, the task of tokenization is to split d into meaningful units called tokens.

What are meaningful units of text?

- Character n-gram. $n \leq 3$ captures aspects of writing style.
- Syllable. Basic unit of speech uttered with a single effort of articulation.
- Morpheme. Minimal unit of meaning in language.
- Word. Sequence of one or more morphemes; an item of vocabulary.
- Phrase. A sequence of words expressing a notion (e.g., names, idioms, figures of speech).

Tokenization

Definition 1 (Tokenization, Token)

Given a text d in the form of a character sequence, the task of tokenization is to split d into meaningful units called tokens.

What are meaningful units of text?

- Character n-gram. $n \leq 3$ captures aspects of writing style.
- Syllable. Basic unit of speech uttered with a single effort of articulation.
- Morpheme. Minimal unit of meaning in language.
- Word. Sequence of one or more morphemes; an item of vocabulary.
- Phrase. A sequence of words expressing a notion (e.g., names, idioms, figures of speech).

Remarks: (type–token distinction)

- In semiotics, a token is “a particular and individual sign, as opposed to the type of which it is an instance”. [Oxford English Dictionary] Non-words can also be tokens (e.g., punctuation, number, abbreviation, etc.).
- Tokens are distinguished from types; the so-called type–token distinction. Example:

  A rose is a rose is a rose.

  In one sense of ‘word’, this sentence contains three words, but in another sense of ‘word’, it contains eight words. In the first sense, the words are types, and in the second sense, they are tokens. In simplified terms, a token may refer to a particular instance of a word printed on a page, while a type refers to the abstract form it instantiates. [Wetzel 2006]

Remarks: (alphabetic vs. logographic alphabets)

- Alphabetic languages typically use a fairly small set of *letters* forming an alphabet to encode sounds as syllables.
- Logographic languages use a fairly large set of letters that encode *symbols*, completely independent of how they are pronounced in speech.
- When using the term “character”, we refer to characters encoded in a computer. Unicode includes both, alphabetic letters and symbol letters of both languages.

Remarks: (units of meaning)

- A syllable is defined as “a vocal sound or set of sounds uttered with a single effort of articulation and forming a word or an element of a word; [...] also, a character or set of characters forming a corresponding element of written language.” [Oxford English Dictionary]
- A morpheme is defined as “a minimal and indivisible morphological unit that cannot be analyzed into smaller units (e.g. ‘in’, ‘come’, and ‘?ing’, forming ‘incoming’).” [Oxford English Dicationary]
- A word is defined as “an element or unit of speech [or] language”. “Any of the sequences of one or more sounds or morphemes (intuitively recognized by native speakers as) constituting the basic units of meaningful speech used in forming a sentence or utterance in a language (and in most writing systems normally separated by spaces); a lexical unit other than a phrase or affix; an item of vocabulary, a vocable.” [Oxford English Dictionary]
- A phrase is defined as “a small group or collocation of words expressing a single notion, or entering with some degree of unity into the structure of a sentence; a common or idiomatic expression.” [Oxford English Dictionary]

Remarks: (heuristic tokenization)

- Morphemes would be the close-to-ideal target unit of meaning for tokenizing a text. Morpheme sequences capture text semantics in an almost lossless way.
- As a first approximation, traditional tokenization therefore targets words, separating them from other kinds syntactical units, such as punctuation.
- As a second approximation, the word-like tokens are then normalized using heuristic morphology reduction techniques.
- Recently, in the wake of developing deep learning-based language models, sub-word tokenization approaches have emerged. These approaches approximate morphemes directly by analyzing sub-word frequencies.
- An extensive review of tokenization approaches in context of their respective purposes is given by Mielke et al. (2021).

Tokenization

Approaches

1. Naïve heuristics:
   - Whitespace: A token is every character sequence separated by whitespace characters.
   - Unix tool: `tr -sc 'A-Za-z' '\n' < text.txt`
   - TREC: A token is every alphanumeric sequence of characters of length > 3, separated by a space or punctuation mark.
2. Linguistic / word tokenization
   - Manually construct a set of rules and apply them in order.
   - Each rule describes how to split a string into smaller tokens.
3. Learned / subword tokenization
   - Split tokens based on observed character sequence frequencies in a training corpus.

Tokenization

Word Tokenization: Special Cases

- **Contractions**

  Apostrophes can be a part of a word, a part of a possessive, or just a mistake

  `it's`, `o'donnell`, `can't`, `don't`, `80's`, `men's`, `master's degree`, `shriner's`

- **Hyphenated compounds**

  Hyphens may be part of a word or a separator.

  Some words refer to the same concept with or without hyphen.

  `winston-salem`, `e-bay`, `wal-mart`, `active-x`, `far-reaching`, `20-year-old`

- **Compounds**

  `wheelchair`, `Computerlinguistik`

Tokenization

Word Tokenization: Special Cases

- Other special characters

  Special characters may form part of words, especially in technology-related text.

  `M*A*S*H`, `I.B.M.`, `Ph.D.`, `C++`, `C#`, ` `, `http://www.example.com`.

- Numbers

  Numbers form tokens of their own, and may contain punctuation as well: `6.5`, `1e+010`

- Phrases, named entities, phone numbers, dates

  `San Francisco, (800) 234-2333, Mar 11,1983`.

Tokenization

Word Tokenization: English

Algorithm: Regular Expression Tokenizer. [Grefenstette, 1999]

Input: d. Document in the form of a string. A. Dictionary of abbreviations.

Output: T. List of tokens in order of appearance in d.

Tokenize(d, A)

1. *consonant* = `"[bcdfghj-np-tvxz]"`, *clitic* = `"(?:'|:|-|'s|'d|'m|'ll|'re|'ve|n't)"`
2. Insert space around punctuation which are unambiguous separators.
3. Insert space around commas that aren't inside numbers.
4. Insert space around single quotes not preceded by letter.
5. Insert space around unambiguous word-suffix clitics and punctuation.
6. Split d by whitespace (**`/\s+/`**) to obtain a list of tokens T.
7. For each token $t \in T$, insert a space before full stops at the end of t, if t is not an abbreviation (i.e., t occurs in A, t is a letter–dot sequence, or t is an upper case letter followed by consonants and a dot).
8. Return a whitespace-separated string of T.

Tokenization

Word Tokenization: English

Algorithm: Regular Expression Tokenizer. [Grefenstette, 1999]

Input: d. Document in the form of a string. A. Dictionary of abbreviations.

Output: T. List of tokens in order of appearance in d.

Tokenize(d, A)

1. *consonant* = `"[bcdfghj-np-tvxz]"`, *clitic* = `"(?:'|:|-|'s|'d|'m|'ll|'re|'ve|n't)"`
2. Apply `s/([?!()\"\/\\|])/␣\1␣/g` to d.
3. Apply `s/(\D),/\1␣,␣/g` and `s/,(\D)/␣,␣\1/g` to d.
4. Apply `s/(\s')/\1␣/g` and `s/(\W)'/\1␣'/g` to d.
5. Apply `s/(`*clitic*`\s)/␣\1/g` and `s/(`*clitic*`)(\W)/␣\1␣\2/g` to d.
6. Split d by whitespace (`/\s+/`) to obtain a list of tokens T.
7. For each $t \in T$: If $t \notin A$, $t \in L($`/\w+\./`$)$, and $t \notin L($`/^(([A-Za-z]\.){2,}|[A-Z][`*consonant*`]+\.)$/`$)$ apply `s/\.$/␣\./` to t.
8. Return T.

Remarks:

- The variables *consonant* and *clitic* are regular expressions for the respective phenomena.
- **s/A/B/g** is a Perl shorthand for searching all occurrences of regular expression A and replace them with B. The Python equivalent is `re.sub(A, B, d)`
- The references \n resolves to the text matched by the n-th capture group `(...)` of regular expression A.
- ␣ indicates that this is a space character.
- The following shell command tokenizes `Thomas' note said, that:  "7:30am isn't great :(":`

```
clitic="(?:'|:|-|'s|'d|'m|'ll|'re|'ve|n't)"
echo "Thomas'␣note said,␣that:␣\"7:30am␣isn't␣great␣:(\"" \
    | perl -pe "s/[?!()\"\/\\|]/␣$&␣/g" \
    | perl -pe "s/(\D),/\1␣,␣/g" \
    | perl -pe "s/,(\D)/␣,␣\1/g" \
    | perl -pe "s/\s'/$&␣/g" \
    | perl -pe "s/(\W)'/\1␣'/g" \
    | perl -pe "s/$clitic\s/␣$&/g" \
    | perl -pe "s/($clitic)(\W)/␣\1␣\2/g"
```

- A more sophisticated approach to tokenization is implemented as part of the Moses statistical machine translation system. [github.com/moses-smt]

Tokenization

Definition 2 (Token Normalization)

Given a token (i.e., a word-like unit), map it to a "basic form" (e.g., its stem, lemma, or morpheme) to which semantically equivalent token variants (e.g., derivations or inflections) are also mapped.

Causes of token variability:

- Orthography. Surface-level variations in spelling and writing.
- Morphology. Variations of word formation due to inflection, derivation, and compounding.
- Lexical-semantic variation. Use of semantically same / similar words in different contexts.

Normalization problems:

- Undergeneralization. Not all equivalent tokens are mapped to the same canonical form.
- Overgeneralization. Non-equivalent tokens are mapped to the same canonical form.
- Language-dependency. Different rule-sets, dictionaries, and algorithms per language.

Tokenization

Definition 2 (Token Normalization)

Given a token (i.e., a word-like unit), map it to a "basic form" (e.g., its stem, lemma, or morpheme) to which semantically equivalent token variants (e.g., derivations or inflections) are also mapped.

Orthographic normalization:

- Case folding. Lower-casing of all tokens.
  "Apple" → "apple": Apple (fruit) at beginning of sentence vs. Apple (company).
- Unicode normalization. Mapping Unicode characters to their closest ASCII equivalent.
  "résumé" → "resume": resume (curriculum vitae) vs. resume (continue).
- Character class reduction. Removal of non-alphanumeric characters.
  "C++" → "C" or "general-purpose" → "generalpurpose": Names, acronyms, and compounds can be distorted.
- Spelling correction. Correction of mistakes and minor variations due to dialect.
  "devops" → "doves": overcorrection of, e.g., names, memes, or technical jargon.

Tokenization

Definition 2 (Token Normalization)

Given a token (i.e., a word-like unit), map it to a "basic form" (e.g., its stem, lemma, or morpheme) to which semantically equivalent token variants (e.g., derivations or inflections) are also mapped.

Morphological analysis:

- Lemmatization. Dictionary-based lookup of the canonical form, the lemma, of a token.
  Dictionaries and mappings need to be assembled for each language.
- Stemming. Rule-based reduction to inflectional base, the stem, of a token.
  "universe" and "university" $\rightarrow$ "univers": Stemming may map unrelated tokens to the same stem.

Lexical-semantic normalization:

- Synonym mapping. Selection of a representative of lemmas with equivalent meanings.
  Typically requires synonym dictionaries, e.g., WordNet or BabelNet.
- Paraphrase mapping. Identification and mapping of paraphrases of a token.

Tokenization

Word Tokenization

Properties:

- Top-down analysis of surface text structure.
- Heuristic operationalization of a linguist's morphological analysis.
- Results in human-interpretable tokens.
- Language-specific knowledge and resources are required: rule sets and dictionaries.
- The vocabulary of tokens grows according to Heap's law.
- Token normalization is an error-prone process.

Applicability:

- Linguistic corpus construction
- Bag of words models and sparse vector representations

Tokenization

Subword Tokenization

Task:

- Learn tokenization rules based on frequently occurring character sequences in a corpus D.

Approaches:

- Byte-Pair Encoding (BPE)
- WordPiece
- Unigram tokenization

Tokenization

Byte-Pair Encoding (BPE)

Byte-Pair Encoding: [Gage 1994] [illustration]

> "The algorithm compresses data by finding the most frequently occurring pairs of adjacent bytes in the data and replacing all instances of the pair with a byte that was not in the original data. The algorithm repeats this process until no further compression is possible, either because there are no more frequently occurring pairs or there are no more unused bytes to represent pairs. The algorithm writes out the table of pair substitutions before the packed data."

Learning tokenization rules: [Sennrich et al., 2015]

- Process token sequences of a corpus D instead of byte sequences.
- Starting point: Every character in the alphabet Σ is a token.
- Iteratively identify the most frequent adjacent token pair, and use them as merging rule.

Remarks:

- Byte-pair encoding has originally been proposed as a data compression algorithm. Sennrich et al., 2015, has adapted it as a heuristic for corpus-relative tokenization. The motivation to do so was to avoid out-of-vocabulary tokens in machine translation, which frequently cannot be translated. By falling back on subword tokens, similar to stems, lemmas, or morphemes, unknown words were represented by substrings, based on the intuition that translators often can form an idea of how to translate an unknown word based on substrings that happen to be similar to known words.

Tokenization

Byte-Pair Encoding (BPE)

Algorithm: Byte-Pair Encoding Tokenizer.

Input: d. A sequence of characters $(c_1, \ldots, c_n)$ over alphabet Σ, the document.

R. A sequence of token pair replacement "rules" $(r_1, \ldots, r_m)$, where $r_i = (t_\mathsf{L}, t_\mathsf{R})$ and $t_\mathsf{L}, t_\mathsf{R} \in V \subseteq \Sigma^*$.

Output: The document as token sequence $(t_1, t_2, \ldots)$.

BPE_tokenize(d, R)

1. $d' =$ *pre_tokenize*(d) `// Sequence of word-like tokens in` d `as character sequences.`
2. **FOR** $i = 1$ **TO** m **DO**
3. $\quad (t_\mathsf{L}, t_\mathsf{R}) = r_i$
4. $\quad d' =$ *replace_all*$(\,(t_\mathsf{L}, t_\mathsf{R}),\ \ t_\mathsf{L} + t_\mathsf{R},\ \ d'\,)$
5. *return*(d')

Tokenization

Byte-Pair Encoding (BPE)

Algorithm: Byte-Pair Encoding Training.

Input: D. A set of sequences of characters over alphabet Σ; the corpus of documents.
k. The maximal size of the vocabulary V, $|V| \leq k$.

Output: V. Set of tokens, where $t \in V \subseteq \Sigma^*$; the vocabulary.
R. Sequence of token pair replacement rules.

BPE_train(D, k)

1. $V = \Sigma$, $R = \emptyset$
2. $D' =$ *pre_tokenize*(D) // Sequence of word-like tokens in D as character sequences.
3. $I =$ *index*(D') // Dictionary that maps tokens t to their token frequency *tf*(t, D).
4. **FOR** $i = 1$ **TO** k **DO**
5. $t_L, t_R =$ *most_frequent_token_sequence*(V, I).
6. $R = R + (t_L, t_R)$
7. $V = V \cup \{ t_L + t_R \}$
8. **FOREACH** $t \in I$ **DO** $t =$ *replace_all*$(\ (t_L, t_R),\ \ t_L + t_R,\ \ t\ )$
9. *return*(V, R)

Remarks:

- *replace_all*(s_1, s_2, s) replaces every occurrence of sequence s_1 with sequence s_2 in sequence s.
- *pre_tokenize*(D) returns the corpus as a sequence of "pre-tokens" D'.
  1. Use of a (naïve) tokenizer for every $d \in D$ and concatenation of the token sequences to obtain D'.

     `My kingdom for a horse!` → `My` `kingdom` `for` `a` `horse` `!`

  2. Typically, a word boundary character $c_{\text{word}} \notin D$ is added as prefix to every token. For languages that indicate a word boundary with a space, the last space ahead of a token is replaced with c_{word}. Adding other boundary characters is a design choice (e.g., for newlines, new paragraphs, new sentences, etc.).

     `My` `kingdom` `for` `a` `horse` `!` → `My` `␣kingdom` `␣for` `␣a` `␣horse` `!`

  3. Each token is split into a sequence of characters.

     `My` `␣kingdom` `␣for` `␣a` `␣horse` `!` → `M y` `␣k i n g d o m` `␣f o r` `␣a` `␣h o r s e` `!`

- Here, a "sequence of characters" refers to individual characters, instead of a string (character sequence). In practice, allocating and merging many immutable string objects is avoided. Instead of splitting and merging strings, the initial pre-token strings (word-like units) are doubled in length and a space character is added before each character. Upon each merging using *replace_all* of two adjacent tokens (initially characters), the space between the pair of tokens is removed.

Remarks: (continued)

- *most_frequent_token_sequence*(V, I) determines the most frequently occurring sequence of tokens $(\, t_{\mathsf{L}}, t_{\mathsf{R}} \,)$ from V across all tokens in I.
- To speed up the the determination of which pair of tokens from V occurs most frequently in D', representing D' as an index I that maps tokens $t \in D'$ to their frequencies of occurrence in D can be utilized.
- An alternative approach is to employ a trie data structure. Upon each merge of two adjacent tokens, the vocabulary I or the index or trie representing it needs to be iteratively updated to reflect the change in token counts.

Tokenization

Byte-Pair Encoding (BPE)

Example: $D = \{$ “A horse, a horse, my kingdom for a horse!” $\}$

$R:$ | k | $(t_{\mathsf{L}}, t_{\mathsf{R}})$ |
|---|---|

$V = \{$

$I = \{$

$\}$

$\}$

Tokenization

Byte-Pair Encoding (BPE)

Example: $D = \{$ “A horse, a horse, my kingdom for a horse!” $\}$

R :

k	(t_L, t_R)

$V = \{$, , !, ␣a, ␣f, ␣h, ␣k, ␣m,
A, d, e, g, h, i,
k, m, n, o, r, s, y,
$\}$

$I = \{$ (A; 1), (␣a; 2), (␣f, o, r; 1),
(␣h, o, r, s, e; 3),
(␣k, i, n, g, d, o, m; 1),
(␣m, y; 1), (,; 2), (!; 1) $\}$

D': ((A), (␣h, o, r, s, e), (,), (␣a), (␣h, o, r, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

Tokenization

Byte-Pair Encoding (BPE)

Example: $D = \{$ “A horse, a horse, my kingdom for a horse!” $\}$

R :

k	(t_L, t_R)
1:	(o, r)

$V = \{$, , !, ␣a, ␣f, ␣h, ␣k, ␣m,
A, d, e, g, h, i,
k, m, n, o, r, s, y,
or, $\}$

$I = \{$ (A; 1), (␣a; 2), (␣f, or; 1),
(␣h, or, s, e; 3),
(␣k, i, n, g, d, o, m; 1),
(␣m, y; 1), (,; 2), (!; 1) $\}$

D': ((A), (␣h, o, r, s, e), (,), (␣a), (␣h, o, r, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

Tokenization

Byte-Pair Encoding (BPE)

Example: $D =$ { “A horse, a horse, my kingdom for a horse!” }

R :

k	(t_L, t_R)
1:	(o, r)
2:	(␣h, or)

$V = \{$, , !, ␣a, ␣f, ␣h, ␣k, ␣m, A, d, e, g, h, i, k, m, n, o, r, s, y, ␣hor, or, $\}$

$I = \{$ (A; 1), (␣a; 2), (␣f, or; 1), (␣hor, s, e; 3), (␣k, i, n, g, d, o, m; 1), (␣m, y; 1), (, ; 2), (!; 1) $\}$

D': ((A), (␣h, o, r, s, e), (,), (␣a), (␣h, o, r, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

Tokenization

Byte-Pair Encoding (BPE)

Example: $D = \{$ “A horse, a horse, my kingdom for a horse!” $\}$

R:

k	(t_L, t_R)
1:	(o, r)
2:	(␣h, or)
3:	(␣hor, s)

$V = \{$,, !, ␣a, ␣f, ␣h, ␣k, ␣m, A, d, e, g, h, i, k, m, n, o, r, s, y, ␣hor, ␣hors, or, $\}$

$I = \{$ (A; 1), (␣a; 2), (␣f, or; 1), (␣hors, e; 3), (␣k, i, n, g, d, o, m; 1), (␣m, y; 1), (,; 2), (!; 1) $\}$

D': ((A), (␣h, o, r, s, e), (,), (␣a), (␣h, o, r, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

Tokenization

Byte-Pair Encoding (BPE)

Example: $D = \{$ “A horse, a horse, my kingdom for a horse!” $\}$

R:

k	(t_L, t_R)
1:	(o, r)
2:	(␣h, or)
3:	(␣hor, s)
	⋮

$V = \{$,, !, ␣a, ␣f, ␣h, ␣k, ␣m, A, d, e, g, h, i, k, m, n, o, r, s, y, dom, ␣for, ␣hor, ␣hors, ␣horse, ␣king, ␣my, or, ...$\}$

$I = \{$ (A; 1), (␣a; 2), (␣for; 1), (␣horse; 3), (␣king, dom; 1), (␣my; 1), (,; 2), (!; 1) $\}$

D': ((A), (␣h, o, r, s, e), (,), (␣a), (␣h, o, r, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

Tokenization

Byte-Pair Encoding (BPE)

Example: $d =$ “A horse, a horse, my kingdom for a horse!”

R :

k	(t_L, t_R)
1:	(o, r)
2:	(␣h, or)
3:	(␣hor, s)
	⋮

$V = \{$,, !, ␣a, ␣f, ␣h, ␣k, ␣m,
A, d, e, g, h, i,
k, m, n, o, r, s, y,
dom, ␣for, ␣hor, ␣hors,
␣horse, ␣king, ␣my, or, ...$\}$

$I = \{$ (A; 1), (␣a; 2), (␣for; 1),
(␣horse; 3),
(␣king, dom; 1),
(␣my; 1), (,; 2), (!; 1) $\}$

d': ((A), (␣h, o, r, s, e), (,), (␣a), (␣h, o, r, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

Tokenization

Byte-Pair Encoding (BPE)

Example: $d =$ “A horse, a horse, my kingdom for a horse!”

R :

k	(t_L, t_R)
1:	(o, r)
2:	(␣h, or)
3:	(␣hor, s)
	⋮

$V = \{$,, !, ␣a, ␣f, ␣h, ␣k, ␣m,
A, d, e, g, h, i,
k, m, n, o, r, s, y,
dom, ␣for, ␣hor, ␣hors,
␣horse, ␣king, ␣my, or, ...$\}$

$I = \{$ (A; 1), (␣a; 2), (␣for; 1),
(␣horse; 3),
(␣king, dom; 1),
(␣my; 1), (,; 2), (!; 1) $\}$

d': ((A), (␣h, o, r, s, e), (,), (␣a), (␣h, o, r, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

1: ((A), (␣h, or, s, e), (,), (␣a), (␣h, or, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, or), (␣a), (␣h, or, s, e), (!))

Tokenization

Byte-Pair Encoding (BPE)

Example: $d =$ “A horse, a horse, my kingdom for a horse!”

R:

k	(t_L, t_R)
1:	(o, r)
2:	(␣h, or)
3:	(␣hor, s)
	⋮

$V = \{$,, !, ␣a, ␣f, ␣h, ␣k, ␣m,
A, d, e, g, h, i,
k, m, n, o, r, s, y,
dom, ␣for, ␣hor, ␣hors,
␣horse, ␣king, ␣my, or, ...$\}$

$I = \{$ (A; 1), (␣a; 2), (␣for; 1),
(␣horse; 3),
(␣king, dom; 1),
(␣my; 1), (,; 2), (!; 1) $\}$

d': ((A), (␣h, o, r, s, e), (,), (␣a), (␣h, o, r, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

1: ((A), (␣h, or, s, e), (,), (␣a), (␣h, or, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, or), (␣a), (␣h, or, s, e), (!))

2: ((A), (␣hor, s, e), (,), (␣a), (␣hor, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, or), (␣a), (␣hor, s, e), (!))

Tokenization

Byte-Pair Encoding (BPE)

Example: $d =$ “A horse, a horse, my kingdom for a horse!”

R:

k	(t_L, t_R)
1:	(o, r)
2:	(␣h, or)
3:	(␣hor, s)
	⋮

$V = \{$,, !, ␣a, ␣f, ␣h, ␣k, ␣m, A, d, e, g, h, i, k, m, n, o, r, s, y, dom, ␣for, ␣hor, ␣hors, ␣horse, ␣king, ␣my, or, ...$\}$

$I = \{$ (A; 1), (␣a; 2), (␣for; 1), (␣horse; 3), (␣king, dom; 1), (␣my; 1), (,; 2), (!; 1) $\}$

d': ((A), (␣h, o, r, s, e), (,), (␣a), (␣h, o, r, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

1: ((A), (␣h, or, s, e), (,), (␣a), (␣h, or, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, or), (␣a), (␣h, or, s, e), (!))

2: ((A), (␣hor, s, e), (,), (␣a), (␣hor, s, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, or), (␣a), (␣hor, s, e), (!))

3: ((A), (␣hors, e), (,), (␣a), (␣hors, e), (,), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, or), (␣a), (␣hors, e), (!))

Tokenization

Byte-Pair Encoding (BPE)

Example: $d =$ “A horse, a horse, my kingdom for a horse!”

R:

k	(t_L, t_R)
1:	(o, r)
2:	(␣h, or)
3:	(␣hor, s)
	⋮

$V = \{$ ␣, !, ␣a, ␣f, ␣h, ␣k, ␣m, A, d, e, g, h, i, k, m, n, o, r, s, y, dom, ␣for, ␣hor, ␣hors, ␣horse, ␣king, ␣my, or, ...$\}$

$I = \{$ (A; 1), (␣a; 2), (␣for; 1), (␣horse; 3), (␣king, dom; 1), (␣my; 1), (␣; 2), (!; 1) $\}$

d': ((A), (␣h, o, r, s, e), (␣), (␣a), (␣h, o, r, s, e), (␣), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, o, r), (␣a), (␣h, o, r, s, e), (!))

1: ((A), (␣h, or, s, e), (␣), (␣a), (␣h, or, s, e), (␣), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, or), (␣a), (␣h, or, s, e), (!))

2: ((A), (␣hor, s, e), (␣), (␣a), (␣hor, s, e), (␣), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, or), (␣a), (␣hor, s, e), (!))

3: ((A), (␣hors, e), (␣), (␣a), (␣hors, e), (␣), (␣m, y), (␣k, i, n, g, d, o, m), (␣f, or), (␣a), (␣hors, e), (!))

⋮

k: ((A), (␣horse), (␣), (␣a), (␣horse), (␣), (␣my), (␣king), (dom), (␣for), (␣a), (␣horse), (!))

Tokenization

Subword Tokenization

Properties:

- Bottom-up analysis of texts.
- Naïve heuristic to identify (frequent) morphemes.
- Results in partly human-interpretable tokens, partly non-interpretable ones.
- Language-specific knowledge and resources may be required: pre-tokenization heuristic.
- The vocabulary of tokens can be bounded.
- Token normalization is optional and depends on the application. [HuggingFace]

Applicability:

- Computing document embeddings in neural networks with (GPU) memory limitations.

Remarks:

- If the tokenizer is used on a document which contains characters that do not occur in the alphabet Σ, no tokens can be formed: This is why a special token for unknown (out-of-vocabulary, OOV) tokens, `[unk]` is often introduced as a catch-all token for such and similar cases.
- A variant of Byte-Pair Encoding (BPE) is used by GPT-2: byte-level BPE. It uses all 256 different bytes as basis vocabulary to avoid the `[unk]` token completely.
- BERT and many of its variants use WordPiece, which is an extension of BPE that uses a different function to find the most likely merge, instead of the most frequent adjacent tokens. This avoids merging subwords that also often appear independently.
- The tokenizers Unigram [Kudo, 2018] and SentencePiece [Kudo and Richardson, 2018] work in reverse to WordPiece: they add all possible tokens to the vocabulary, then iteratively remove tokens until the desired vocabulary size is reached.
- Providers of closed-source language models often share their tokenizer implementation open source, because customers need to be able to reliably calculate the number of tokens they may process in order to reliably estimate usage costs. [OpenAI tiktoken]

Tokenization

Token Removal

Undesired tokens (so-called stop words) are removed (process called stopping) to reduce data size, sparsity, and to improve performance on downstream tasks.

- Frequent tokens (per corpus)
  E.g., remove one occurrence of `Wikipedia` from each Wikipedia article. Why not remove all occurrences?
- Function word tokens (per language)
  E.g., `the, of, and,` etc.; but what happens to a phrase like `to be or not to be` ?
- Punctuation-only tokens
  This would remove sequences like '`;-)`', which are often meaningful.
- Number-only tokens
- Short tokens
  `xp, ma, pm, ben e king, el paso, master p, gm, j lo, ...`

Stop words are often collected in domain-specific lists. [Terrier stopword list]

Tokenization

Token Removal (continued)

Source text: (34 tokens)

```
The idea of giving computers the ability to process human language is as old as the
idea of computers themselves.  This book is about the implementation and implications
of that exciting idea.
```

Stopped text: (16 tokens)

```
The idea of giving computers the ability to process human language is as old as the
idea of computers themselves.  This book is about the implementation and implications
of that exciting idea.
```

Tokenization

Token Removal (continued)

Source text: (34 tokens)

```
The idea of giving computers the ability to process human language is as old as the
idea of computers themselves.  This book is about the implementation and implications
of that exciting idea.
```

Stopped text: (16 tokens)

```
idea giving computers ability process human language old idea computers themselves
book implementation implications exciting idea
```