Chapter NLP:III

III. Text Models

- Text Preprocessing
- Text Representation
- Text Similarity
- Text Classification
- Language Modeling
- Sequence Modeling

Text Preprocessing

Overview

The goal of text preprocessing is its conversion into a canonical form.

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

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.

Machine translation

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:

Question answering

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


```
P R E L I M I N A R Y P R O O F S.
c
Unpublished Work �2008
by Pearson Education, Inc. To be published by Pearson Prentice Hall,
Pearson Education, Inc., Upper Saddle River, New Jersey. All rights reserved. Permission to use
this unpublished Work is granted to individuals registering through Melinda_Haggerty@prenhall.com
for the instructional purposes not exceeding one academic term or semester.

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

FT

D
RA

Conversational
agent

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.
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).
Let's turn to another useful language–related task, that of making available to nonEnglish–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.
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:

Dialogue system

Machine
translation

Question
answering

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

```

Text Preprocessing

Overview

The goal of text preprocessing is its conversion into a canonical form.

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

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.

Machine translation

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:

Question answering

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

screenshot-jurafsky08-speech-and-language-processing-cleaned.txt

Open | Save

```
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.

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).

Let's turn to another useful language-related task, that of making available to nonEnglish-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.

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?
```

Plain Text | Tab Width: 2 | Ln 14, Col 1 | INS

Text Preprocessing

Overview

Preprocessing is required to:

- Normalized text for subsequent processing

  Example: Preprocessing extracts HTML text from PDFs, so an indexing pipeline of a search engine is only implemented for HTML documents.

- Reduce language variety

  Example: Preprocessing corrects spelling mistakes to reduce vocabulary dimensionality.

- Avoid processing errors and model bias

  Example: Preprocessing removes artifacts from PDF conversion, so a classification model can learn from the text alone.

Text Preprocessing

Overview

Preprocessing is required to:

- Normalized text for subsequent processing

  Example: Preprocessing extracts HTML text from PDFs, so an indexing pipeline of a search engine is only implemented for HTML documents.

- Reduce language variety

  Example: Preprocessing corrects spelling mistakes to reduce vocabulary dimensionality.

- Avoid processing errors and model bias

  Example: Preprocessing removes artifacts from PDF conversion, so a classification model can learn from the text alone.

Constraints:

Task-dependence	Preprocessing depends on the task and source documents.
Provenance	Determine where a preprocessed text was in a raw corpus.
Reversibility	Render a preprocessed text in a human-readable form.

Text Preprocessing

Preprocessing Pipeline

Typical steps in a preprocessing pipeline:

1. **Extraction** and conversion to plain text.
   - ❑ Encoding detection and unification
   - ❑ Line break unification (`\n` in UNIX *vs.* `\r\n` in Windows)
   - ❑ Extraction of main content and meta information
2. **Text normalization**.
   - ❑ Canonicalization Prune whitespace, check spelling and grammar, . . .
   - ❑ Expansion and/or abstraction Expand abbreviations, translate, . . .
3. **Tokenization**. Segmenting text into paragraphs, sentences, (sub)words . . .
4. **Annotation** of basic text and document features.
   - ❑ Syntactic units: phonemes, morphemes, words, sentences
   - ❑ Discourse units: paragraphs, sections, chapters
   - ❑ Typographic units: lines, pages (layout, meta-information), documents
   - ❑ Meta-information: title, authors, date, properties, ...

Remarks:

- Annotation is skipped when the annotations are not needed for further processing.
- Extraction is skipped when the data is already created and collected as plain text.
- Text normalization is sometimes undesirable when the non-normality of the text is relevant for the task, like stylometric markers are for authorship or dialects are for computational social science.

Text Preprocessing
Token Normalization

Application of heuristic rules to each token in an attempt to unify them.

- Lower-casing
  Problem: Capitalization may carry distinctions between word semantics.
  Examples: `Bush` vs. `bush`, `Apple` vs. `apple`.
- Removal of special characters
  Example: `U.S.A.` → `USA`
- Removal of diacritical marks
  Example: `café` → `cafe`
- Spelling correction
  Example: `My gramma got die of beaties` → `My grandma got diabetes`
- Reduction of morphology
  Lemmatization or stemming heuristics

Text Preprocessing

Token Normalization: Regular Expressions [WT:IV-86 ff., NLP:V-12 ff.]

Token normalization is often done with (sophisticated) regular expressions (regex).

- A regex defines a regular grammar over an alphabet Σ through a sequence of characters and metacharacters. They are a form of programming language to describe finite automata.
- A regex can be used to describe general structures of a language and find spans of text that match the description.

Token Normalization: Regular Expressions [WT:IV-86 ff., NLP:V-12 ff.]

Token normalization is often done with (sophisticated) regular expressions (regex).

- Regular characters denote the terminal symbols from the alphabet Σ. Without metacharactes, they literally match characters in a string.

  `the` matches `the`

- Metacharacters encode constructs like disjunctions or negations.

  `[tT]` matches `T` or `t`

  `[^a-zA-Z]` matches any character that is not a letter

- In regex syntax, non-terminal symbols and production rules are directly encoded in the expression using regular and metacharacters:

  `[tT]` $\Leftrightarrow$ `[tT]` $\rightarrow$ `t | T`

Text Preprocessing

Token Normalization: Regular Expressions [WT:IV-86 ff., NLP:V-12 ff.]

Token normalization is often done with (sophisticated) regular expressions (regex).

- Regular characters denote the terminal symbols from the alphabet Σ. Without metacharactes, they literally match characters in a string.

  `the` matches `the`

- Metacharacters encode constructs like disjunctions or negations.

  `[tT]` matches `T` or `t`

  `[^a-zA-Z]` matches any character that is not a letter

- In regex syntax, non-terminal symbols and production rules are directly encoded in the expression using regular and metacharacters:

  `[tT]` $\Leftrightarrow$ `[tT]` $\rightarrow$ `t | T`

Two regex for (all) instances of `the` in a text:

Regex	`the`	`The`	`atheist`
`the`	x	–	x
`[^a-zA-Z][tT]he[^a-zA-Z]`	x	x	–

Token Normalization: Regular Expressions (continued)

Character Classes.

- Brackets `[]` specify a character class.

  `[wod]` matches `w` or `o` or `d`

  `[wW]` matches `w` or `W`

- Disjunctive ranges of characters can be specified with a hyphen `-`.

  `[a-zA-Z]` matches any letter

  `[0-8]` matches any digit except for `9`

- Several common classes are predefined.

  `\d` matches any decimal digit ⇔ `[0-9]`.
  `\D` matches any non-digit character ⇔ `[^0-9]`.
  `\s` matches any whitespace character ⇔ `[\t\n\r\f\v]`.
  `\S` matches any non-whitespace character ⇔ `[^\t\n\r\f\v]`.
  `\w` matches any alphanumeric character ⇔ `[a-zA-Z0-9]`.
  `\W` matches any non-alphanumeric character ⇔ `[^a-zA-Z0-9]`.

Text Preprocessing

Token Normalization: Regular Expressions (continued)

Negation.

- The caret `[^]` inside brackets negates the specified character class.

  `[^0-9]` matches anything except digits

  `[^wo]` matches any character except `w` and `o`

- Outside brackets, the caret `^` is interpreted as a regular character.

  `woodchuck^` matches `woodchuck^`

OR.

- The pipe `|` specifies a boolean OR-disjunction of string sequences.

  `groundhog|woodchuck` matches `groundhog` or `woodchuck`

- Character classes are equivalent to OR-concatenated strings of characters.

  `[a-d]` $\Leftrightarrow$ `a|b|c|d`

Text Preprocessing

Token Normalization: Regular Expressions (continued)

Wildcards.

- The period `.` matches any character. Match literal periods via escape `\.`

  `w..dchuck` matches `woodchuck`, `weedchuck`, ...

- The asterisk `*` repeats the previous character zero or more times.

  `wo*dchuck` matches `wdchuck`, `wodchuck`, `wooooodchuck`, ...

- The plus `+` repeats the previous character one or more times.

  `wo+dchuck` matches `wodchuck`, `wooooodchuck`, ...

- The question mark `?` repeats the previous character zero or one time.

  `woo?dchuck` matches `wodchuck` and `woodchuck`

- Curly brackets `{n,m}` specify the number of repetitions.

  `wo{2,3}dchuck` matches `woodchuck` and `wooodchuck`

Text Preprocessing

Token Normalization: Regular Expressions (continued)

Grouping.

- Parentheses `()` can be used to (semantically) group parts of a regex.

  `w(ood)*chuck` matches `wchuck`, `woodoodchuck`, ...

- The part of the string that matches the group can later be backreferenced.

  s/`([0-9])`/`␣$1`/**g** replaces any number with a space and the matched number

Token Normalization: Regular Expressions (continued)

Combining Metacharacters

- Match many different woodchucks.

  ```
  [wW][oO][oO]+[dD][cC][hH][uU][cC][kK][sS]? | groundhog
  ```

- Match email addresses, excluding whose with special characters.

  ```
  [\w]+@[\w][\w]+(\.[\w]+)*\.(de|org|net)
  ```

- Match time expressions?

  ```
  August 25th, 2022
  in the next five years
  2023-03-03T12:56:51Z
  ```

Text Preprocessing

Token Normalization: Regular Expressions (continued)

Complete regular expressions to parse time expressions (1/2):

```
((((([iI]n|[wW]ithin|[tT]o\s\s?the|[tT]o|[fF]or\s\s?the|[fF]or|[fF]rom|[sS]ince|[aA]fter|[bB]efore|[bB]etween|[aA]t|[oO]n|[oO]ver|[pP
]er)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]he|[tT]his|[tT]hese|[tT]hose|[iI]ts))?)(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?(((0?[123456789
]|[12]\d|3[01])(\.|/))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?)(((month|time(span)?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(from(\s+(\r(\n)?|\n)
?|(\r(\n)?|\n))\s*)?)?([Jj]anuary|[Jj]an\.|[Jj]an|[Ff]ebruary|[Ff]eb\.|[Ff]eb|[Mm]arch|[Mm]ar\.|[Mm]ar|[Aa]pril|[Aa]pr\.|[Aa]pr|[Mm]a
y|[Jj]une|[Jj]un\.|[Jj]un|[Jj]uly|[Jj]ul\.|[Jj]ul|[Aa]ugust|[Aa]ug\.|[Aa]ug|[Ss]eptember|[Ss]ep\.|[Ss]ep|[Oo]ctober|[Oo]ct\.|[Oo]ct|[
Nn]ovember|[Nn]ov\.|[Nn]ov|[Dd]ecember|[Dd]ez\.|[Dd]ez|[Ss]pring|[Ss]ummer|[Aa]utumn|[Ff]all|[Ww]inter))|((0?[123456789]|1[012])(\.|/
)))(()?((19|20)?\d2))?)|((((([iI]n|[wW]ithin|[tT]o\s\s?the|[tT]o|[fF]or\s\s?the|[fF]or|[fF]rom|[sS]ince|[aA]fter|[bB]efore|[bB]etween
|[aA]t|[oO]n|[oO]ver|[pP]er)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]he|[tT]his|[tT]hese|[tT]hose|[iI]ts))?)(\s+(\r(\n)?|\n)?|(\r(\n)?|
\n))\s*(((([sS]tart|[bB]egin|[Ss]tart|[Bb]egin|[Ee]nd|[eE]nd|[Mm]idth|[mM]idth)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]he|[tT]his|[tT]h
ese|[tT]hose|[iI]ts))?)(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?|(([sS]tart|[bB]egin|[Ss]tart|[Bb]egin|[Ee]nd|[eE]nd|[Mm]idth|[mM]idth)((\s
+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]he|[tT]his|[tT]hese|[tT]hose|[iI]ts))?)(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)([[a-z]]+(\s+(\r(\n)?|\n)
?|(\r(\n)?|\n))\s*)?)?((((([lL]ast|[pP]receding|[pP]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext))((\s+(\r(
\n)?|\n)?|(\r(\n)?|\n))\s*(((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even
|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|
[hH]undred|[aA]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[
eE]ighth|[nN]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|
[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[s
S]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]our
ty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred)))?))?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)|(((
(1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE
]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred)
)|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[
eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]int
h|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hre
e|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixt
y|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred)))?)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(([lL]ast|[pP]receding|[pP]ast|[cC
]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext)))?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*))?(((Q(1|2|3|4)|H(1|2)(\/(19|20)?
\d2)?|(((\w([a-z])*(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?(year|quarter))([a-z])*))|((month|time(span)?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(
from(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?)?([Jj]anuary|[Jj]an\.|[Jj]an|[Ff]ebruary|[Ff]eb\.|[Ff]eb|[Mm]arch|[Mm]ar\.|[Mm]ar|[Aa]pril|[A
a]pr\.|[Aa]pr|[Mm]ay|[Jj]une|[Jj]un\.|[Jj]un|[Jj]uly|[Jj]ul\.|[Jj]ul|[Aa]ugust|[Aa]ug\.|[Aa]ug|[Ss]eptember|[Ss]ep\.|[Ss]ep|[Oo]ctobe
r|[Oo]ct\.|[Oo]ct|[Nn]ovember|[Nn]ov\.|[Nn]ov|[Dd]ecember|[Dd]ez\.|[Dd]ez|[Ss]pring|[Ss]ummer|[Aa]utumn|[Ff]all|[Ww]inter))|(([Rr]epo
rted\s\s?time\s\s?span|[Rr]eported\s\s?time\s\s?span|[Rr]eported\s\s?time|[rR]eported\s\s?time|[Tt]ime\s\s?span|[tT]ime\s\s?span|[Ss]p
an|[sS]pan|[Dd]ecade|[dD]ecade)))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((19|20)\d2(/(19|20)?\d2)?|\d2/\d2))?|((19|20)\d2(/(19|20)?\d2)?|\
d2/\d2)))|((((([lL]ast|[pP]receding|[pP]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext))((\s+(\r(\n)?|\n)?|(\r
(\n)?|\n))\s*(((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN
]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[a
A]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]
inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[e
E]ighth|[nN]inth|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth
|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[
fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|
```

Text Preprocessing

Token Normalization: Regular Expressions (continued)

Complete regular expressions to parse time expressions (2/2):

```
[aA]\s\s?hundred)))?))?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)|((((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hre
e|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixt
y|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]our
th|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]
hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5
|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT
]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred)))?)((\s+(
\r(\n)?|\n)?|(\r(\n)?|\n))\s*(([lL]ast|[pP]receding|[pP]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext)))?(\s+
(\r(\n)?|\n)?|(\r(\n)?|\n))\s*))?(((Q(1|2|3|4)|H(1|2)(\/(19|20)?\d2)?|(((\w([a-z])*(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?(year|quarter))(
[a-z])*))|((month|time(span)?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(from(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?)?([Jj]anuary|[Jj]an\.|[Jj]an|
[Ff]ebruary|[Ff]eb\.|[Ff]eb|[Mm]arch|[Mm]ar\.|[Mm]ar|[Aa]pril|[Aa]pr\.|[Aa]pr|[Mm]ay|[Jj]une|[Jj]un\.|[Jj]un|[Jj]uly|[Jj]ul\.|[Jj]ul|
[Aa]ugust|[Aa]ug\.|[Aa]ug|[Ss]eptember|[Ss]ep\.|[Ss]ep|[Oo]ctober|[Oo]ct\.|[Oo]ct|[Nn]ovember|[Nn]ov\.|[Nn]ov|[Dd]ecember|[Dd]ez\.|[D
d]ez|[Ss]pring|[Ss]ummer|[Aa]utumn|[Ff]all|[Ww]inter))|(([Rr]eported\s\s?time\s\s?span|[Rr]eported\s\s?time\s\s?span|[Rr]eported\s\s?t
ime|[rR]eported\s\s?time|[Tt]ime\s\s?span|[tT]ime\s\s?span|[Ss]pan|[sS]pan|[Dd]ecade|[dD]ecade)))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((
19|20)\d2(/(19|20)?\d2)?|\d2/\d2))?|((19|20)\d2(/(19|20)?\d2)?|\d2/\d2))))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]o|[aA]nd|[oO]r|[oO]n
|[aA]t|[oO]f\s\s?the|[oO]f|[tT]he|[tT]his|[iI]ts|[iI]nstead\s\s?of)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(([sS]tart|[bB]egin|[Ss]tart|[Bb
]egin|[Ee]nd|[eE]nd|[Mm]idth|[mM]idth)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]he|[tT]his|[tT]hese|[tT]hose|[iI]ts))?))?((\s+(\r(\n)?|\
n)?|(\r(\n)?|\n))\s*[[a-z]]+)?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(((((([lL]ast|[pP]receding|[pP]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]o
llowing|[sS]ucceeding|[nN]ext))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[
tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[
sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[
fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econ
d|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2
|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]lev
en|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred)))?)
)?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)|((((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS
]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ight
y|[nN]inety|[hH]undred|[aA]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|
[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]if
th|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[
sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]h
irty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred)))?)((\s+(\r(\n)?|\n)?|(\r(\n)?|\
n))\s*(([lL]ast|[pP]receding|[pP]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext)))?(\s+(\r(\n)?|\n)?|(\r(\n)?|
\n))\s*))?(((Q(1|2|3|4)|H(1|2)(\/(19|20)?\d2)?|(((\w([a-z])*(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?(year|quarter))([a-z])*))|((month|time
(span)?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(from(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?)?([Jj]anuary|[Jj]an\.|[Jj]an|[Ff]ebruary|[Ff]eb\.|[F
f]eb|[Mm]arch|[Mm]ar\.|[Mm]ar|[Aa]pril|[Aa]pr\.|[Aa]pr|[Mm]ay|[Jj]une|[Jj]un\.|[Jj]un|[Jj]uly|[Jj]ul\.|[Jj]ul|[Aa]ugust|[Aa]ug\.|[Aa]
ug|[Ss]eptember|[Ss]ep\.|[Ss]ep|[Oo]ctober|[Oo]ct\.|[Oo]ct|[Nn]ovember|[Nn]ov\.|[Nn]ov|[Dd]ecember|[Dd]ez\.|[Dd]ez|[Ss]pring|[Ss]umme
r|[Aa]utumn|[Ff]all|[Ww]inter))|(([Rr]eported\s\s?time\s\s?span|[Rr]eported\s\s?time\s\s?span|[Rr]eported\s\s?time|[rR]eported\s\s?tim
e|[Tt]ime\s\s?span|[tT]ime\s\s?span|[Ss]pan|[sS]pan|[Dd]ecade|[dD]ecade)))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((19|20)\d2(/(19|20)?\d2)
?|\d2/\d2))?|((19|20)\d2(/(19|20)?\d2)?|\d2/\d2))))*))
```

Text Preprocessing

Token Normalization: Regular Expressions Summary

<table>
<tr><th>Char</th><th>Concept</th><th>Example</th></tr>
<tr><td>[ ]</td><td>Character Classes</td><td>[Ww]oodchuck</td></tr>
<tr><td>-</td><td>Ranges in classes</td><td>There are [0-9]+ woodchucks\.</td></tr>
<tr><td>|</td><td>Disjunction of regexes</td><td>woodchuck|groundhog</td></tr>
<tr><td>^</td><td>Negation</td><td>[^0-9]</td></tr>
<tr><td>.</td><td>Any Character</td><td>What a (.)* woodchuck</td></tr>
<tr><td>()</td><td>Grouping of regex parts</td><td>w(oo)+dchuck</td></tr>
<tr><td>\</td><td>Special (sets of) characters</td><td>\swoodchuck\s</td></tr>
<tr><td>*</td><td>Zero or more repetitions</td><td>wooo*dchuck</td></tr>
<tr><td>+</td><td>One or more repetitions</td><td>woo+dchuck</td></tr>
<tr><td>?</td><td>Zero or one repetition</td><td>woodchucks?</td></tr>
</table>

Text Preprocessing
Tokenization

Tokenization turns a sequence of characters into a sequence of tokens.

Example:

```
Friends, Romans, Countrymen, lend me your ears !
```

$\boxed{\texttt{Friends}}$ $\boxed{\texttt{,}}$ $\boxed{\texttt{Romans}}$ $\boxed{\texttt{,}}$ $\boxed{\texttt{Countrymen}}$ $\boxed{\texttt{,}}$ $\boxed{\texttt{lend}}$ $\boxed{\texttt{me}}$ $\boxed{\texttt{your}}$ $\boxed{\texttt{ears}}$ $\boxed{\texttt{!}}$

Terminology: (simplified)

- A $\boxed{\text{token}}$ is a character sequence forming a useful semantic unit.
- A type is to a token what a class is to an object.

Token-granularity:

- Word-level: may or may not inlcude whitespace between words
- Phrase-level: identification of multi-term named entities and common phrases
- Sentence-level: one token corresponds to one clause, or one sentence

Text Preprocessing

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, a separator, and some words refer to the same concept with or without hyphen: `winston-salem`, `e-bay`, `wal-mart`, `active-x`, `far-reaching`, `loud-mouthed`, `20-year-old`.

- **Compounds**
  English: `wheelchair`, German: `Computerlinguistik` for computational linguistics.

- **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`.

- **Phrase tokens: named entities, phone numbers, dates**
  `San Francisco`, `(800) 234-2333`, `Mar 11,1983`.

Remarks:

- A related philosophical concept is the type-token distinction (see unit about corpus linguistics in this course). Here, a token is a specific instance of a word (i.e., its specific written form), and a type refers to its underlying concept as a whole. This is comparable to the distinction between class and object in object-oriented computer programming. For example, the sentence “A rose is a rose is a rose.” comprises nine token instances but only four types, namely “a”, “rose”, “is”, and “.”. [Wikipedia]

- Tokenization is strongly language-dependent. English is already among the easiest languages to be tokenized, and there are still many problems to be solved. In Chinese, for example, words are not separated by a specific character, rendering the process of determining word boundaries much more difficult.

Text Preprocessing

Tokenization: Approaches

1. Heuristics

   Whitespace: A token is every character sequence separated by whitespace characters.

   TREC: A token is every alphanumeric sequence of characters of length > 3, separated by a space or punctuation mark.

2. Rule-based

   Manually construct a set of rules and apply them in order.

   Each rule describes how to split a string into smaller tokens.

3. Frequency-based

   Split tokens based on observed frequencies in a training corpus.

Text Preprocessing

Tokenization: Rule-based [Jurafsky and Martin, 2007] [Grefenstette, 1999]

Algorithm: Tokenization with Regular Expressions.

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

Output: The document with space in-between its tokens.

Tokenize(d, A)

1. `alnum = [A-Za-z0-9]; nalnum = [^A-Za-z0-9]; alwayssep = [?!()";/\|`]`
2. `clitic = ('|:|-|'S|'D|'M|'LL|'RE|'VE|N'T|'s|'d|'m|'ll|'re|'ve|n't)`
3. // Put whitespace around unambiguous separators.
4. // Put whitespace around commas that aren't inside numbers.
5. // Segment single quotes not preceded by letter (not apostrophes).
6. // Segment unambiguous word-final clitics and punctuation.
7. Split d by whitespace (**/\s+/**) to obtain a list of tokens T.
8. // Segment periods from each $t \in T$ that isn't an abbreviation in A or like one (letter period sequence or letter followed by consonants).
9. // Optionally expand clitics to normalize them.
10. Return a whitespace-separated string of T.

Text Preprocessing

Tokenization: Rule-based [Jurafsky and Martin, 2007] [Grefenstette, 1999]

Algorithm: Tokenization with Regular Expressions.

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

Output: The document with space in-between its tokens.

Tokenize(d, A)

1. `alnum = [A-Za-z0-9]; nalnum = [^A-Za-z0-9]; alwayssep = [?!()";/\|`]`
2. `clitic = ('|:|-|'S|'D|'M|'LL|'RE|'VE|N'T|'s|'d|'m|'ll|'re|'ve|n't)`
3. Apply `s/$alwayssep/␣$&␣/g` to d.
4. Apply `s/([^0-9]),/$1␣,␣/g` and `s/,([^0-9])/␣,␣$1/g` to d.
5. Apply `s/^'/$&␣/g` and `s/($nalnum)'/$1␣'/g` to d.
6. Apply `s/$clitic$/␣$&/g` and `s/$clitic($nalnum)/␣$1␣$2/g` to d.
7. Split d by whitespace (`/\s+/`) to obtain a list of tokens T.
8. Apply `s/\.$/␣\./` to $t \in T$ if t matches `/$alnum\./` and is not in A and doesn't match `/^([A-Za-z]\.([A-Za-z]\.)+|[A-Z][bcdfghj-np-tvxz]+\.)$/`.
9. Optionally expand clitics: `s/'ve/have/` and `s/'m/am/` and so on.
10. Return a whitespace-separated string of T.

Remarks:

- The variables `alnum, nalnum, nalnum, and clitic` are regular expressions that capture the respective phenomena.
- The syntax **`s/A/B/`** stems from Perl and SED and commands to replace all occurrences of A with B. The Python equivalent is `re.sub(A, B,` d`)`
- The backreference $& resolves to the complete text matched by A.
- The backreferences $&1 resolves to the text matched by the first group (. . .) of the RegEx.

Text Preprocessing

Problems of Rule-based Tokenization

1. The vocabulary grows fast.
   - Most applications limit the vocabulary (i.e to ca. 50.000 for deep learning).
   - This makes dense representations very sparse and memory intensive.
   - Limiting the vocabulary removes named entities, rare words, typos, . . .
2. The construction cost is high.
   - Rules must be hand-crafted.
   - Rules differ for each genre, text source, and language.

Text Preprocessing

Tokenization: Byte-Pair Encoding [Sennrich et al., 2015]

Idea: Merge adjecent symbols to tokens if they are often in tokens together.

1. Split a string into symbols.
2. Apply all merge rules. In order, most frequent rule first.
3. Replace all out-of-vocabulary tokens with the unknown token `[UNK]`.

```
0.  a␣horse!␣a␣horse!␣my␣kingdom␣for␣a␣horse!
1.  a,␣h,o,r,s,e,!,␣a,␣h,o,r,s,e,!,␣m,y,...
```

Merge Rules

1:	o	r	or
2:	␣h	or	␣hor
3:	␣hor	s	␣hors
4:	␣hors	e	␣horse
5:	m	y	my
	...		

Vocabulary: $V = \{$`!, a, ..., y, dom, ␣my, ␣horse, ␣king, ␣for, [UNK]`$\}$

Text Preprocessing

Tokenization: Byte-Pair Encoding [Sennrich et al., 2015]

Idea: Merge adjecent symbols to tokens if they are often in tokens together.

1. Split a string into symbols.
2. Apply all merge rules. In order, most frequent rule first.
3. Replace all out-of-vocabulary tokens with the unknown token `[UNK]`.

```
0.  a␣horse!␣a␣horse!␣my␣kingdom␣for␣a␣horse!
1.  a,␣h,o,r,s,e,!,␣a,␣h,o,r,s,e,!,␣m,y,...
2.  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,!
```

Merge Rules

1:	o	r	or
2:	␣h	or	␣hor
3:	␣hor	s	␣hors
4:	␣hors	e	␣horse
5:	m	y	my
	...		

Vocabulary: $V = \{$`!, a, ..., y, dom, ␣my, ␣horse, ␣king, ␣for, [UNK]`$\}$

Text Preprocessing

Tokenization: Byte-Pair Encoding [Sennrich et al., 2015]

Idea: Merge adjecent symbols to tokens if they are often in tokens together.

1. Split a string into symbols.
2. Apply all merge rules. In order, most frequent rule first.
3. Replace all out-of-vocabulary tokens with the unknown token `[UNK]`.

```
0.  a␣horse!␣a␣horse!␣my␣kingdom␣for␣a␣horse!
1.  a,␣h,o,r,s,e,!,␣a,␣h,o,r,s,e,!,␣m,y,...
2.  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,!
```

Merge Rules

1:	o	r	or
2:	␣h	or	␣hor
3:	␣hor	s	␣hors
4:	␣hors	e	␣horse
5:	m	y	my
...			

Vocabulary: $V = \{$`!, a, ..., y, dom, ␣my, ␣horse, ␣king, ␣for, [UNK]`$\}$

Text Preprocessing

Tokenization: Byte-Pair Encoding [Sennrich et al., 2015]

Idea: Merge adjecent symbols to tokens if they are often in tokens together.

1. Split a string into symbols.
2. Apply all merge rules. In order, most frequent rule first.
3. Replace all out-of-vocabulary tokens with the unknown token `[UNK]`.

0. `a␣horse!␣a␣horse!␣my␣kingdom␣for␣a␣horse!`
1. `a,␣h,o,r,s,e,!,␣a,␣h,o,r,s,e,!,␣m,y,...`
2. `a,␣hor,s,e,!,␣a,␣hor,s,e,!,␣m,y,`
   `␣k,i,n,g,d,o,m,␣f,or,␣a,␣hor,s,e,!`

Merge Rules

1:	o	r	or
2:	␣h	or	␣hor
3:	␣hor	s	␣hors
4:	␣hors	e	␣horse
5:	m	y	my
	...		

Vocabulary: $V = \{$`!, a, ..., y, dom, ␣my, ␣horse, ␣king, ␣for, [UNK]`$\}$

Text Preprocessing

Tokenization: Byte-Pair Encoding [Sennrich et al., 2015]

Idea: Merge adjecent symbols to tokens if they are often in tokens together.

1. Split a string into symbols.
2. Apply all merge rules. In order, most frequent rule first.
3. Replace all out-of-vocabulary tokens with the unknown token `[UNK]`.

```
0.  a␣horse!␣a␣horse!␣my␣kingdom␣for␣a␣horse!
1.  a,␣h,o,r,s,e,!,␣a,␣h,o,r,s,e,!,␣m,y,...
2.  a,␣horse,!,␣a,␣horse,!,␣my,
   ␣king,dom,␣for,␣a,␣horse,!
3.  Does not apply here.
```

Tokenized Sentence:

`a ␣horse ! ␣a ␣horse ! ␣my`
`␣king dom ␣for ␣a ␣horse !`

Merge Rules

1:	o	r	or
2:	␣h	or	␣hor
3:	␣hor	s	␣hors
4:	␣hors	e	␣horse
5:	i	n	in
	...		
5:	m	y	my
	...		

Vocabulary: $V = \{$`!, a, ..., y, dom, ␣my, ␣horse, ␣king, ␣for, [UNK]`$\}$

Text Preprocessing

Tokenization: Byte-Pair Encoding Rule Finding [Sennrich et al., 2015]

1. Create an initial tokenization of a training corpus. i.e. using whitespaces.
2. Create a index I of all tokens and their counts.
3. Split each token into symbols, add them to the vocabulary V.
4. Add a merge rule to R for the pair i, j of adjecent symbols fulfilling:

$$\text{next\_merge} = \arg\max_{\langle i,j \rangle} \sum_{t \in I} \text{count}(\langle i, j \rangle \in t) \cdot \text{count}(t)$$

5. Apply the new rule to I. Add the merged symbol to V. Repeat from 4.
6. Stop if V or R reach a predefined size. e.g. 50,000

a,␣horse,!,␣a,␣horse,!,␣my,␣kingdom,␣for,␣a,␣horse,!

$I = \{ \quad \}$

$V = \{ \quad \}$

Merge Rules R

Text Preprocessing

Tokenization: Byte-Pair Encoding Rule Finding [Sennrich et al., 2015]

1. Create an initial tokenization of a training corpus. i.e. using whitespaces.
2. Create a index I of all tokens and their counts.
3. Split each token into symbols, add them to the vocabulary V.
4. Add a merge rule to R for the pair i, j of adjecent symbols fulfilling:

$$\text{next\_merge} = \arg\max_{\langle i,j \rangle} \sum_{t \in I} \text{count}(\langle i,j \rangle \in t) \cdot \text{count}(t)$$

5. Apply the new rule to I. Add the merged symbol to V. Repeat from 4.
6. Stop if V or R reach a predefined size. e.g. 50,000

`a,␣horse,!,␣a,␣horse,!,␣my,␣kingdom,␣for,␣a,␣horse,!`

I = { (a; 1), (␣horse; 3), (␣a; 2), (␣my; 1), (␣kingdom; 1), (␣for; 1), (!; 3) }

V = { }

Merge Rules R

Text Preprocessing

Tokenization: Byte-Pair Encoding Rule Finding [Sennrich et al., 2015]

1. Create an initial tokenization of a training corpus. i.e. using whitespaces.
2. Create a index I of all tokens and their counts.
3. Split each token in I into symbols, add them to the vocabulary V.
4. Add a merge rule to R for the pair i, j of adjecent symbols fulfilling:

$$\text{next\_merge} = \underset{\langle i,j \rangle}{\arg\max} \sum_{t \in I} \text{count}(\langle i, j \rangle \in t) \cdot \text{count}(t)$$

5. Apply the new rule to I. Add the merged symbol to V. Repeat from 4.
6. Stop if V or R reach a predefined size. e.g. 50,000

`a,␣horse,!,␣a,␣horse,!,␣my,␣kingdom,␣for,␣a,␣horse,!`

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

V = {!, a, ..., y [UNK]}

Merge Rules R

Text Preprocessing

Tokenization: Byte-Pair Encoding Rule Finding [Sennrich et al., 2015]

1. Create an initial tokenization of a training corpus. i.e. using whitespaces.
2. Create a index I of all tokens and their counts.
3. Split each token in I into symbols, add them to the vocabulary V.
4. Add a merge rule to R for the pair i, j of adjecent symbols fulfilling:

$$\text{next\_merge} = \arg\max_{\langle i,j \rangle} \sum_{t \in I} \text{count}(\langle i, j \rangle \in t) \cdot \text{count}(t)$$

5. Apply the new rule to I. Add the merged symbol to V. Repeat from 4.
6. Stop if V or R reach a predefined size. e.g. 50,000

`a,␣horse,!,␣a,␣horse,!,␣my,␣kingdom,␣for,␣a,␣horse,!`

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

V = {!, a, ..., y, [UNK]}

Merge Rules R			
1:	o	r	or

Text Preprocessing

Tokenization: Byte-Pair Encoding Rule Finding [Sennrich et al., 2015]

1. Create an initial tokenization of a training corpus. i.e. using whitespaces.
2. Create a index I of all tokens and their counts.
3. Split each token in I into symbols, add them to the vocabulary V.
4. Add a merge rule to R for the pair i, j of adjecent symbols fulfilling:

$$\text{next\_merge} = \arg\max_{\langle i,j \rangle} \sum_{t \in I} \text{count}(\langle i, j \rangle \in t) \cdot \text{count}(t)$$

5. Apply the new rule to I. Add the merged symbol to V. Repeat from 4.
6. Stop if V or R reach a predefined size. e.g. 50,000

`a,␣horse,!,␣a,␣horse,!,␣my,␣kingdom,␣for,␣a,␣horse,!`

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

Merge Rules R		
1:	o r	or

$V = \{$!, a, ..., y, or, [UNK]$\}$

Text Preprocessing

Tokenization: Byte-Pair Encoding Rule Finding [Sennrich et al., 2015]

1. Create an initial tokenization of a training corpus. i.e. using whitespaces.
2. Create a index I of all tokens and their counts.
3. Split each token in I into symbols, add them to the vocabulary V.
4. Add a merge rule to R for the pair i, j of adjecent symbols fulfilling:

$$\text{next\_merge} = \arg\max_{\langle i,j \rangle} \sum_{t \in I} \text{count}(\langle i, j \rangle \in t) \cdot \text{count}(t)$$

5. Apply the new rule to I. Add the merged symbol to V. Repeat from 4.
6. Stop if V or R reach a predefined size. e.g. 50,000

`a,␣horse,!,␣a,␣horse,!,␣my,␣kingdom,␣for,␣a,␣horse,!`

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

V = {!, a, ..., y, or, ␣hor, [UNK]}

Merge Rules		R
1: o	r	or
2: ␣h	or	␣hor

Text Preprocessing

Tokenization: Byte-Pair Encoding Rule Finding [Sennrich et al., 2015]

1. Create an initial tokenization of a training corpus. i.e. using whitespaces.
2. Create a index I of all tokens and their counts.
3. Split each token in I into symbols, add them to the vocabulary V.
4. Add a merge rule to R for the pair i, j of adjecent symbols fulfilling:

$$\text{next\_merge} = \arg\max_{\langle i,j \rangle} \sum_{t \in I} \text{count}(\langle i, j \rangle \in t) \cdot \text{count}(t)$$

5. Apply the new rule to I. Add the merged symbol to V. Repeat from 4.
6. Stop if V or R reach a predefined size. e.g. 50,000

```
a,␣horse,!,␣a,␣horse,!,␣my,␣kingdom,␣for,␣a,␣horse,!
```

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

V = {!, a, ..., y, or, ␣hor, ␣hors, [UNK]}

Merge Rules			R
1:	o	r	or
2:	␣h	or	␣hor
3:	␣hor	s	␣hors

Text Preprocessing

Tokenization: Byte-Pair Encoding Rule Finding [Sennrich et al., 2015]

1. Create an initial tokenization of a training corpus. i.e. using whitespaces.
2. Create a index I of all tokens and their counts.
3. Split each token in I into symbols, add them to the vocabulary V.
4. Add a merge rule to R for the pair i, j of adjecent symbols fulfilling:

$$\text{next\_merge} = \arg\max_{\langle i,j \rangle} \sum_{t \in I} \text{count}(\langle i, j \rangle \in t) \cdot \text{count}(t)$$

5. Apply the new rule to I. Add the merged symbol to V. Repeat from 4.
6. **Stop if V or R reach a predefined size.** e.g. 50,000

`a,␣horse,!,␣a,␣horse,!,␣my,␣kingdom,␣for,␣a,␣horse,!`

I = { (`a`; 1), (`␣horse`; 3), (`␣a`; 2), (`␣my`; 1),
(`␣king,dom`; 1), (`␣for`; 1), (`!`; 3), . . . }

V = {`!, a, ..., y, or, ␣hor, ␣hors, ..., [UNK]`}

Merge Rules			R
1:	o	r	or
2:	␣h	or	␣hor
3:	␣hor	s	␣hors
	...		

Remarks:

- A variant of Byte-Pair Encoding (BPE) is used by `GPT-2`: byte-level BPE. It uses all 256 Bytes as basis vocabulary to avoid the [UNK] token completely.

- `BERT` and many of it's variants use WordPiece, which is an extention of BPE. It adapts the merge_select function to find the most likely merge, instead of the most common one. This avoids merging subwords that also often appear independently.

$$\text{next\_merge} = \underset{\langle i,j \rangle}{\arg\max} \frac{\sum \langle i, j \rangle}{\sum i \cdot \sum j}$$

- The tokenizers Unigram [Kudo, 2018] and SentencePiece [] work in reverse to WordPiece: they add all possible tokens to the Vocabulary, then iteratively remove tokens until the desired vocabulary size is reached.

Text Preprocessing

Tokenization: Token Removal

Remove undesired tokens (stop words) to reduce data size, sparsity, and improve performance on downstream tasks (Stopping).

- Frequent tokens (collection-specific)

  `Wikipedia` when processing Wikipedia.

- Function word tokens (language-dependent)

  `the`, `of`, `and`, ...

  `to be or not to be`?

- Punctuation-only tokens

  `;-)`

- Number-only tokens

- Short tokens

  `xp`, `ma`, `pm`, `ben e king`, `el paso`, `master p`, `gm`, `j lo`, ...

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

Text Preprocessing

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.
```

Text Preprocessing

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
```