CS 481

Artificial Intelligence Language Understanding

January 19, 2023

Announcements / Reminders

- Please follow the Week 02 To Do List instructions
- Quiz #01 due on Sunday 01/22/23 at 11:59 PM CST
- Exam dates:
  - Midterm: 03/02/2023 during Thursday lecture time
  - Final: 04/27/2023 during Thursday lecture time

Plan for Today

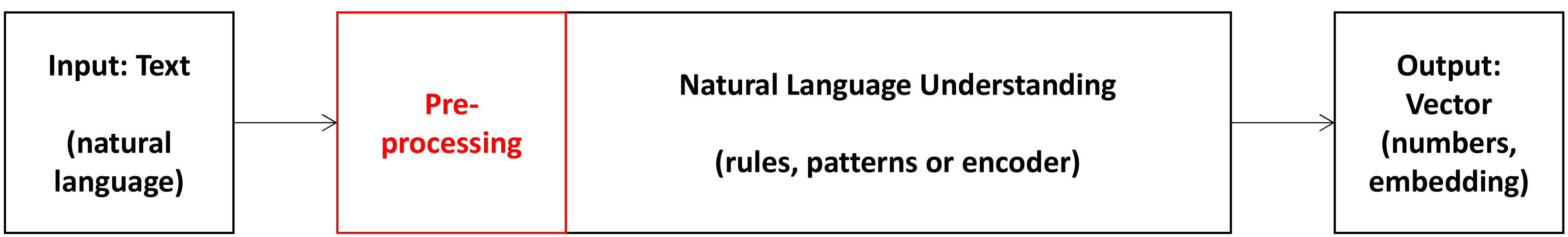
- Text Pre-processing - continued
  - Byte Pair Encoding
- Regular Expressions (RegEx)
  - Introduction
  - RegEx for basic text pre-processing
- Python libraries / packages for NLP
- Text corpora

Parsing

The task of determining the parts of speech, phrases, clauses, and their relationship to one another is called parsing.

Automated Text Processing

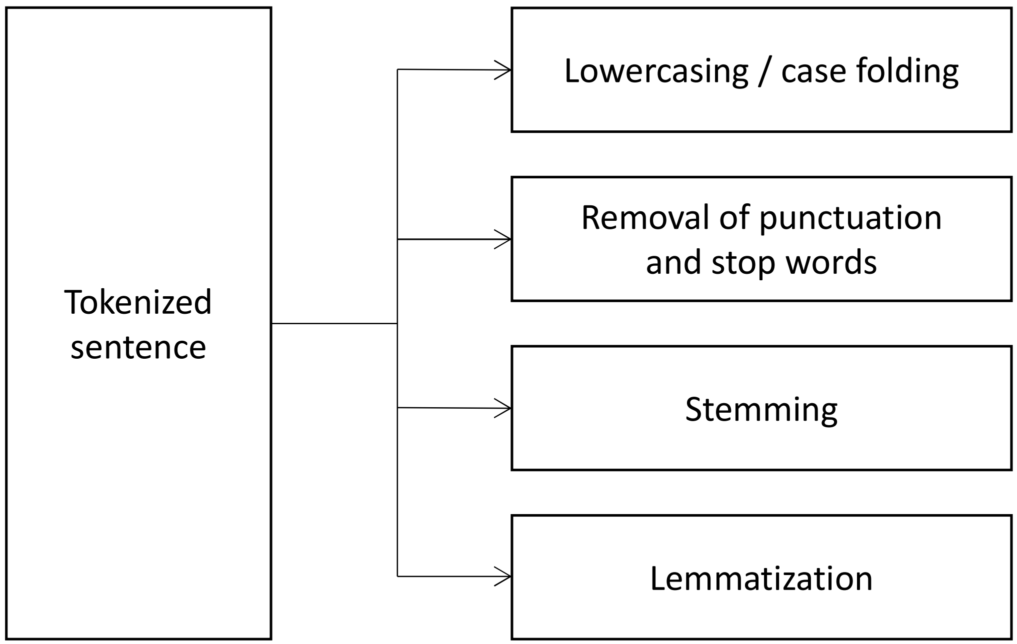
The task of automatic processing of text is to extract a numerical representation of the meaning of that text. This is the natural language understanding (NLU) part of NLP. The numerical representation of the meaning of natural language usually takes the form of a vector called an embedding.



Basic Pre-Processing: Normalization

Document(s) / text level:

Tokenized sentence level:



Note: depending on the nature of data, additional pre-processing steps may be required / important.

Pre-processing: Normalization

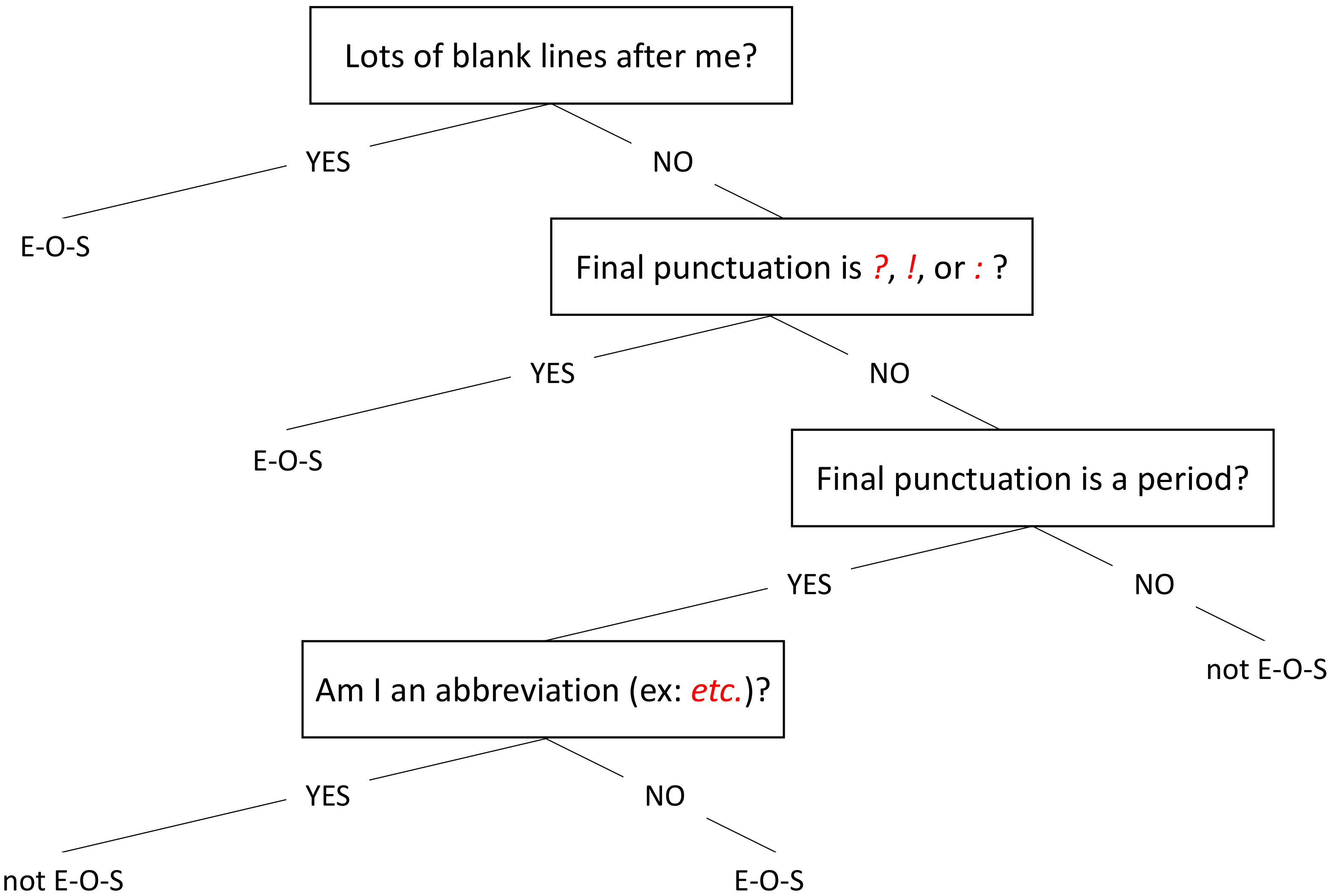
Every NLP task needs to do text normalization:

- Segmenting / tokenizing sentences in a document
- Segmenting / tokenizing words in sentences
- Normalizing word formats
  - lowercasing / case folding
  - stemming
  - lemmatization
  - etc.

Segmentation and Tokenization

- **Text segmentation (text into sentence): breaking up text into sentences at the appearance of full stops, exclamation and question marks.**
  - **abbreviations, forms of addresses (*Mrs.*), ellipses (*...*), numbers (*.02%*) are problematic**
    - **potential solution: build a "end of sentence" classifier**
  - **tools:** `.split()` **method, RegEx, NLP library specific**
- **Sentence tokenization (sentence to words): breaking up sentences into tokens based on the presence of whitespaces and punctuation marks (others possible).**
  - **tools:** `.split()` **method, RegEx, NLP library specific**

Segmentation with Decision Trees



E-O-S: End-Of-Sentence

Tokenizaton: Type vs. Token

- **Type: an element of the vocabulary.**
- **Token: an instance of that type in runninga type in text.**

Vocabulary: a set of types

Example: *"A good course is a course that you like"*

- **9 tokens**
- **7 types (*a* and *course* are repeated)**

- **Type/token ratio (TTR): 7/9**

(White) Space-based Tokenization

For languages that use space characters between words

- Arabic-, Cyrillic-, Greek-, Latin-, based writing systems (and others)
- Segment off a token between instances of spaces

This approach does not require a lot of knowledge about the language.

Tokenization: Other Cases / Options

Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!

How do we decide where the token boundaries should be?

Instead of

- **white-space segmentation**
- **single-character segmentation**

use the data to tell us how to tokenize $\rightarrow$ subword tokenization

- **tokens can be parts of words and whole words)**

Subword Tokenization

Three common algorithms:

- Byte-Pair Encoding (BPE) (Sennrich et al., 2016)
- Unigram language modeling tokenization (Kudo, 2018)
- WordPiece (Schuster and Nakajima, 2012)

All three have two components:

- A token learner that takes a raw training corpus and induces a vocabulary (a set of tokens).
- A token segmenter that takes a raw test sentence and tokenizes it according to that vocabulary

Tokenization: Problematic Cases

NLP applications should handle problematic cases during tokenization:

- **tokens containing periods:** *Dr.*, *xyz.com*
- **hyphens:** *rule-based*
- **clitics (connected word abbreviations):** *couldn't*, *we've*
- **numerical expressions and dates:** *(123) 555-5555*, *August 7th, 2019*
- **emojis, hashtags, email and web addresses (URLs)**

NLP libraries differ in this regard.

Tokenization: Problematic Cases

Finland's capital	→	*Finland* **or** *Finlands* **or** *Finland's* **??**
what're, I'm, isn't	→	*What are*, *I am*, *is not*
Hewlett-Packard	→	*Hewlett Packard* **??**
state-of-the-art	→	*state of the art* **??**
Lowercase	→	*lower-case* **or** *lowercase* **or** *lower case* **??**
San Francisco	→	**one token or two ??**
m.p.h., *PhD.*	→	**??**

Other languages (German example):

- *Lebensversicherungsgesellschaftsangestellter* **is a noun compound for "***life insurance company employee***"**

Tokenization Approaches

(White)space-based tokenization:

Learning
Learned

Deep	learning

Issues:

- some languages don't use (white)space to separate words (Chinese)
- punctuations, etc. (e.g *let's*)
- compound words

Subword tokenization:

Learn	##ing
Learn	##ed

De	ep	learn	##ing

Issues:

- needs a rule based system for affixes ("*##s*", "*##ing*", "*##ify*", "*un##*", etc.)
- some generated subwords may never appear in your text

Character-based tokenization:

L	e	a	r	n	i	n	g

L	e	a	r	n	e	d

D	e	e	p	l	e	a	r	n	i	n	g

Issues:

- requires more computation
- may not preserve semantics (meaning) well
- not every NLP task (after tokenization will handle it

Vocabulary size →

Byte Pair Encoding: Compression

The idea comes from a compression algorithm:

- **Consider a string:** `aaabdaaabac`
- **the byte pair "`aa`" occurs most often → replace it by a byte that is not used in the data (for example "`Z`") to get**

`ZabdZabac` **where** `Z = aa`

- **repeat for pair "`ab`", replacing it with "Y" to get:**

`ZYdZYac` **where** `Y = ab` **and** `Z = aa`

- **and so on...**

Byte Pair Encoding: The Algorithm

BPE consists of two components:

- **the Token Learner: train / create new vocabulary V based on the corpus C (training set)**
- **the Token Segmenter uses new vocabulary V to segment new text (test set)**

Token Learner algorithm:

function BPE(corpus C, number of merges k) returns vocabulary V

$V \leftarrow$ all unique characters in C # initial vocabulary

for $i = 1$ **to** k **do** # merge tokens k times

$t_L, t_R \leftarrow$ most frequent pair of adjacent characters in C

$t_{NEW} \leftarrow t_L + t_R$ # create new token (concatenate t_L, t_R)

$\mathrm{V} \leftarrow \mathrm{V} + \mathrm{t}_{\mathrm{NEW}}$ # update vocabulary V

Replace each occurence of (t_L, t_R) in C with $\mathrm{t}_{\mathrm{NEW}}$

return V

BPE Token Learner: Example

Training corpus C**:**

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Initial vocabulary V**:**

$V = \{d, e, i, l, n, o, r, s, t, w\}$

Initial vocabulary V **with added stop token '_':**

$V = \{\_, d, e, i, l, n, o, r, s, t, w\}$

Number of merges k**: 8**

BPE Token Learner: Example

Before merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

After e r to er merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

BPE Token Learner: Example

Before merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

After er _ to er_ merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

BPE Token Learner: Example

Before merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

After n e to ne merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
2	l o w e s t _	
6	ne w er_	
3	w i d er_	
2	n e w _	

BPE Token Learner: Example

Subsequent merges

Merge	Vocabulary V
ne w → **new**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, **new**
l o → **lo**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, **lo**
lo w → **low**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, **low**
new er_ → **newer_**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, **newer_**
low _ → **low_**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, **low_**

BPE Token Segmenter: Algorithm

On the test set, run each merge learned from the training set (corpus C):

- greedily
- in the order we learned them
- ignore test set frequencies

So: merge every `e r` to `er`, then merge `er _` to `er_`, etc.

Result:

- test set "`n e w e r _`" would be tokenized as a full word
- test set "`l o w e r _`" would be two tokens: "`low er_`"

BPE Tokenization: Properties

BPE will usually:

- **capture frequent words**
- **capture frequent subwords**
  - **prefixes, suffixes, etc,**
  - **which are also often morphemes (the smallest meaning-bearing unit of a language) like *-est* or *–er***
    - **for example the word *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est***

Tokenization Approaches

(White)space-based tokenization:

Learning	
Learned	
Deep	learning

Issues:

- some languages don't use (white)space to separate words (Chinese)
- punctuations, etc. (e.g *let's*)
- compound words

Subword tokenization:

Learn	##ing		
Learn	##ed		
De	ep	learn	##ing

Issues:

- needs a rule based system for affixes ("*##s*", "*##ing*", "*##ify*", "*un##*", etc.)
- some generated subwords may never appear in your text

Character-based tokenization:

L	e	a	r	n	i	n	g					
L	e	a	r	n	e	d						
D	e	e	p	l	e	a	r	n	i	n	g	

Issues:

- requires more computation
- may not preserve semantics (meaning) well
- not every NLP task (after tokenization will handle it

Vocabulary size →

Byte Pair Encoding: Compression

The idea comes from a compression algorithm:

- **Consider a string:** `aaabdaaabac`
- **the** **byte pair** **"`aa`" occurs most often → replace it by a byte that is not used in the data (for example "`Z`") to get**

`ZabdZabac` **where** `Z = aa`

- **repeat for pair "`ab`", replacing it with "`Y`" to get:**

`ZYdZYac` **where** `Y = ab` **and** `Z = aa`

- **and so on...**

Byte Pair Encoding: The Algorithm

BPE consists of two components:

- **the Token Learner: train / create new vocabulary V based on the corpus C (training set)**
- **the Token Segmenter uses new vocabulary V to segment new text (test set)**

Token Learner algorithm:

function BPE(corpus C, number of merges k) returns vocabulary V

$V \leftarrow$ all unique characters in C # initial vocabulary

for $i = 1$ **to** k **do** # merge tokens k times

$t_L, t_R \leftarrow$ most frequent pair of adjacent characters in C

$t_{NEW} \leftarrow t_L + t_R$ # create new token (concatenate t_L, t_R)

$\mathrm{V} \leftarrow \mathrm{V} + \mathrm{t}_{\mathrm{NEW}}$ # update vocabulary V

Replace each occurence of (t_L, t_R) in C with $\mathrm{t}_{\mathrm{NEW}}$

return V

BPE Token Learner: Example

Training corpus C**:**

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Initial vocabulary V**:**

$\mathrm{V} = \{\mathrm{d, e, i, l, n, o, r, s, t, w}\}$

Initial vocabulary V **with added stop token '_':**

$\mathrm{V} = \{\_, \mathrm{d, e, i, l, n, o, r, s, t, w}\}$

Number of merges k**: 8**

BPE Token Learner: Example

Before merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

After e r to er merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

BPE Token Learner: Example

Before merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

After er _ **to** er_ **merge:**

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

BPE Token Learner: Example

Before merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

After n e to ne merge:

f	"word"	Vocabulary V
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
2	l o w e s t _	
6	ne w er_	
3	w i d er_	
2	n e w _	

BPE Token Learner: Example

Subsequent merges

Merge	Vocabulary V
ne w → **new**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, **new**
l o → **lo**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, **lo**
lo w → **low**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, **low**
new er_ → **newer_**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, **newer_**
low _ → **low_**	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, **low_**

BPE Token Segmenter: Algorithm

On the test set, run each merge learned from the training set (corpus C):

- **greedily**
- **in the order we learned them**
- **ignore test set frequencies**

So: merge every `e r` **to** `er`**, then merge** `er _` **to** `er_`**, etc.**

Result:

- **test set "**`n e w e r _`**" would be tokenized as a full word**
- **test set "**`l o w e r _`**" would be two tokens: "**`low er_`**"**

BPE Tokenization: Properties

BPE will usually:

- **capture frequent words**
- **capture frequent subwords**
  - **prefixes, suffixes, etc,**
  - **which are also often morphemes (the smallest meaning-bearing unit of a language) like *-est* or *–er***
    - **for example the word *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est***

Pre-processing: Lowercasing

Some applications (eg. Information Retrieval, search) reduce all letters to lower case:

- users tend to use lower case
- possible exception: upper case in mid-sentence?
  - General Motors
  - Fed vs. fed

For sentiment analysis, topic modeling:

- preserving case is important (US vs. us)

Pre-processing: Stemming

Stemming refers to the process of removing suffixes and reducing the word to some base form such that all different variants of that word can be represented by the same base form (*car* and *cars* are reduced to *car*).

- **use a set of rules to accomplish stemming**
  - **if the word ends in "*-es*", remove "*-es*"**
- **final base form may NOT be linguistically correct**
  - *airliner* $\rightarrow$ *airlin*
- **commonly used by search engines and in text classification**

Stemming: Before and After

Before:

For example compressed and compression are both accepted as equivalent to compress.

After:

For exampl compress and compress ar both accept as equival to compress.

Stemming: Porter Stemmer

Based on a series of rewrite rules run in series

- **a cascade, in which output of each pass (different rewrite rule applied) fed to next pass**

Some sample rules:

$$\begin{aligned} \text{ATIONAL} &\rightarrow \text{ATE} \quad \text{(e.g., relational} \rightarrow \text{relate)} \\ \text{ING} &\rightarrow \epsilon \quad \text{if stem contains vowel (e.g., motoring} \rightarrow \text{motor)} \\ \text{SSES} &\rightarrow \text{SS} \quad \text{(e.g., grasses} \rightarrow \text{grass)} \end{aligned}$$

see https://tartarus.org/martin/PorterStemmer/def.txt **for more**

Pre-processing: Lemmatization

Lemmatization is a process of **mapping all the different forms of a word to its base word**, or **lemma**. In other words: reduce inflections or variant forms to base form

- has to find correct dictionary headword form

Stemming vs. Lemmatization

Stemming	Lemmatization
adjustable → adjust	was → (to) be
	better → good
meeting → meet	meeting → meeting
studies → studi	studies → study
studying → study	studying → study

Tokenization / Lemmatization Example

Sentence input:

Chaplin wrote, directed, and composed music for most of his films.

Tokenization:

Chaplin	wrote	,	directed	,	and	composed	music	for	most	of	his	films	.

Chaplin wrote, directed, and composed music for most of his films.

Lemmatization:

Chaplin	write	,	direct	,	and	compose	music	for	most	of	he	film	.

Chaplin wrote, directed, and composed music for most of his films.

Pre-processing: Stop Words

Very common words (articles, propositions, pronouns, conjunctions,etc.) that do not add much information (but take up space) are called stop words and are frequently filtered out.

- **Examples in English:** *an*, *the*, *a*, *for*, *is*
- **Filtering based on the stop (word) list**
  - **generated based on collection frequency**
- **Tools: RegEx + stop list, NLP libraries have their own stop lists**
- **Careful: sometimes it may lead to removing important information**

Additional Pre-processing Steps

- Additional normalization
  - in addition to stemming, lemmatization:
    - standardizing abbreviations (eg. expanding), hyphenations, digits to text (9 to nine) conversions, etc.
- Language detection
- Code mixing
  - embedding of linguistic units such as phrases, words, and morphemes of one language into an utterance of another language
- Transliteration
  - converting between different writing systems

Regular Expressions (RegEx)

A regular expression (RegEx, regex or regexp) is a sequence of characters that specifies a search pattern in text.

- **patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation.**
- **a technique developed in theoretical computer science and formal language theory. Related to the concept of a regular language (a formal language that can be defined by a regular expression).**
- **very efficient for pre-processing tasks**
- **there are two key RegEx libraries:**
  - **re (built-in)**
  - **and regex (external: https://pypi.org/project/regex/)**

Regular Expressions: Disjunction

- **Disjunction**: characters within square brackets []

Pattern	Matches
`[wW]oodchuck`	**W**oodchuck **OR** **w**oodchuck
`[1234567890]`	Any digit (1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9 or 0)

- **Disjunction**: a **range** of characters, square brackets [] and dash -

Pattern	Matches	Example
`[A-Z]`	An upper case letter	**D**renched Blossoms
`[a-z]`	A lower case letter	**m**y beans were impatient
`[0-9]`	A single digit	Chapter **1**: Down the Rabbit Hole

Regular Expressions: Disjunction

- **Disjunction**: a pipe | also means OR

Pattern	Matches
`woodchuck|groundhog`	woodchuck **OR** groundhog
`yours|mine`	yours **OR** mine
`a|b|c`	same as `[abc]` or `[a-c]`

Regular Expressions: Negation

- **Negation**: a caret ^ means negation (has to be first within [])

Pattern	Matches	Example
`[^A-Z]`	NOT an upper case letter	Oyfn pripetchik
`[^Ss]`	Neither 'S' nor 's'	I have no reason for it

Regular Expressions: Optionality

- **Optionality**: the question mark ? marks optionality of previous

Pattern	Matches	Example
`woodchucks?`	woodchuck OR woodchucks	nice woodchuck!
`colou?r`	color OR colour	beautiful colour

Regular Expressions: . wildcard

- **. wildcard**: period . represents ANY character

Pattern	Matches	Example
`beg.n`	any character between *beg* and *n*	begin beg'n begun

Regular Expressions: Other Operators

- **Some additional and useful operators:**

Operator	Expansion	Match	Examples
`\d`	`[0-9]`	any digit	Party of <u>5</u>
`\D`	`[^0-9]`	any non-digit	<u>B</u>lue moon
`\w`	`[a-zA-Z0-9_]`	any alphanumeric / underscore	<u>D</u>aiyu
`\W`	`[^\w]`	a non-alphanumeric	<u>!</u>!!!!
`\s`	`[ \r\t\n\f]`	whitespace (space, tab)	
`\S`	`[^\s]`	non-whitespace	<u>i</u>n Chicago

Regular Expressions: Backslash

- **Some characters need to be backlashed (operators in RegEx)**

Pattern	Matches	Comment
`\*`	an asterisk *	K*A*P*L*A*N
`\.`	a period .	Dr. Livingston, I presume
`\?`	a question mark	What is the time?
`\n`	a newline character	
`\t`	a tab	

Regular Expressions: Anchors

- **Anchors**: anchor regular explations to specific places in a string

Pattern	Matches	Comment
`^[A-Z]`	Palo Alto	Start of string anchor (^) **First** character has to be uppercase letter
`^[^A-Za-z]`	1 "Hello"	Start of string anchor (^) **First** character cannot be a letter
`\.$`	The end.	End of string anchor ($) Note the \ before . **Last** character has to be .
`.$`	The end? The end!	End of string anchor ($) **Last** character can be anything
`\b`		word boundary
`\B`		word non-boundary

Regular Expressions: Example

Find an instance of the word 'the' within input string.

RegEx patterns:

- `the` **: will miss capitalized 'The'**
- `[tT]he` **: will match substrings 'the' and 'The' within other words (o*ther*, *the*m)**
- `[^a-zA-Z][tT]he[^a-zA-Z]` **: this will do it**

Fixed two type of errors (to increase precision and recall):

- **Type I: matching strings we shouldn't have (false positive)**
- **Type II: not matching strings we should have matched (false negative)**

Python re Module / Library

Python's re module / library is built-in. Documentation:

`https://docs.python.org/3/library/re.html`

Key functions / methods:

- `match()`**: checks for matching string at the beginning**
- `search()`**: find first location of a matching string**
- `findall()`**: returns all non-overlapping matches**
- `split()`**: splits string by occurences of a pattern**
- `sub()`**: "replace"**

Python NLP Libraries / Packages

- Natural Language Toolkit (NLTK) [more academic]
- TextBlob
- CoreNLP
- Gensim
- spaCy [industry / production]
- Polyglot
- scikit-learn (machine learning)
- pyTorch (machine learning)
- Pattern
- PyNLPl

Natural Language Toolkit (NLTK)

"NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum."

Link:	**https://www.nltk.org/**
Anaconda:	**https://anaconda.org/anaconda/nltk**
Install:	**https://www.nltk.org/install.html**

TextBlob

“TextBlob is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.”

Link:	**https://textblob.readthedocs.io/en/dev/**
Anaconda:	**https://anaconda.org/conda-forge/textblob**
Install:	**https://textblob.readthedocs.io/en/dev/install.html**

CoreNLP

"CoreNLP is your one stop shop for natural language processing in Java! CoreNLP enables users to derive linguistic annotations for text, including token and sentence boundaries, parts of speech, named entities, numeric and time values, dependency and constituency parses, coreference, sentiment, quote attributions, and relations. CoreNLP currently supports 8 languages: Arabic, Chinese, English, French, German, Hungarian, Italian, and Spanish."

Link: https://stanfordnlp.github.io/CoreNLP/

Anaconda: https://anaconda.org/auto/corenlp

Gensim

"Gensim is a Python library for topic modelling, document indexing and similarity retrieval with large corpora. Target audience is the natural language processing (NLP) and information retrieval (IR) community."

Link:	**https://github.com/RaRe-Technologies/gensim**
Anaconda:	**https://anaconda.org/anaconda/gensim**
Install:	**https://github.com/RaRe-Technologies/gensim**

spaCy

“spaCy is a free, open-source library for advanced Natural Language Processing (NLP) in Python.”

Link:	**https://spacy.io/**
Anaconda:	**https://anaconda.org/conda-forge/spacy**
Install:	**https://spacy.io/usage**

Polyglot

“Polyglot is a natural language pipeline that supports massive multilingual applications.”

Link: https://polyglot.readthedocs.io/en/latest/index.html

Anaconda: https://anaconda.org/syllabs_admin/polyglot

Install: https://polyglot.readthedocs.io/en/latest/Installation.html

scikit-learn

"Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy."

Link:	**https://scikit-learn.org/stable/index.html**
Anaconda:	**https://anaconda.org/anaconda/scikit-learn**
Install:	**https://scikit-learn.org/stable/install.html**

Pattern

"Web mining module for Python, with tools for scraping, natural language processing, machine learning, network analysis and visualization."

Link:	**https://github.com/clips/pattern**
Anaconda:	**https://anaconda.org/conda-forge/pattern**
Install:	**https://github.com/clips/pattern**

PyNLPl

"PyNLPl (Python Natural Language Processing library), pronounced as 'pineapple', is a Python library for Natural Language Processing. It contains various modules useful for common, and less common, NLP tasks."

Link: https://github.com/proycon/pynlpl

Anaconda: N/A?

Install: https://github.com/proycon/pynlpl

Text Corpora

In linguistics, a corpus (Latin for "body" | plural: corpora) or text corpus is a language resource consisting of a large and structured set of texts (nowadays usually electronically stored and processed), written or transcribed.

Text corpora are:

- **usually purposefully collected**
- **usually structured**
- **usually annotated (part of speech tags, etc.)**

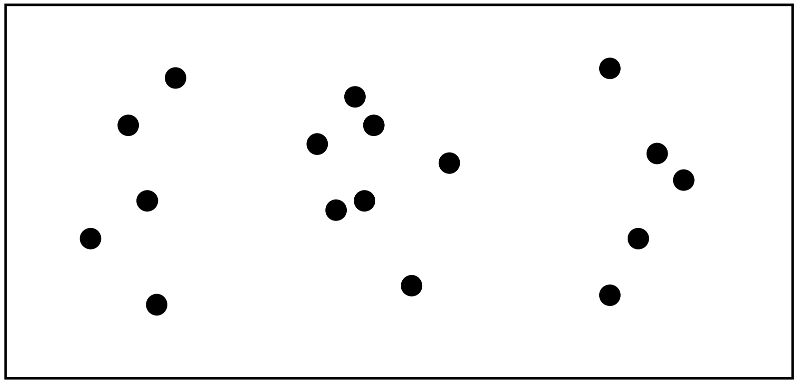
Text Corpora

Words / documents are produced within a context.

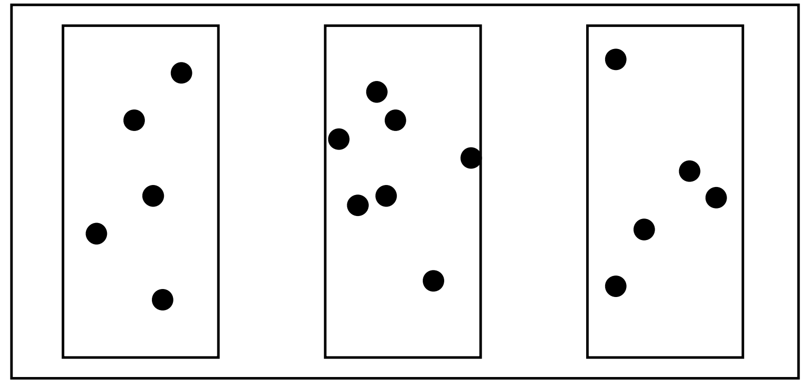
A text is generated by:

- **a specific writer(s),**
- **at a specific time,**
- **in a specific variety,**
- **of a specific language,**
- **for a specific function.**

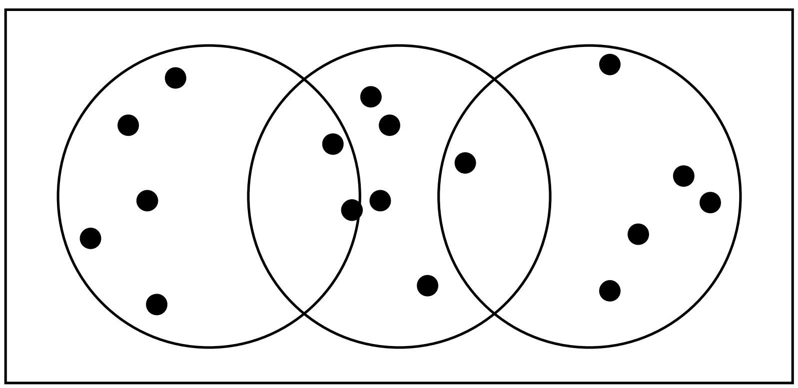
Text Corpora Structures



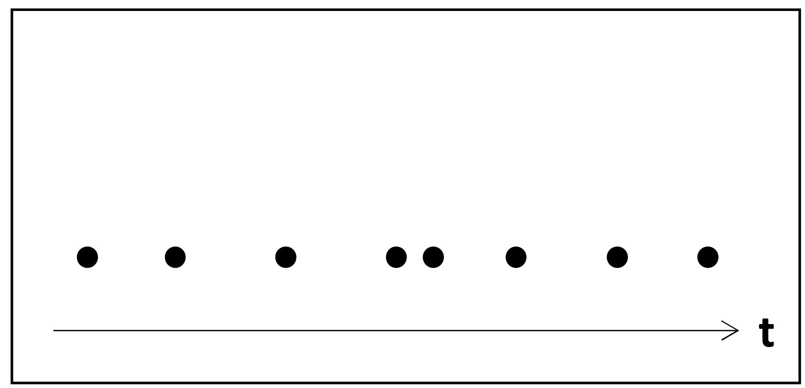
Isolated (e.g. Gutenberg)



Categorized (e.g. Brown)



Overlapping (e.g. Reuters)



Temporal (e.g. Inaugural Address)

Text Corpora Variation

Text corpora can contain a lot of variation:

- **Language:** 7097 languages in the world
- **Variety, like African American Language varieties.**
- **Twitter posts:** might include forms like "*iont*" (*I don't*)
- **Code switching (e.g., Spanish/English, Hindi/English):**
  - ***S/E: Por primera vez veo a @username actually being hateful! It was beautiful:)***
  - *[For the first time I get to see @username actually being hateful! it was beautiful:)]*
  - ***H/E: dost tha or ra- hega ... dont wory ... but dherya rakhe***
  - *["he was and will remain a friend ... don't worry ... but have faith"]*
- **Genre:** newswire, fiction, scientific articles, Wikipedia
- **Author(s) demographics:** writer's age, gender, ethnicity

Text Corpora Sizes

V = vocabulary = set of types

$|V|$ = *size* (cardinality) of vocabulary

N = number of tokens (instances of types)

Heaps Law / Herdan's Law: $|V| = kN^{\beta}$

where (often): $0.67 < \beta < 0.75$ (i.e., vocabulary size grows with > square root of the number of word tokens)

Corpus	Tokens = N	Types = $|V|$
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA (Corpus of Contemporary American English)	440 million	2 million
Google N-grams	1 trillion	13+ million

Text Corpora Datasheet

Text corpora should be described by:

- **Motivation**: Why was the corpus collected, by whom, and who funded it?
- **Situation**: When and in what situation was the text written/spoken?
- **Language variety**: What language (including dialect/region) was the corpus in?
- **Speaker demographics**: What was, e.g., age or gender of the authors of the text?
- **Collection process**: How big is the data? If it is a subsample how was it sampled? Was the data collected with consent? How was the data pre-processed, and what metadata is available?
- **Annotation process**: What are the annotations, what are the demographics of the annotators, how were they trained, how was the data annotated?
- **Distribution**: Are there copyright or other intellectual property restrictions?

English Corpora: Online Tour

English-Corpora.org

home corpora users related resources my account **upgrade** help

The most widely used online corpora: guided tour, overview, search types, variation, virtual corpora (quick overview) ~~BYU~~.

The links below are for the online interface. But you can also download the corpora for use on your own computer.

Corpus (online access)	Download	# words	Dialect	Time period	Genre(s)
News on the Web (NOW)		14.3 **billion+**	20 countries	2010-yesterday	Web: News
iWeb: The Intelligent Web-based Corpus		14 **billion**	6 countries	2017	Web
Global Web-Based English (GloWbE)		1.9 **billion**	20 countries	2012-13	Web (incl blogs)
Wikipedia Corpus		1.9 **billion**	(Various)	2014	Wikipedia
Coronavirus Corpus		1.3 **billion+**	20 countries	Jan 2020-yesterday	Web: News
Corpus of Contemporary American English (COCA)		1.0 **billion**	American	1990-2019	Balanced
Corpus of Historical American English (COHA)		475 million	American	1820-2019	Balanced
The TV Corpus		325 million	6 countries	1950-2018	TV shows
The Movie Corpus		200 million	6 countries	1930-2018	Movies

Source: https://www.english-corpora.org/

NLTK Corpora

NLTK Corpora

NLTK has built-in support for dozens of corpora and trained models, as listed below. To use these within NLTK we recommend that you use the NLTK corpus downloader, `>>> nltk.download()`

Please consult the README file included with each corpus for further information.

1. *Australian Broadcasting Commission 2006* [download | source]
   id: `abc`; size: 1487851; author: Australian Broadcasting Commission; copyright: ; license: ;
2. *Alpino Dutch Treebank* [download | source]
   id: `alpino`; size: 2797255; author: ; copyright: ; license: Distributed with permission of Gertjan van Noord;
3. *Averaged Perceptron Tagger* [download | source]
   id: `averaged_perceptron_tagger`; size: 2526731; author: ; copyright: ; license: ;
4. *Averaged Perceptron Tagger (Russian)* [download | source]
   id: `averaged_perceptron_tagger_ru`; size: 8628828; author: ; copyright: ; license: ;
5. *Grammars for Basque* [download | source]
   id: `basque_grammars`; size: 4704; author: Kepa Sarasola; copyright: ; license: ;
6. *BioCreAtIvE (Critical Assessment of Information Extraction Systems in Biology)* [download | source]
   id: `biocreative_ppi`; size: 223566; author: ; copyright: Public Domain (not copyrighted); license: Public Domain;
7. *BLLIP Parser: WSJ Model* [download | source]
   id: `bllip_wsj_no_aux`; size: 24516205; author: ; copyright: ; license: ;
8. *Grammars from NLTK Book* [download | source]
   id: `book_grammars`; size: 9103; author: Ewan Klein; copyright: ; license: ;
9. *Brown Corpus* [download | source]
   id: `brown`; size: 3314357; author: W. N. Francis and H. Kucera; copyright: ; license: May be used for non-commercial purposes.;

Source: https://www.nltk.org/nltk_data/

NLTK: Brown Corpus

The Brown University Standard Corpus of Present-Day American English (or just Brown Corpus) is an electronic collection of text samples of American English, the first major structured corpus of varied genres.

NLTK: Reuters Corpus

The Reuters Corpus (overlapping corpus):

- **10,788 news documents,**
- **1.3 million words,**
- **documents have been classified into 90 topics**

NLTK: Gutenberg Corpus

NLTK includes a small selection of texts from the Project Gutenberg electronic text archive, which contains electronic books (hosted at http://www.gutenberg.org/)