Lecture 2: Basics of Text Processing

Instructor: Stephanie Schoch

CSCI 680: Natural Language Processing
Department of Computer Science, College of William & Mary
August 28, 2026

WILLIAM & MARY

Many slides are adapted from Jurafsky & Martin (3rd ed.)

Learning Objectives

By the end of the lecture, students should be able to:

- Understand how regular expressions relate to **preprocessing**.
- Define **tokenization** and **describe challenges** with word-based tokenization.
- Explain **byte pair encoding.**

Lecture Outline

- Regular Expressions
- Words
- Tokenization

Regular Expressions

Regular expressions are used everywhere!

Regular expressions: A formal language for specifying text strings

- Part of most text processing tasks
  - Useful pre-processing or text formatting step (e.g. BPE pre-tokenization)
- Very useful for data analysis of text
- Used to search for a pattern in a string

Example: Python function

re.search(pattern,string)

scans through the string and returns the first match inside it for the pattern.

Regular Expressions: Disjunctions

Letters inside square brackets []

Pattern	Matches
r”[mM]ary”	mary or Mary
r”[1234567890]”	Any one digit

Ranges using the dash [A-Z]

Pattern	Matches	
r”[A-Z]”	An uppercase letter	Drenched Blossoms
r”[a-z]”	A lowercase letter	my beans were impatient
r”[0-9]”	A single digit	Chapter 1: Down the Rabbit Hole

Regular Expressions: More Disjunctions

Groundhog is another name for woodchuck!

The pipe symbol | for disjunction.

Pattern	Matches
r”groundhog|woodchuck”	woodchuck
r”[yours|mine]”	yours
r”a|b|c”	= [abc]
r”[gG]roundhog|[Ww]oodchuck”	Woodchuck

Regular Expressions: Negation in Disjunction

Carat as first character in [] negates the list

- Carat means negation only when it's first in []

Pattern	Matches	
r"[^A-Z]"	Not uppercase	Oyfn pripetchik
r"[^Ss]"	Neither 'S' nor 's'	I have no exquisite reason.
r"[^.]"	Not a period	Our resident.
r"[e^]"	Either e or ^	Look up ^ now

Regular Expressions: Anchors, Counting, Wildcards

Anchors

Pattern	Matches
r”^[A-Z]”	Williamsburg
r”\$”	The end.

Counting, Wildcards

Pattern	Matches	
r”baa*”	0 or more	baa, baaa, baaaa
r”baa+”	1 or more	baa, baaa, baaaa,
r”.”	Anything	anything…

Regular Expressions: Example

Find me all instances of the word “the” in a text.

`the`

Misses capitalized examples

`[tT]he`

Incorrectly returns `other` **or** `Theology`

`[^a-zA-Z0-9][tT]he[^a-zA-Z0-9]`

Errors: False Positives and Negatives

- The process we just went through was based on fixing two kinds of errors:
  1. **False negatives (Type II Errors):**
     Not matching things that we should have matched (The)

  2. **False positives (Type I Errors):**
     Matching strings that we should not have matched (there, then, other)

- In NLP, we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves to antagonistic efforts:
  - Increasing coverage (or *recall*) (minimizing false negatives)
  - Increasing accuracy (or *precision*) (minimizing false positives)

Substitutions & Capture Groups

Substitutions: To change every instance of cherry to apricot in string:

```
re.sub(r"cherry", r"apricot", string)
```

Capture Group: Use parentheses to capture (store) the values that we matched in the search

- Groups have numbers: in replacement we refer back to that group with a number command.

```
re.sub(r"(\d{2})/(\d{2})/(\d{4})", r"\2-\1-\3", string)}
```

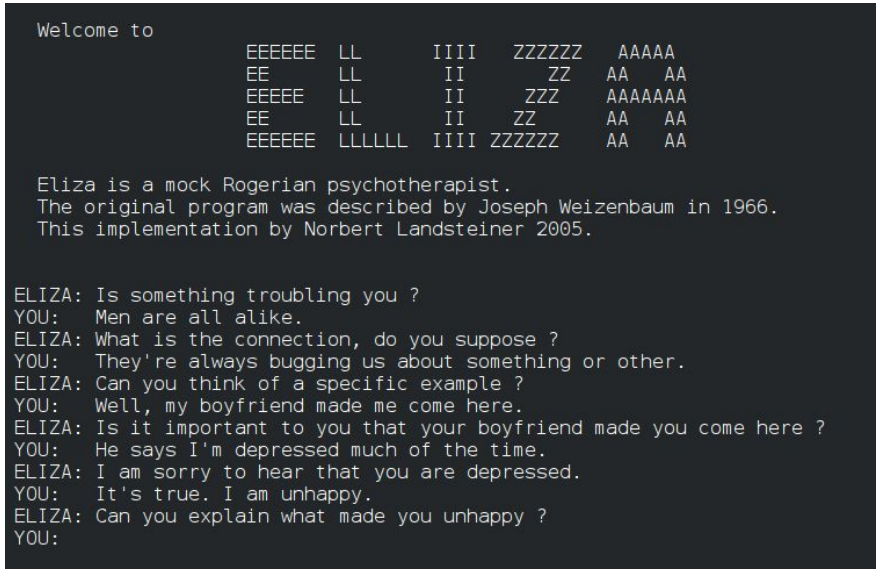
10/15/2011 → 15-10-2011

Simple Application: ELIZA

- Early NLP system that imitated a Rogerian psychotherapist (Joseph Weizenbaum, 1966).

- Uses pattern matching to match, e.g.,:
  “I need X”

  and translates them into, e.g.
  “What would it mean to you if you got X?



Simple Application: ELIZA

Men are all alike.
IN WHAT WAY

They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED

How ELIZA Works

re.sub(r".* I'M (depressed|sad) .*",r"I AM SORRY TO HEAR YOU ARE \1",input)

re.sub(r".* I AM (depressed|sad) .*",r"WHY DO YOU THINK YOU ARE \1",input)

re.sub(r".* all .*",r"IN WHAT WAY?",input)

re.sub(r".* always .*",r"CAN YOU THINK OF A SPECIFIC EXAMPLE?",input)

DEMO: https://www.masswerk.at/eliza/

Summary

- Regular expressions play a surprisingly large role in NLP
  - Part of most text processing tasks, even for big neural language model pipelines
    - Including text formatting and preprocessing
  - Very useful for data analysis of any text data

Lecture Outline

- Regular Expressions
- **Words**
- Tokenization

How Many Words: In A Sentence?

They picnicked by the pool, then lay back on the grass and looked at the stars.

- 16 words: if we don’t count punctuation marks as words
- 18 words: if we count punctuation

How Many Words: In An Utterance?

"I do uh main- mainly business data processing"

- **Disfluencies:**
  - Fragments *main-*
  - Filled pauses: *uh and um*

Should we consider these to be words?

How Many Words: In A Sentence?

They picnicked by the pool, then lay back on the grass and looked at the stars.

- **Type:** an element of the vocabulary V
  - The number of types is the vocabulary size |V|
- **Instance:** an instance of that type in running text.
  - 14 types and 16 instances (if we ignore punctuation).

More questions: Are They and they the same word?

How Many Words: In A Corpus?

Corpus	Types = |V|	Instances = N
Shakespeare	31 thousand	884,000
Brown Corpus	38 thousand	1 million
Switchboard Conversations	20 thousand	2.4 million
COCA	2 million	440 million
Google N-Grams	13+ million	1 trillion

The bigger the corpus, the more word types!

How Many Words: In A Corpus? Heaps Law / Herdan's Law

N = number of instances

$|V|$ = number of types in vocabulary V

$$|V| = kN^{\beta}$$ ← Roughly 0.5

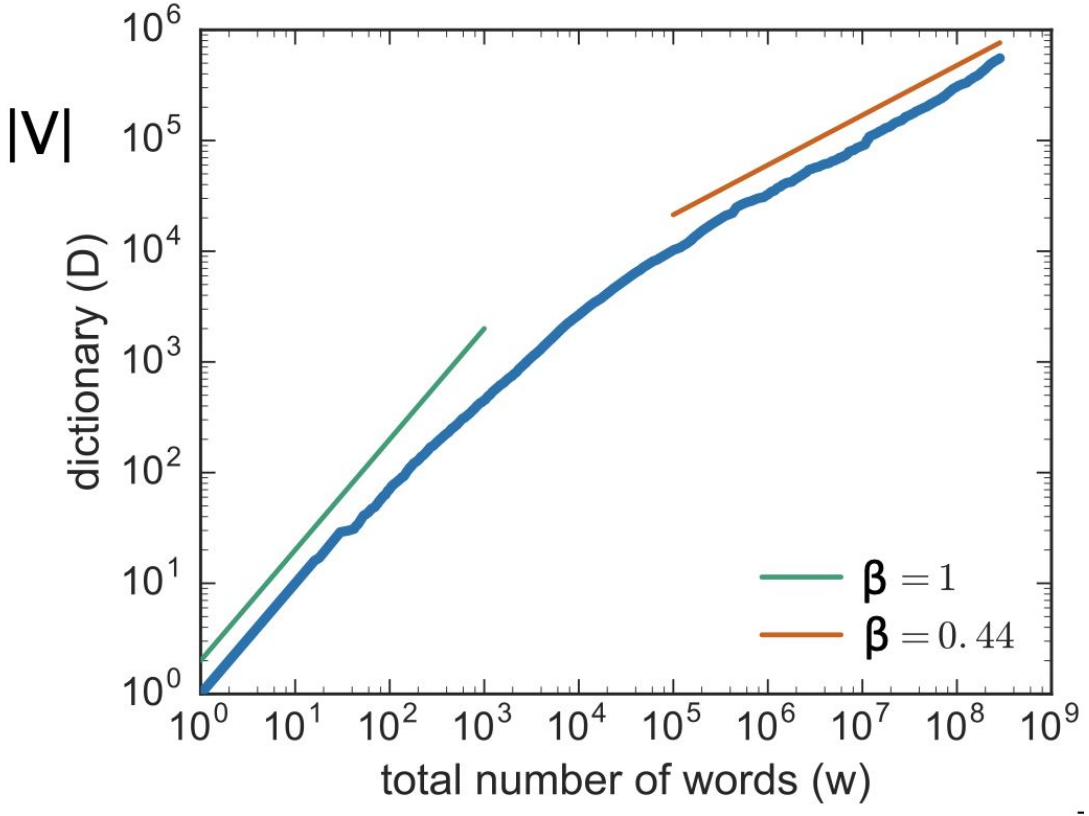
Vocab size for a text goes up with the square root of its length in words

How Many Words: In A Corpus? Heaps Law / Herdan's Law

Two Kinds of Words:

Function words: of, the, is, and, una, 是

Content words: mango, braise, snowy, feliz, 北京



Tria, Loreto, Servedio, 2018

Corpora

- Words don’t appear out of nowhere!

- A text is produced by:
  - a specific writer(s),
  - at a specific time,
  - in a specific variety,
  - of a specific language,
  - for a specific function.

Corpora Vary Along Different Dimensions

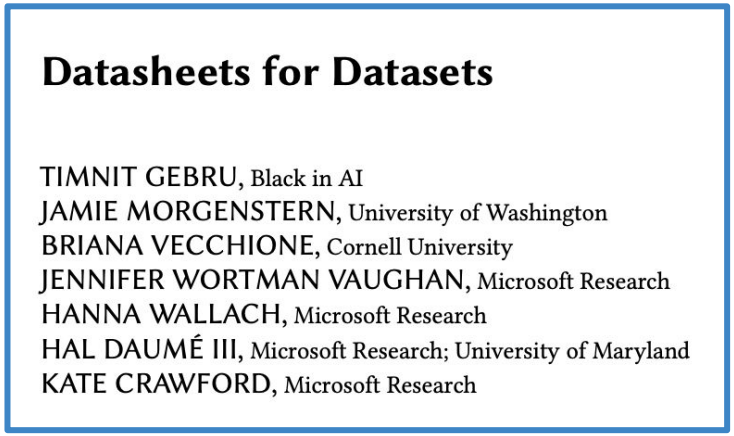
- **Language:** 7097 languages in the world
- **Variety:** like African American Language varieties
  - AAE Twitter posts might include forms like *“iont” (I don’t)*
- **Genre:** newswire, fiction, scientific articles, Wikipedia
- **Author demographics:** writer’s age, gender, ethnicity, SES
- **Code switching**: Speakers use multiple languages in same utterance
  - Common around the world, especially in spoken language and some genres (e.g. texting, social media)
  - Spanish/English: 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:)]

Corpus Datasheets

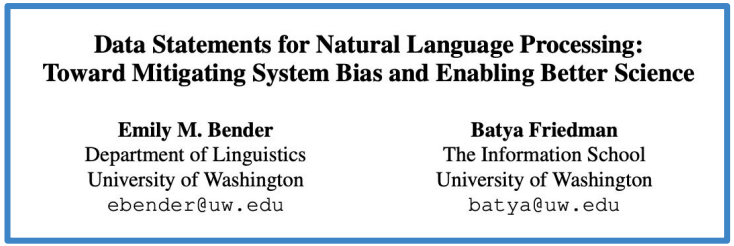
Gebru et al (2020), Bender and Friedman (2018)

- **Motivation:**
  - Why was the corpus collected?
  - By whom?
  - Who funded it?
- **Situation:** In what situation was the text written?
- **Collection process:** How was it sampled? Was there consent? Preprocessing?
- **Annotation process, language variety, demographics, etc.**



Datasheets for Datasets

TIMNIT GEBRU, Black in AI
JAMIE MORGENSTERN, University of Washington
BRIANA VECCHIONE, Cornell University
JENNIFER WORTMAN VAUGHAN, Microsoft Research
HANNA WALLACH, Microsoft Research
HAL DAUMÉ III, Microsoft Research; University of Maryland
KATE CRAWFORD, Microsoft Research



Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science

Emily M. Bender
Department of Linguistics
University of Washington
ebender@uw.edu

Batya Friedman
The Information School
University of Washington
batya@uw.edu

Lecture Outline

- Regular Expressions
- Words
- **Tokenization**

Word-based Tokenization

Rule-based tokenization:

- Pre-define a standard and implement rules for the kind of tokenization we are interested in
  - Parsing algorithms (need grammatical words as input)
  - Linguistic applications
  - Social science applications

Space-based tokenization:

- A very simple way to tokenize
  - For languages that use space characters between words
    - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
  - Segment off a token between instances of spaces

Issues for Tokenization

- We can’t just blindly remove punctuation:
  - m.p.h., Ph.D., AT&T, cap’n
  - prices ($45.55)
  - dates (01/02/06)
  - URLs (http://stephanieschoch.com/nlp)
  - hashtags (#nlproc)
  - email addresses (someone@wm.edu)
- Numbers are tokenized differently across languages:
  - English 555,500.50 = French 555 500,50
- Clitics: a word that doesn't stand on its own
  - "are" in we're, French "je" in j'ai, “le" in l'honneur
- Multiword expressions:
  - New York, rock ’n’ roll

Tokenization in NLTK

Bird, Loper and Klein (2009), *Natural Language Processing with Python*. O'Reilly

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     (?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(?:-\w+)*        # words with optional internal hyphens
...   | \$?\d+(?:\.\d+)?%?  # currency, percentages, e.g. $12.40, 82%
...   | \.\.\.              # ellipsis
...   | [][.,;"'?():_`-]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Tokenization in Languages without Spaces

- Not every language uses spaces to separate words
  (e.g. Chinese, Japanese, Thai)

  Where should the token boundaries be?

- **Example: Word Tokenization in Chinese**

  姚明进入总决赛 “Yao Ming reaches the finals”

  - Words are composed of characters called “**hanzi**” (or sometimes just “**zi**”)
    - Each represents a morpheme: average of 2.4 per word
    - Deciding what counts as a word is complex and not agreed upon

How to do Word Tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?

姚明 进入 总决赛

YaoMing reaches finals

Chinese Treebank

5 words?

姚 明 进入 总 决赛

Yao Ming reaches overall finals

Peking University

7 words?

姚 明 进 入 总 决 赛

Yao Ming enter enter overall decision game

Just use characters

Tokenization Across Languages

- In Chinese, we often use characters (zi) as tokens
- Word segmentation is more complex in other languages (e.g. Thai, Japanese)

Word-based Tokenization →

Subword-based Tokenization

Subword Tokenization

Instead of:

- White-space segmentation / orthographic words
  - Many languages don’t have these
  - # words grows without bound
- Unicode character segmentation
  - Too small as tokens for many purposes
- Morphemes
  - Hard to define

Use the data to tell us how to tokenize.

Subword tokenization (tokens can be parts of words, as well as whole words)

Subword Tokenization

Common Algorithms

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

All have 2 parts

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

Byte-Pair Encoding (BPE) Token Learner

Iteratively merge frequent neighboring tokens to create longer tokens.

Repeat:

- Choose most frequent neighboring pair ('A', 'B')
- Add a new merged symbol ('AB') to the vocabulary
- Replace every 'A' 'B' in the corpus with 'AB'.

Until 𝑘 merges

Vocabulary

[A, B, C, D, E]
[A, B, C, D, E, AB]
[A, B, C, D, E, AB, CAB]

Corpus

A B D C A B E C A B
AB D C AB E C AB
AB D CAB E CAB

BPE Token Learner Algorithm

Iteratively merge frequent neighboring tokens to create longer tokens.

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

BPE: Additional Information

- Generally, subword algorithms are run **within** space-separated words.

- Don't merge across word boundaries
  - Separate corpus by whitespace or similar, using specialized regular expressions.
  - Set of starting strings, with whitespace attached to start of each string.
  - Counts come from the corpus, but can only merge within strings.

- **BPE Tokens:**
  - Usually include frequent words & frequent subwords (e.g. morphemes like *-est* or *-er*)

BPE Token Learner

Original corpus:

set ␣ new ␣ new ␣ renew ␣ reset ␣ renew

Put space token at start of words

corpus		vocabulary
2	␣ n e w	␣, e, n, r, s, t, w
2	␣ r e n e w	
1	s e t	
1	␣ r e s e t	

BPE Token Learner

corpus

```
2   ␣ n e w
2   ␣ r e n e w
1   s e t
1   ␣ r e s e t
```

vocabulary

```
␣, e, n, r, s, t, w
```

Merge **n e** to **ne** (count 4 = 2 **new** + 2 **renew**)

corpus

```
2   ␣ ne w
2   ␣ r e ne w
1   s e t
1   ␣ r e s e t
```

vocabulary

```
␣, e, n, r, s, t, w, ne
```

BPE Token Learner

corpus	vocabulary
2 ␣ ne w	␣, e, n, r, s, t, w, ne
2 ␣ r e ne w	
1 s e t	
1 ␣ r e s e t	

Merge **ne w** to **new** (count 4)

corpus	vocabulary
2 ␣ new	␣, e, n, r, s, t, w, ne, new
2 ␣ r e new	
1 s e t	
1 ␣ r e s e t	

BPE Token Learner

corpus	vocabulary
2 ␣ new	␣, e, n, r, s, t, w, ne, new
2 ␣ r e new	
1 s e t	
1 ␣ r e s e t	

Merge ␣ r to ␣r (count 3) and ␣r e to ␣re (count 3)

corpus	vocabulary
2 ␣ new	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re
2 ␣re new	
1 s e t	
1 ␣re s e t	

System has learned the prefix re- !

BPE Token Learner

The next merges are:

merge	current vocabulary
(␣, new)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new
(␣re, new)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new, ␣renew
(s, e)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new, ␣renew, se
(se, t)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new, ␣renew, se, set

BPE Encoder Algorithm

- **Tokenize a test sentence:** run each merge learned from the training data:
  - Greedily, in the order we learned them
  - (test frequencies don't play a role)
- **First:** segment each test word into characters
- **Then run rules:**
  - (1) merge every **n e** to **ne**
  - (2) merge **ne w** to **new**
  - (3) **␣ r**
  - (4) **␣ re**
  - …
- **Result:**
  - Recreates training set words
  - Learns subwords like **␣ re** that may appear in new words (e.g. **rearrange**)

Pretokenization for BPE

```
>>> import regex as re
>>> pat = re.compile(
... # Contractions: 't and 'm are tokens
...     r"'s|'t|'re|'ve|'m|'ll|'d|"
... # Words:  sequence of Unicode letters (after optional space)
...     r" ?\p{L}+|"
... #Number: sequence of digits (after optional space)
...     r" ?\p{N}+|"
... # Punctuation: sequence of non-alphanumeric/non-space
...                           #(after optional space)
...     r" ?[^\s\p{L}\p{N}]+|"
... # whitespace
...     r"\s+(?!\S)|\s+"
... )
>>> text = "We're 350 dogs! Um, lunch?"
>>> print(pat.findall(text))
['We', "'re", ' 350', ' dogs', '!', ' Um', ',', ' lunch', '?']
>>>
```

Visualizing GPT-4o tokens

Tat Dat Duong's Tiktokenizer visualizer

Tokens: 11865, 8923, 11, 31211, 6177, 23919, 885, 220, 19427, 7633, 18887, 147065, 0

- Most words are tokens, w/initial space
- Clitics like **'s**
  - Are segmented off **Jane**
  - But part of frequent words like **she's**
- Numbers segmented into chunks of 3 digits
- **Anyhow** and **·anyhow** are segmented differently
- Some of this is from preprocessing
  - regular expressions for chunking digits, stripping clitics

Tokenizing Across Languages

Tat Dat Duong's Tiktokenizer visualizer

- Even though BPE tokenizers are multilingual, LLM training data is vastly dominated by English
  - Most BPE tokens used for English, leaving less for other languages
  - Words in other languages are often split up
  - Tokenization is generally better in English

English: 19 tokens; 1/16 words are split into multiple tokens

```
In·a·deep·bowl,·mix·the·orange·juice·with·the·sugar,·g
inger,·and·nutmeg.
```

Spanish: 33 tokens; 6/16 words are split

```
En·un·recipiente·hondo,·mezclar·el·jugo·de·naranja·con
·el·azúcar,·jengibre,·y·nuez·moscada.
```

Tokenizing Across Languages

Tat Dat Duong's Tiktokenizer visualizer

TikTokenizer DEMO (Using Google Translate):

https://translate.google.com/?sl=auto&tl=en&op=translate

Exit Survey (from last class… if you haven't already!):
https://forms.gle/Xa1yksySRyN4vRXJ7

TODOs:

- Start brainstorming for your project.

Next Lecture: **N-Gram Language Models (I)**