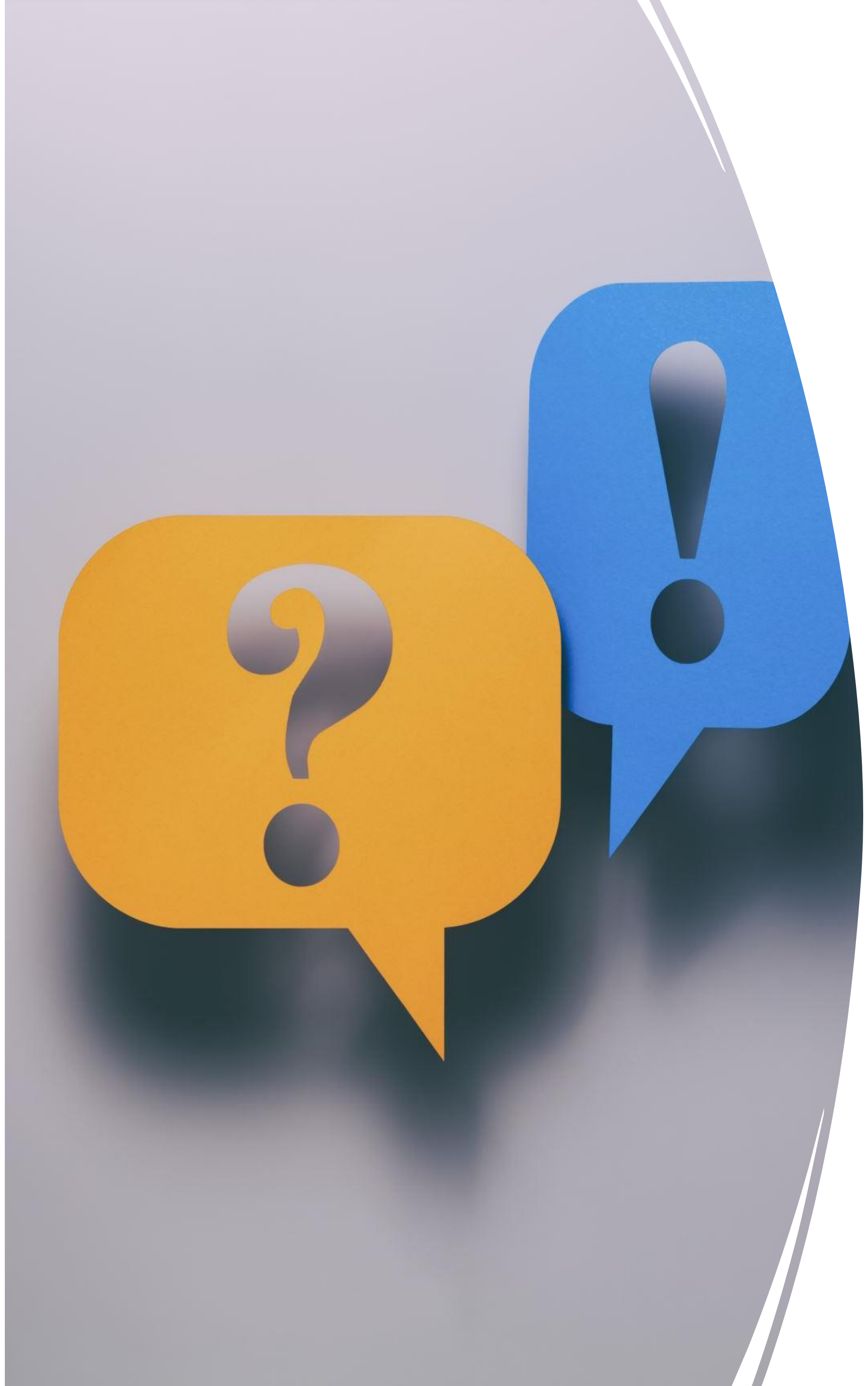
UNIVERSITÉ DU LUXEMBOURG

Natural Language Processing

Lecture 2: Basic Text Preprocessing

Salima Lamsiyah
University Luxembourg, FSTM, DCS, MINE Research Group
Salima.lamsiyah@uni.lu



Revisions

1. What is NLP ?
2. Give some examples of NLP applications
3. Why NLP is hard?
4. What are the different levels of ambiguity ?
5. How to deal with ambiguity?
6. What are the main components of NLP?

Lecture Plan

1. Corpora and Words
2. Text Normalization
    a. Tokenization
    b. Stemming
    c. Lemmatization
3. Sentence Segmentation
4. Regular Expressions

Corpora and Words

Corpora

- A corpus is a collection of text\speech
  - Often annotated in some way
  - Sometimes just lots of text

- 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 dimension like

- **Language**: 7097 languages in the world
- **Variety**, like African\Mainstream American Language varieties.
- **Code switching**, e.g., Spanish/English, Hindi/English:
- **Genre:** newswire, fiction, scientific articles, Wikipedia
- **Author Demographics**: writer's age, gender, ethnicity, socioeconomic class , …
- **Time:** language changes over time

Corpus datasheets| data statement

Motivation:
- Why was the corpus collected?
- By whom?
- Who funded it?

Situation: In what situation was the text written?

Collection process: If it is a subsample how was it sampled? Was there consent? Pre-processing?

- +**Annotation process, language variety, demographics, etc.**

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

How many words in a sentence?

- Example: "They lay back on the San Francisco grass and looked at the stars and their dreams took flight"

- **Tokens** are the total number of running words.
- **Types** are the number of distinct words in a corpus.

- How Many:
  - 18 tokens (or 17)
  - 16 types (or 15)

How many words in a sentence?

- I'm
- **Orthographically** one word (in the English writing system)

- But **grammatically** two words:
  1. the subject pronoun I
  2. the verb ’m, short for am.

How many words in a corpus?

- ***N*** = number of tokens
- ***V*** = vocabulary = set of types, **|V|** is size of the vocabulary.
- ***Heaps'Law*** **=** $|V| = kN^{\beta}$ ***where often β =0.5, 10<k<100***
- **Vocab size for a text goes up with the square root of its length in words**

Corpus	**Tokens = N**	**Types = |V|**
Switchboard phone conversations	2.4 million	
Shakespeare	884,000	
COCA	440 million	
Google N-grams	1 trillion	

How many words in a corpus?

- ***N*** = number of tokens
- ***V*** = vocabulary = set of types, **|V|** is size of the vocabulary.
- ***Heaps'Law =*** $|V| = kN^{\beta}$ ***where often β =0.5, 10<k<100***
- **Vocab size for a text goes up with the square root of its length in words**

Corpus	**Tokens = N**	**Types = |V|**
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Text Normalization

Text Normalization

- Every NLP application requires **Text Normalization**:

1. Word Tokenization
2. Word Normalization (normalizing word formats)
   - Stemming
   - Lemmatization
   - Lowercasing
3. Sentence Segmentation

Text Tokenization

Word Tokenization: **Space-based tokenization**

- **Word Tokenization** consists of splitting a sentence into an ordered list of individual words, usually referred to as tokens.

- A very simple way for tokenization **Space-based tokenization**
  - For languages that use space characters between words
  - Examples: Arabic, Latin, Greek, ...

- Unix Tools for space-based tokenization
  - The "**tr**" command
  - Inspired by Ken Church's Unix for Poets
  - Given a text file, output the word tokens and their frequencies

Tokenization in languages without spaces

- Many languages (like Chinese, Japanese, Thai) don’t use spaces to separate words!
- How do we decide where the token boundaries should be?

How to choose tokens in Chinese

- Chinese words are composed of characters called "**hanzi" (汉字)** (or sometimes just "**zi**")
- Each one represents a meaning unit called a morpheme.
- Each word has on average 2.4 of them.
- But deciding what counts as a word is complex and not agreed upon.

How to do choose tokens in Chinese?

- 姚明进入总决赛 “Yao Ming reaches the finals”
  - yáo míng jìn rù zǒng jué sài

- 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

- So Chinese uses characters (zi) as tokens

  - But that doesn't work for, e.g., Thai and Japanese
  - These differences make it hard to use words as tokens

- And there's another reason why we don't use words as tokens!

Morphological Typology

- Dimensions along which languages vary

- Two are salient for tokenization:
  1. number of morphemes per word
  2. how easy it is to segment the morphemes

Words have parts

- **Morpheme**: a minimal meaning-bearing unit in a language.
  - fox: one morpheme
  - cats: two morphemes cat and –s
- **Morphology**: the study of morphemes

Types of morphemes

- **root**: central morpheme of the word; supplying the main meaning

- **affix**: adding additional meanings

Examples:

worked

root work

ffix -ed

glasses

root glass

affix -es

Types of affixes

- **Inflectional** morphemes
  - grammatical morphemes
  - often syntactic role like agreement

    –ed past tense on verbs

    -s/-es plural on nouns

- **Derivational** morphemes
  - more idiosyncratic in application and meaning
  - often change grammatical class

    care (noun)

    \+ -*full* → careful (*adjective*)

    \+ -*ly* → *carefully (adverb)*

Issues in Tokenization

- **Can't just blindly remove punctuation:**
  - m.p.h., Ph.D.,
  - prices ($45.55)
  - dates (20/12/1992)
  - URLs (http://www.stanford.edu)
  - hashtags (#nlproc)
  - email addresses (someone@cs.colorado.edu)
- **Clitic: a word that doesn't stand on its own**
  - "are" in we're, French "je" in j'ai, "le" in l'honneur
- **When should multiword expressions (MWE) be words?**
  - New York, rock 'n' roll

Word Tokenization: **Subword Tokenization**

- Instead of
  - white-space segmentation
  - single-character segmentation

- **Use the data** to tell us how to tokenize.

- **Subword tokenization** (because tokens can be parts of words as well as whole words)

Subword Tokenization Methods: Examples

- Three common algorithms:
  - **WordPiece Tokenizer** (Schuster and Nakajima, 2012)
  - **Byte-Pair Encoding (BPE) Tokenizer** (Sennrich et al., 2016)
  - **SentencePiece Tokenizer** (Kudo, 2018)

- All have 2 parts:
  - A token **learner** that takes a raw training corpus and induces a vocabulary (a set of unique tokens).
  - A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Subword Tokenization: Byte Pair Encoding

- **Byte Pair Encoding** is a subword tokenization method that splits words into smaller parts by merging the most frequent pairs of characters or subwords.

- **How It Works**:
  - It starts by treating each character in a word as its own token.
  - The most frequent pair of characters is merged into a single token.
  - This process repeats until a desired vocabulary size is reached.

- **Example**: Let's tokenize **"lowering"** using BPE:
  - Step 1: Start with individual characters: l o w e r i n g
  - Step 2: Merge frequent pairs:
  - "low" + "er" + "ing"
  - Final tokens: low, er, ing

- So, **"lowering"** becomes: **low er ing**

Subword Tokenization: WordPiece Tokenizer

- **WordPiece** starts with individual characters and learns which subword combinations to merge based on maximizing the likelihood of the training data.

- **How It Works**:
  - Words are split into the smallest units or subwords.
  - These subwords are merged based on their likelihood in the data, and if a word is not in the vocabulary, it will be split into smaller known parts.

- **Example**: For **"playing"**:
  - WordPiece might split it into: play, ##ing
  - The ## symbol indicates that **"ing"** is a suffix attached to another part of the word.
- So, **"playing"** becomes**: play ##ing**

Subword Tokenization: SentencePiece Tokenizer

- **SentencePiece** is a tokenization algorithm that doesn't require space as a boundary for words. It treats the entire sentence as a stream of characters and breaks it into subwords or tokens based on frequency.

- **How It Works**:
  - It works on raw text (without spaces as word boundaries).
  - It segments the text into subword units using a probabilistic model.

- **Example**: Let’s use **"I am learning"** with SentencePiece:
  - SentencePiece could split it into: I, _am, _learn, ing
  - The _ symbol before **"am"** and **"learn"** represents a space that was part of the original sentence.
- So, **"I am learning"** becomes: **I _am _learn ing**

Subword Tokenization Methods

Check this link for more details and examples

Tokenization in NLTK

```
from nltk.tokenize import word_tokenize
text = "I am learning NLP."
tokens = word_tokenize(text)
print(tokens)
```

```
['I', 'am', 'learning', 'NLP', '.']
```

Treebank Tokenizer

```
from nltk.tokenize import TreebankWordTokenizer

tokenizer = TreebankWordTokenizer()
sentence = "They'll meet at 5:00 p.m. on Jan. 5th, 2021."
tokens = tokenizer.tokenize(sentence)
print(tokens)
```

```
['They', "'ll", 'meet', 'at', '5:00', 'p.m.', 'on', 'Jan.', '5th', ',', '2021', '.']
```

Regular Expression Tokenization

```
from nltk.tokenize import regexp_tokenize
text = "The price is $45.50 for a book."
tokens = regexp_tokenize(text, r'\w+|\$[\d\.]+')
print(tokens)
```

```
['The', 'price', 'is', '$45.50', 'for', 'a', 'book']
```

Tokenization in SpaCy

```python
import spacy

# Load the English language model
nlp = spacy.load('en_core_web_sm')

text = "I’m learning NLP with spaCy!"
doc = nlp(text)

# Tokenizing the text
tokens = [token.text for token in doc]
print(tokens)
```

```
['I', '’m', 'learning', 'NLP', 'with', 'spaCy', '!']
```

Word Normalization

Word Normalization

- Putting words/tokens in a standard format
  - U.S.A. or USA
  - making or made
  - Fed or fed
  - am, is, be, are
  - studies, studying, studied, study
  - …

Word Normalization: Case Folding

- Applications like Information Retrieval: reduce all letters to lower case is helpful

- For Named Entity Recognition, Machine Translation, Part-of-speech tagging, and other applications

  - Case is not helpful (**US** versus **us** is important)

Word Normalization: Stemming

- **Stemming** is the process of chopping off affixes (prefix, suffix, infix) crudely from a word in order to obtain a word **stem (root)**.

- Porter Stemmer is the most used stemming method, Developed by Martin Porter in the 1980s
  - Based on a series rules
  - Some sample rules:
    - **ATIONAL => ATE** (e.g., communicational => communicate)
    - **ING => remove it if stem contains vowel** (e.g., playing => play)
    - **SSES => SS** (e.g., processes => process)

Word Normalization: Stemming

- **Example**

This was not the map we found in Billy Bones’s chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.

Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note
.

Word Normalization: Lemmatization

Lemmatization returns the canonical form, dictionnary form of a word. The output we will get after lemmatization is called 'lemma'.

- Am, are, is => be
- Cat, cats, cat's, cats' => cat
- I was in a meeting => I be in a meeting
- We were meeting constantly => We be meet constantly

Sentence Segmentation

- **Sentence segmentation** consists of splitting the text into a set of sentences using a specific sentence splitter.
- !, ? mostly unambiguous but period "." is very ambiguous because:
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
- Common algorithm: Tokenize first: use rules or Machine Learning to classify a period as either (a) part of the word or (b) a sentence-boundary.
  - An abbreviation dictionary can help
- Sentence segmentation can then often be done by rules based on this tokenization

Text Preprocessing Libraries

1. Natural Language Toolkit (NLTK): https://www.nltk.org/
2. spaCy Library: https://spacy.io/

https://www.activestate.com/blog/natural-language-processing-nltk-vs-spacy/

Other Text Preprocessing Techniques

- Stop-words removal
- Special Character Removal
- Sometimes Removing Emails, URLS, XML tags, …

https://kavita-ganesan.com/text-preprocessing-tutorial/#.YzBavuxBy3K

Regular Expressions

Regular Expressions

- Regular expression (RE) is a formal language that specifies text search strings.

- Formally, a *regular expression* is an algebraic notation for characterizing a set of strings.
- Very useful for searching in texts, when we have a ***pattern*** to search for and a ***corpus*** of texts to search through.

- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions: Concatenation

- Concatenation

Reglaur Expression	**Example Patterns Matched**
/woodchuck/	"groundhogs, also known as **woodchuck**, are rodent of the family Sciuridae"
/a/	"Bring me **a** cup of te**a**"
/!/	” Just enjoy the NLP course**!** "

Regular Expressions: Disjunction

- Disjunction

The use of the brackets [] to specify a disjunction of characters.

Reglaur Expression	**Match**
/**[wW]**oodchuck/	**w**oodchuck or **W**oodchuck
/[abc]/	‘a’, ‘b’, or, ‘c’
/[1234567890]/	Any digit

The use of the brackets [] Plus the dash - to specify a range.

Reglaur Expression	**Match**
/**[A-Z]**/	An upper case letter
/**[a-z]**/	A lower case letter
/**[0-9]**/	A single digit

Regular Expressions: Negation in Disjunction

- Negations [^a]:
  - Caret ^ means negation only when it is the first symbol in []

Reglaur Expression	Match (single character)
/[^A-Z]/	Not an upper case letter
/[^aA]/	Neither ’a’ nor ’A’
/[^e^]/	Neither ’e’ nor ’^’
/[e^]/	Either ’e’ or ’^’
/a^b/	The pattern a^b

Regular Expressions: ? * + .

- The question mark **?** means the preceding character or nothing.
- The asterisk ***** indicates zero or more occurrences of the preceding element.
- The plus sign **+** indicates one or more occurrences of the preceding element.
- The period (/**.**/), a wildcard expression that matches any single character except a carriage return character \r.

Reglaur Expression	**Match**
/woodchucks**?**/	woodchuck or woodchucks
/ab*****c/	ac, abc, abbc, abbbc, and so on.
/ab**+**c/	abc, abbc, abbbc, and so on, but not "ac"
/beg**.**n/	Begin, began, beg'n, and so on.

Regular Expressions: Anchors ^ $

- Anchors are special characters that don’t match any characters but anchor regular expressions to particular places in a string.
- The most common anchors are the caret **^** and the dollar **$**
- The caret **^** matches the start of the line.
- **Remark:** The caret has 3 uses:
    - To match a start of a line,
    - To indicate a negation inside of square brackets,
    - And just mean a caret.
- The dollar sign **$** matches the end of a line.

Regular Expressions: Anchors ^ $

- Examples:

Reglaur Expression	Match	Example Pattern Matched
/^[A-Z]/	Start of the line	Salima Lamsiyah
/^[^A-Za-z]/	Start of the line	1. “HELLO”
\.$	End of line	The end.
.$	End of line	The end? The end!
/^The dog\. $/	Start and end of the line	The dog.
\b	Word boundary	/\bthe\b/ returns **the**

Regular Expressions: More Disjunction

- Woodchuck is another name for groundhog!
  - The pipe | for disjunction

Reglaur Expression	**Match**
	Either the string **woodchuck** or **groundhog**
	Either the string **guppy** or **guppies**
	Woodchuck, woodchuck, Groundhog, groundhog

Regular Expressions: More Disjunction

- Woodchuck is another name for groundhog!
  - The pipe | for disjunction

Reglaur Expression	**Match**
/woodchuck|groundhog /	Either the string **woodchuck** or **groundhog**
/gupp (y|ies)/	Either the string **guppy** or **guppies**
[gG]roundhog|[Ww]oodchuck	Woodchuck, woodchuck, Groundhog, groundhog

Regular Expressions

- Regular Expression Operator precedence hierarchy

<table>
<tr><td>Parenthese</td><td>()</td></tr>
<tr><td>Counters</td><td>* + ? {}</td></tr>
<tr><td>Sequences and anchors</td><td>The ^my end$</td></tr>
<tr><td>Disjunction</td><td>|</td></tr>
</table>

More Regular Expressions Operators

RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression
{,m}	up to *m* occurrences of the previous char or expression

Example

- Find all the instances of the word "there" in a text.

- You can use RegEx in Python
- https://regex101.com/
- https://www.w3schools.com/python/python_regex.asp

Example

- Find me all the instances of the word "there" in a text.
  - /there/ => Misses capitalized examples
  - /[Tt]here/ => Incorrectly returns *therefore*
  - /*\b*[Tt]here\b/ => will not return (there_ or there10)
  - /[^a-zA-Z][Tt]here[^a-zA-Z]/ => Might not return there when it begins a line
  - Final Regular Expression: /(^|[^a-zA-Z])[Tt]here([^a-zA-Z])|$/

Error

- The process we just went through was based on fixing two kinds of errors:
  1. Matching strings that we should not have matched (therefore, weathered, feathered)

     False positives (Type I errors)

  2. Not matching string that we should have matched (There)

     False negatives (Type II errors)

Error

- In NLP we are always dealing with these kinds of errors.

- Reducing the error rate for an application often involves two antagonistic efforts:

  - **Increasing accuracy or precision** (minimizing false positives)

  - **Increasing coverage or recall** (minimizing false negatives).

Exercice

- Write regular expressions for the following languages.

1. The set of all alphabetic strings;
2. The set of all lower case alphabetic strings ending by b;
3. All strings that start at the beginning of the line with an integer and that end at the end of the line with a word;
4. Check date in DD/MM/YYYY and DD.MM.YYYY formats
5. Write a regular expression that matches Column 1 Column 2 Column 3

You can use https://regex101.com/

More Regular Expressions (Optional)

Substitutions

- Substitution in Python and UNIX commands:

s/regexp1/pattern/

e.g. **s**/colour/color/

More Regular Expressions

Capture Groups

- Say we want to put angles around all numbers:

  e.g. the 40 boxes => the <40> boxes

- Use () to "capture" a pattern into a numbered register (1, 2, 3…)
- Use \1 to refer to the contents of the register

  S / ([0-9]+) /<\1>/

More Regular Expressions

Capture Groups: Multiple Registers

/the (.*)er they (.*), the \1er we \2/

- Matches:

  the faster they ran, the faster we ran

- But not

  the faster they ran, the faster we played

More Regular Expressions

But suppose we don't want to capture?

- Parentheses have a double function: grouping terms, and capturing
- Non-capturing groups: add a **?:** after parentheses:

/(?:some|a few) (people|cats) like some \1/

- matches

◦ some cats like some cats

but not

◦ some cats like some people

Simple Regular Expressions Application: ELIZA

Chatbot ELIZA

- ELIZA an early chatbot 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 Regular Expressions Application: ELIZA

Chatbot ELIZA

- Let’s chat with ELIZA https://web.njit.edu/~ronkowit/eliza.html.

User$_1$:	Men are all alike.
ELIZA$_1$:	IN WHAT WAY
User$_2$:	They’re always bugging us about something or other.
ELIZA$_2$:	CAN YOU THINK OF A SPECIFIC EXAMPLE
User$_3$:	Well, my boyfriend made me come here.
ELIZA$_3$:	YOUR BOYFRIEND MADE YOU COME HERE
User$_4$:	He says I’m depressed much of the time.
ELIZA$_4$:	I AM SORRY TO HEAR YOU ARE DEPRESSED

a sample conversation with ELIZA in 1966

Simple Regular Expressions Application: ELIZA

How ELIZA Works

```
s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing application

- For hard tasks, we use machine learning classifiers
  - But regular expressions are still used for pre-processing, or as features in the classifiers
  - Can be very useful in capturing generalizations

Reading

- Please refer to this document : Chapter 2

- Regular Expressions:

https://docs.python.org/3/library/re.html#
https://en.wikipedia.org/wiki/Regular_expression
https://docs.python.org/3/library/re.html#
https://regex101.com/
https://regexr.com/
https://www.youtube.com/watch?v=V_BozMwoYe4&ab_channel=techTFQ
https://www.w3resource.com/python-exercises/re/

- ELIZA Chatbot

https://web.njit.edu/~ronkowit/eliza.html
http://psych.fullerton.edu/mbirnbaum/psych101/eliza.htm

Next Class

- **Edit Minimum Distance**
- **Text Representation Part 1**