

Preprocessing

NLP Week 2

Thanks to Dan Jurafsky for some of the slide inspirations!

Plan for today

1. What is preprocessing?
2. Characterizing strings with RegEx
3. Tokenization
4. Word normalization and lemmatization
5. *Group exercises*

What is preprocessing?

Preprocessing is the first step to all NLP tasks and models ; It involves *normalizing* text data.

Google is set to file its own suggestions for fixing the search monopoly by Dec. 20. Both sides can modify their requests before Judge Mehta is expected to hear arguments on the remedies this spring. He is [expected to rule](#) by the end of the summer.

Kent Walker, Google's president of global affairs, called the government's proposal "extreme."

"D.O.J.'s wildly overbroad proposal goes miles beyond the court's decision," he said in a blog post. "It would break a range of Google products — even beyond Search — that people love and find helpful in their everyday lives."

What is preprocessing?

Preprocessing is the first step to all NLP tasks and models ; It involves *normalizing* text data.

Google is set to file its own suggestions for fixing the search monopoly by **Dec. 20**. Both sides can modify their requests before Judge Mehta is expected to hear arguments on the remedies this spring. He is expected to rule by the end of the summer.**\n**

Kent Walker, Google's president of global affairs, called the government's proposal "extreme."

"D.O.J.'s wildly overbroad proposal goes miles beyond the court's decision," he said in a blog post. "It would break a range of Google products — even beyond Search — that people love and find helpful in their everyday lives."

What is preprocessing?

Preprocessing is the first step to all NLP tasks and models ; It involves *normalizing* text data.

Google is set to file its own suggestions for fixing the search monopoly by **Dec. 20**. Both sides can modify their requests before Judge Mehta is expected to hear arguments on the remedies this spring. He is expected to rule by the end of the summer.**\n**

Kent Walker, Google's president of global affairs, called the government's proposal "extreme."

"D.O.J.'s wildly overbroad proposal goes miles beyond the court's decision," he said in a blog post. "It would break a range of Google products — even beyond Search — that people love and find helpful in their everyday lives."

What about splitting text into words? Subwords?

What is preprocessing?

Preprocessing is the first step to all NLP tasks and models ; It involves *normalizing* text data, by:

- **Cleaning formatting**
- **Normalizing words**
- **Segmenting text into subwords, words, sentences, ...**

Regular Expressions (Regex)

Regex is a formal language for specifying text strings

Eg. How can we search for mentions of these cute animals in text?

- woodchuck
- woodchucks
- Woodchuck
- Woodchucks
- Groundhog
- groundhogs



Can we write a formal specification that accounts for all of these?

RegEx: Disjunctions (Or)

- Letters or ranges inside brackets [] will represent disjunctive expressions

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any one digit

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

RegEx: Disjunctions (Or)

- The pipe symbol | is also for disjunction

Pattern	Matches
<code>groundhog woodchuck</code>	woodchuck
<code>yours mine</code>	yours
<code>a b c</code>	= <code>[abc]</code>
<code>[gG]roundhog [Ww]oodchuck</code>	Woodchuck

RegEx: wildcards, etc: . ? * +

- Wildcards can match any symbol for a defined number of times.

Pattern	Matches	Examples
beg.n	Any char	<u>begin</u> <u>begun</u> <u>beg3n</u>
woodchucks?	Optional s	<u>woodchuck</u> <u>woodchucks</u>
to*	0 or more of previous	<u>t</u> <u>to</u> <u>too</u> <u>tooo</u>
to+	1 or more of previous	<u>to</u> <u>too</u> <u>tooo</u> <u>toooo</u>

Regular Expressions (RegEx)

RegEx is a formal language for specifying text strings

Eg. How can we search for mentions of these cute animals in text?

- woodchuck
- woodchucks
- Woodchuck
- Woodchucks
- Groundhog
- groundhogs



[gG] roundhogs? | [Ww] oodchucks?

RegEx: Negation (Not)

Carat (^) as first character in [] negates the list:

- Carat means negation only when it's first in []
- Other Special characters (., *, +, ?) lose their special meaning inside []

Pattern	Matches	Examples
[^A-Z]	Not an upper case	O <u>y</u> fn pripetchik
[^Ss]	Neither 'S' nor 's'	<u>I</u> have no exquisite
[^.]	Not a period	<u>O</u> ur resident Djinn
[e^]	Either e or ^	Look up <u>^</u> now

RegEx: Anchors ^ \$

- ^ at the beginning of a regex is for at beginning of string
- \$ marks the end of string

Pattern	Matches
<code>^[A-Z]</code>	<u>P</u> alo Alto
<code>^[^A-Za-z]</code>	<u>1</u> "Hello"
<code>\. \$</code>	The end <u>.</u>
<code>. \$</code>	The end <u>?</u> The end <u>!</u>

RegEx aliases

- Common regex have shortcuts.

Pattern	Expansion	Matches	Examples
<code>\d</code>	<code>[0-9]</code>	Any digit	Fahrenheit <u>4</u> 51
<code>\D</code>	<code>[^0-9]</code>	Any non-digit	<u>B</u> lue Moon
<code>\w</code>	<code>[a-zA-Z0-9_]</code>	Any alphanumeric or _	<u>D</u> aiyu
<code>\W</code>	<code>[^\w]</code>	Not alphanumeric or _	Look <u>!</u>
<code>\s</code>	<code>[\r\t\n\f]</code>	Whitespace (space, tab)	Look <u>_</u> up
<code>\S</code>	<code>[^\s]</code>	Not whitespace	<u>L</u> ook up

RegEx in Python

Regex and Python both use backslash "\" for special characters.

- Must type an extra backslash for regexes in Python!
 - Eg. `"\\d+"` to search for 1 or more digits
- **OR:** use Python's **raw string notation** for regex:
 - `r"\d+"` matches one or more digits (instead of `"\\d+"`)

Why RegEx?

Widely used in both academics and industry:

1. Part of most text processing tasks, even for big LLM pipelines
2. Very useful for data analysis of any text data

.

First chatbot: ELIZA (1966)

- ELIZA was an early NLP system meant to imitate a Rogerian psychotherapist.
- It worked via **pattern matching**:
 - **Input:** “I need X.”
 - **Output:** “What would it mean if you got X?”

Welcome to

```
EEEEEE LL      IIII  ZZZZZZ  AAAAA
EE      LL      II    ZZ      AA   AA
EEEEEE LL      II    ZZZ     AAAAAA
EE      LL      II    ZZ      AA   AA
EEEEEE LLLLLL  IIII  ZZZZZZ  AA   AA
```

Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.

```
ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

First chatbot: ELIZA (1966)

- Pattern matching examples from ELIZA:
 - `r".* I'm (depressed|sad)" -> "I am sorry to hear that you are X"`
 - `r".* all .*" -> "In what way?"`
 - `r".* always .*" -> "Can you think of a specific example?"`

Tokenization

Tokenization is to segment text into tokens (or meaningful units for processing).

Sample Data:

"This is tokenizing."

Character Level

[T] [h] [i] [s] [i] [s] [t] [o] [k] [e] [n] [i] [z] [i] [n] [g] [.]

Word Level

[This] [is] [tokenizing] [.]

Subword Level

[This] [is] [token] [izing] [.]

Why is it important?

- **Tokenization is the first step to all NLP language models.**
- Language is a symbolic system that takes continuous signal (e.g. audio speech, or hand gestures) and transforms it into a sequence of meaningful symbols (e.g. morphemes, signs, or words) -> **tokenization is what decides what those meaningful units are in an NLP system.**

Tokens vs types

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

Eg. Assume we are tokenizing on the word-level:

“The NLP course students enjoyed the course very much”

How many tokens?

How many types?

Tokens vs types

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

Eg. Assume we are tokenizing on the word-level:

“The NLP course students enjoyed the course very much”

How many tokens? 9

How many types?

Tokens vs types

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

Eg. Assume we are tokenizing on the word-level:

“The NLP course students enjoyed the course very much”

How many tokens? 9

How many types? 7

Simple tokenization

- A very simple word tokenization algorithm could split text on spaces and punctuation marks.

```
str = "He is expected by the end of the summer."
```

```
delimiter = r"[,\?\.!]\s?|\s"
```

```
tokens = str.split(delimiter)
```

```
["He", "is", "expected", "by", "the", "end", "of", "the", "summer"]
```

Issues with tokenization

Can't just blindly remove punctuation

- m.p.h., Ph.D., AT&T, cap'n
- prices (\$45.55)
- dates (01/02/06)
- URLs (<http://www.hec.ca>)
- hashtags (#hec)
- email addresses (someone@cs.colorado.edu)
- **Clitic:** a word that doesn't stand on its own, eg. "are" in we're, French "je" in j'ai, "le" in l'honneur

What about spaces? When should multiword expressions be words?

- New York, rock 'n' roll

Tokenization with NLTK

- **Alternative:** instead of matching delimiters between words, lets try to match sets of characters that can be words!

```
>>> 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 and 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 other languages

But... Not all languages uses spaces or single characters!

So what determines token boundaries in other languages?

The example of Chinese

Chinese words are composed of characters called "**hanzi**" (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.

Tokenization in other languages

So how should we tokenize 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

5 words?

姚 明 进入 总 决赛

Yao Ming reaches overall finals

7 characters? (don't use words at all):

姚 明 进 入 总 决 赛

Yao Ming enter enter overall decision game

Statistical tokenization

Can we be *smarter* with tokenization and try to *learn* the relevant morphemes of a language based on a corpus?

Three common algorithms:

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

Statistical tokenization

- There are two parts to any statistical tokenization algorithm:
 1. **A learner** which *learns* the vocabulary or set of tokens for a language given a train corpus
 2. **A segmenter** which *segments* a test corpus into a sequence of learnt vocabulary tokens.

Byte-Pair Encoding (BPE)

THE LEARNER

1. Initialize the vocabulary as the list of all individual characters:
 $V = [A, B, C, D, \dots, a, b, c, d, \dots]$
2. Then repeat the following steps until termination condition is reached:
 1. Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
 2. Add a new merged symbol 'AB' to the vocabulary
 3. Replace every adjacent 'A' 'B' in the corpus with 'AB'.
3. Termination condition: the vocabulary desired size is reached.

Byte-Pair Encoding (BPE)

THE LEARNER

```
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  $V$ 
```


Byte-Pair Encoding (BPE)

THE LEARNER

Note

To also account for existing signal coming from spaces in many languages a special end-of-word symbol '___' is commonly added to all words before spaces in the training corpus, before training the learner.

e.g. Tokenization_ is_ the_ first_ step_ to_ all_ NLP_ language_ models_ .

Byte-Pair Encoding (BPE)

THE SEGMENTER

- Now that we have learnt a set of tokens, how do we actually segment our test corpus?
- Looping over the vocabulary, with merged tokens **in the order in which we learned them**:
 1. For every learnt merged symbol, eg, 'AB', in the the vocabulary:
 - 1.1. Greedily replace every adjacent 'A' 'B' in the corpus with 'AB'.

Byte-Pair Encoding (BPE)

```
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  $V$ 
```

If your train corpus = your test corpus, this last step takes care of the segmenter part of the algorithm.

Word Normalization

Tokenization is a preprocessing step; there are other forms of preprocessing we might want to consider, such as various forms of **word normalization**.

- **Case folding:** putting all characters in the same case (e.g. NLP -> nlp)

Note: Case can be useful for some semantic-based tasks
(e.g. US versus us mean different things)

- **Lemmatization:** replacing words by their dictionary word (e.g. am, are, is -> be)

Note: Syntactic cues like tense, plural markers etc. can be useful for tasks like language generation.

- **Stemming:** removing affixes from words, keeping only stems (similar purpose to lemmatization)

Lemmatization

Represent words by their ***lemma***, their shared root= dictionary headword form:

am, are, is -> be

car, cars, car's, cars' -> car

He is reading detective stories -> He be read detective story

Lemmatization

Lemmatization is done using a morphological parser (most often heuristic rule-based parsers that are language dependent) that splits words into morphemes.

Recall, last week we talked about ***morphemes***, the smallest units of meaning in a language. There are two types:

1. **Stems** : the core meaning-bearing units
2. **Affixes** : subwords that must attach to a stem, often have grammatical functions.

heavenly -> heaven/-ly

cars -> car/-s

Incompetent -> in-/competent

Stemming

Stemming is a slightly simpler approach with the same ethos as lemmatization where you remove all affixes, reducing all words to their stems.

heavenly -> *heaven*

cars -> *car*

Incompetent -> *competent* ?? *Incompetent*

In practice, stemming is often crude and selective in which affixes are removed, dependent of the set of heuristic rules used)

Stemming

In practice, stemming is often crude and selective in which affixes are removed, dependent of the set of heuristic rules used.

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 .

[15 minute break]

Let's try preprocessing

Team up!

Open exercises/week 2 in your course folder and start writing/running code!