

Words & Regexes

CS 485, Fall 2026

Applications of Natural Language Processing

Brendan O'Connor

College of Information and Computer Sciences
University of Massachusetts Amherst

- To do NLP you need to get started with text data
 1. Text normalization: cleaning up text to reasonable sequences of words (tokens)
 2. Regular expressions: a computational tool to help, and even do rule-based NLP

Text normalization

- Every NLP task needs text normalization
 - 1. Segment sections of text, possibly paragraphs/sentences/etc.
 - 2. Segment/tokenize words in running text
 - 3. Normalizing word formats (e.g. case sensitive or not?)

Raw text

DISCHARGE CONDITION: The patient was able to oxygenate on room air at 93% at the time of discharge. She was profoundly weak, but was no longer tachycardic and had a normal blood pressure. Her respirations were much improved albeit with transmitted upper airway sounds.

DISCHARGE STATUS: The patient will be discharged to [**Hospital **] for both pulmonary and physical rehabilitation.

DISCHARGE MEDICATIONS:

1. Levothyroxine 75 mcg p.o. q.d.
2. Citalopram 10 mg p.o. q.d.
3. Aspirin 81 mg p.o. q.d.
4. Fluticasone 110 mcg two puffs inhaled b.i.d.
5. Salmeterol Diskus one inhalation b.i.d.
6. Acetaminophen 325-650 mg p.o. q.4-6h. prn.

Preprocessing: Text cleaning

DISCHARGE CONDITION: The patient was able to oxygenate on room air at 93% at the time of discharge. She was profoundly weak, but was no longer tachycardic and had a normal blood pressure. Her respirations were much improved albeit with transmitted upper airway sounds.

DISCHARGE STATUS: The patient will be discharged to [**Hospital1 **] for both pulmonary and physical rehabilitation.

DISCHARGE MEDICATIONS:

1. Levothyroxine 75 mcg p.o. q.d.
2. Citalopram 10 mg p.o. q.d.
3. Aspirin 81 mg p.o. q.d.
4. Fluticasone 110 mcg two puffs inhaled b.i.d.
5. Salmeterol Diskus one inhalation b.i.d.
6. Acetaminophen 325-650 mg p.o. q.4-6h. prn.



The patient was able to oxygenate on room air at 93% at the time of discharge. She was profoundly weak, but was no longer tachycardic and had a normal blood pressure. Her respirations were much improved albeit with transmitted upper airway sounds.

- Remove unwanted structure to just the sentences/ paragraphs you want to analyze
- Get text out of weird formats like HTML, PDFs, or idiosyncratic formatting (e.g. in these EHRs)

This step is usually specific to your dataset

Preprocessing: Tokenization

The patient was able to oxygenate on room air at 93% at the time of discharge. She was profoundly weak, but was no longer tachycardic and had a normal blood pressure. Her respirations were much improved albeit with transmitted upper airway sounds.

↓

```
['The', 'patient', 'was', 'able', 'to', 'oxygenate',  
'on', 'room', 'air', 'at', '93', '%', 'at', 'the',  
'time', 'of', 'discharge', '.', 'She', 'was',  
'profoundly', 'weak', ',', 'but', 'was', 'no',  
'longer', 'tachycardic', 'and', 'had', 'a',  
'normal', 'blood', 'pressure', '.', 'Her',  
'respirations', 'were', 'much', 'improved',  
'albeit', 'with', 'transmitted', 'upper', 'airway',  
'sounds', '.']
```

- Words are (usually) the basic units of analysis in NLP.
- In English, words are delineated as tokens via space and punctuation conventions, recognizable via moderately simple rules
- Tokenization: from text string to sequence of word strings
- Sentence splitting: harder but sometimes done too

*There are good off-the-shelf tokenizers
(NLTK, SpaCy, Stanza, Twokenizer, Huggingface...)*

Preprocessing: Normalization

- Often:
 - Lowercase words (“She” -> “she”)
- Sometimes:
 - Remove numbers (“93” -> “NUMBER_NN”)
 - Correct misspellings / alternate spellings (“color” -> “colour”)
- Problem specific:
 - Resolve synonyms / aliases (if you know them already)
 - Remove “stopwords”
 - Punctuation and grammatical function words (“if”, “the”, “by”), and
 - Very common words in your domain that don’t add much meaning

Demo: is tokenization hard?

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
 - woodchuck
 - woodchucks
 - Woodchuck
 - Woodchucks



Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
<code>[wW]oodchuck</code>	Woodchuck, woodchuck
<code>[1234567890]</code>	Any digit

- Ranges `[A-Z]`

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

Regular Expressions: Negation in Disjunction

- Negations `[ ^Ss ]`
 - Carat means negation only when first in []

Pattern	Matches	
<code>[^A-Z]</code>	Not an upper case letter	O <u>y</u> fn pripetchik
<code>[^Ss]</code>	Neither 'S' nor 's'	<u>I</u> have no exquisite reason"
<code>[^e^]</code>	Neither e nor ^	Look h <u>e</u> re
<code>a^b</code>	The pattern a carat b	Look up <u>a^b</u> now

Regular Expressions: More Disjunction

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

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



Regular Expressions: ? * + .

Pattern	Matches	
colou?r	Optional previous char	<u>color</u> <u>colour</u>
oo*h!	0 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
o+h!	1 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
baa+		<u>baa</u> <u>baaa</u> <u>baaaa</u> <u>baaaaa</u>
beg.n		<u>begin</u> <u>begun</u> <u>begun</u> <u>beg3n</u>



Stephen C Kleene

Kleene *, Kleene +

- Regex good enough for...?
 - Token boundaries?
 - Names?

Demo: regexes in *grep* and look at some data

```
grep -Eoi '#[ ^]*livesmatter'
```

```
grep -Eoi '#[a-z0-9]*livesmatter'
```

grep option: you may like
"extended" (grep -E) or "perl-style" (grep -P)
grep -P is closest to Python *re* library

```
cat 2016-03-01.text.txt | grep -Poi '#[a-z0-9]*livesmatter' | sort | uniq -c | less
```

```
3 #ALLLIVESMATTER
1 #AllJonasLivesMatter
81 #AllLivesMatter
1 #Alllivesmatter
2 #AmericanLivesMatter
2 #ArmenianLivesMatter
20 #BLACKLIVESMATTER
1 #BLACKLivesMatter
4 #BlackLivesMatter
1 #BearLivesMatter
1 #BeerLivesMatter
1 #BeigeLivesMatter
948 #BlackLivesMatter
1 #BlackLlivesMatter
1 #BlackMuslimLivesMatter
1 #BlackTransLivesMatter
1 #BlacklLivesMatter
2 #BlacklivesMatter
26 #Blacklivesmatter
1 #Blackslivesmatter
2 #BlueLivesMatter
90 #BlueLivesMatter
1 #Bluelivesmatter
1 #BookoutLivesMatter
2 #BrownLivesMatter
1 #BugLivesMatter
1 #CatsLivesMatter
1 #Chickenlivesmatter
```

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-Z][tT]he[^a-zA-Z]`

Errors

- The process we just went through was based on **fixing two kinds of errors**
 - Matching strings that we should not have matched (**there**, **then**, **other**)
 - False positives (Type I)
 - Not matching things that we should have matched (The)
 - False negatives (Type II)

stopped here 9/21

Errors cont.

- 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).

Simple Tokenization in UNIX

- (Inspired by Ken Church's UNIX for Poets.)
- Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt
```

Change all non-alpha to newlines

```
| sort
```

Sort in alphabetical order

```
| uniq -c
```

Merge and count each type

1945	A	25	Aaron
72	AARON	6	Abate
19	ABBESS	1	Abates
5	ABBOT	5	Abbess
		6	Abbey
...	...	3	Abbot
			...

Issues in Tokenization

- Finland's capital → Finland Finlands 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 lowercase lower case ?
- San Francisco → one token or two?
- m.p.h., PhD. → ??

Tokenization: language issues

- French
 - *L'ensemble* → one token or two?
 - *L ? L' ? Le ?*
 - Want *l'ensemble* to match with *un ensemble*
- German noun compounds are not segmented
 - *Lebensversicherungsgesellschaftsangestellter*
 - 'life insurance company employee'
 - German information retrieval needs **compound splitter**

Summary

- Regular expressions play a surprisingly large role
 - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For many hard tasks, we use machine learning classifiers
 - But regular expressions are used as features in the classifiers
 - Can be very useful in capturing generalizations