Words and regular expressions

COMPSCI 485, Applications of Natural Language Processing

Katrin Erk
College of Information and Computer Sciences
University of Massachusetts Amherst

Text normalization

- Our focus is on text data
- To do NLP on text data, you often need to **normalize** it first:
  - Segment it into paragraphs, sentences, words
  - Remove junk, for example HTML formatting
  - Maybe remove sensitive data if you're dealing with medical data
- One important tool for normalization: **regular expressions**
- What can you do with normalized text? **Word counts**, which are amazingly useful

Forms of normalization

Text normalization

Every NLP task needs text normalization

1. Segment sections of text, possibly paragraphs/sentences/etc.

   Why do we want to do that?

2. Segment/tokenize words in running text

3. Normalize word formats, for example: case sensitive or not?

   Why would we not want to retain case?

Preprocessing: Segmenting and selecting 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 [**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. Keep just the sentences/ paragraphs you want to analyze
- Get text out of weird formats like HTML, PDFs, or idiosyncratic formatting. Here: medications

This step is usually specific to your dataset

Preprocessing: Word 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', '.']

You can just split text on whitespace.
Or use a good off-the-shelf tokenizer:
NLTK, SpaCy, Stanza, Twokenizer, Huggingface...

- Words are (usually) the basic units of analysis in NLP.
- In English, words are separated through 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

Preprocessing: normalization

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

Demo: is tokenization hard?

- tokenization.ipynb, tokenization.html, available on Canvas

Regular expressions

Regular expressions: a quick overview

- A formal language for specifying text strings
- For example: how can we search for occurrences of any of the following:
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions: Disjunctions

- Letters inside square brackets []

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

- Ranges `[A-Z]`

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Regular Expressions: Negation in Disjunction

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

Pattern	Matches	
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither ‘S’ nor ‘s’	I have no exquisite reason”
[^e^]	Neither e nor ^	Look here
a^b	The pattern a carat b	Look up a^b now

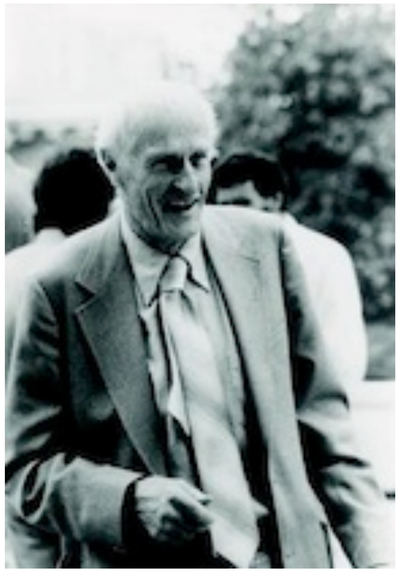
Regular Expressions: More Disjunction

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

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

Regular Expressions: ? * + .

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



Stephen C Kleene

Kleene *, Kleene +

How would you use regular expressions to search for…

- Proper names?
- Emojis?
- Phone numbers?

- Using regular expressions in Python: notebook Regular_expressions.ipynb, available on Canvas

Regular expressions at the command line

Demo: using regular expressions with grep

Data: NLTK collection of tweets, from Britain around the time of Brexit.
File available on Canvas under Files -> Lectures -> 1-normalization:
tweets.20150430-223406.json

We'll analyze hashtags. I noticed at a quick skim that Nigel Farage, a British politician who was pro-Brexit, seemed to appear a lot.

```
grep -Eoi '#[^ ]*farage[^ ]*' ~/nltk_data/corpora/twitter_samples/
tweets.20150430-223406.json
```

-E: extended (you may also like: -P, perl-like, closest to Python re)
-o: print only matches
-i: ignore case

Regular expressions at the command line

```
$ grep -Eoi '#[^ ]*Farage[^ ]*' ~/nltk_data/corpora/twitter_samples/tweets.20150430-223406.json

...

#Farage
#AskNigelFarage
#AskNigelFarage
#AskNigelFarage
#AskNigelFarage
#AskNigelFarage
#AskNigelFarage
#AskNigelFarage
#AskNigelFarage",
#AskNigelFarage",
#AskNigelFarage
#AskNigelFarage
#AskNigelFarage
#AskNigelFarage
#BetterThanFarage\u00e1Trois",
#AskFarage
#AskNigelFarage
#AskFarage
#AskNigelFarage
#AskNigelFarage
#AskNigelFarage
```

How frequent are the different hashtags?

Regular expressions at the command line

```
grep -Eoi '#[^ ]*Farage[^ ]*' ~/nltk_data/corpora/twitter_samples/tweets.20150430-223406.json | sort | uniq -c
```

...

```
 29 #Farage",
  3 #faragefail",
  1 #FARAGEFOREVER",
  1 #farageical",
  1 #FarageQuestions
  1 #HailFarage
  3 #ItsFarageT\u2026",
  4 #ItsFarageTime",
  2 #nigel_farage
  1 #Nigel_Farage
  5 #nigelfarage
 86 #NigelFarage
  7 #NigelFarage",
  3 #TeamFarage
```

| (pipe): output of the lhs script becomes input to the rhs script

sort: sort.

uniq -c: collapse consecutive repeating items, and count

But what are the most frequent tags?

Regular expressions at the command line

```
grep -Eoi '#[^ ]*Farage[^ ]*' ~/nltk_data/corpora/twitter_samples/tweets.20150430-223406.json | sort | uniq -c | sort | less

...

  10 #asknigelfarage",
  11 #Farage:
  17 #AskNigelFarage,
  29 #Farage",
  53 #AskFarage",
  86 #NigelFarage
 106 #AskNigelFarage!
 130 #Farage
 327 #AskFarage
 663 #AskNigelFarage",
1411 #AskNigelFarage
```

More preprocessing at the command line

See document on Canvas: Ken Church, UNIX for Poets

- tr: translate characters

  tr -sc ‘A-Za-z’ ‘\n’ : translate all characters other than A-Za-z (-c) to newline, squeezing multiple occurrences into one (-s)

  that is: tokenize

Fixing and evaluating regular expressions

Creation of regular expressions is often a multi-step approach

- Find all occurrences of “the”:
  - the misses capitalized determiners
  - [Tt]he also returns words that include ‘the’, such as “theology”, “other”
  - [^A-Za-z][Tt]he[^A-Za-z]

  This example addresses two types of errors:
  - Missing strings we should have matched, such as “The”: **false negatives**
  - Matching strings we shouldn’t have matched, such as “other”: **false positives**

Errors continued

- In NLP we are always dealing with these two kinds of errors
- Reducing the error rate in models we build often involve two efforts that go in opposite directions:
  - Increasing accuracy or precision: reducing false positives
  - Increasing coverage or recall: reducing false negatives

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. → ??

[Slide: SLP3]

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**

[Slide: SLP3]

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

[Slide: SLP3]

Different normalization techniques

Normalizations

- Word tokenization: regular expressions, or specialized tool
  - NLTK word_tokenize(), spacy…
- Lowercasing
- Lemmatization
  - Reduce inflected forms to base form: houses -> house, said -> say, are -> be…
    Lemma : dictionary headword
- Stemming: quick-and-dirty lemmatization, cut off what may be affixes

Normalizations

- Morphology:
  - Morphemes: smallest meaningful units that make up words
  - Stems and affixes: house-s, dis-informat-ion
- Does anyone speak languages with more complex morphology than English? What do you see there?
- Why is morphology relevant to NLP?

Using word counts

What can you find out with just word counts?

What can you find out with just word counts?

- Cultural trends: What were people searching for? https://trends.withgoogle.com/year-in-search/2025/us/?hl=en

- Analyzing politicians' speeches:

  - Word clouds: rough topic analyses, for example https://wordwatchers.wordpress.com/2020/09/09/the-race-is-on-a-content-analysis-of-the-2020-presidential-nomination-acceptance-speeches/

  - Word counts over time: the trajectory of a speech, for example https://wordwatchers.wordpress.com/2020/09/19/optimism-and-certainty-in-trump-and-bidens-acceptance-speeches/

- Diving deeper: the language of real estate listings: https://languagelog.ldc.upenn.edu/nll/?p=4681 and https://languagelog.ldc.upenn.edu/nll/?p=4686

How many words?

Why is the number of tokens important?
Why the number of types?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Church and Gale (1990): $|V| > O(N^{½})$

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

Word Frequencies: Alice in Wonderland

Word	Frequency (f)
the	1629
and	844
to	721
a	627
she	537
it	526
of	508
said	462
i	400
alice	385

Alice's Adventures in Wonderland, by Lewis Carroll

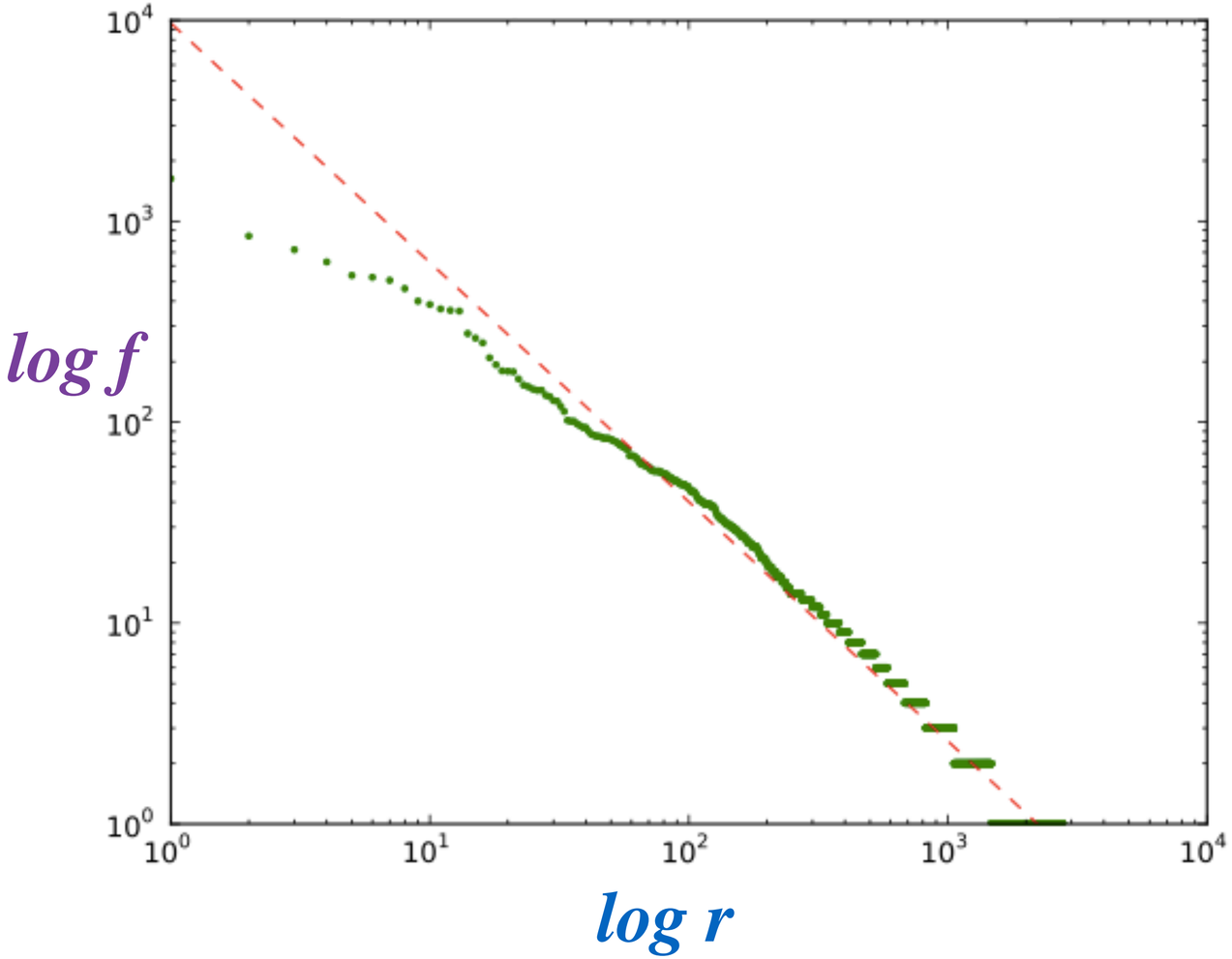
Zipf's Law

- When word types are ranked by frequency, then frequency (f) * rank (r) is roughly equal to some constant (k)

$$f \times r = k$$

Rank (r)	Word	Frequency (f)	$r \cdot f$
1	the	1629	1629
2	and	844	1688
3	to	721	2163
4	a	627	2508
5	she	537	2685
6	it	526	3156
7	of	508	3556
8	said	462	3696
9	i	400	3600
10	alice	385	3850
20	all	179	3580
30	little	128	3840
40	about	94	3760
50	again	82	4100
60	queen	68	4080
70	don't	60	4200
80	quite	55	4400
90	just	51	4590
100	voice	47	4700
200	hand	20	4000
300	turning	12	3600
400	hall	9	3600
500	kind	7	3500

Plot: log frequencies



How is Zipf's law important for NLP?