Basic Text Processing

Regular Expressions

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
`[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

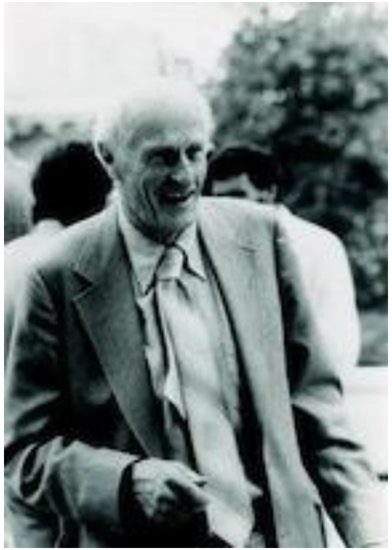
Woodchuck is another name for groundhog!

The pipe | for disjunction

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

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 +

Regular Expressions: Anchors ^ $

Pattern	Matches
`^[A-Z]`	Palo Alto
`^[^A-Za-z]`	1 "Hello"
`\.$`	The end.
`.$`	The end? The end!

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:

1. Matching strings that we should not have matched (there, then, other)

False positives (Type I errors)

2. Not matching things that we should have matched (The)

False negatives (Type II errors)

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).

Substitutions

Substitution in Python and UNIX commands:

```
s/regexp1/pattern/
```

e.g.:

```
s/colour/color/
```

Capture Groups

- Say we want to put angles around all numbers:
  the 35 boxes → *the <35> boxes*
- Use parens () 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>/
```

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 ate

But suppose we don't want to capture?

Parentheses have a double function: grouping terms, and capturing

Non-capturing groups: add a ?: after paren:

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

matches

- `some cats like some cats`

But not

- `some cats like some a few`

Simple Application: ELIZA

Early NLP system that imitated a Rogerian psychotherapist (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

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 text

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

Lookahead assertions

`(?= pattern)` is true if pattern matches, but is **zero-width; doesn't advance character pointer**

`(?! pattern)` true if a pattern does not match

How to match, at the beginning of a line, any single word that doesn’t start with “Volcano”:

`/^(?!Volcano)[A-Za-z]+/`

Basic Text Processing

Words and Corpora

How many words?

"I do uh main- mainly business data processing"

- Fragments, filled pauses

"Seuss's cat in the hat is different from other cats!"

- **Lemma**: same stem, part of speech, rough word sense
  - cat and cats = same lemma
- **Wordform**: the full inflected surface form
  - cat and cats = different wordforms

How many words?

they lay back on the San Francisco grass and looked at the stars and their

Type: an element of the vocabulary.

Token: an instance of that type in running text.

How many?

- 15 tokens (or 14)
- 13 types (or 12) (or 11?)

How many words?

N = number of tokens

V = vocabulary = set of types, **|*V*|** is size of vocabulary

Heaps Law = Herdan's Law = $|V| = kN^{\beta}$ where often .67 < β < .75

i.e., vocabulary size grows with > square root of the number of word tokens

	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

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 dimension like

- **Language**: 7097 languages in the world
- **Variety**, like African American Language varieties.
  - AAL Twitter posts might include forms like "*iont*" *(I don't)*
- **Code switching**, e.g., Spanish/English, Hindi/English:

S/E: 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:)]

H/E: dost tha or ra- hega ... dont wory ... but dherya rakhe

["he was and will remain a friend ... don't worry ... but have faith"]

- **Genre:** newswire, fiction, non-fiction, scientific articles, Wikipedia
- **Author Demographics**: writer's age, gender, race, socioeconomic status, etc.

Corpus **datasheets**

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

Motivation: Why was the corpus collected, by whom, and 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, speaker demographics

Basic Text Processing

Word tokenization

Text Normalization

Every NLP task requires text normalization:

1. Tokenzing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

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
     | sort
     | uniq -c
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

```
1945 A
  72 AARON
  19 ABBESS
   5 ABBOT
 ... ...
```

```
25 Aaron
 6 Abate
 1 Abates
 5 Abbess
 6 Abbey
 3 Abbot
....  …
```

The first step: tokenizing

```
tr -sc ’A-Za-z’ ’\n’ < shakes.txt | head
```

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...
```

The second step: sorting

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

```
A
A
A
A
A
A
A
A
A
...
```

More counting

Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

```
23243 the
22225 i
18618 and
16339 to
15687 of
12780 a
12163 you
10839 my
10005 in
8954  d
```

What happened here?

Issues in Tokenization

Can't just blindly remove punctuation:

- m.p.h., Ph.D., AT&T, cap’n.
- prices ($45.55) and dates (01/02/06); URLs; (http://www.stanford.edu), hashtags (#nlproc), email addresses (someone@cs.colorado.edu).

Clitics: a part of a word that can't stand on its own

- we're → we are, French j'ai, l'honneur

Can "Multiword Expressions (MWE) be words?

- New York, rock ’n’ roll

Tokenization in NLTK

Bird et al. (2009)

```
>>> 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 without spaces

Chinese, Japanese, Thai, don't use spaces to separate words

Word tokenization in Chinese

Chinese words are composed of characters called **hanzi**

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 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

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

Word tokenization

So in Chinese it's common not to do word segmentation at all

But in Thai and Japanese, it's required

The standard algorithms are neural sequence models trained by supervised machine learning.

Basic Text Processing

Byte Pair Encoding tokenization

A third option for word segmentation

Use the data to tell us how to tokenize.

Subword tokenization (because tokens are often parts of words)

Can include common morphemes like *-est* or *-er*.

- (A morpheme is the smallest meaning-bearing unit of a language; *unlikeliest* has morphemes *un-*, *likely*, and *-est*.)

Subword tokenization

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

Byte Pair Encoding (BPE)

Let vocabulary be the set of all individual characters

= {A, B, C, D,…,a, b, c, d….}

Repeat:

- choose the two symbols that are most frequently adjacent in training corpus (say ‘A’, ‘B’),
- adds a new merged symbol ‘AB’ to the vocabulary
- replace every adjacent ’A’ ’B’ in corpus with ‘AB’.

Until *k* merges have been done.

BPE token learner algorithm

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)

Most subword algorithms are run inside white-space separated tokens.

So first add a special end-of-word symbol '__' before whitespace in training corpus

Next, separate into letters.

BPE token learner

Original (very fascinating🙄) corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Add end-of-word tokens and segment:

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

BPE token learner

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

Merge e r to er

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

BPE

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

Merge er _ to er_

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

BPE

corpus

5 l o w _
2 l o w e s t _
6 n e w er_
3 w i d er_
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

Merge n e to ne

corpus

5 l o w _
2 l o w e s t _
6 ne w er_
3 w i d er_
2 ne w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_, ne

BPE

The next merges are:

Merge	**Current Vocabulary**
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

BPE token learner algorithm

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every e r to er, then merge er _ to er_, etc.

Result:

- Test set "n e w e r _" would be tokenized as a full word
- Test set "l o w e r _" would be two tokens: "low er_"

Basic Text Processing

Word Normalization and other issues

Word Normalization

Putting words/tokens in a standard format

- U.S.A. or USA
- uhhuh or uh-huh
- Fed or fed
- am, is be, are

Case folding

Applications like IR: reduce all letters to lower case

- Since users tend to use lower case
- Possible exception: upper case in mid-sentence?
  - e.g., ***General Motors***
  - ***Fed*** vs. ***fed***
  - ***SAIL*** vs. ***sail***

For sentiment analysis, MT, Information extraction

- Case is helpful (***US*** versus ***us*** is important)

Lemmatization

Represent all words as their shared root, = dictionary headword form:

- *am, are, is* $\rightarrow$ *be*
- *car, cars, car's, cars'* $\rightarrow$ *car*
- Spanish quiero (‘I want’), quieres (‘you want’) $\rightarrow$ querer ‘want'

He is reading detective stories $\rightarrow$ *He be read detective story*

Lemmatization is done by Morphological Parsing

Morphemes:

- The small meaningful units that make up words
- **Stems**: The core meaning-bearing units
- **Affixes**: Parts that adhere to stems, often with grammatical functions

Morphological Parsers:

- Parse *cats* into two morphemes *cat* and *s*
- Parse Spanish *amaren* (‘if in the future they would love’) into morpheme *amar* ‘to love’, and the morphological features *3PL* and *future subjunctive*.

Stemming

Reduce terms to stems, chopping off affixes crudely

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 .

Porter Stemmer

Based on a series of rewrite rules run in series

- A cascade, in which output of each pass fed to next pass

Some sample rules:

$$
\begin{array}{rcll}
\text{ATIONAL} & \rightarrow & \text{ATE} & \text{(e.g., relational} \rightarrow \text{relate)} \\
\text{ING} & \rightarrow & \epsilon & \text{if stem contains vowel (e.g., motoring} \rightarrow \text{motor)} \\
\text{SSES} & \rightarrow & \text{SS} & \text{(e.g., grasses} \rightarrow \text{grass)}
\end{array}
$$

Dealing with complex morphology is necessary for many languages

- E.g., the Turkish word:
- Uygarlastiramadiklarimizdanmissinizcasina
- `(behaving) as if you are among those whom we could not civilize'
- Uygar `civilized' + las `become'

+ tir `cause' + ama `not able'

+ dik `past' + lar 'plural'

+ imiz 'p1pl' + dan 'abl'

+ mis 'past' + siniz '2pl' + casina 'as if'

Sentence Segmentation

!, ? are relatively unambiguous but **period** "." is quite ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Common Algorithm: decide (using rules or ML) whether a period is part of the word or is a sentence-boundary marker.

- An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.