Natural Language Processing
CSCI 4152/6509 — Lecture 5
N-grams and Morphology

Instructors: Vlado Keselj
Time and date: 14:35 – 15:55, 9-Oct-2025
Location: Studley LSC-Psychology P5260

Previous Lecture

- Introduction to text processing with Perl
- Regular expressions in Perl
  - Use of special variables
  - Backreferences, shortest match

Back Reference Examples (repeated)

Consider examples:

`/(a+(b+))(c+(d+))\4/` and `/(a+(b+))(c+(d+))\3/`

Shortest Match

- default matching: left-most, longest match
- e.g., consider /\d+/
- Shortest match is sometimes preferred
  - e.g., consider: /<div>.*<\/div>/ or /<[^>]*>/ vs. /<.*>/
  - and: /<div>.*?<\/div>/ and /<.*?>/
- Shortest match iterators:
  *? +? ?? {n}? {n,m}?

Regular Expression Substitutions

- syntax: s/*re*/*sub*/*options*
- Some substitution options
  - c – do not reset search position after /g fail
  - e – evaluate replacement as expression
  - g – replace globally (all occurrences)
  - i – case-insensitive pattern matching
  - m – treat string as multiple lines
  - o – compile pattern only once
  - s – treat string as a single line
  - x – use extended regular expressions

Text Processing Example

- Let us consider some simple text processing examples
- Use of Regular Expressions
- Convenient string manipulation
- Associative arrays
- Example: Counting Letters

Experiments on "Tom Sawyer"

- File: `TomSawyer.txt`:
  ```
  The Adventures of Tom Sawyer

                                  by

                  Mark Twain (Samuel Langhorne Clemens)

  Preface
  MOST of the adventures recorded in this book really occurred;
  one or two were experiences of my own, the rest those of boys
  who were schoolmates of mine. Huck Finn is drawn from life;
  Tom Sawyer also, but not from an individual -- he is a
  combination of the characteristics of three boys whom I knew,
  and therefore belongs to the composite order of architecture.
  ```

Letter Count Total

```perl
#!/usr/bin/perl
# Letter count total

my $lc = 0;

while (<>) {
  while (/[a-zA-Z]/) { ++$lc; $_ = $'; }
}
print "$lc\n";

# ./letter-count-total.pl TomSawyer.txt
# 296605
```

Letter Frequencies

```perl
#!/usr/bin/perl
# Letter frequencies

while (<>) {
  while (/[a-zA-Z]/) {
    my $l = $&; $_ = $';
    $f{$l} += 1;
  }
}

for (keys %f) { print "$_ $f{$_}\n" }
```

Letter Frequencies Output

```
./letter-frequency.pl TomSawyer.txt
S 606
a 22969
T 1899
N 324
K 24
d 14670
Y 214
E 158
j 381
y 6531
u 8901
...
```

Letter Frequencies Modification

```
#!/usr/bin/perl
# Letter frequencies (2)

while (<>) {
  while (/[a-zA-Z]/) {
    my $l = $&; $_ = $';
    $f{lc $l} += 1;
  }
}

for (sort keys %f) { print "$_ $f{$_}\n" }
```

New Output

```
./letter-frequency2.pl TomSawyer.txt
a 23528
b 4969
c 6517
d 14879
e 35697
f 6027
g 6615
h 19608
i 18849
j 639
k 3030
...
```

Letter Frequencies Modification (3)

```perl
#!/usr/bin/perl
# Letter frequencies (3)

while (<>) {
  while (/[a-zA-Z]/) {
    my $l = $&; $_ = $';
    $f{lc $l} += 1; $tot ++;
  }
}

for (sort { $f{$b} <=> $f{$a} } keys %f) {
  print sprintf("%6d %.4lf %s\n",
              $f{$_}, $f{$_}/$tot, $_); }
```

Output 3

```
35697 0.1204 e
28897 0.0974 t
23528 0.0793 a
23264 0.0784 o
20200 0.0681 n
19608 0.0661 h
18849 0.0635 i
17760 0.0599 s
15297 0.0516 r
14879 0.0502 d
12163 0.0410 l
 8959 0.0302 u
```

...

Elements of Morphology

- Reading: Section 3.1 in the textbook, "Survey of (Mostly) English Morphology"
- *morphemes* — smallest meaning-bearing units
- *stems* and *affixes*; stems provide the "main" meaning, while affixes act as modifiers
- affixes: prefix, suffix, infix, or circumfix
- cliticization — clitics appear as parts of a word, but syntactically they act as words (e.g., 'm, 're, 's)
- tokenization, stemming (Porter stemmer), lemmatization

Tokenization

- Text processing in which plain text is broken into words or *tokens*
- Tokens include non-word units, such as numbers and punctuation
- Tokenization may normalize words by making them lower-case or similar
- Usually simple, but prone to ambiguities, as most of the other NLP tasks

Stemming

- Mapping words to their *stems*
- Example: *foxes* → *fox*
- Use in Information Retrieval and Text Mining to normalize text and reduce high dimensionality
- Typically works by removing some suffixes according to a set of rules
- Best known stemmer: Porter stemmer

Lemmatization

- *Surface word form:* a word as it appears in text (e.g., working, are, indices)
- *Lemma:* a canonical or normalized form of a word, as it appears in a dictionary (e.g., work, be, index)
- *Lemmatization:* word processing method which maps surface word forms into their lemmas

Morphological Processes

- Morphological Process = changing word form, as a part of regular language transformation
- Types of morphological processes
  1. inflection
  2. derivation
  3. compounding

1. Inflection

Examples: dog → dogs
          work → works
                 working
                 worked

- small change (word remains in the same category)
- relatively regular
- using suffixes and prefixes

2. Derivation

- Typically transforms word in one lexical class to a related word in another class
- Example: wide (adjective) $\rightarrow$ widely (adverb)
  but, similarly: old $\rightarrow$ oldly (*) is incorrect.
  more ex.: accept (verb) $\rightarrow$ acceptable (adjective)
    acceptable (adjective) $\rightarrow$ acceptably (adverb)
      teach (verb) $\rightarrow$ teacher (noun)

- Derivation is a more radical change (change word class)
- less systematic
- using suffixes

Some Derivation Examples

Derivation type	Suffix	Example		
noun-to-verb	-fy	glory	$\rightarrow$	glorify
noun-to-adjective	-al	tide	$\rightarrow$	tidal
verb-to-noun (agent)	-er	teach	$\rightarrow$	teacher
verb-to-noun (abstract)	-ance	delivery	$\rightarrow$	deliverance
verb-to-adjective	-able	accept	$\rightarrow$	acceptable
adjective-to-noun	-ness	slow	$\rightarrow$	slowness
adjective-to-verb	-ise	modern	$\rightarrow$	modernise (Brit.)
adjective-to-verb	-ize	modern	$\rightarrow$	modernize (U.S.)
adjective-to-adjective	-ish	red	$\rightarrow$	reddish
adjective-to-adverb	-ly	wide	$\rightarrow$	widely

3. Compounding

Examples: news + group = newsgroup
down + market = downmarket
over + take = overtake
play + ground = playground
lady + bug = ladybug

Characters, Words, N-grams

- We saw some experiments with counting characters

- Let us look at Counting Words

- N-grams and Counting N-grams

Counting Words and Zipf's Law

Word	Freq (f)	Rank (r)
the	3331	1
and	2971	2
a	1776	3
to	1725	4
of	1440	5
was	1161	6
it	1030	7
I	1016	8
that	959	9
he	924	10
in	906	11
's	834	12
you	780	13
his	772	14
Tom	763	15
't	654	16
⋮	⋮	

- We looked at code for counting letters, words, and sentences

- We can look again at counting words; e.g., in "Tom Sawyer":

- We can observe: Zipf's law (1929): $r \times f \approx$ const.

Counting Words

```perl
#!/usr/bin/perl
# word-frequency.pl

while (<>) {
  while (/'?[a-zA-Z]+/g) { $f{$&}++; $tot++; }
}

print "rank    f    f(norm) word        r*f\n".
      ('-'x35)."\n";
for (sort { $f{$b} <=> $f{$a} } keys %f) {
  print sprintf("%3d. %4d %lf %-8s %5d\n",
          ++$rank, $f{$_}, $f{$_}/$tot, $_,
          $rank*$f{$_});
}
```

Program Output (Zipf's Law)

rank	f	word	r*f		rank	f	word	r*f
					18.	516	for	9288
					19.	511	had	9709
1.	3331	the	3331		20.	460	they	9200
2.	2971	and	5942		21.	425	him	8925
3.	1776	a	5328		22.	411	but	9042
4.	1725	to	6900		23.	371	on	8533
5.	1440	of	7200		24.	370	The	8880
6.	1161	was	6966		25.	369	as	9225
7.	1130	it	7910		26.	352	said	9152
8.	1016	I	8128		27.	325	He	8775
9.	959	that	8631		28.	322	at	9016
10.	924	he	9240		29.	313	she	9077
11.	906	in	9966		30.	303	up	9090
12.	834	's	10008		31.	297	so	9207
13.	780	you	10140		32.	294	be	9408
14.	772	his	10808		33.	286	all	9438
15.	763	Tom	11445		34.	278	her	9452
16.	654	't	10464		35.	276	out	9660
17.	642	with	10914		36.	275	not	9900

Graphical Representation of Zipf's Law
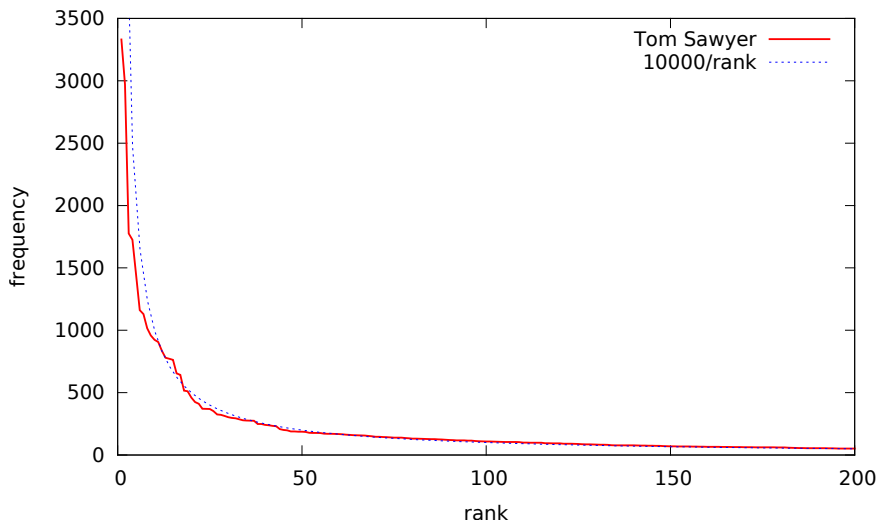
Zipf's Law (log-log scale)
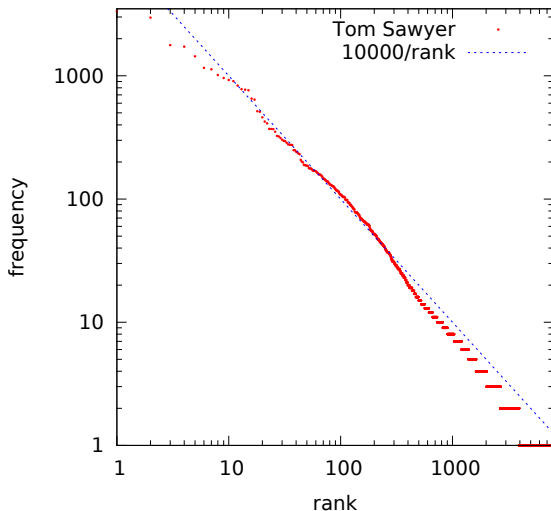
Character N-grams

- Consider the text:
  The Adventures of Tom Sawyer

- Character n-grams = substring of length n

- $n = 1 \Rightarrow$ *unigrams*: T, h, e, _ (space), A, d, v, ...

- $n = 2 \Rightarrow$ *bigrams*: Th, he, e_, _A, Ad, dv, ve, ...

- $n = 3 \Rightarrow$ *trigrams*: The, he_, e_A, _Ad, Adv, dve, ...

- and so on; Similarly, we can have word n-grams, such as
  $(n = 3)$: The Adventures of, Adventures of Tom, of Tom
  Sawyer ...

- or normalized into lowercase

Experiments on "Tom Sawyer"

- Consider the Tom Sawyer novel:

                    The Adventures of Tom Sawyer

                                 by

              Mark Twain (Samuel Langhorne Clemens)

Preface
MOST of the adventures recorded in this book really occurred;
one or two were experiences of my own, the rest those of boys
who were schoolmates of mine. Huck Finn is drawn from life;
Tom Sawyer also, but not from an individual -- he is a

Word and Character N-grams ($n = 3$)

```
Word tri-grams              Character tri-grams
-------------------         -------------------
the adventures of           T h e      _ o f
adventures of tom           h e _      o f _
of tom sawyer               e _ A      f _ T
tom sawyer by               _ A d      _ T o
sawyer by mark              A d v      T o m
by mark twain               d v e      o m _
mark twain samuel           v e n      m _ S
twain samuel langhorne      e n t      _ S a
samuel langhorne clemens    n t u      S a w
langhorne clemens preface   t u r      a w y
clemens preface most        u r e      w y e
preface most of             r e s      y e r
most of the                 e s _      e r _
...                         s _ o      ...
```

A Program to Extract Word N-grams

```perl
#!/usr/bin/perl
# word-ngrams.pl


$n = 3;

while (<>) {
  while (/'?[a-zA-Z]+/g) {
     push @ng, lc($&); shift @ng if scalar(@ng) > $n;
     print "@ng\n" if scalar(@ng) == $n;
  }
}

# Output of: ./word-ngrams.pl TomSawyer.txt
# the adventures of
# adventures of tom
# ...
```

Some Perl List Operators

- push @a, 1, 2, 3; — adding elements at the end
- pop @a; — removing elements from the end
- shift @a; — removing elements from the start
- unshift @a, 1, 2, 3; — adding elements at the start
- scalar(@a) — number of elements in the array
- $#a — last index of an array, by default $#a = scalar(@a) - 1
- To be more precise, this is always true: scalar(@a) == $#a - $[+ 1
- $[(by default 0) is the index of first element of an array
- Arrays are dynamic: examples: $a[5] = 1, $#a = 5, $#a = -1

Extracting Character N-grams (attempt 1)

```perl
#!/usr/bin/perl
# char-ngrams1.pl - first attempt

$n = 3;

while (<>) {
  while (/\S/g) {
      push @ng, $&; shift @ng if scalar(@ng) > $n;
      print "@ng\n" if scalar(@ng) == $n;
  }
}

# Output of: ./char-ngrams1.pl TomSawyer.txt
# T h e    A d v    e n t
# h e A    d v e    n t u
# e A d    v e n    ...
```

Extracting Character N-grams (attempt 2)

```perl
#!/usr/bin/perl
# char-ngrams2.pl - second attempt

$n = 3;

while (<>) {
  while (/\S|\s+/g) {
    my $token = $&;
    if ($token =~ /^\s+$/) { $token = '_' }
    push @ng, $token;
    shift @ng if scalar(@ng) > $n;
    print "@ng\n" if scalar(@ng) == $n;
  }
}
```

```
# Output of: ./char-ngrams2.pl TomSawyer.txt
# _ T h    f _ T    _ _ _
# T h e    _ T o    _ _ M
# h e _    T o m    _ M a
# e _ A    o m _    ...
# _ A d    m _ S       This may be what we want, but
# A d v    _ S a       probably not.
# d v e    S a w
# v e n    a w y
# e n t    w y e
# n t u    y e r
# t u r    e r _
# u r e    r _ _
# r e s    _ _ _
# e s _    _ _ b
# s _ o    _ b y
# _ o f    b y _
# o f _    y _ _
```