Natural Language Processing
CSCI 4152/6509 — Lecture 6
Counting N-grams

Instructors: Vlado Keselj
Time and date: 16:05 – 17:25, 23-Sep-2024
Location: Carleton Tupper Building Theatre C

Previous Lecture

- Regular expressions in Perl
  - Use of special variables
  - Backreferences, shortest match
- Text processing examples
  - tokenization
  - counting letters
- Elements of Morphology

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

3. Compounding

Examples:
$$\begin{aligned} \text{news} + \text{group} &= \text{newsgroup} \\ \text{down} + \text{market} &= \text{downmarket} \\ \text{over} + \text{take} &= \text{overtake} \\ \text{play} + \text{ground} &= \text{playground} \\ \text{lady} + \text{bug} &= \text{ladybug} \end{aligned}$$

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

- 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): $r \times f \approx \text{const.}$

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
⋮	⋮	

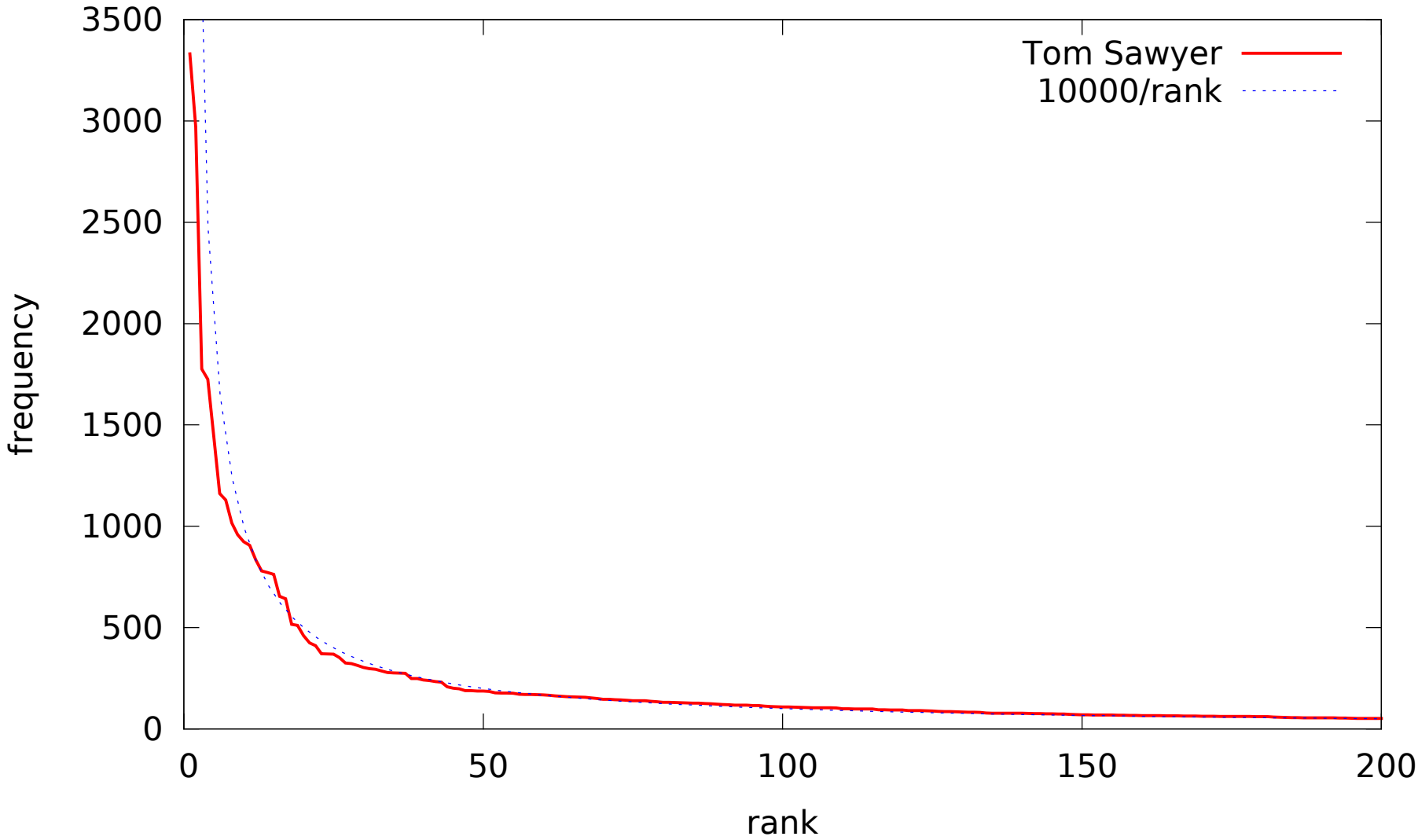
Counting Words

```
#!/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 %1f %-8s %5d\n",
              ++$rank, $f{$_}, $f{$_}/$tot, $_,
              $rank*$f{$_});
}
```

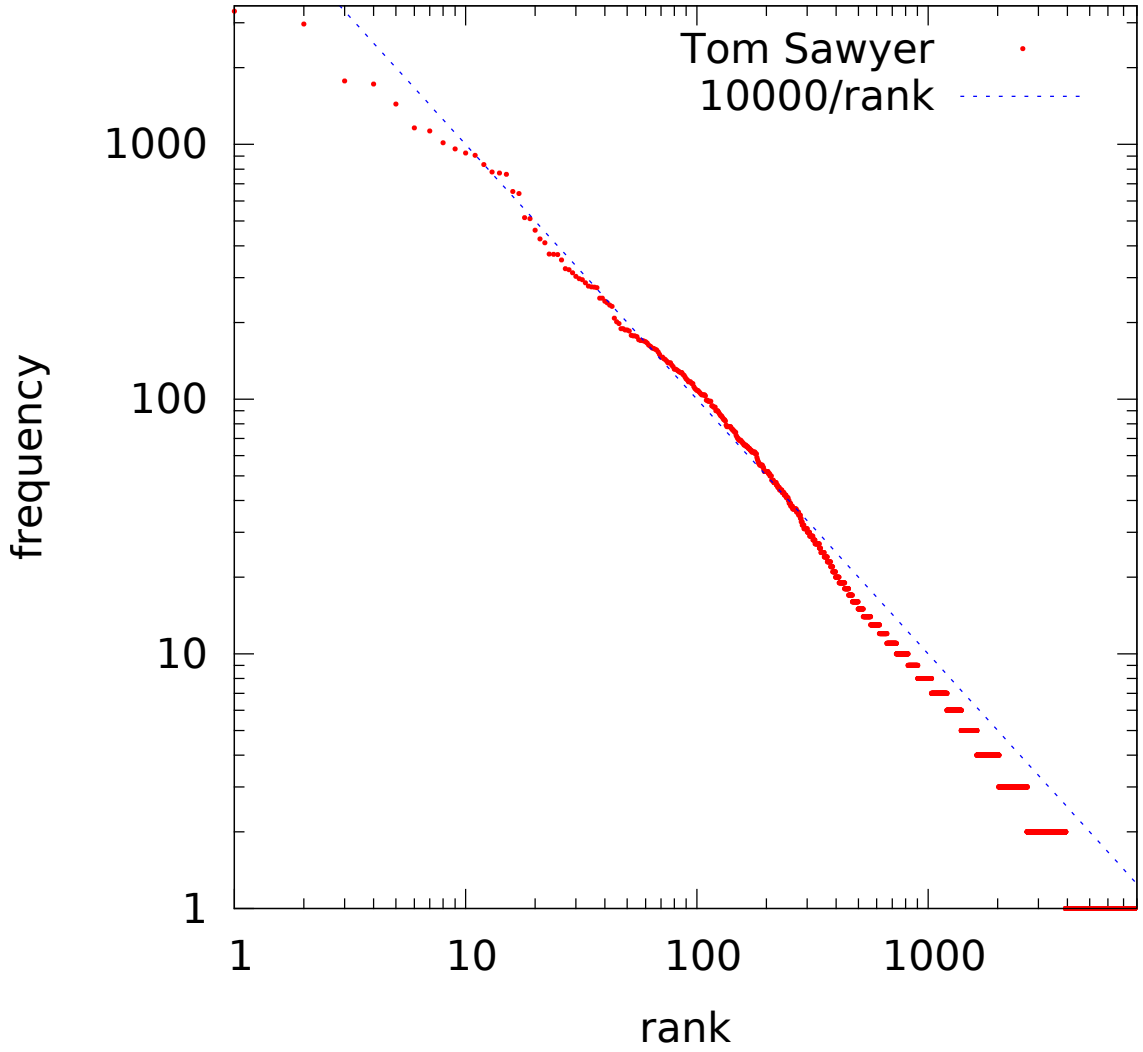
Program Output (Zipf's Law)

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

```
#!/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)

```
#!/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)

```
#!/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 _ _
```

Extracting Character N-grams (attempt 3)

```
#!/usr/bin/perl
# char-ngrams3.pl - third attempt

$n = 3;
$_ = join('',<>); # notice how <> behaves differently
            # in an array context, vs. scalar context

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-ngrams3.pl TomSawyer.txt
# _ T h   f _ T   a r k
# T h e   _ T o   r k _
# h e _   T o m   k _ T
# e _ A   o m _   _ T w
# _ A d   m _ S   T w a
# A d v   _ S a   w a i
# d v e   S a w   a i n
# v e n   a w y   i n _
# e n t   w y e   n _ (
# n t u   y e r   _ ( S
# t u r   e r _   ( S a
# u r e   r _ b   S a m
# r e s   _ b y   a m u
# e s _   b y _   m u e
# s _ o   y _ M   u e l
# _ o f   _ M a   e l _
# o f _   M a r   ...
```

Extracting Character N-grams by Line

- We need to handle whitespace spanning multiple line
- Generally, any token may span multiple lines
- Could be done but leads to a bit more complex code

Word N-gram Frequencies

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

$n = 3;

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

sub collect {
  my $ng = "@_";
  $f{$ng}++; ++$tot;
}
```

```
print "Total $n-grams: $tot\n";

for (sort { $f{$b} <=> $f{$a} } keys %f) {
  print sprintf("%5d %lf %s\n",
                $f{$_}, $f{$_}/$tot, $_);
}

# Output of: ./word-ngrams-f.pl TomSawyer.txt
# Total 3-grams: 73522
#    70 0.000952 i don 't
#    44 0.000598 there was a
#    35 0.000476 don 't you
#    32 0.000435 by and by
#    25 0.000340 there was no
#    25 0.000340 don 't know
#    24 0.000326 it ain 't
```

```
#     22 0.000299 out of the
#     22 0.000299 i won 't
#     21 0.000286 it 's a
#     21 0.000286 i didn 't
#     21 0.000286 i can 't
#     20 0.000272 it was a
#     19 0.000258 and i 'll
#     18 0.000245 injun joe 's
#     18 0.000245 you don 't
#     17 0.000231 i ain 't
#     17 0.000231 he did not
#     16 0.000218 he had been
#     15 0.000204 out of his
#     15 0.000204 all the time
#     15 0.000204 it 's all
#     15 0.000204 to be a
#     15 0.000204 what 's the
#     14 0.000190 that 's so
#...
```

Character N-gram Frequencies

```
#!/usr/bin/perl
# char-ngrams-f.pl

$n = 3;
$_ = join('',<>); # notice how <> behaves differently
            # in an array context, vs. scalar context

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

```
sub collect {
  my $ng = "@_";
  $f{$ng}++; ++$tot;
}

print "Total $n-grams: $tot\n";

for (sort { $f{$b} <=> $f{$a} } keys %f) {
  print sprintf("%5d %lf %s\n",
                $f{$_}, $f{$_}/$tot, $_);
}

# Output of: ./char-ngrams-f.pl TomSawyer.txt
# Total 3-grams: 389942
#  6556 0.016813 _ t h
#  5110 0.013105 t h e
#  4942 0.012674 h e _
#  3619 0.009281 n d _
```

```
#  3495 0.008963 _ a n
#  3309 0.008486 a n d
#  2747 0.007045 e d _
#  2209 0.005665 _ t o
#  2169 0.005562 i n g
#  1823 0.004675 t o _
#  1817 0.004660 n g _
#  1738 0.004457 _ a _
#  1682 0.004313 _ w a
#  1673 0.004290 _ h e
#  1672 0.004288 e r _
#  1592 0.004083 d _ t
#  1566 0.004016 _ o f
#  1541 0.003952 a s _
#  1526 0.003913 _ ‘ ‘
#  1511 0.003875 ’ ’ _
#  1485 0.003808 a t _
# ...
```

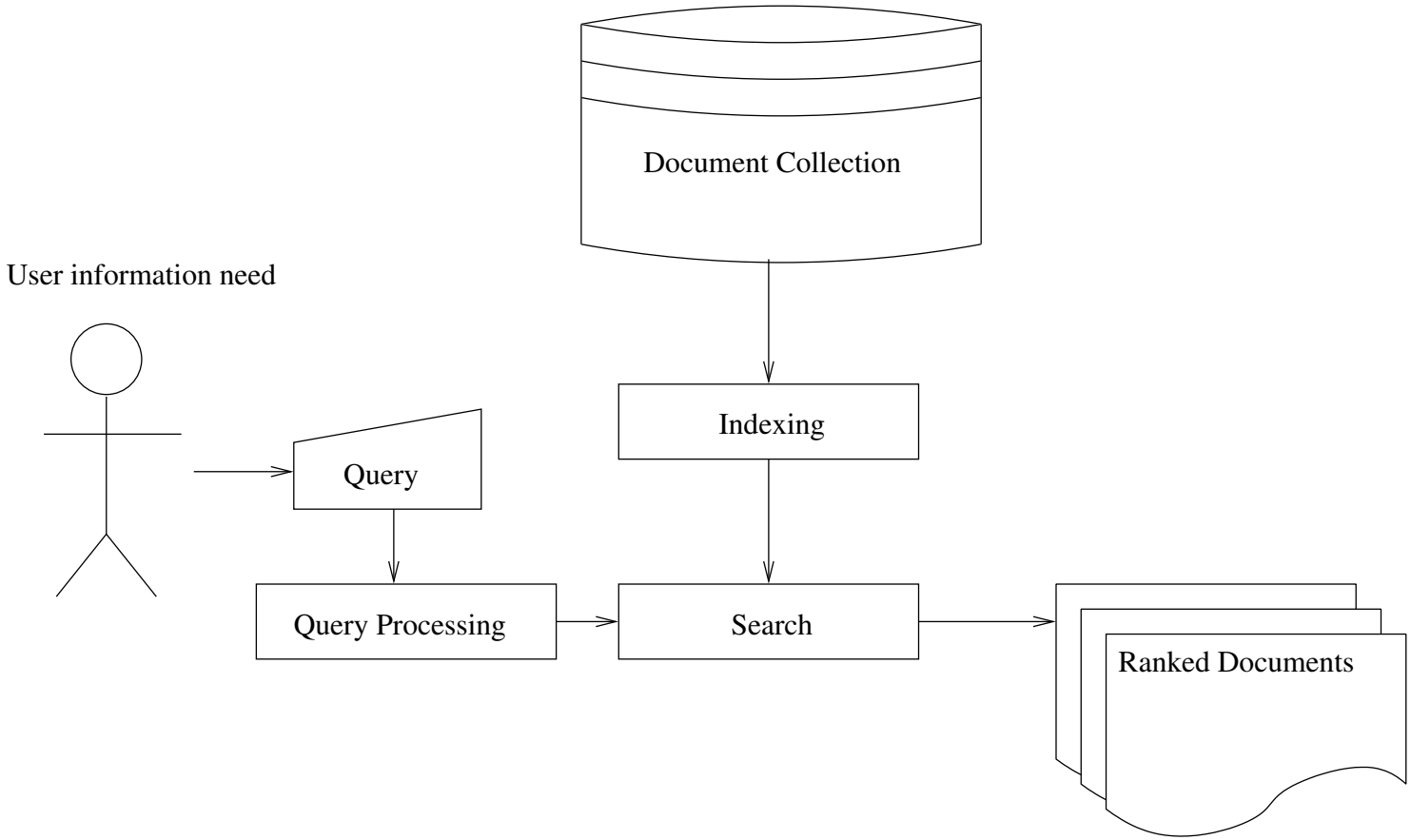
Using Ngrams Module

- Using Perl module: `Text::Ngrams`
- Flexible use for several types of n-grams, e.g.: character, word, byte
- Use `ngrams.pl` or use module from a program
- Details covered in the lab

Elements of Information Retrieval

- Reading: [JM] Sec 23.1, ([MS] Ch.15)
- Information Retrieval: area of Computer Science concerned with finding a set of relevant documents from a document collection given a user query.
- Basic task definition (ad hoc retrieval):
  - User: information need expressed as a query
  - Document collection
  - Result: set of relevant documents

Typical IR System Architecture



Steps in Document and Query Processing

- a "bag-of-words" model
- stop-word removal
- rare word removal (optional)
- stemming
- optional query expansion
- document indexing
- document and query representation;
  e.g. sets (Boolean model), vectors

Vector Space Model in IR

- We choose a global set of terms $\{t_1, t_2, \ldots, t_m\}$
- Documents and queries are represented as vectors of weights:

$$\vec{d} = (w_{1,d}, w_{2,d}, \ldots, w_{m,d}) \quad \vec{q} = (w_{1,q}, w_{2,q}, \ldots, w_{m,q})$$

  where weights correspond to respective terms
- What are weights? Could be binary (1 or 0), term frequency, etc.
- A standard choice is: *tfidf* — term frequency inverse document frequency weights

$$tfidf = tf \cdot \log\left(\frac{N}{df}\right)$$

- *tf* is frequency (count) of a term in document, which is sometimes log-ed as well
- *df* is document frequency, i.e., number of documents in the collection containing the term

Example: Binary Weights

Consider documents:

```
d1:  dog cat dog dog
d2:  book sky dog book
d3:  cat cat sky cat
```

Example: *tf* Weights

Consider documents:

```
d1:  dog cat dog dog
d2:  book sky dog book
d3:  cat cat sky cat
```

Example: *tfidf* Weights

Consider documents:

```
d1:  dog cat dog dog
d2:  book sky dog book
d3:  cat cat sky cat
```

Cosine Similarity Measure

$$sim(q,d) = \frac{\sum_{i=1}^{m} w_{i,q} w_{i,d}}{\sqrt{\sum_{i=1}^{m} w_{i,q}^2} \cdot \sqrt{\sum_{i=1}^{m} w_{i,d}^2}} = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| \cdot |\vec{d}|}$$