Natural Language Processing
CSCI 4152/6509 — Lecture 7
Elements of Information Retrieval and Text Mining

Instructors: Vlado Keselj
Time and date: 16:05 – 17:25, 26-Sep-2023
Location: Rowe 1011

Previous Lecture

- Letter frequencies program (finished)
- Elements of Morphology
- Zipf's Law

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 _
# ...
```

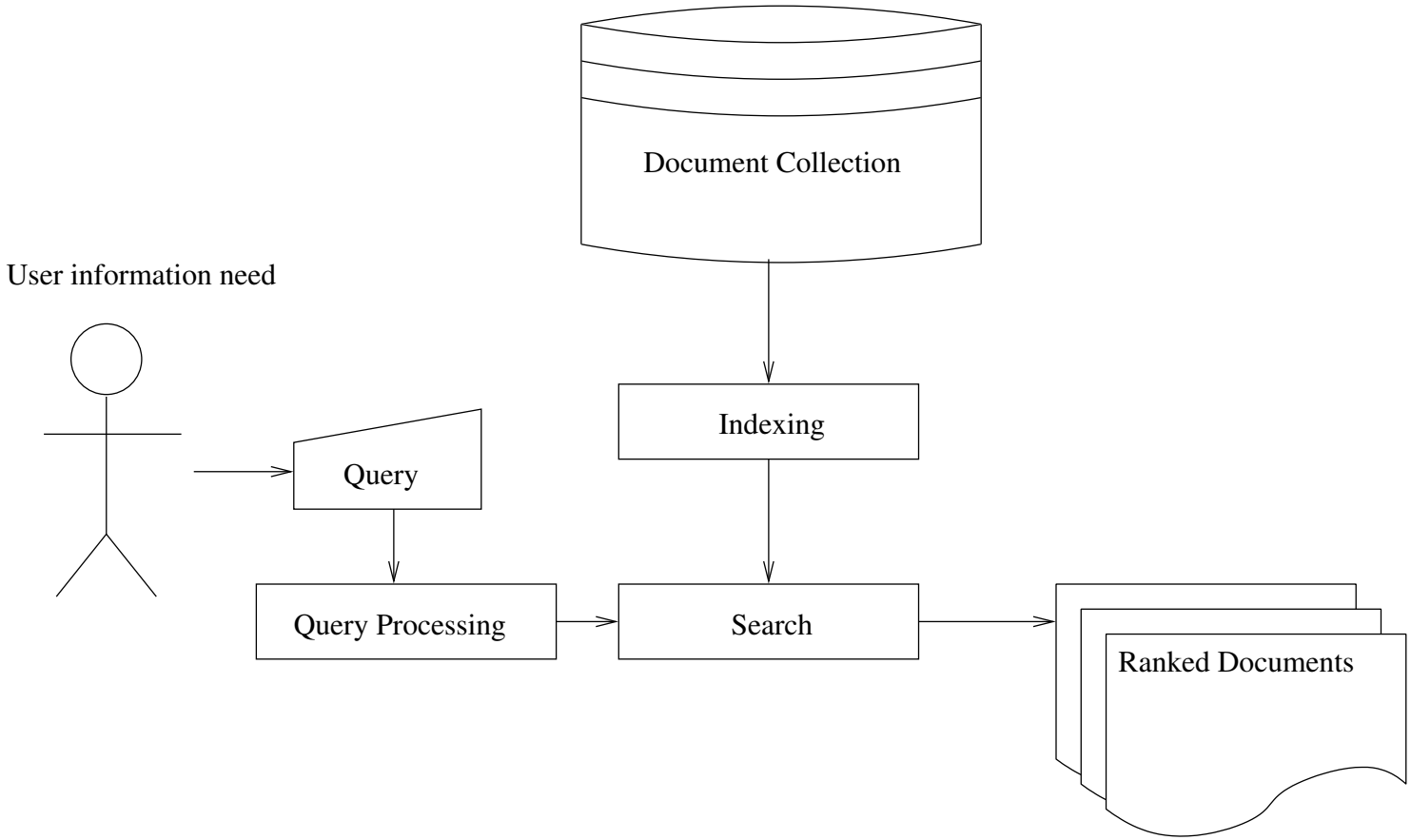
Using Ngrams Module

- 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
```