Natural Language Processing
CSCI 4152/6509 — Lecture 6
Text Similarity and Applications

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

Previous Lecture

- Back Reference Examples (continued)
- RegEx: Shortest match, Substitutions
- Text processing examples
  - tokenization
  - counting letters
- Elements of Morphology
  - Lemmatization, Morphological Processes
- Word Counting and Zipf's Law
- N-grams definition
- Extracting and Analyzing n-grams in Perl

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 _
# ...
```

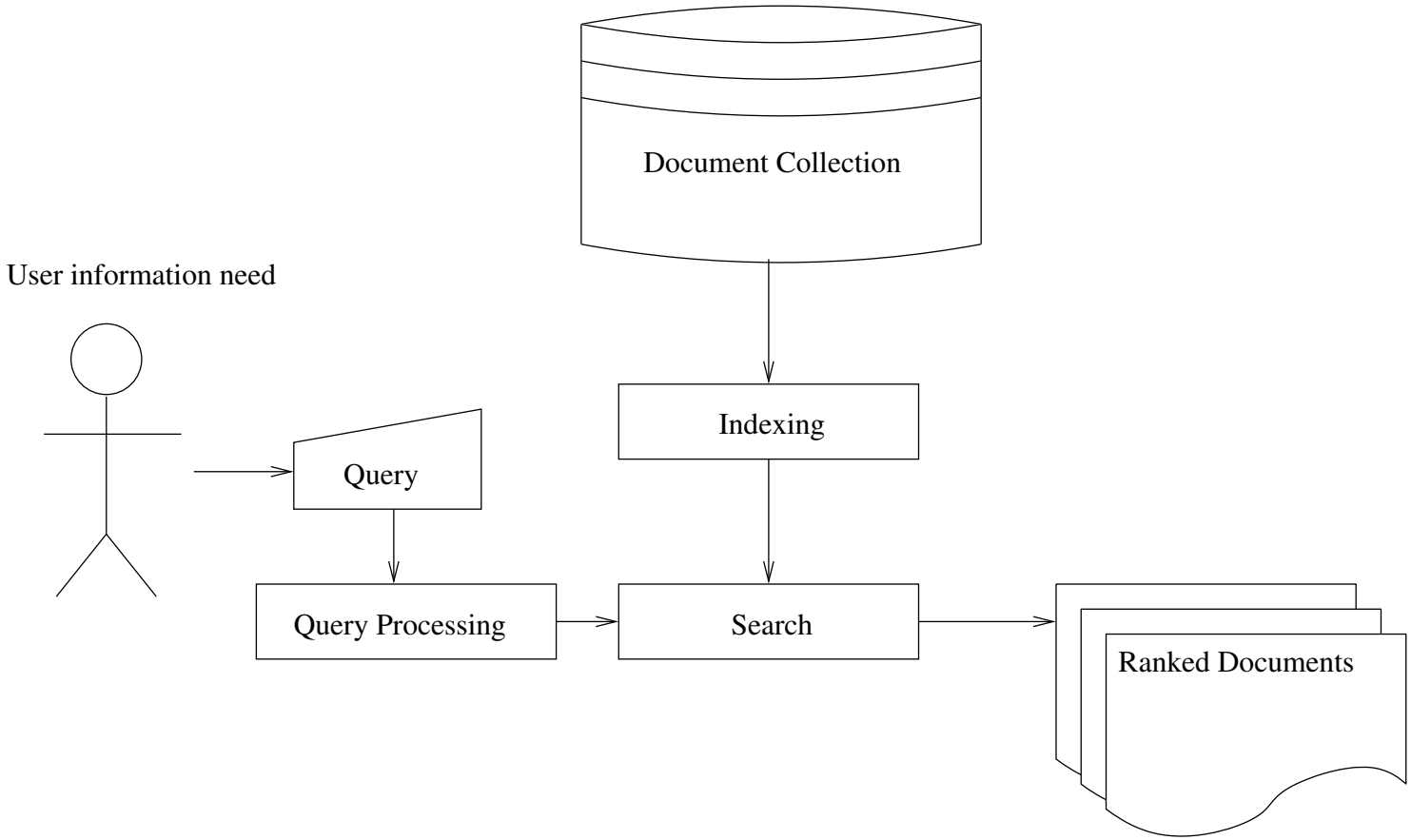
Using Ngrams Module

- 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 a 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
```

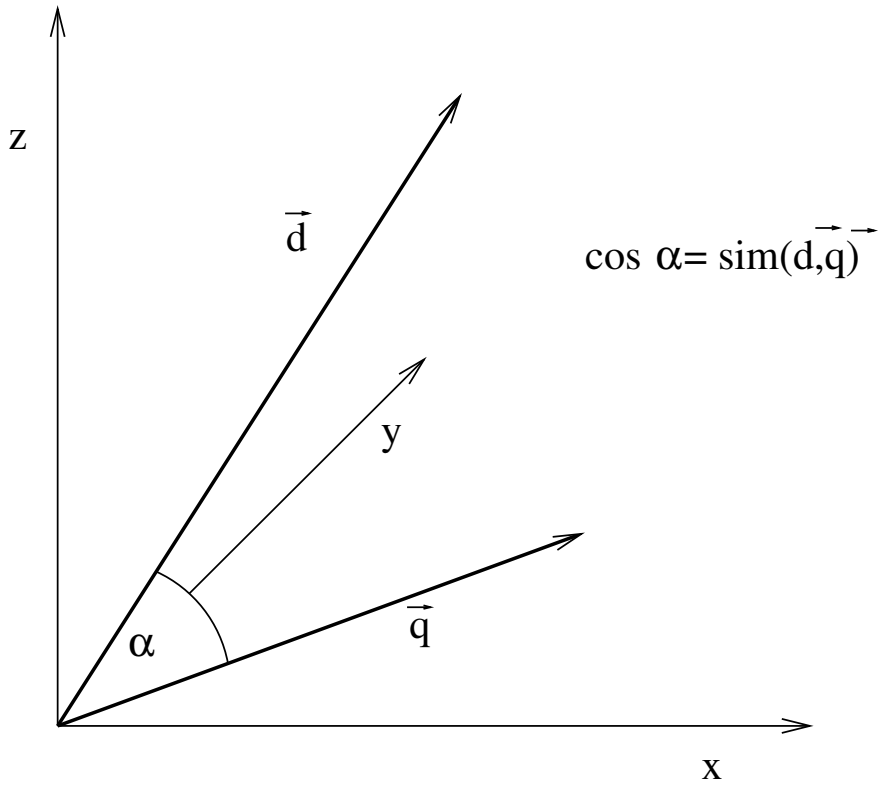
Example: *tfidf* Weights

Consider documents:

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

Cosine Similarity Measure

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



Side Note: Lucene and IR Book

- Lucene search engine
- `http://lucene.apache.org`
- Open-source, written in Java
- Uses the vector space model
- Another interesting link: Introduction to IR on-line book covers well text classification:

  `http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html`