Faculty of Computer Science, Dalhousie University *26-Sep-2023*

CSCI 4152/6509 — Natural Language Processing

Lecture 7: Elements of Information Retrieval and Text Mining

Location: Rowe 1011 Instructor: Vlado Keselj
Time: 16:05 – 17:25

Previous Lecture

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

7.2 Counting N-grams

Given a sequence of tokens $T = (t_1, t_2, t_3, \ldots, t_k)$ an n-gram is an arbitrary subsequence of n such tokens, such as $(t_1, t_2, \ldots, t_n)$, or $(t_2, t_3, \ldots, t_{n+1})$, and so on. Given a textual document, it could be broken into a list of character or list of words, if we assume that a character is a token, or that a word is a token. In those cases, we look at character n-grams or word n-grams, respectively. We typically want to collect all n-grams from a text, and a way to visualize this is to imagine a sliding window over the text.

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

For example, if we take another look at 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 combination of the characteristics of three boys
```

```
whom I knew, and therefore belongs to the composite order of
architecture.

The odd superstitions touched upon were all prevalent among children and
slaves in the West at the period of this story -- that is to say, thirty
or forty years ago.

Although my book is intended mainly for the entertainment of boys and
girls, I hope it will not be shunned by men and women on that account,
for part of my plan has been to try to pleasantly remind adults of what
they once were themselves, and of how they felt and thought and talked,
and what queer enterprises they sometimes engaged in.
...
```

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

The following Perl program `word-ngrams.pl` lists all word ngrams extracted from the standard input. We set variable `$n` to 3 as we want word 3-grams to be extracted. The first while-loop reads input line by line, and in the second while-loop we match string that we assume would be words. We choose any sequence of letters to be a word, and possibly starting with an apostrophe (`'`). As we will see later, this will recognize usual words, but it will also break complex words like `I'm`, `you're`, or `man's` into words `I` and `'m`, `you` and `'re`, and `man` and `'s`.

```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`

Since the first element of an array `@a` is `$a[0]` and the last element is `$a[$#a]` the number of elements is obviously `$#a+1$`. Another way to obtain the number of elements of an array is `scalar(@a)`. The Perl function `scalar` enforces a scalar context on an expression, and in a scalar context an array is interpreted just a number representing its length.

Perl arrays are dynamic, they expand and also can shrink easily. For example, after the command '`$a[5] = 1`' the array `@a` will be expanded if needed to at least six elements. The command '`$#a = 5`' sets array `@a` to exactly six elements. Similarly, '`$#a = -1`' erases an array by reducing it to zero elements, so it is equivalent to the command '`@a = ();`'.

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 _ _
```

This output may be what we want, but probably not. Since we already reduced repeated whitespace characters to one underscore ('_'), we probably want to treat the new line in the same way.

An easy way to solve the problem is to treat the whole file as one line:

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   ...
```

These days computers have very large working memories (RAM, or Random Access Memories), so reading a whole file in memory as in the above example is normally not a problem. If we want to avoid reading the whole file into memory and still recognize multi-line whitespace as one space character, it can be done but we will leave it for reader as an exercise. One approach would be to write a function `next_char` that keeps the current line and on each call reads the next character and returns it. When encoutering a whitespace character, it would read as many lines as needed until a non-whitespace character is found, and it would return a space.

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

If the files are very large, we may not want to read the whole file in memory and still want to handle multi-line whitespace. This problem may happen in general if any tokens that are part of n-grams span multiple lines. Again, the issue can handled in a brief way if we allow reading the whole file. We want to read the file line by line, without accumulating too many lines at a time, we would need to identify when a token may be spanning multiple lines and only then read lines ahead.

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 _
# ...
```

7.3 Using Ngrams Module

This section is covered in the lab, but you can also read here about the basic use of the Ngrams module.

We will now discuss how different kinds of n-grams can be collected using a Perl module named Text::Ngrams. A program associate with this module is named `ngrams.pl`, and both files, `Ngrams.pm` and `ngrams.pl`, can be found in the directory `~prof6509/public` on bluenose. They can also be found on the course website under the tab 'Misc'. If you use the web-site, for technical reasons the file `ngrams.pl` was renamed to `ngrams-pl.txt` and if you download it, you will need to rename it back to `ngrams.pl`.

The module and the program are open-source code, and can be found in the CPAN archive. The newest version is available on bluenose. The modules are typically installed system-wide and the Perl is configured in such way that it can easily find them. Since you do not have administrative permissions on bluenose, we need to use a way to use the module locally. The Perl modules can be installed on a per-user basis, either in a more systematic way or in more ad-hoc way. We will use here a local ad-hoc installation. You will cover the steps of installing the module in more details in the lab, but for now, we will assume that you are in a convenient sub-directory of your your home directory on bluenose. You would first copy the appropriate files using the commands:

```
cp ~prof6509/public/ngrams.pl .
cp ~prof6509/public/Ngrams.pm .
```

These files may actually be installed system-wide on bluenose, but to be sure to use the local version, we will do a couple additional operations and checks. First, create a subdirectory `Text` and copy the module there:

```
mkdir Text
cp Ngrams.pm Text
```

Check Local `ngrams.pl`

- Use command: `more ngrams.pl`

Let us take a look at the version of `ngrams.pl` that we use here. (This version is slightly different from the version in the CPAN archive.) We can use the command '`more ngrams.pl` and the beginning of the file should look as follows:

```
#!/usr/bin/perl -w

use strict;
use vars qw($VERSION);
$VERSION = 2.005;
# $Revision: 1.26 $

use lib '.';

use Text::Ngrams;
use Getopt::Long;
...
```

The line '`use lib '.';`' is important, since it directs Perl to give priority in finding the module in the current directory, rather than some other versions that may be available in the system. You can test the program `ngrams.pl` but typing:

```
./ngrams.pl
```

then typing some input, and pressing '`C-d`'; i.e., Control-D combination of keyboard keys. For example, if you type input:

```
natural language processing
```

you should get the output:

```
BEGIN OUTPUT BY Text::Ngrams version 2.005

1-GRAMS (total count: 28)
FIRST N-GRAM: N
 LAST N-GRAM: _
------------------------
_ 3
A 4
C 1
E 2
G 3
I 1
L 2
N 3
O 1
P 1
R 2
S 2
T 1
U 2

2-GRAMS (total count: 27)
FIRST N-GRAM: N A
 LAST N-GRAM: G _
------------------------
_ L 1
_ P 1
A G 1
A L 1
A N 1
A T 1
C E 1
E _ 1
E S 1
G _ 1
G E 1
G U 1
I N 1
L _ 1
L A 1
N A 1
N G 2
O C 1
P R 1
R A 1
R O 1
S I 1
S S 1
```

```
T U 1
U A 1
U R 1

3-GRAMS (total count: 26)
FIRST N-GRAM: N A T
 LAST N-GRAM: N G _
-------------------------
_ L A 1
_ P R 1
A G E 1
A L _ 1
A N G 1
A T U 1
C E S 1
E _ P 1
E S S 1
G E _ 1
G U A 1
I N G 1
L _ L 1
L A N 1
N A T 1
N G _ 1
N G U 1
O C E 1
P R O 1
R A L 1
R O C 1
S I N 1
S S I 1
T U R 1
U A G 1
U R A 1

END OUTPUT BY Text::Ngrams
```

This are the character n-grams of up to the size 3 of the given text, with their counts.

Verifying Version of Ngrams.pm

To test that the program is using the correct version of the module `Ngrams.pm` we can edit the file `Text/Ngrams.pm` and temporarily insert a ‘die’ command at the beginning of the module. The beginning of the module should look as follows:

```
# (c) 2003-2014 Vlado Keselj http://web.cs.dal.ca/~vlado
#
# Text::Ngrams - A Perl module for N-grams processing

die;

package Text::Ngrams;
```

```
use strict;
require Exporter;
use Carp;
...
```

- If we run `ngrams.pl` it should report error
- Delete '`die;`' command from the Ngrams.pm file

It is important to note that this is the copy of the module in the subdirectory `Text`. After this small test, do not forget to remove again the line '`die;`'.

8 Elements of Information Retrieval and Text Mining

In the previous sections, we looked at some methods for processing text in a stream mode. Many language processing tasks can be solved in this way, by using mainly regular expressions, extracting some pieces of text, and collecting basic statistics. We will now look at some techniques for working with the documents as whole units withing large collections. First we will look at the task of Information Retrieval, and then the area of Text Mining, with a particular emphasis on Text Classification and brief mentioning of Text Clustering.

The term *Text Mining* was coined at about the same time as *Data Mining*, and it consists of methods for a coarse-grained management of text documents, such as classification and clustering; but also some finer-grained mining of information, such as in information extraction.

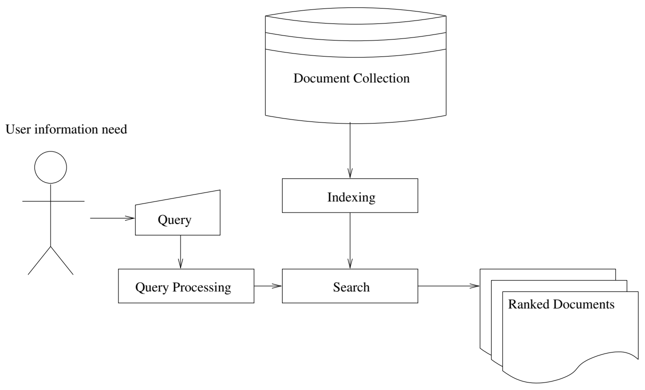
8.1 Elements of Information Retrieval

– Reading: [JM] Sec 23.1, ([MS] Ch.15)

Information Retrieval is an area of Computer Science mainly concerned with the task of finding a set of relevant documents from a document collection given a user query. A search engine, such as Google, is a information retrieval system.

Basic Information Retrieval problem definition: The basic definition of the problem or task of Information Retrieval is also called *ad hoc retrieval* and is given as follows: We are given a set of documents called a *document collection*, where each document is a natural language text. A user has an *information need* which she or he will need to express as a *query*, which is a short text, possibly in natural language or some more specialized format. The task of an Information Retrieval system is to return a subset of documents from the document collection that are *relevant* to the user query. The relevant documents should also be sorted by relevancy, starting from the most relevant document; i.e., a *ranked list of 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

The document semantics is reduced to the set of stems of content-bearing words.

8.2 Vector Space Model

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

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

- *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