Open Source Text Mining

Mathew Flynn, PhD
Louise Francis, FCAS, MAAA

Rationale For Paper

- Text mining is a new and promising technology for analyzing unstructured text data
- Commercial text mining software can be expensive and difficult to learn
- Several free open source languages can perform text mining, but without help thay can be difficult to learn
- In this session we will provide a mini- tutorial to 2 open source products

Two Open Source Products for Text Mining

- Perl – a text processing language
- R – a statistical and analytical language with text mining functionality provided by a text mining package tm

The Data

- Text mining can be applied to many common tasks
  - Internet searches
  - Screening emails for spam
  - Analyzing free form fields in underwriting and claims files
  - Analyzing survey data
- We illustrate the last 2
- Survey data can be downloaded from CAS web site.

Mini Tutorial

- We will give tutorial on using Perl and R for text mining
- Download the survey data
- Follow our examples

The Survey Data

- From 2008 CAS Quinquennial Survey
- What are the top two issues that will impact the CAS in the next five years?

Survey Question: Top Two Issues Affecting CAS
A crisis that could affect our ability to "regulate" ourselves.
A need to deal more thoroughly with non-traditional risk management approaches
Ability of members to prove they are more than just number crunchers
ability to convince non-insurance companies of the value/skills offered by CAS members.

Perl

- Go to www.Perl.org
- Download Perl
- Run execute file (or run active perl)
- the Windows command search must be correct for Windows to find the desired perl.exe

www.Perl.org

Good References

- *Practical Text Mining with Perl* by Bilisoly (2008) is an excellent resource for text mining in Perl
- *Perl for Dummies* (Hoffman, 2003) provides a basic introduction including needed header information

Some Key Things

- Perl must be run from DOS. One gets to DOS by finding the Command Prompt on the Programs menu
- Before running Perl switch to the Perl directory (i.e., if Perl was installed and it is in the folder named Perl, in DOS, type “cd C:\Perl”).
- Programs need to be saved in text processing software. We recommend Notepad rather than Word, as some of the features of Word cause unexpected results when running a program. We recommend using the extension .pl.

Some Key Things cont.

- The header line of a Perl program is dependent on the operating system.
- To run a Perl program type the following at the command prompt:
  - Perl program_name input_file_name, output_file_name[1]
- The input and output files are only required if the program requires a file, and the file name is not contained in the program itself.
- The input and output file may be contained in the program code, as illustrated in some of our examples, rather than entered at runtime.

Parsing Text

- Identify the spaces, punctuation and other non alphanumeric characters found in text documents and separating the words from these other characters
- Most computer languages (and spreadsheets) have text functions that perform the search and substring functions to do this
- Perl has special functions for parsing text

The split function

- split(*/separating character(s)/, string*)
- Example
  - $Response = "Ability of members to prove they are more than just number crunchers";
  - @words =split (/ /, $Response);

Complications of split function

- More than one space
  - @words =split (/ [\s+]/, $Response);
- Other separators
  - Use substitute function

Simple parse program: Parse2.pl

- #!perl -w
- # Parse2.pl
- # Program to parse text string using one or more spaces as separator
- $Response = "Ability of members to prove they are more than just number crunchers";
- @words =split (/\s+/, $Response); #parse words in string
- # Loop through words in word array and print them
- foreach $word (@words) {
- print "$word\n";
- }

Less Simple parse program: Parse3.pl

- #!perl -w
- # Parse3.pl
- # Program to parse a sentence and remove punctuation
- $Test = "A crisis that could affect our ability to 'regulate' ourselves.";# a test string with punctuation
- @words =split (/[\s+]/, $Test); # parse the string using spaces
- # Loop through words to find non punctuation characters
- foreach $word (@words) {
- while ($word =~ /(\w+)/g) {
- # match by 1 or more alphanumeric characters. These will be the words excluding punctuation
- print "$1 \n"; #print the first match which will be the word of alphanumeric characters
- }
- }

Read in survey data and parse

- #!perl -w
- # Enter file name withtext data here
- $TheFile ="Top2lss.txt";
- # open the file
- open(INFILE, $TheFile) or die "File not found";
- # read in one line at a time
- while(<INFILE>) {
- chomp; # eliminate end of line charachter
- s/[.?!"()'{},&;]//g; # replace punctuation with null
- s/\// /g; # replace slash with space
- s/\-//g; #replace dash with null
- s/^ //g; #replace beginning of line space
- print "$_\n"; # print cleaned line out
- @word=split(/[\s+]/); # parse line
- }

Print it out also

- #!perl -w
- #parsecomplex.pl
- # Enter file name withtext data here
- $TheFile ="Top2lss.txt";
- # open the file
- open(INFILE, $TheFile) or die "File not found";
- # read in one line at a time
- while(<INFILE>) {
- chomp; # eliminate end of line charachter
- s/[.?!"()'{},&;]//g; # replace punctuation with null
- s/\// /g; # replace slash with space
- s/\-//g; #replace dash with null
- s/^ //g; #replace beginning of line space
- print "$_\n"; # print cleaned line out
- @word=split(/[\s+]/); # parse line
- foreach $word (@word) {
- print "$word,"}
- print "\n";
- }

Word Search

- First, read in the accident description field
- For each claim
  - Read in each word
  - If the lower case of the target word is found output a 1 for the new indicator variable, otherwise output a 0.

SearchTarget.pl

```
SearchTarget.pl
$target = “(regulaton)”;
# initialize file variable containing file with text data
$TheFile =“Top2Iss1.txt”;
open(INFILE, $TheFile) or die "File not found"; # open the file
# initialize identifier variables used when search is successful
$i=0;
$flag=0;
# read each line
while(<INFILE>) {
 chomp;
++$i;
# put input line into new variable
$Sentence = $_;
# parse line of text
@words = split(/[\s+]/,$Sentence);
$flag=0;
foreach $x (@words) {
if (lc($x) =~ /$target/) {
$flag=1;
 }
}
# print lines with target variable to screen
print "$i $flag $Sentence \n";
}
```

Using Target in Analysis

Homeowner Claim	Mean Severity
No	2,376.6
Yes	6,221.1

Text Statistics

- The length of each word is tabulated within a loop. A key line of code is:
- $count[length($x)] +=1; #increment counter for words of this length

Perl Program for Word Lengths

Length.pl

```
#!perl -w
# Enter file name with text data here
$TheFile ="Top25bs.txt";
# open the file
open(INFILE, $TheFile) or die "File not found";
# read in one line at a time
while(<INFILE>) {
 chomp; # eliminate end of line character
 s/[.?!;*()]/ /g; # replace punctuation with null
 s/\// /g; # replace slash with space
 s/-//g; #replace  dash with null
 s/^ //g; #replace beginning of line space
 print "$_\n"; # print cleaned line out
 @word=split(/\s+/);  # parse line
# count length of each word in array @count
                foreach $x  (@word) {
                        $count[length($x)] +=1 ;}
 }
$mxcount=$#count;
# print out largest word size and frequency of each count
print "Count $mxcount\n";
for ($i = 0; $i <= $#count; ) {
# does word of that size exist?
if ( exists($count[$i]) ) {
print "There are $count[$i] words of length $i\n";
}
$i += 1;  # increment loop counter
}
```

Hashes

- A hash is like an array, but can be distinguished from an array in a number of ways.
- An array is typically indexed with zero and integer values, while a hash can be indexed with a letter or word.
- Instead of an index the hash has a key that maps to a specific array value.
  - For instance, while the first entry in a Perl array is $array[0], the first entry in a hash might be $hash{'a'} or $hash{'x'} or even $hash{'hello'}. (Note that the order is not relevant.)
- Because the time to locate a value on a hash table is independent of its size, hashes can be very efficient for processing large amounts of data

Hashes cont.

- A hash variable begins with a %
- A hash holding a list of words might be denoted %words.
- A hash holding the counts of words from a document might be %count, and the indices of the hash can be the words themselves.
- A specific value of a hash is referenced by using a dollar sign ($) in front of the hash variable name, and referencing the specific item with brackets.
  - For example, the specific word homeowner is referenced as $words{'homeowner'}.
  - Thus the indices of the hash are strings, not numbers.

Use Hash Distribution of Word Frequencies

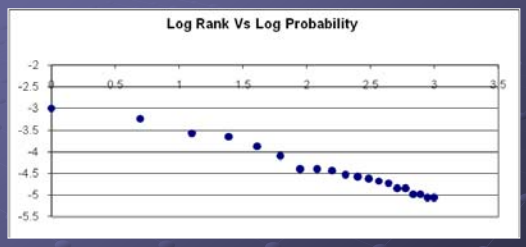
```
Testhash.pl
#!perl -w
# Testhash.pl
# Usage:  testhash.pl <datafile> <outputfile>
# input datafile must be present and a command line arg such as Top2Iss.txt
open(MYDATA, $ARGV[0]) or die("Error: cannot open file '$ARGV[0]'\n");
# output datafile must be present and a cmd line arg
open(OUTP, ">$ARGV[1]") or die("Cannot open file '$ARGV[1]' for writing\n");
print OUTP "Output results for ".$ARGV[0]."\n";
# read in the file, get rid of newline and punctuation chars
while( $line = <MYDATA> ){
            chomp($line);
# eliminate punctuation
            $line =~ s/[-.?!"()'{}&;]//g;
            $line =~ s/\s+/ /g;
            @words = split(/ /,$line);

            foreach $word (@words) {
      ++$counts{lc($word)};
            }
}
# sort by value (lowest to highest using counts for the key)
# and write the output file and screen
foreach $value (sort {$counts{$a} cmp $counts{$b} }
       keys %counts)
{
# print the word and the count for the word
            print "$value $counts{$value} \n";
            print OUTP "$value $counts{$value} \n"
}
# close the files
close MYDATA;
close OUTP;
```

Word Frequencies

Rank	Word	Count	P(Rank=k)
1	of	102	0.05
2	the	80	0.04
3	to	57	0.03
4	and	53	0.03
5	in	42	0.02
6	actuaries	34	0.02
7	other	27	0.01
8	from	26	0.01
9	for	25	0.01
10	erm	24	0.01
726	alternative	1	0.00
727	thin	1	0.00
728	information	1	0.00
729	industries	1	0.00
730	retire	1	0.00

Zipf's Law



Stop Words

- Frequently occurring words
- The
- A
- To
- It
- Do not contribute to meaning of record of text
- Eliminate

Substitution operator

- Thus to eliminate the word "the", use the code

  s/the//g;
- Apply to multiple terms you want to eliminate

  s/[-.?!"()'{}&;]//g;

Term Document Matrix

- A Table of indicator variables
- If a word is present, a 1, otherwise a 0

Term Data Matrix

Ourselves	cas	Not	That	communicators/executive	our	approaches
1	0	0	1	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Word Lengths

Length	GL Data	Survey Data
1	1,062	21
2	4,172	309
3	5,258	298
4	5,418	215
5	2,982	153
6	2,312	143
7	2,833	213
8	1,572	161
9	1,048	216
10	591	146
11	111	92
12	156	44
13	78	61
14	19	2
15	0	3
16	1	1
17	2	0
18	1	0
19	1	0

Stopwords.pl

```
# StopWords.pl
# This program eliminates stop words and  computes the term-document matrix
# a key part is to tabulate the indicator/count of every term - usually a word
# it may then be used to find groupings of words that create content
# This would be done in a separate program
# Usage:  termdata.pl <datafile> <outputfile>
$TheFile = "Top2Iss.txt";
#$Outp1 = "OutInd1.txt";
open(MYDATA, $TheFile ) or die("Error: cannot open file");
open(OUTP1, ">OutInd1.txt") or die("Cannot open file for writing\n");
open(OUTP2, ">OutTerms.txt") or die("Cannot open file for writing\n");
# read in the file each line and create hash of words
# create grand dictionary of all words
# initialize line counter
  $i=0;
while (<MYDATA> ){
                  chomp($_);
            s/[-.?!"()'{}&;]//g;
            s/^ //g;
            s/,//g;
            s/\d/ /g;
            s/(\sof\s)/ /g;
           s/(\sto\s)/ /g;
           s/(\sthe\s)/ /g;
           s/(\sand\s)/ /g;
           s/(\sin\s)/ /g;
           s/(The\s)/ /g;
           s/(\sfor\s)/ /g;
           s/(\as\s)/ /g;
             s/(A\s)/ /g;
           s/(\sin\s)/ /g;
               s/(\swith\s)/ /g;
               s/(\san\s)/ /g;
               s/(\swith\s)/ /g;
               s/(\sare\s)/ /g;
```

Stopwords.pl cont.

```
               s/(\sthey\s)/ /g;
               s/(\sthan\s)/ /g;
               s/(\sas\s)/ /g;
               s/(\sby\s)/ /g;
                s/\s+/ /g;
            if (not /^$/) { #ignore empty lines
                              @words = split(/ /);
                        foreach $word (@words) {
                      ++$response[$i]{lc($word)};
                      ++$granddict{lc($word)};
                              }
                       ++$i;
             }
}
$nlines = $i-1;
for $i (0..$nlines) {
     foreach $word (keys %granddict) {
             if (exists($response[$i]{$word}))
             {
             ++$ indicator[$i]{$word}; }
              else
             {
              $indicator[$i]{$word}=0;
                 }
print  OUTP1 "$indicator[$i]{$word},";
              }
  print OUTP1 "\n";
}
foreach $word (keys %granddict) {
         print OUTP2 "$word,$granddict{$word}\n";
         }
# close the files
close MYDATA;
close OUTP1;
close OUTP2;
```

OutPut Matrix

1	0	0	1	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0