Open Source Text Mining

Mathew Flynn, PhD
Louise Francis, FCAS, MAAA

Rationale For Paper

- Text mining is a 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 introduce 2 open source products and mention a third

Two Open Source Products for Text Mining

- R – a statistical and analytical language with text mining functionality provided by a text mining package tm along with other packages that provide additional capability
- Python – a n analytical language used by computer scientists, data scientists and engineers.
- Perl – Historically recognized for its string processing capabilities

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
- Accident description field in a claim file
- Survey data can from a 2008 CAS quinquennial survey.

Mini Tutorial

- We will give tutorial on using R and Python for Various aspects of text mining
- We give a brief background on Perl
- We introduce the 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.

We Begin with Perl

- We begin with Perl to illustrate key text mining concepts and procedures
- Go to www.Perl.org to download
- Our “Open Source Text Mining” paper made heavy use of Perl (find at www.casact.org)
- Much of the analytics community now uses Python instead of Perl

The Perl Programming Language

HOME | LEARN | DOCUMENTATION | CPAN | COMMUNITY | GET INVOLVED

25,000 extensions on CPAN

That's why we love Perl 5

Get started

DOWNLOAD PERL

Perl 5 is a highly capable, feature-rich programming language with over 29 years of development. More about why we love Perl...

Learning Perl 5

With free online books, over 25,000 extension modules, and a large developer community, there are many ways to learn Perl 5.

The Perl Community

Perl has an active world wide community with over 300 local groups, mailing lists and support/discussion websites.

Text Mining Steps

- Data Preprocessing
  - Clean data: remove misspellings, punctuation, numbers, convert to lower case
  - Split individual words from spaces, punctuation
  - Remove stop words
  - Create document term matrix with results
- Data Exploration
- Use analytic techniques to derive meaning
- Use for prediction

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

Regular Expressions

- A language for string pattern description
- There can be some variations across languages such as Perl, Python, R
- There are various shorthand characters to denote types of strings including
- ‘/d for digit
- /b for blank at beginning of a word
- /w for an alphanumeric character
- /^ at beginning denotes beginning of string

Simple parse program: Parse2.pl

- #!perl -w
- # Parse2.pl
- # Program to parse text string using one or more spaces as separator
- The split function uses a Regular Expression (\s+) to capture one or more spaces
- $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 =“Top2Iss.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
- }

Word Search

- First, read in the data
- 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";
}
```

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
- Cycle through every record in the data
- And every word found at least once
- 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

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