

# Text Mining Handbook

Louise Francis, FCAS, MAAA, and Matt Flynn, PhD

---

## Abstract

**Motivation.** Provide a guide to open source tools that can be used as a reference to do text mining

**Method.** We apply the text processing language Perl and the statistical language R to two text databases, an accident description database and a survey database.

**Results.** For the accident description data new variables are extracted from the free-form text data that are used to predict the likelihood of attorney involvement and the severity of claims. For the survey data, the text mining identified key themes in the responses. Thus, useful information that would not otherwise be available was derived from both databases.

**Conclusion.** Open source software can be used to apply text mining procedures to insurance text data.

**Availability.** The Perl and R programs along with the CAS survey data will be available on the CAS Web Site.

**Keywords.** Predictive modeling, data mining, text mining

---

## 1. INTRODUCTION

Text mining is an emerging technology that can be used to augment existing data in corporate databases by making unstructured text data available for analysis. An excellent introduction to text mining is provided by Weiss, et al. (2005). Francis (2006) provides a short introduction to text mining with a focus on insurance applications. One of the difficulties in getting started with text mining is acquiring the tools, i.e., the software for implementing text mining. Much of the software is expensive and/or difficult to use. For instance, some of the software requires purchase of expensive data mining suites. Other commercial software is more suited to large scale industrial strength applications such as cataloging and searching academic papers. One innovation that has made budget-friendly software available to text miners is open source software. Two of the very popular open source products that will be featured in this paper are R and Perl. R is a programming language that has wide acceptance for statistical and applied mathematical applications. Perl is a programming language that is particularly strong in text processing. Both languages can be easily downloaded from the Internet. Moreover, a large body of literature now exists to guide users through specialized applications such as text mining. This paper will rely heavily on information in the book *Practical Text Mining in Perl* by Roger Bilisoy (2008) when illustrating text mining applications in Perl.

It will also rely heavily on the R **tm** library. While this library is described by Feinerer, Hornik, and Meyer (2008), the applications are not insurance applications, and a significant amount of trial and error can be required to achieve successful application of the software to insurance problems.

We hope to make it easier for potential users to employ Perl and/or R for insurance text mining projects by illustrating their application to insurance problems with detailed information on the code and functions needed to perform the different text mining tasks.

In this paper we will illustrate the application of the text mining software using two applications: 1) analyzing a free-form accident description field in an insurance claims database and 2) analyzing free-form comments in a survey of actuaries. Both applications involve relatively small text fields, compared to those anticipated by many text mining products. We will also show how text mining can be used to generate new information from the unstructured text data.

## **1.1 Research Context**

The field of text mining is rapidly evolving, but at this time is not yet widely used in insurance. Kolyshkina and Rooyen (2006) presented the results of an analysis that applied text mining on an insurance claims database. They applied text mining to a free-form claim comment field to derive “concepts” from the description. Francis (2006) presented an introduction to text mining that provided a simple illustration using a liability claim description field.

A number of texts introduce text mining to less technical audiences. Inmon and Nesavich (2007) provide a nontechnical overview that describes some of the history and applications of text mining. This book gives an overview of the procedures used to mine data, from preprocessing and integrating the text to using data mining procedures like self-organizing maps. Weiss, et al. (2005) provide a comprehensive introduction that includes pseudo code to implement the different parts of the process as well as access to a simple software tool that implements many of the procedures in the book. Weiss’s software tool provides an inexpensive way to gain insights into text mining and to perform some text processing and pattern recognition procedures. However, the software has some limitations. It requires download and installation of Java along with the text mining software, can only work with data in xml format, produces output that requires further processing by another programming language before common analyses can be performed and has a steep learning curve. A more elaborate description of difficulties encountered when utilizing text mining software, both open source and commercial, is provided by Francis (2006). Francis suggests that text miners may want to learn one of the languages specifically oriented towards text processing such as Perl or Python.

The rapidly evolving field of text mining has seen advances in the open source tools available for text mining. These tools include 1) introductions to programming in Perl such as Bilisoly (2008),

which specifically focus on text mining, 2) interfaces to Perl in some of the popular statistical programming languages such as SAS, 3) new text mining functions in the popular analytical programming language R, and 4) interfaces to R in some of the popular programming languages. This paper will introduce text mining users to two of these open source tools and provide two detailed examples of their application. The two examples will be of particular interest to actuaries and insurance professionals.

## **1.2 Objective**

The objective of this paper is to improve the accessibility and usability of text mining for insurance applications. By documenting two open-source software languages that can be used for text mining we hope to reduce the significant learning curve that can be associated with text mining software. We focus on two popular open-source languages that can be downloaded for free. This paper will describe in some detail two applications of text mining in insurance. Helpful references for those new to Perl and R will also be provided, as this paper is not a self-contained introduction to these languages.

## **1.3 Outline**

The remainder of the paper proceeds as follows:

- Section 2 will discuss the first phase of text mining: preprocessing of text data and will present techniques that are programmed in Perl;
- Section 3 introduces the R text mining library and will apply it to the GL accident description data;
- Section 4 will apply the R text mining functions to the survey data.
- Both sections 3 and 4 will introduce the second or analytical phase of text mining along with their implementation using R statistical functions.
- Section 5 will summarize the analysis and present conclusions.

## **2. BACKGROUND AND METHODS**

There are two main phases to text mining: 1) Preprocessing and integration of unstructured data, and 2) statistical analysis of the preprocessed data to extract content from the text. Section 2 will cover the preprocessing phase and show how Perl can be used for the preprocessing.

## 2.1 Data and Software

Two different data sets will be used to illustrate the open source tools: a claim description database and a survey response database. The claim description data is a field from a general liability (GL) database. Table 2.1 provides a sample of several records of the claim description data.

**Table 2.1**

**Accident Description**

---

Crew was hooking a jumper pipe between fire line and water line in order to perform a flush test. when the line was charged the jumper pipe slipped off causing water to spray on newly hung dry wall. Damage to one piece of dry wall and few pieces of i.

---

CAT 345 traveling under a guide wire when the back of the boom caught the wire. When the wire became taut it caused the power pole to break and wire to snap.

---

Insd was working and damaged buried service wires at a customer's residence.

The GL data also contains two additional variables: the claim's severity developed to ultimate and trended to a common point in time and an indicator variable encoding whether or not an attorney is involved in the claim. Both variables are potential dependent variables in a predictive model. That is, we may be interested in what the characteristics are of claims that tend to have attorney involvement, as the claims adjusters might be better able to manage claims if they had such information. In addition, the claim severity variable is a key outcome variable of the underwriting and claims adjustment process and may assist a company to better understand and manage this variable.

The survey data is from the Casualty Actuarial Society Quinquennial Survey conducted in 2008. Numerous questions on the survey allowed for write-in answers. One of the questions asked was: "What are the top two issues that will impact the CAS in the next five years?" Table 2.2 shows an example of some of the answers to the question.

**Table 2.2**

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

The survey data does not contain any potential dependent variables. When analyzing free-form survey responses, text mining is used to group the unique response into categories of responses, as an aid in reducing the many individual responses to a smaller set that still faithfully represents the responses supplied.

### **2.1.1 The Software - Perl**

Two software programs are featured in this paper: Perl and R. Each must be downloaded from the appropriate Web site and then installed, before it can be used.

The Perl Web Site is [www.Perl.org](http://www.Perl.org). Figure 3 displays the screen for the home page of the Web site. By clicking on “Download” on the top left side, you will be redirected to a site where you can download Perl. Perl must then be installed by running the execute file. A commonly used alternative is ActivePerl from <http://www.activestate.com/activeperl/>. One common “gotcha” with either install (or, battling multiple Perl installs/versions on Windows platforms) is that the Windows command search must be correct for Windows to find the desired perl.exe file. This executable search path is governed by the search path for the commands. If things are amiss, one can check/edit the setting of one’s Window’s environmental “PATH” variable. The current setting can be examined from a cmd prompt by simply typing “PATH”. The path variable itself can be easily set in standard Windows fashion from the desktop, right-click on “My Computer”, “Properties”, “Advanced”, “Environmental Variables” button, scroll-down the tiny window to find the path variable and click on the “Edit” button, and adjusting the very long string in the tiny edit window.

While free documentation for using Perl is available on the Perl Web Site, there are many books that will guide the new user through key aspects of its use that may be difficult to understand solely from the online documentation. The book *Practical Text Mining with Perl* by Bilisoly (2008) is an excellent resource for text mining in Perl that requires no prior exposure to Perl. However the book does not cover a few introductory topics such as installation and header lines required at the beginning of Perl programs. *Perl for Dummies* (Hoffman, 2003) covers these topics in its first few chapters.

Key things to keep in mind when using Perl are:

- Perl must be run from DOS. One gets to DOS by finding the Command Prompt on the

Programs menu.<sup>1</sup>

- 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”.
- The header line of a Perl program is dependent on the operating system. Appendix B provides examples of the header line that executes successfully on a Windows computer.
- To run a Perl program type the following at the command prompt:

Perl program\_name input\_file\_name, output\_file\_name<sup>2</sup>

- 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.
- A few aspects of the syntax to help readers follow the Perl code are:
  - Each line ends with a semicolon.
  - The # is used for comments.
  - Scalar variable names begin with a \$.
  - Vector variable names begin with the @.
  - A specific value of an array is accessed with brackets; \$Array [1] accesses a specific value of an array. Note the use of the dollar sign when accessing a specific array value.

---

<sup>1</sup> The operating system used with Perl for this paper was Windows XP. The command prompt is found by clicking on “Programs” and then “Accessories.” Placing a shortcut on the desktop will make it more efficient to get to DOS.

<sup>2</sup> The input and output file may be contained in the program code, as illustrated in some of our examples, rather than entered at runtime.

Figure 2.1

The Perl Web Site



## 2.2 Step 1: Preprocessing the Data

The data that most analysts are accustomed to working with is structured data. Text data is unstructured data. Unstructured data has the following characteristics:

- It has no specified format.
  - .txt, .html, .xml, .pdf, .doc are among the possible formats for text data.
- It has variable length.
  - One record might contain a phrase of a few words, a few sentences or an academic paper of many pages.

- It can have variable spelling.
  - Misspellings, abbreviations, and singular versus plural forms of words are among the reasons for variable spelling .
- Punctuation and other nonalphanumeric characters may be in the data.
  - Periods, question marks, dashes, equal signs, and quotation marks are among the characters that may be found.
- The contents are not predefined and do not have to adhere to a predefined set of values.
  - Even when restricted to one discipline such as actuarial science, a paper can be on a variety of topics such as ratemaking, reserving, reinsurance, enterprise risk management, or data mining. An accident description field may refer to the part injured, the injury, circumstances of the accident, etc.

To make text data useful, unstructured text data is converted into structured data for further processing. There are several steps involved in the preprocessing:

- Parse the data. That is, extract the words from the data, typically discarding spaces and punctuation.
- Eliminate articles and other words that convey little or no information.
- Replace words that are synonyms, and plural and other variants of words with a single term.
- Create the structured data, a table where each term in the text data becomes a variable with a numeric value for each record.

### **2.3 Parsing the Text Data**

Parsing text involves identifying the spaces, punctuation, and other non alphanumeric characters found in text documents, and separating the words from these other characters. Most programming and statistical languages contain character procedures that can be used to parse the text data. For instance, if the text data contains no punctuation and the words are separated by a space the algorithm for parsing words from spaces is:

- Initialize an array to hold the words that are parsed.
- Search for the first space and record its position.



- Extract the string from the first position to the position before the next space. This is the first word.
- Find the next space and record its position.
- Use the positions of the previous and subsequent space to extract the second word.
- Repeat until all words are parsed.

In Perl a special text function helps one to quickly parse text data. The function is the *split* function, which splits words from spaces and other unwanted characters. The split function takes as its arguments a character string used to separate words (contained between slashes) and the text string to be parsed.

For instance, the following code (saved in the program Parse1.pl) can be used to parse the survey response, “Ability of members to prove they are more than just number crunchers.”

```
$Response3 = “Ability of members to prove they are more than just number crunchers”;  
  
@words =split (/ /, $Response);
```

When the print function is used (i.e., within a loop containing print “\$words<sup>4</sup>\n”;) the result is a printout of the word array with each entry on a different line as in Figure 2.2.

A natural question to ask is, “What happens if there are two spaces instead of one?” The following code parses a text string with more than one space separating the words where the string expression “\s+” is used to denote one or more spaces.

```
$Response = “Ability of members to prove they are more than just number crunchers”;  
  
@words =split (/ [\s+]/, $Response);
```

The notation “\s” denotes a single space and is the same as simply placing a space between the slashes in the split function. However, the notation “\s+” denotes one or more spaces, and can thus be used where there are a variable number of spaces. In addition, the split parameters “\s+” have been enclosed in brackets<sup>5</sup>. The code was saved in the program “Parse2.pl.

The expression “\s+” is referred known as ”regular expression.” Regular expressions can be used to 1) parse text containing more complicated structures such as multiple spaces and punctuation, 2)

---

<sup>3</sup> Note the use of quotation marks used when giving a string variable a value within a program

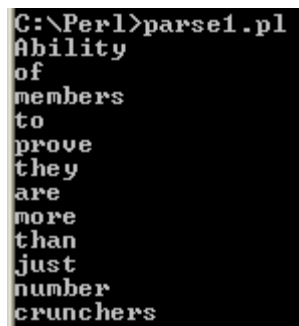
<sup>4</sup> \$words is used to print an array element, rather than the entire array. See sample program Parse1.pl.

<sup>5</sup> Without a bracket around the “\s+”, the code does not always work as expected.

search for specific strings of text in a large database, and 3) replace certain strings with other strings (or the empty string). Because regular expressions use special variables to specify character patterns, such as expressions that denote digits, letters, etc., we have included a section in Appendix A that contains a glossary defining the common variables.

Figure 2.2

Output of Pars1.pl shown on DOS screen



```
C:\Perl>parse1.pl
Ability
of
members
to
prove
they
are
more
than
just
number
crunchers
```

Regular expressions can be used to eliminate unwanted punctuation from text data. For instance, suppose you are parsing the expression “A crisis that could affect our ability to ‘regulate’ ourselves.” This express contains quotes and a period that we want to remove. This can be accomplished by first using the split function then using an additional regular expression to retain only the alphanumeric characters, not punctuation. An example of this is supplied in the program Parse3.pl. In addition, the substitution function, `s`, can be used to find and replace unwanted text. For instance if the text is:

```
$Response="Desire/ability for participation from a wide spectrum of members."
```

Typically the substitution operator has the following form: `s /string/replacement string/`. If we want to replace all periods with a space we would use `s/./ /g`. The `g` following the operator stands for ‘global’ and will replace all occurrences of the string. Additional modifiers may be required, i.e., when the character removed is the `/`. We can remove the `/` with the following code which must use the escape symbol `\` before the `/`.

```
Response=~ s/\\//g;
```

More complicated expressions are used to remove all unwanted punctuation, and examples are provided in the sample Perl programs and appendices<sup>6</sup> supplied with this paper.

---

<sup>6</sup> Many of the Perl programs referenced in this paper are included in Appendices.

More complicated regular expressions can also be used to extract strings *not* containing a specific set of characters, extract strings with pre-specified patterns of letters or numbers, such as a phone number or social security number, and to determine where sentences begin and end, etc. The reader who wishes to pursue this subject further should refer to Chapter 2 of Bilisoly (2008), Chapter 12 of Hoffman (2003) or the more extensive coverage in Frenz (2005).

The early days of text mining involved simple word searches for a key word in some text. For instance, suppose we noticed that a number of GL claims involved damage to homeowners and we wished to further investigate these claims. Perl could be used to search for the word “homeowner” in the accident description field. The following is the procedure for doing this:

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

Code for this is displayed in the program SearchTarget.pl. Inputting data is covered briefly in Appendix A. Table 2.3 below shows that claims involving homeowners are more than twice as severe as other claims:

**Table 2.3**

<b>Homeowner Claim</b>	<b>Mean Severity</b>
No	2,376.6
Yes	6,221.1

## 2.4 Simple Text Statistics

One of the first things an analyst may want to do when examining text is to produce simple text statistics. For instance, what are the length statistics of words and the most common words in the text? In Perl the length function can be used to evaluate the length of words. A count variable is updated with the count of words of length  $n$ . Table 2.4 presents the length statistics for both the GL Claims data and the survey response data. The program length.pl contains the code for computing the length of words. 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.
```

(Note: in this code the variable \$x holds the word after the line has been parsed.)

Table 2.4

Distribution of Length of Words

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

To compile length statistics one needs to create a list of every word in a database, loop through the database, and tabulate the number of times each word is found. To efficiently create a list of words, a hash structure is used in Perl. 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 (Hoffman 2003, Wikipedia, 2009).

In Perl 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. A counter for each word can be updated by referencing the word directly, rather than trying to locate its position in an array. This eliminates the need to loop through an array to find a specific word before updating the count for the word in a document. Suppose we wanted to update the count for the word “actuary”. This could be accomplished using the code \$count(“actuary”) +=1; In general, to count words using a conventional array we would need to do the following:

- Loop through each response and each word in each response.
- Find all unique words, and number of unique words.
- Assign each word an index  $I$  in an array.
- Create an array to hold count for each word. Assign word from array to index.
- Loop through record of text. For each word in each record:
  - Find the array index  $I$  for the word.
  - Update count stored in counter( $I$ ) by 1.
- Continue until last word on last response is counted.

With hashes the process requires less looping. The procedure is:

- Loop through each record.
- Loop through each word for each record.
- The index for the word in the hash is the word, i.e., the index for the word actuary is ‘actuary’. Update the counter by 1.
- Continue until last word on last response is counted.

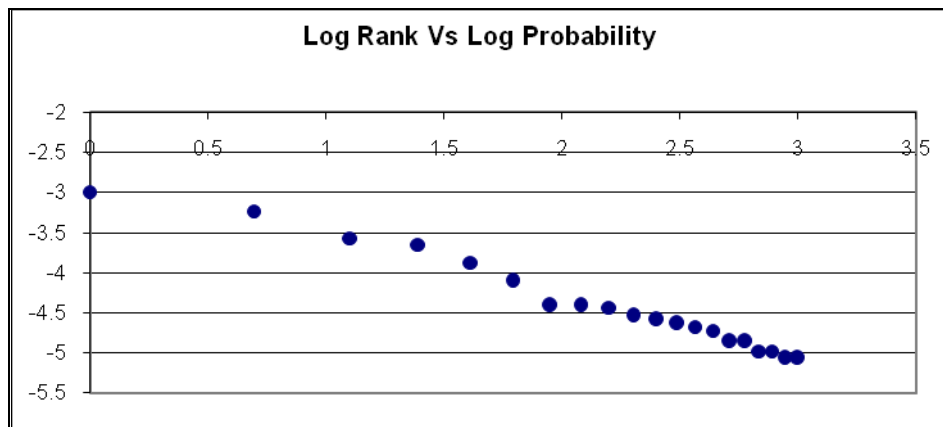
Sample code that was used to tabulate the words in the CAS survey response is shown in Testhash.pl. Table 2.5 displays the frequency of each word in the GL data and in the survey response. Figure 2.3 shows that an approximate linear relationship holds between the log ranks and the log probabilities for the survey data, as would be expected under Zipf’s law.<sup>7</sup>

---

<sup>7</sup> Zipf’s Law provides a probability distribution for the frequency of words in text. It is like a discrete version of the Pareto distribution. A feature of Zipf’s law is that a plot of the frequency of words versus the rank of the word on a log scale will be approximately linear. Perl can be used to tabulate the frequencies of words in a document or database to see if they follow Zipf’s law.

**Table 2.5**

<b>Word Frequencies for Survey Data</b>				
<b>Rank</b>	<b>Word</b>	<b>Count</b>	<b>P(Rank=k)</b>	
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	



**Figure 2.3**

## 2.5 Term Data and Stop Words

In order to perform the statistical procedures that analyze text for content, it is necessary to set up a database that contains information about the “terms” in each response. Under this approach,<sup>8</sup> a “bag of words” is used to extract the important content from text. This means that the order of the words in the text is discarded and only the presence/absence of particular words is recorded. In

<sup>8</sup> Note that other approaches based on linguistics take sentence structure into account

its simplest form, the terms are the complete word collection on a particular record, but later we show adjustments that are made to reduce the number of terms. One could set up a database, where each row is a record and each column is a word in the text on the record. A simple database that can be constructed from text information is a term frequency database, where the frequency of each word in each record is recorded. An example of such a “term-document” matrix is shown below.

**Table 2.6 Example of Term-Document Matrix – Survey Data**

<b>Ourselfs</b>	<b>cas</b>	<b>Not</b>	<b>That</b>	<b>communicators/executive</b>	<b>our</b>	<b>approaches</b>
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

To produce the matrix of term indicators it is necessary to 1) create a list of all words in the data base (which will be referred to as the grand dictionary), 2) check each record (either claim description or survey response) for the presence of each word, 3) create an indicator value of 1 if the word is present, otherwise a zero is recorded and 4) print the results to a file for further processing.

The code in TermDocMatrix.pl is used to perform this task. In this example, the use of hashes improves the efficiency of programming the term-document matrix.

Another preprocessing task involves the elimination of “stop” words. Stop words are very common words like the articles “the” and “a” that do not typically add to the informational content of a text string and are therefore are eliminated. Note that certain types of natural language processing techniques that evaluate grammatical structures may utilize such words but the simple statistical approaches we are illustrating eliminate these words.

A simple approach to the elimination of stop words is to use the substitution operator. Thus to eliminate the word “the”, use the code

```
s/the//g,
```

which substitutes the null character for the word being eliminated. This is illustrated in the program “StopWords.pl” where separate substitutions are performed for the most common stop words

found in our text data. A more comprehensive approach uses a list of stop words from an external file and eliminates each word in the list. A further task that is performed in text mining is “stemming.” When stemming a word, the “stem” or main root of the word replaces the numerous grammatical permutations of the word, such as plural versus singular, past and futures tenses, etc. As part of the process of stemming, synonyms are generally replaced with a single term. For instance, in the survey data, some respondents use the abbreviation ERM when referring to enterprise risk management. Since our two example databases are relatively small, we simply find the multiple versions of the same term and use the substitute operator to substitute a single term for the multiple versions. Many text mining programs reference a database of root terms and synonyms as part of the stemming process. We will illustrate the use of one such implementation in the section on text mining in R. It should be noted that most repositories of synonyms used for stemming will not contain some of the relationships that appear in a given unique database. This is particularly true when misspellings are considered. Thus the stemming part of preprocessing must necessarily be tailored to a given application and code may need to be manually developed to complete the task.

## 2.6 Search Techniques

One of the more common interactions that Internet users have with text mining is search technology. Though information retrieval (IR) technology is not a focus of this paper, a brief illustration of a simple search process will be provided. The search will utilize a similarity measure to compare a short text string and rank a database of text records with respect to how closely they match the search string. Suppose we are comparing the phrase “Credibility of the Profession” to the phrase “Credibility of the CAS,” Table 2.7 shows the term-document matrix for these two expressions. One measure of similarity between the two rows is correlation, which can be measured by the Pearson product moment correlation. The correlation computed between the two rows below is (-.25), even though three out of the five words overlap. This is a consequence of the correlation coefficient measuring the strength of linear relationship, while the relationship between the two rows below is non-linear, as a result of the (1,0) and (0,1) cells.

**Table 2.7**

Credibility	of	The	profession	CAS
1	1	1	1	0
1	1	1	0	1



The text mining literature frequently uses the “cosine” measure as the similarity statistic. Its formula is:

$$(2.1) \quad \cos(\theta) = \frac{\mathbf{A} * \mathbf{B}}{\|\mathbf{A}\| * \|\mathbf{B}\|}, \mathbf{A}, \mathbf{B} = \text{word frequencies}^9$$

It is customary to replace the simple word frequency with a weighted frequency before computing cosine and other statistics. The weighted frequency statistic is the TF-IDF (term frequency – inverse document frequency) statistic.

This statistic computes a weight for each term that reflects its importance. This statistic is more relevant when IR technology is applied to academic papers or Web pages where the “records” is an entire paper or the contents of one Web site. For such “records,” many terms appear multiple times, whereas most terms in our two examples appear at most once in a given record. The relevance of the term to a particular document will depend on how many words are in the document. This is why the denominator of the TF-IDF is adjusted for the number of words in the document. It is also adjusted for the number of records (or documents) containing the word (i.e., terms that appear on many records are down-weighted).

The TF-IDF formula is:

$$(2) \quad \text{TF-IDF}(i) = \text{Frequency}(i) * N / \text{df}(i),$$

where df is the frequency of word (i) in all documents  
N is the number of words in the record/document

The TF-IDF can be computed for a phrase and for a database of records or documents. Then, the records can be ranked. The program matchline.pl in Appendix E searches a database (in this case the 10 line database in Top10.txt accompanying this paper). It finds the record most similar to the input phrase “Credibility of the CAS”, using the cosine, measure, though other similarity measures could be used instead. The procedure used is:

- Read the database.
- Create a hash of all words on each record.

---

<sup>9</sup> The norm of a vector in the denominator of the formula is the square root of the sum of the squares of the elements. It appears that the cosine differs from the more familiar correlation coefficient in not centering the variables by subtracting their means.

- Create a hash of all words in the database.
- Compute the TF-IDF statistic for each term on each record of the database.
- Read the search string.
- Compute the TF-IDF for the search string.
- Compute the cosine correlation between the TF-IDF of the search string and each record in the database.
- Determine which record is the closest match and print it out.

## **2.7 Next Steps: Statistical Analysis to Derive Content**

To derive content from text, techniques referred to as unsupervised learning are used. With unsupervised learning techniques, the statistical analysis procedure has no dependent variable to fit a model to. Instead the values of the variables in the data are used to group like records together. For instance, in the GL claims data, all records with the words indicating a vehicle accident might be grouped together. In the survey response data, all records invoking the word “credibility” might be grouped together. All responses using “ERM” or “Enterprise Risk Management” might be grouped together, but in a separate group from those with the word “credibility”. A procedure referred to as clustering is used to perform the statistical analysis of the term-document matrix to group similar records. Bilisoly (2008) illustrates using the data from the Perl preprocessing (such as the term-document matrix) within an R program to perform the required statistical analysis. Bilisoly (2008), Francis (2006) and Weiss et al. (2005) provide a more detailed description of how to apply clustering and other unsupervised techniques to text mining applications. The next section of this paper will introduce R and its use in text mining. Though Perl can be used to preprocess the data and perform simple text analytics, we will introduce the R functions that read in and preprocess data as well as the functions that perform the statistical analysis of the data that is required for content analysis.

## **3 The Software – R and the GL Database**

### **3.1 –Introduction to R**

One of the most popular open source analytical languages is R. This language is widely used to perform statistical and data mining analyses. R can be downloaded and installed on one’s computer from the CRAN Web Site: <http://cran.r-project.org/>. Figure 3.1 displays the screen for the home page of the Web site. By clicking on your operating system: Linux, Mac OS X, or Windows, under “Download and Install R” on the top center, you can download and install R. While free documentation for using R is available on the R Web Site, there are many books that the new user might enjoy and benefit from. We recommend *Introductory Statistics with R* by Peter Daalgaard, *Modern Applied Statistics with S* by Venables and Ripley, *Data Analysis and Graphics Using R* by Maindonald and

Braun, and *Statistics, an Introduction using R* by Crawley.

The CRAN Web Site includes a wealth of material for learning R including FAQs, tutorials, introductory papers.

Of particular note is the link on the left side of the main page to Contributed Packages or Libraries of specialized functions. We will need to download and install the main text mining package **tm** as well as a number of helper packages. One of our motivations for introducing the R text mining procedures in this paper is that they can be confusing to figure out and produce useful results from, in the absence of guidance from those who have already struggled with these issues.

### 3.2 The **tm** package

R is a statistical computing environment best known for its ability to analyze structured data. The key package that makes analysis of unstructured data available to the R user that will be used throughout this section is the text mining package **tm**. Feinerer et al. (2008) provides an introduction to this tool. However, the authors of this paper experienced minor difficulties in using **tm** based solely on Feinerer et al. and the **tm** help documentation. Note that while this paper was in progress, the **tm** package was updated and previous code had to be changed to conform to the current version. One objective of this paper is to make the process easier for others new to the R package **tm**.

One can install packages quite conveniently in R via the /Packages/Install Packages menu command. Currently, the CRAN package repository features over 1,800 packages! Several additional packages might be of interest to CAS members, including **actuar** by Vincent Goulet.<sup>10</sup>

---

<sup>10</sup> <http://www.actuar-project.org/>

Figure 3.1

CRAN – Comprehensive R Archive Network



Once required packages are downloaded and installed, we first will access the libraries of functions. Note that several of the packages used for specialized graphics in this paper cannot be found on the CRAN Web Site. For these, we provide documentation in our accompanying R programs about where to download these packages.<sup>11</sup> Once they are downloaded, the user must use the “Install Packages from Local Zip Files” option of the Packages menu. While R has some skeletal GUI functionality, the R programming language is the primary vehicle for using R. The R command line begins with the symbol “>”. Examples of R code presented in this section will begin with a line that starts with “>” and use a different font.

The “library” command loads the bundled functions of the desired library. Help describing those functions is a short command away.

```
>library(tm) #loads library  
>help(package=tm)
```

The ability to replicate the examples in this paper is dependent on the user installing all packages that we reference in the text and code. For example, checking the results of the “help” command

<sup>11</sup> We use the Tinnr editor, but our code can be read in Word.

above, the `tm` package “Suggests: filehash, proxy, Rgraphviz, Rmpi, RWeka, snow, Snowball, XML”, that is, `tm` depends on a number of additional packages for some of its functionality. Moreover, some of the packages are updated periodically; therefore the R user should occasionally check for and, if necessary, update using the update packages feature of the R packages menu. In addition, it is recommended that one install version 2.9.2 or later of R, as many of our illustrations do not work on earlier versions.

A number of text data sources are available for use within the R text mining package. One or more of the text sources is used to illustrate the application of data in some of the papers cited in the reference section (Weiss, 2005 Bilisoly 2008). In addition the documentation for the `tm` packages utilizes some of these databases (Feinerer et al., 2008). For an up-to-date list of available data sources and readers available within the `tm` package use the following command in R:

```
> getSources()
[1] "DataframeSource" "DirSource"      "GmaneSource"   "ReutersSource"
[5] "URISource"      "VectorSource"
```

Before the analyst can analyze data, whether structured data, such as a pricing database, or unstructured data, such as accident descriptions or responses to survey questions, the data must be read into R. The most common function used to read in data is the `read.table` function, which reads data that is tab-, comma-, or space-delimited, or uses a delimiter specified by the programmer. (It defaults to the space/tab delimiter). However, since data is commonly found in other formats than the classical text or ASCII files a variety of other readers are now available.

```
> getReaders()
[1] "readDOC"          "readGmane"
[3] "readHTML"        "readNewsgroup"
[5] "readPDF"          "readReut21578XML"
[7] "readReut21578XMLasPlain" "readPlain"
[9] "readRCV1"         "readTabular"
```

One can utilize one of the available readers to read in our source data (which is in the form of a single ASCII file) with a single column of text data, laid out in one logical record per row. (Note the forward slashes in the file name). The `read.csv` function (which reads in comma-delimited data that was saved in csv format in Excel) is used for our data. Note that we want each row of data to be

read in its entirety as a single string, and if spaces are used as a delimiter, each word or string of characters not separated by a space will be read in as a separate variable value.

```
>txt.csv <- read.csv("c:/temp/GLAccDesc.csv"); # or  
>txt.csv<-read.csv("c:/temp/Top2.csv" to read in as tab delimited)
```

In order to apply many of the capabilities of R, it is next necessary to convert the file that was read in to the format that the **tm** function requires. To do this the file must be converted to a corpus. A corpus, as understood in natural language processing, is a body or collection of text information (Manning and Schutze, 2003). Wikipedia defines a corpus as “a large and structured set of texts (now usually electronically stored and processed). [Corpus] are used to do statistical analysis and hypothesis testing, checking occurrences or validating linguistic rules and a specific universe.” (Wikipedia, 2009). For instance a collection of Edgar Allen Poe’s short stories can be considered a corpus. In fact, the Poe corpus is utilized by Bilisoly (2008) in many of the text mining examples in his book. Another corpus might be a collection of news articles written and disseminated by Reuters. Within each corpus, each article or story is treated as a separate record by most text mining programs. Thus each record of this paper’s claims and survey data is treated the same way as an entire Poe story by the software. We can define a Corpus as follows:

```
>txt <- Corpus(DataframeSource(txt.csv)),
```

which converts the text read by the read.csv function into a corpus. We can then summarize and inspect that collection. The summary function provides some basic information about the text data. For the 1617 rows of the GL data corpus, the summary function code and output are:

```
>summary(txt)  
  
A corpus with 1617 text documents  
  
The metadata consists of 2 tag-value pairs and a data frame  
  
Available tags are:  
  
  create_date creator  
  
Available variables in the data frame are:  
  
  MetaID
```

Metadata is data about data, i.e., a description of the types of variables, their functions, permissible values, etc. Certain types of data formats, such as html and xml contain tags and other

data structures that provide some metadata. For our simple text file there is very little descriptive information about the data in the file, hence the relatively brief and uninformative output of the summary function. The inspect function allows the text miner to print out and inspect a subset of the data. It is a good practice to inspect a sample of the data, both to detect data quality issues and to become familiar with the contents of the data. Below we show the code and output of two records from the GL claims data.

```
>inspect(txt[1:2])
```

```
[[1]] Crew was hooking a jumper pipe between fire line and water line in order to perform a flush test. When the line was charged the jumper pipe slipped off causing water to spray on newly hung dry wall. Damage to one piece of dry wall and few pieces of l...
```

```
[[2]] While exposing Verizon line - employee used hands to pull line from trench causing it to tear. ...
```

As described in the Perl section of this paper, it is necessary to perform some preprocessing to prep the data for text mining. Example tasks include converting text to lower case, removing stop words, stemming and identifying synonyms. To see all available text transformations currently available within **tm** use:

```
>getTransformations()
```

Some applications of available transformations (output) are:

```
>txt <- tm_map(txt, tolower)
```

```
>txt <- gsub(Old Pattern, New Pattern, txt)for (j in 1:length(txt)) txt[[j]] <- gsub("[&|_|/\\|()|\\.|", " ", txt[[j]]);
```

```
# Replace enterprise risk management
```

```
>for (j in 1:length(txt)) txt[[j]] <- gsub("enterprise risk management", "erm",txt[[j]])
```

```
>for (j in 1:length(txt)) txt[[j]] <- gsub("off shoring", "offshoring", txt[[j]]);
```

```
# remove stopwords
```

```
>txt <- tm_map(txt, removeWords, stopwords("english"))
```

```
>txt <- tm_map(txt, removeNumbers)
```

```
>txt <- tm_map(txt, removePunctuation)
```

The transformation function **tm\_map** can convert text to lowercase. That is, `tm_map(txt, tolower)` functions the same way as the Perl function **lc**. With Perl, the text data was preprocessed with the substitution operator to remove unwanted punctuation and patterns. With R, it is easiest to apply to transformations sequentially. To remove unwanted characters in the text data apply the **gsub** transformation. For instance, the following replaces the “/” separating some of the words in the survey data with a space.

```
>for (j in 1:length(txt)) txt[[j]] <- gsub("/", " ",txt[[j]])
```

Then apply the remove punctuation transformation. Without replacing the slash first, two words will be combined into one when punctuation is removed.

Next, remove the stopwords. The **tm** library comes with a list of stopwords in a number of different languages including English. As alternative to using the list, the user can supply a list of stopwords, i.e.:

```
>newstopwords <-c("and", "for", "the", "to", "in", "when", "then", "he", "she", "than")
```

```
> txt <- tm_map(txt, removeWords, newstopwords)
```

The **tm** package stemming function uses an algorithm that removes common word endings for English words, such as “es”, “ed” and “s”.

The output from the transformation code above is:

```
>inspect(txt[1:3])
```

```
[[1]] crew hook jumper pipe fire line water line perform flush test line charg jumper pipe slip caus  
water spray newli hung dri wall damag piec dri wall piec
```

```
[[2]] expos verizon line - employe hand pull line trench caus tear
```

```
[[3]] cat 345 travel guid wire boom caught wire wire taut - caus power pole break wire snap
```



When comparing this to the lines of unprocessed output on the last page it is apparent that the ending “ing” was removed from “hooking” and “charged” was replaced with “charg”, etc.

The next set of tasks for text mining includes creating a Document by Term matrix (or DTM), identifying frequently occurring words, and removing sparse terms.

```
>dtm <- DocumentTermMatrix(txt)
```

Produces a matrix such as:

**Table 3.1**

**a. Subset of Term-Document Matrix – GL Claim Data**

	alleg	caus	damag	Travel	truck
	0	1	1	0	0
	0	1	0	0	0
	0	1	0	1	0
	0	1	0	1	0
	0	0	1	0	0

**b. Subset of Term-Document Matrix – Survey Data**

	abil	account	actuari	Topic	Valu
	1	0	0	0	0
	0	0	0	0	0
	1	0	0	0	0
	1	0	0	0	1
	1	0	0	0	0

Document by term matrices are generally quite sparse – they contain lots of zeros. As illustrated even in the tiny example above, many terms appear in only a few records. The function `removeSparseTerms` removes terms that occur infrequently in a database. Terms which occur only once or twice are likely to consume a lot of computational resources without adding anything useful to the analysis. We can greatly reduce the size of the dtm without losing much useful information. The following function:

```
>dtm3 <- removeSparseTerms(dtm, 0.94)
```

reduces the number of terms from over 2,200 to about 280 in the GL data. The sparsity parameter of 0.94 says to remove all terms from the dtm with zeros for the term count in 94% of the documents. When the remove sparse terms function is applied to the survey data, the number

of terms is reduced from approximately 400 to approximately 250.

Note that at this point the remainder of the text mining is the same whether the term-document matrix was created in Perl or in R. Thus if there was a document term matrix created in Perl named “dtm.csv”, it would be necessary to use the read function to read it in.

```
>dtm.csv<-read.csv("c:/Directory/dtm.csv")
```

Now that the text data has been preprocessed into a more convenient and compact form we can begin to investigate relationships with simple statistics. In section 2, we illustrated how to compute word frequencies in Perl using hashes. In the R **tm** package, there are ready-made functions to perform simple statistical analyses of the term data.

In section 2 we illustrated how to compute correlations between a phrase and the records in a text database. A simple exploratory text mining statistic to compute is the correlation between a single term and all the other terms in a database. If the two terms always occur together, i.e., if both terms are always present or absent together, the correlation will be 1.0. If the two terms never occur together the correlation will be 0.0. Thus correlation is an indicator of how closely related two terms are. The R **tm** package has a function that finds terms that are related to a word of interest to the analyst. We can identify terms that are correlated with a target variable, in this example “travel”. The following function:

```
>findAssocs(dtm2, " travel ", 0.15)
```

finds all words with a correlation of at least 0.15 with the target term. In general, the terms correlated with “travel” appear to describe vehicular accidents.

**Table 3.2 Words Correlated With The Word “Travel”**

Term	Correlation
Travel	1.00
vehicl	0.34
windshield	0.28
north	0.27
south	0.27
Tire	0.24
lane	0.20
hit	0.19
onto	0.19
flew	0.18
near	0.18
rte	0.18
claimant	0.17
debri	0.15
east	0.15
front	0.15

The correlation measure is a “similarity” measure. That is, it measures how closely related two variables are. It is one of many measures of similarity. Other similarity measures are the Chi-Squared statistic, which measures how closely related two categorical variables are, and phi, which measures the correlation between binary categorical variables. Similarity measures can be applied across rows as well as across columns of a database. When applied across rows the similarity indicates how similar two records are. In the case of text mining, a similarity measure would indicate how many words the two rows have in common.

Complimentary statistics to the similarity measure are dissimilarity measures. Dissimilarity statistics measure how far apart or dissimilar two columns or rows of a database are. A common dissimilarity measure is the Euclidian distance measure, which measures the squared distance between each variable (i.e., for our text data each term) for record  $i$  and each term for record  $j$ .

The Euclidian Distance Formula:

$$d_{i,j} = \left( \sum_{k=1}^m (x_{i,k} - x_{j,k})^2 \right)^{1/2} \quad i, j = \text{records}, m = \text{number of variables.} \quad (3.1)$$

A similar measure, which used absolute value differences rather than the squared difference is the taxicab or Manhattan distance. Kaufman (1990) and Francis (2006) provide a more comprehensive

background on the common dissimilarity measures.

The R proxy library can be used to define similarity and distance measures. Note that although the library uses the dissimilarity function, this function measures dissimilarity and similarity, depending on the measure used. Below is an example of code for computing the cosines between the rows of the GL data.

```
>library(proxy)
>dissimilarity(dtm3, method = "cosine")[1:10]
```

### 3.3 Clustering for content analysis

The dissimilarity measure is a key input into a statistical procedure that is commonly used in data mining to create content. The procedure is clustering. Clustering separates records into groups that are similar with respect to the terms contained in each record. The algorithm attempts to maximize the dissimilarity between the groups and minimize the dissimilarity within groups, hence it relies heavily on a dissimilarity measure to perform the grouping. Sanche (2006) describes how clustering can be used for variable reduction. Clustering is used to reduce a large number of variables to a smaller set that maintains the predictive information of the larger set. The clustering procedure finds groups of variables that are strongly related to each other, that is, they are highly correlated and tend to convey similar information. After clustering, the group is replaced by a single variable. For instance, in Sanche's example, the two variables, population density and car density, were replaced by one variable encoding the density information. A similar concept applies in text mining: where multiple words refer to the same concept, and therefore can be grouped together, as one variable representing the concept.

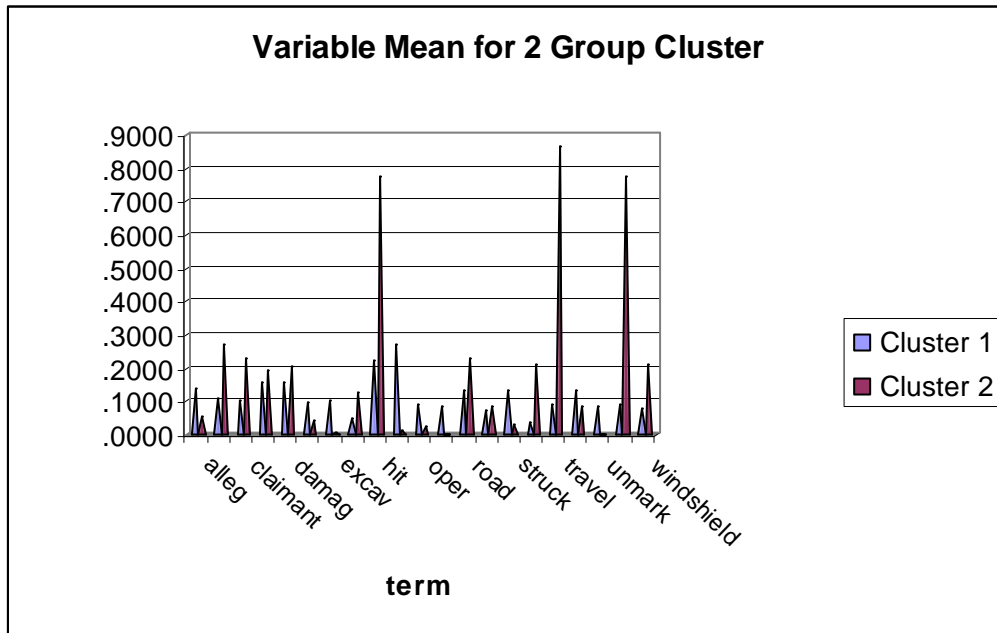
The R stats library contains functions that can be used to identify groups of related terms via the key clustering methods, hierarchal and k-means clustering. Load the stats library, then run the clustering procedure. Only one of the two procedures, k-means clustering, will be applied to the GL data. The hierarchical technique will be illustrated on the survey data in Section 4. To use either procedure, first load the stats library, then run the clustering procedure. The function for k-means clustering is **kmeans**.

The analyst may specify a distance measure or use the default Euclidian measure. To apply k-means to a document-term matrix (dtm3) use the following:

```
>library(stats)  
>glKmeans <- kmeans(dtm3, 5)
```

K-means clustering “which aims to partition the points into k groups such that the sum of squares from points to the assigned cluster centers is minimized.” (R Documentation for the “stats” package). The procedure is used to break out the term-document matrix into subsets based on clusters. With k-means clustering the user must specify “k”, the number of clusters, as well as the distance measure. Francis (2006) discusses several approaches to determining the number of groups to use. We used the k-means procedure to develop several different-sized groupings. With the GL (as opposed to the survey) data, our ultimate objective is to extract information from the accident description that can be used in predictive modeling. First we note that the text miner usually attempts to understand the groupings by examining descriptive statistics, especially the average number of appearances for the terms in each group. Figure 3.2 displays the statistics for the two cluster grouping.

**Figure 3.2 Average # Times Term Is Used for Two Clusters**



From Figure 3.2, it is clear that the claims in cluster 2 tend to have a higher frequency of the words “hit” and “travel” (as well as vehicle) and thus appear to have a high representation of car

accidents (suggesting two groupings of vehicle accidents versus everything else). While not shown here, other cluster groupings display a different, but typically informative set of descriptive statistics.

### 3.4 Use of clusters in prediction

The clusters computed by the clustering procedure can be incorporated into modeling databases and used to predict a variable of interest. The decision tree procedure is a popular data mining procedure that can be implemented in R to predict a dependent variable of interest. Brieman et al. (1993), DeVile (2006), and Derrig and Francis (2008) provide detailed introductions to the tree modeling technique. Two reasons for the popularity of decision tree techniques are (1) the procedures are relatively straightforward to understand and explain and (2) the procedures address a number of data complexities, such as nonlinearities and interactions that commonly occur in real data.

In the GL dataset used for this paper, besides the text data, we have two target variables of interest: an indicator of attorney involvement in the claim and the amount of the claim or its severity. An alternative that we will not explore in this paper is to directly apply the tree models to cluster the terms. A good paper discussing this topic is Himmel, Reincke, and Michelmann (2008).

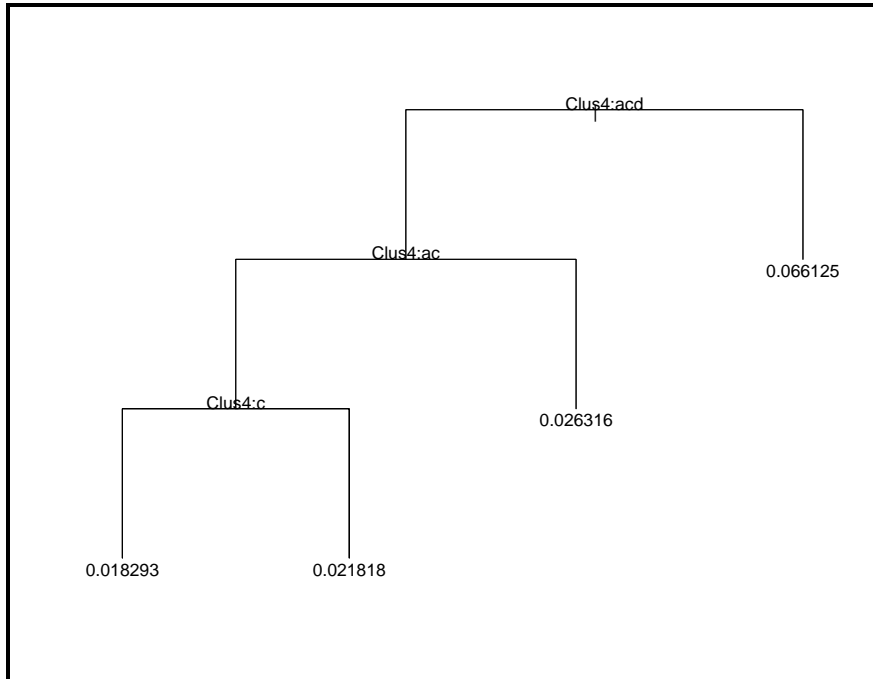
We can fit a CART-style tree model with the **rpart** library. First we remerge the attorney and severity columns back with the newly formed document-term matrix and then fit a tree model using recursive partitioning.

```
>GLAccDesc.matrix <- as.matrix(dtm3)
>attorney <- txt.csv$attorney
>severity <- txt.csv$TrendSev
>GLAccDesc<- data.frame(GLAccDesc.matrix,attorney)
>GLAccDesc.rpart <- rpart(attorney ~ ClusterGroup12) #fit tree using clusters as predictors
```

---

<sup>12</sup> ClusterGroup denotes one of the new variables (cluster groupings) created from the clustering procedure.

**Figure 3.3**  
**RPART tree for attorney=0/1. Four Group Cluster is Used for Prediction**



A convenient feature for the `rpart` procedure is that one can export the resulting tree model to `pmml` (perhaps for import into SAS EMiner™ or another environment for scoring.)

```
>library(pmml)
>rpart.pmml <- pmml(GLAccDesc.rpart)
>saveXML(pmml(GLAccDesc.rpart), file="C:/temp/rpart.xml")
```

The tree in Figure 3.3 graphically summarizes how the data was partitioned into groups, and the terminal branch or terminal node displays the predicted value for each partition.<sup>13</sup> Note that the tree procedure uses all four clusters (although alternative structures could have been found) in predicting attorney involvement and that there are significant differences between the groups in the propensity for attorney involvement. The tree summary (created using the `summary` function) indicated that the lowest probability of attorney involvement is associated with cluster 3 and highest probability is associated with cluster 2. Table 3 displays the top terms associated with each cluster based on descriptive output. This table indicates that words indicating a vehicular accident are in cluster 3

---

<sup>13</sup> To know the specific values of the cluster variable used for partitioning the data the `summary` function must be used.

while cluster 2 appears to be an “all other” category.

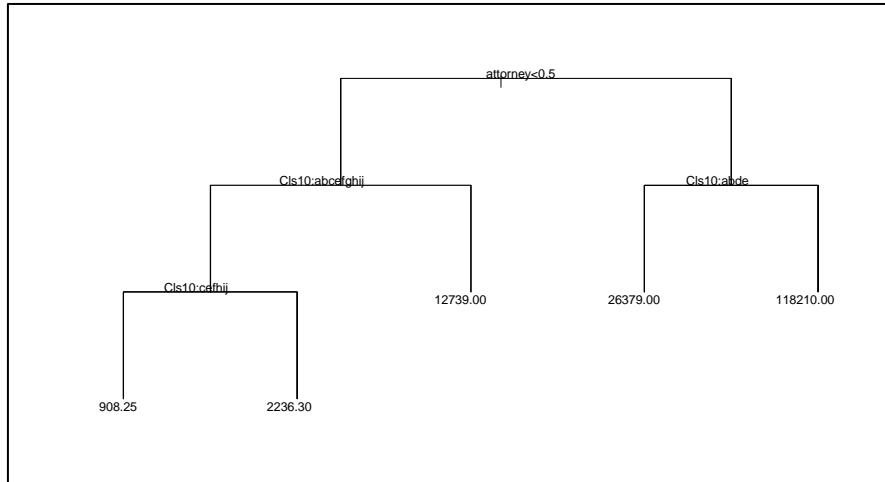
**Table 3. Cluster Content, 4 Group Cluster**

Cluster	Key Words
1	excavate, operate, line
2	all other (moderate representation of many words)
3	hit, travel vehicle
4	allege

Next tree analysis is used to model claim severity. The tree in Figure 3.7 utilized the two, four, seven, and 10 group clusters computed with the k-means procedure, along with attorney involvement as potential predictors. The ten group cluster (but not the other clusters variables) is used in the fitted model to predict severity. This would suggest that the different-sized clusters extracted different content from the text data and that the 10 group cluster was the most predictive of claim severity. Note that the average severity for each terminal node varies from a low of \$908 to a high of \$26,379. The tree summary indicates the lowest severities are associated with no attorney and cluster 4 (of 10). Cluster 4 has a relatively low average content of most of the words, though it loaded higher on operator than other terms. After inspecting output we found that most of the other terminal nodes contained multiple clusters, with the lowest severities in the “no attorney” branches.



Figure 3.7 Severity Tree Using Several Cluster Variables as Predictors



## 4 Using R to Text Mine the 2008 CAS Quinquennial Membership Survey

Braithwaite (2009) provides a short overview of the survey results in the August 2009 *Actuarial Review*. The full report of the survey task force, Braithwaite et al. (2009) is available in the Summer 2009 *CAS E-Forum*. Survey responders provided extensive written comments. Now we will apply our text-mining R skills to summarizing the free-form text responses to the question, “What are the top two issues affecting actuaries and the CAS in the next five years?”

### 4.1 Preprocessing the Survey Data

First, we’ll read in the text and create a document collection, or Corpus.

```

> txt.csv <- read.csv(file="C:/temp/Top 2 Issues.txt", header=FALSE)
> txt <- Corpus(DataframeSource(txt.csv), dbControl=list(useDb=TRUE, dbName="txtcsv",
dbType="DB1"))
> inspect(txt[1:2])
[[1]] A crisis that could affect our ability to regulate ourselves.
[[2]] A need to deal more thoroughly with non-traditional risk management approaches
  
```

Next, we’ll apply some transformations, converting to lower case and removing stopwords in our 336 sample documents. Notice from the output above that we’ll have to do something about

removing punctuation from our survey written comments. The text is particularly free-form, including punctuation, etc., so we'll need to use the **gsub** function to take out slashes, backslashes, dashes, parentheses, periods, etc., to allow the words to stand alone. (Note that in R we must use a double-backslash to escape characters as the backslash is itself a reserved character.)

Next we'll create a document by term matrix, or dtm. The resulting matrix is quite sparse, with the 336 rows representing documents and the 489 columns for each term. We can tighten up our dtm by restricting ourselves to the more frequently occurring terms.

```
>dtm <- DocumentTermMatrix(txt)
```

Note that this matrix has 336 rows and 489 columns, so we want to remove sparse terms as follows:

```
>dtm3 <- removeSparseTerms(dtm, 0.99)
```

```
>nrow(dtm3); ncol(dtm3)
```

```
[1] 336
```

```
[1] 79
```

Before analyzing the data further we can export the term-document matrix (for possible use in other software) using R's write.csv function. It is also helpful for many analyses to convert the matrix into a data frame (an R database).

```
>dtm2<-data.frame(dtm)
```

## 4.2 Statistical Analyses of Survey Data

Next we examine the frequency of terms in the survey data. The **sapply** function is used to sum each column and count the number of responses containing the term. This is then sorted and the top 10 terms are plotted in Figure 4.1:

```
>numwords <- 30
```

```
>v <- as.matrix(sort(sapply(top2, doit),decreasing=TRUE)[1:numwords],  
colnames=count);v[1:numwords]
```

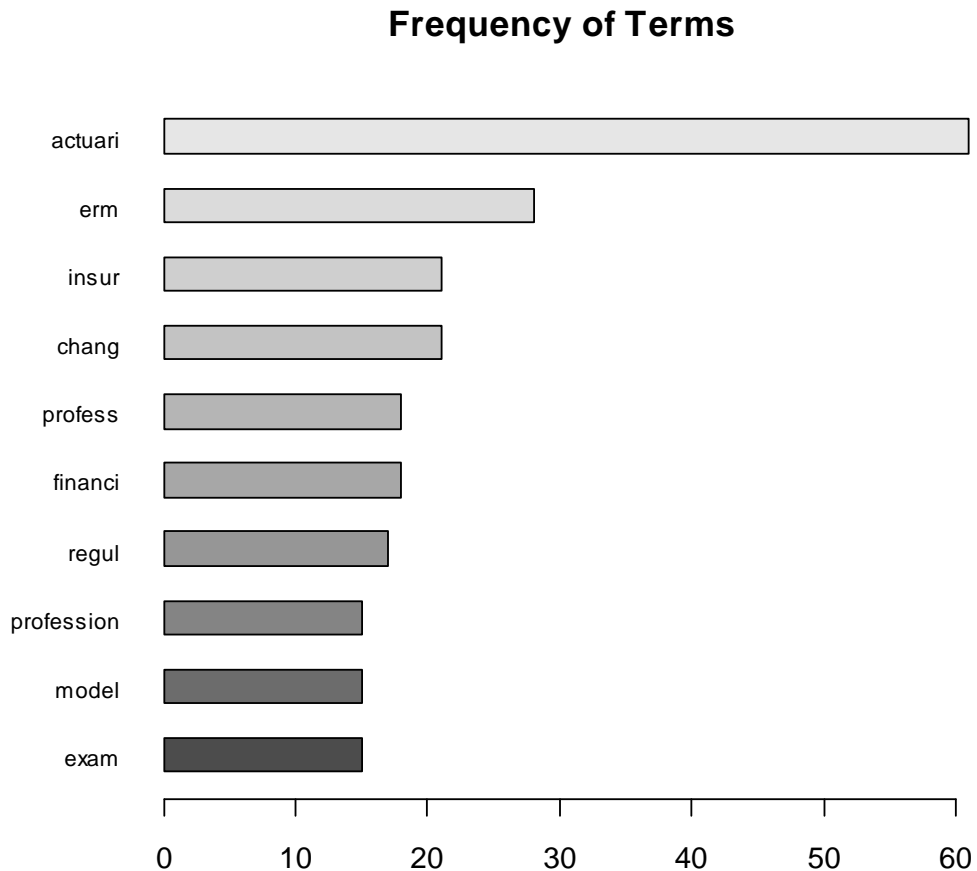
```
>w <- rownames(v); length(w); w
```

```
>require(grDevices); # for colors
```

```
>x <- sort(v[1:10,], decreasing=FALSE)
```

```
>barplot(x, horiz=TRUE, cex.names=0.5, space=1, las=1, col=grey.colors(10), main="Frequency of  
Terms")
```

Figure 4.1



Another way to profile collections is to use a “tag cloud” or a “word cloud” or “weighed list,” a list where the font size or color of the words represents the term frequency. Tag clouds were originally popular on photo collections to index the content of the photos. A second variation uses random placement of the words. A third jazzy variation in the “fun” package creates a spinning 3-D tag cloud for an html page using flash and javascript.

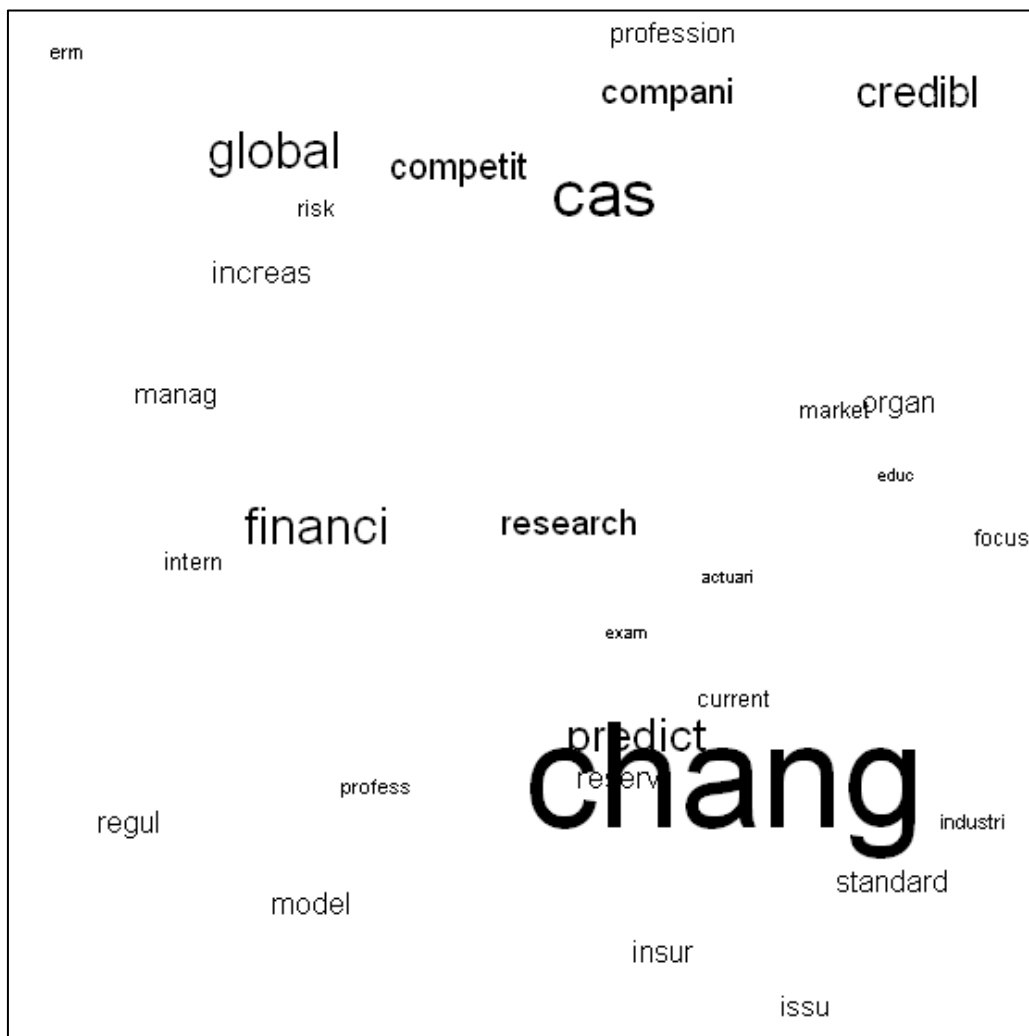
In Figure 4.2, the package “snippets” is used to randomly plot terms, with the font size representing the frequency of the term. A number of themes are revealed by the frequency graphs including change, ERM, regulation, credibility (of the profession), and prediction.

```
>require(snippets)
```

```
>set.seed(221)
```

```
>x=runif(length(w))  
>y=runif(length(w))  
>plot(x, y, type="n", axes=FALSE)
```

Figure 4.2 Cloud Map of Word Frequency in Survey



The most frequently occurring word in the survey response is “actuari”. This is not surprising, as the respondents are actuaries commenting about their profession. It is common in text mining to drill down on some text by extracting “snippets” or some of the text associated with a word of interest. In R, the subset function can be used to find and print the text in responses containing the word actuary.

```
>rownames(subset(top2, actuari>=1)) #which rows have actuari>=1  
  
>txt.actuari <- txt.csv[rownames(subset(top2, actuari>=1)),]  
  
>txt.actuari; #print out records with actuary
```

Table 4.1 displays a sample of the text associated with the word “actuary.” It is clear that a number of different topics are revealed, including professionalism/reputation of actuaries, relevance of actuaries, education, and the financial crisis.

**Table 4.1 Text associated with term “actuary”**

Actuarial Malpractice
Actuarial Malpractice
Actuarial Students of today are not good communicators/executive material.
Actuaries regain status as "masters of risk"
Relevance of actuaries as providers of solutions for various business problems.
Relevance to the practicing actuary
Relevance to the practicing actuary
Reputation issues arising from current financial crises
reputation of actuarial profession and professionals
Rise of other International actuarial organizations
The current financial crisis - actuaries need to protect their role in insurance and risk management
Training and educating actuaries

### 4.3 Unsupervised Methods for Content Analysis

We applied the k-means clustering to the GL claims data. In this section we will illustrate another clustering technique, hierarchical clustering. We will use the technique to identify common themes in the survey responses. In this illustration, hierarchical clustering will be applied to the terms or columns, rather than the records (responses). Note that the transpose function (for instance  $\mathbf{x}' \leftarrow t(\text{term\_document\_matrix})$ ) must be applied to the data before clustering. Hierarchical clustering is a recursive procedure that sequentially groups terms into larger and larger groups, by combining together terms with the smallest dissimilarity. This will group words that tend to occur

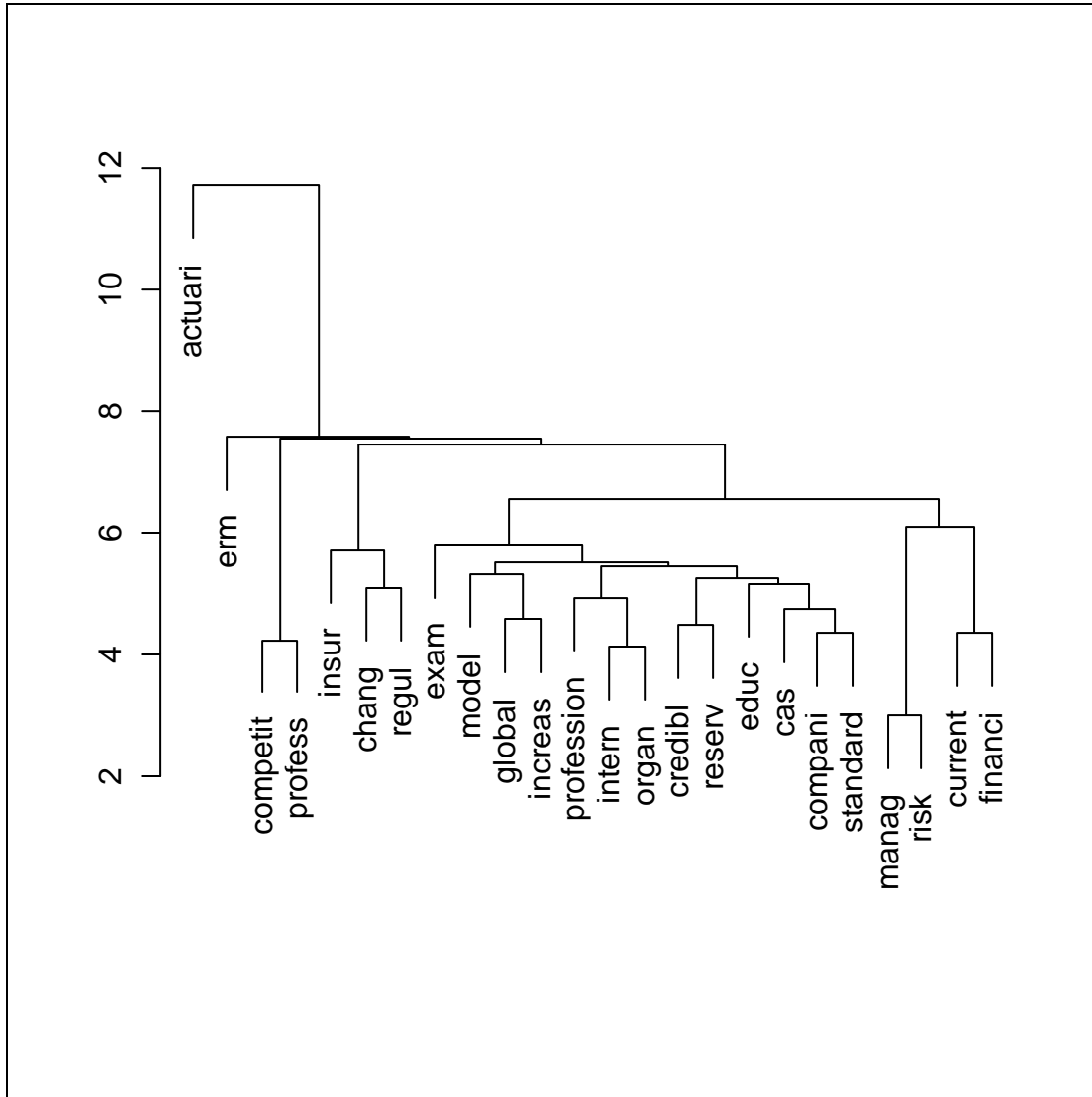
together into the same groups. Hierarchical clustering produces a graph that looks like a tree. Each level of the tree displays a step in the sequence of combining terms into larger(smaller) groupings. The largest group containing all the data is displayed at the top of the graph. At the bottom of the tree, each term is its own group.

We note that change and regulation (changes in regulation) group together as do current and research (research current topics). It appears that the term “actuari” seems to separate early from all other terms. In addition the term “increas” is associated with both education and research. We also note that “professio” and “profession” occur as separate words. This is an artifact of the stemming function where profession was stemmed to “profess”, while professional was stemmed to “profession”.

The word “intern” (international) is associated with the stemmed professional, while the word “competition” is associated with profession (i.e., competition from other professions). Other topics displayed on the clustering tree are exams, reserves, and modeling. The term “financial” also appeared frequently. Using the **findAssoc** function to find terms that occur frequently with “financial,” we find financial crisis and financial analysis, including dynamic financial analysis are common themes.

The tree graph can guide the user in determining how many groupings to maintain. That is by examining the tree, the analyst can group together terms that logically seem to belong together. Alternatively, the number of groups can be subjectively determined. For instance, if one decides to use only two groups, we might split the data into responses that contains the word “actuary” versus everything else. Unfortunately the two grouping scenario would not be very informative, so a larger number of groups would be needed.

Figure 4.3 Hierarchical Cluster of Survey Words



The next technique we apply to the survey data is principal components. Principal components is an unsupervised method that is used for variable reduction. The original variables (in our case, after removing sparse terms, there are 35 terms remaining) are replaced by a smaller number of variables that capture the essential features of the original variables. For instance, the words “risk” and “manage” frequently occur together, and the two can be replaced by one term or “component” in principal components terminology representing the concept “risk management”. A description of the methodology used in principal components techniques is outside the scope of this paper (see Kim, 1978 for more information).

Note that the correlation matrix between terms is a key input into the calculation and highly correlated terms tend to be associated with the same factor.

```
>require(stats)
>prcomp(top2, scale=TRUE)
>summary(prcomp(top2, scale=TRUE))
```

The key information for the analysis is the new “components” or variables estimated from the procedure and the “loadings,” or coefficients of each term on the component. Each component is essentially a weighted average of some of the terms, and the terms that have high loadings are the key contributors to the component and can be used in identifying the concept underlying each component. For instance the words “research” and “practical” tend to have a high loading on the first component, suggesting that they represents the concept “practical research”. To identify the concept associated with each factor, it is necessary to access the loading of each term on the top factors (we used output statistics to retain the top 15 components). The loading is part of the principal components object referenced with “PrincipalComponentsObject\$rotation”. The following code finds and prints out the top four terms associated with each factor. These are used to construct Table 4.3, the theme associated with each factor.

```
>.for (i in 1:15) {
  > top4[[i]] <- sort(survey.prcomp$rotation[,i], decreasing=TRUE)[1:4]
}
>top4.
```

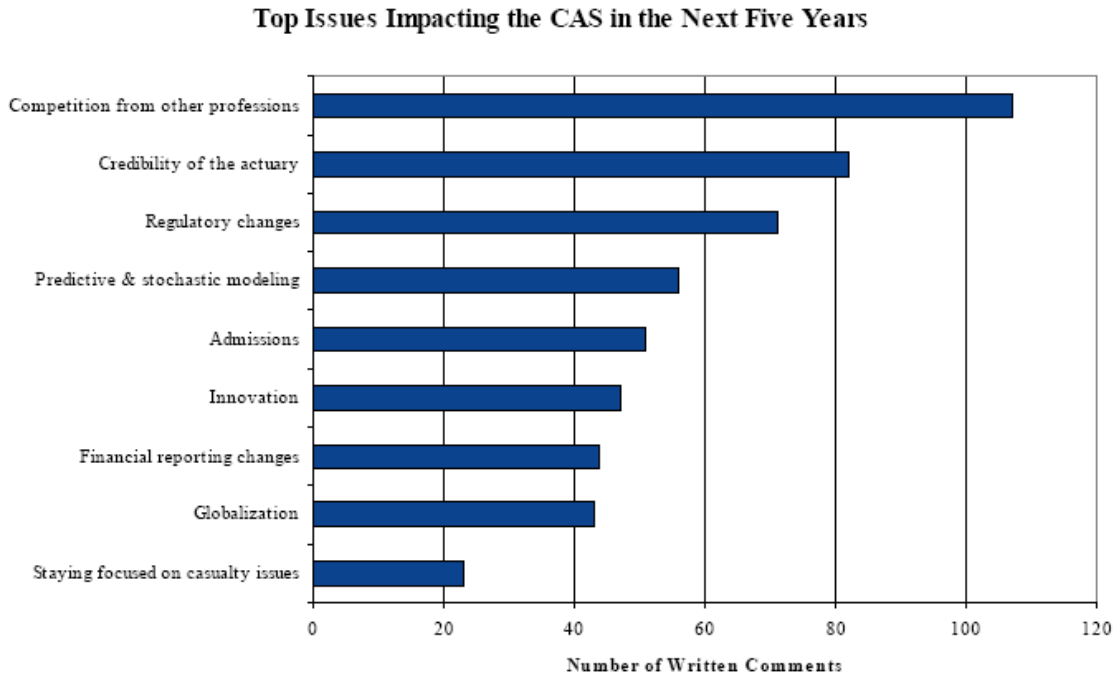


**Table 4.2: Concept Associated With Each Component**

<b>Component</b>	<b>Concept</b>
1	practical research
2	predictive modeling
3	maintain standards
4	demand for actuarial skills
5	risk management
6	exam structure
7	actuarial organization
8	reputation/financial issues
9	economic capital
10	competition - other professions
11	opportunities for actuaries
12	capital
13	predictive modeling
14	globalism/competition
15	reserving and credibility

These concepts show a significant overlap with the top issues identified in the write-up of the survey published by the task force which are displayed in the graph below. In order to summarize the responses, a CAS volunteer examined classified and tabulated responses. While this review provides context that automated procedures does not provide, nonetheless, the text mining procedures were able to identify key themes in the survey response. In addition, text mining uncovered additional themes, such as economic capital and risk management.

**Figure 4.4: Top Issues From Quinquennial Survey Summary**



## 5 Conclusions

In the previous three sections, we provided an introduction to utilizing Perl and R for text mining, including brief examples of code that the programmer needs to successfully use these tools. Text files containing the code used in the paper will be posted with the paper. Using two sample text databases we showed that an analyst can:

- Extract information from an accident description field that can be use to model the propensity for attorneys to be involved in a claim and claim severity
- Analyze survey response data to efficiently identify major themes

A goal of this paper is to make it easier for those interested in text mining to use open source software in their work. This paper is not intended to replace reference books that introduce the two languages (which we encourage the reader to acquire). However, without specific guidance focused on text mining, these tools can be challenging to use. In Table 5.1 we compare the capabilities of the two open source languages for text mining. Table 5.1 presents the procedures we illustrated for each language.

**Table 5.1 Features of Procedures Illustrated**

<b>Task</b>	<b>Perl</b>	<b>R</b>
parse data	yes	yes
convert to lower case	yes	yes
eliminate punctuation	yes	yes
eliminate stopwords	yes	yes
stemming	yes	yes
compute frequencies	yes	yes
compute correlations	yes	yes
cluster terms	no	yes
apply multivariate models	no	yes
efficient for large databases	yes	no

## **Acknowledgment**

The authors thank Vicky Tuite for her editorial and debugging assistance and Edward Vandenberg for his editorial assistance.

## **Supplementary Material**

Survey datasets and R code will accompany this paper

## **Appendix A: Perl Procedures with Glossary of Perl Terms**

This section is provided to assist the reader in understanding some of the unique structures and variables used in the Perl language. Additional online help for beginners can be found at <http://www.perlmonks.org/index.pl?node=Tutorials#Getting-Started-with-Perl>. The section on “strings” is particularly helpful with text manipulations, and the section on Pattern Matching, Regular Expressions, and Parsing is a good resource on parsing.

In the text below, Perl code is displayed in italics.

### Input Files

The following code opens a file for printing and prints the data. The directory and file name must be specified. Note the use of the `chomp` function to remove end of line characters.

```
$TheFile="c:\perl\GLACC.txt";  
open (INFILE, $Thefile);  
while <INFILE> {  
  chomp; # eliminate end of line character  
  $Sentence=$_; # read text into new variable, one line at a time  
  print "$Sentence\n"; } # print line
```

Files can also be input from the command line when running a program as in:

```
perl progname.pl <datafile> <outputfile>
```

To open the file supplied at the command line the following code is needed:

```
open(MYDATA, $ARGV[0]) or die("Error: cannot open file '$ARGV[0]'\n");  
# opens data file  
open(OUTP, ">$ARGV[1]") or die("Cannot open file '$ARGV[1]' for writing\n"); # opens  
output file
```

### Printing

Many of the examples used in the paper print directly to the computer screen. The command to do this is “print”. Thus

```
print "hello";  
prints the word hello to the screen.
```

```
print "hello\n";  
prints hello and a carriage return, so that the next printed line will be on a new line.
```

To print to a file, the programmer must first open a file for output. The character > before the file name denotes that the file is opened for output:

```
open(OUTP1, ">OutInd1.txt") or die("Cannot open file for writing\n");  
will open the output file “OUTP1” or print an error message if the file cannot be opened.  
print OUTP1 "hello\n";  
will print the word hello along with a carriage return to the file.
```

When printing to an output file, the command “printf” rather than “print” can also be used.

### Looping

The most common looping command in the code in this paper is the “while” command. Its structure is:

```
while (expression is true) {code to perform}.
```

Thus:

```
while($line = <MYDATA> ){  
code to perform tasks  
}
```

will read each line of data from the file MYDATA and assign each line, one at a time to the string variable \$line. The code within braces will then (usually) perform other tasks with the data, such as print it out. The code will stop executing when the end of the file is reached.

Another common looping structure is “foreach”.

```
foreach $word (@words) {  
code to perform tasks  
}
```

will loop through each word in the array @words and perform the tasks specified in the code within brackets, such as replace unwanted characters or print the word. Again the brackets are used as with the while structure to specify the code that is executed within the loop.

Regular Expressions - In the body of the paper, regular expressions were introduced as a way to pattern match a text string. Below is a guide to special variables and characters that are regularly used in Perl regular expressions:

//

The forward slash is a delimiter used in pattern matching. Within it the characters defining the pattern are specified. Thus split (/ /, \$Test) will split a string expression using the space as a separator.

\

A single backward slash is used as an escape operator. Thus /(\.)/ when used in a regular expression denotes the period. The \ must be used with some special characters, such as the period and the backwards slash itself.

[ ] brackets are used within expressions as parentheses, i.e., to subset. See the next example.

^ denotes not in a regular expression. Thus the code:

([^.]\*\.)

denotes a text string that is not period, such as words or spaces, followed by a period. This could be used to find a regular sentence structure that ends in a period. Note that the \* operator denotes zero or more of the character preceding it (i.e., of non-period characters).

\$\_

is a special variable that holds the current line of text read in from a file by the Perl.

\$1

is a special match variable. It contains all text from the left most parenthesis in the end of the last match. The sample code following the special variable g matches the text up to the first period (and also illustrates the period and g modifier).

.

The period is a wildcard character. If one wanted to match the letter a followed by anything followed by d, one would use /a.d/. For instance, the word “and” would be a match.

g is a modifier that stands for “global” and indicates a need to search the entire record of test. Below is an example using the previous two terms.

```
$Test = "A crisis that could affect our ability to 'regulate'. ourselves." ;# a test string
```

```
while ( $Test =~ /([^\s]+)/g ) {  
  print "$1 \n"; }
```

It prints out:

A crisis that could affect our ability to 'regulate'.

`=~`

is similar the equal operator. It binds a pattern match on the right to a variable on the left.

`\w`

denotes one letter or number. Letters only, including both small and capital, are denoted `/a-zA-Z/`.

`\w(2)`

denotes two alphanumeric characters.

`\w(n)`

denotes *n* alphanumeric characters.

`\w+`

denotes one or more letters or numbers.

`\d`

denotes one digit. It is the same as `[0-9]`.

`\d+`

denotes one or more digits.

`\s+`

denotes one or more spaces.

`\b`

denotes a word boundary, such as the space or tab characters.

`\D`

denotes any non-digit character. It is the same as `[^0-9]`.

`\W`

denotes any nonalphanumeric character.

`split(/Regular expression/,text string or variable)`

is used to split a longer string, such as a sentence or paragraph into its component word. It is common to use the space or one or more spaces (specified as `[\s+]`) as the delimiter. An example is

```
$StringVar="Ability of members to prove they are more than just number crunchers";  
split(/ /, $StringVar)
```

## Appendix B

### Example Code Using Perl Regular Expressions

#### I Simple parsing programs

##### Parse1.pl

```
#!/perl -w  
# Program Parse1.pl  
# This program provides a simple example of parsing  
# It uses the space as the separator in parsing words in the string  
# Set scalar variable to text string  
$Response = "Ability of members to prove they are more than just number crunchers";  
@words =split (/ /, $Response); # parse words and put in array  
foreach $word (@words) {  
  print "$word\n";  
} # print to screen, one word per line from array
```

##### 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";  
}
```

##### 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  
  }  
}
```

##### Simple Search Program

##### SearchTarget.pl



```
$target = "(homeowner)";  
# initialize file variable containing file with text data  
$TheFile = "GLACC1.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";  
}
```

## Appendix C

### Code for Simple Text Statistics

#### I Frequency of Word Lengths

##### Length.pl

```
#!/perl -w  
# Enter file name with text data here  
$TheFile = "Top2Iss.txt";  
# open the file  
open(INFILE, $TheFile) or die "File not found";  
# read in one line at a time  
while(<INFILE>) {  
  chomp; # eliminate end of line character  
  s/[?!"(){};&]//g; # replace punctuation with null  
  s/\\/ /g; # replace slash with space  
  s/\\-//g; #replace dash with null  
  s/^ //g; #replace beginning of line space  
  print "$_ \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
```

## Text Mining Handbook

```
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
}
```

## II Zipf's Law – 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;
```

## Appendix D: Term-Document Matrix and Stop Words

Program to Create Term-Document Matrix.

TermDocMatrix.pl

```
#!/perl -w
#
# Program TermDocData.pl
# This program 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>
$TheFile = "Top9.txt";
#$Outp1 = "OutInd1.txt";
# open input file with text data
open(MYDATA, $TheFile) or die("Error: cannot open file");
# open first output file
open(OUTP1, ">OutInd1.txt") or die("Cannot open file for writing\n");
# open second output file
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
  $i=0;
# loop through data and convert to lower case and add to dictionary using hash
while (<MYDATA> ) {
    chomp($_);
    $_ = ~ lc($_);
    s/[-.!'"(){}&];//g;
    s/\s+/ /g;
    @words = split(/ /);
    foreach $word (@words) {
        ++$response[$i]{lc($word)}; # get freq of each word on line
        ++$granddict{lc($word)};
    }
    ++$i;
}
# record no of lines in file
$lines = $i-1;
print " no of lines is $lines\n";
# print statistics to screen
for ($j=0; $j<= $lines; ++$j) {
    print "$j ";
    foreach $word (keys %{$response[$j]})
        { print "$word, ${response[$j]}{$word}"; }
    print "\n";
}
# compute term-document matrix
# if term exists on record count frequency, else record gets a zero for the ter,
for $i (0..$lines) {
    foreach $word (keys %granddict) {
        if (exists($response[$i]{$word}))
```

## Text Mining Handbook

```
{
  ++$ indicator[$i]{$word}; }
else
{
  $indicator[$i]{$word}=0;
}
print OUTP1 "$indicator[$i]{$word},";
}
print OUTP1 "\n";
}
# print stats to file
foreach $word (keys %granddict) {
  print OUTP2 "$word,$granddict{$word}\n";
}
# close the files
close MYDATA;
close OUTP1;
close OUTP2;
```

### Program to Eliminate Stop Words When Creating Term-Document Matrix. Stopwords.pl

```
#!/perl -w
#
# 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;
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;
```

## APPENDIX E

### MATCHLINE.PL

#### **#PROGRAM MATCHLINE.TXT TO SEARCH FOR THE PHRASE THAT MOST CLOSELY MATCHES A PHRASE.**

```
#!/perl -w
# matchline.pl
# Usage: matchline.pl <datafile> <in phrase file > <outputfile>
# datafile must be present and a cmd line arg
# create a dictionary of all words in a file and alphabetize them
open(MYDATA, $ARGV[0]) or die("Error: cannot open file '$ARGV[0]'\n");
open(INPH, $ARGV[1]) or die("Error: cannot open file '$ARGV[1]'\n");
open(OUTP, ">$ARGV[2]") or die("Cannot open file '$ARGV[2]' for writing\n");

print OUTP "#Output results for ".$ARGV[0]."\n";

$nphr = 0;

# read in the file, get rid of newline and punctuation chars
while( $line = <MYDATA> ){
    chomp($line);
    $line =~ s/[-?!"(){}'//g;
    @words = split(/ /,$line);

    foreach $word (@words) {
        ++$granddict{lc($word)}; # this is the hash assignment lc is lowercase
        ++$tf[$nphr]{lc($word)};
    }
    $nphr++;
}

# Read in the input phrase from file
$linecnt = 0;
while( $line = <INPH> ){
    print OUTP "Input Phrase: " . $line . "\n";
    chomp($line);
    $line =~ s/[-?!"(){}'//g;
    @words = split(/ /,$line);
    $linecnt++;
}
# FIXME if ($linecnt != 1) die("Input phrase file must contain only 1 line");
print "inph linecount ". $linecnt . "\n";

foreach $word (@words){
    ++$inph{lc($word)};
}
```

```

}

#print %tf[0];
# compute document frequencies
foreach $word (sort (keys(%granddict))){
    $sum = 0;
    for $i (0 .. $nphr) {
        #print $word . "\n";
        if ( exists $tf[$i]{$word} ) {
            ++$sum;
        }
    }
    $df{$word} = $sum;
}

# Step 3 Compute tf-idf weights
$n = $nphr + 1;
foreach $word (sort keys %granddict) {
    for $i (0 .. $nphr) {

        if (exists $tf[$i]{$word} ) {
            $tf_val = $tf[$i]{$word};
        }
        else {
            $tf_val = 0;
        }
        #print OUP "Word ". $word. " ". $tf_val . " df: " . $df{$word}. "\n";
        $weight[$i]{$word} = $tf_val * log($n / $df{$word}) / log(2);
        #print "Weight ". $weight[$i]{$word}. " " . "\n";
    }
}

# Compute weight of input phrase
foreach $word (sort keys %granddict) {
    if (exists $inph{$word} ) {
        $tf_val = $inph{$word};
    }
    else {
        $tf_val = 0;
    }

    $inph_weight{$word} = $tf_val * log($n / $df{$word}) / log(2);
}

# Step 4 Normalize the column of weights
for $i (0 .. $nphr - 1){
    $len2 = 0;
    foreach $word ( sort keys %granddict){
        $len2 += $weight[$i]{$word}**2;
        #print $word . " len2 " . $len2 . "\n";
    }
    $len = sqrt($len2);
    foreach $word (sort keys %granddict){

```

## Text Mining Handbook

```
        $unit[$i]{$word} = $weight[$i]{$word}/$len;
    }
}

# Normalize input weight so it can be compared with the others
foreach $word (sort keys %granddict){
    $len2 += $inph_weight{$word};
    #print "inph ". $word . " len2 " . $len2 . "\n";
}

$len = sqrt($len2);
foreach $word (sort keys %granddict){
    $inph_unit{$word} = $inph_weight{$word}/$len;
}

#Step 5 Compute cosine similarities between input phrase and other phrases
$best = 0;
$best_idx = 0;
for $i ( 0 .. $nphr-1 ){
    $sum = 0;
    foreach $word (sort keys %granddict) {
        $sum += $unit[$i]{$word} * $inph_unit{$word};
    }
    $inph_cosine[$i] = $sum;
    printf "INPH %d %.5f", $i, $inph_cosine[$i];
    printf OUTP "INPH %d %.5f \n", $i, $inph_cosine[$i];
    if ($inph_cosine[$i] > $best) {
        $best = $inph_cosine[$i];
        $best_idx = $i;
    }
}
printf "\nBest Match %.5f, %d\n", $best, $best_idx;
printf OUTP "\nBest Match %.5f, %d\n", $best, $best_idx;

# reopen the data file to get the best line since we didn't store it to save memory
open(MYDATA, $ARGV[0]) or die("Error: cannot open file '$ARGV[0]'\n");

$linecnt = 0;
while( $line = <MYDATA> ){
    print $line . " linecount: " . $linecnt . "\n";
    if( $linecnt == $best_idx ){
        #print $line;
        print OUTP "Best Line: " . $line . "\n";
        $linecnt++;
        last;
    }
    else {
        $linecnt++;
    }
}
# close the files
close MYDATA;
close INPH;
```



close OUP;

## MATCHLINE PROGRAM OUTPUT

**Table E-1**

Input Phrase: Credibility of the CAS	
0	0%
1	1%
2	27%
3	0%
4	0%
5	1%
6	0%
7	35%
8	31%
9	13%

Best Match 0.34764, 7  
Best Line: Credibility of the profession

**Table D-2 Program Input**

### Input Data for Matchline.pl Program

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.  
Ability to help sort out property pricing problems  
Actuarial Malpractice (2)  
Actuarial Students of today are not good communicators/executive material.  
Admission requirements and Membership value in an increasingly globalized socio-economic network  
Credibility of the profession  
Credibility of the actuarial profession  
Credibility of reserve estimates

## APPENDIX F

### R Resources for Beginners

There are a great number of online resources for learning R. The first place to begin is at the CRAN Web Site: <http://cran.r-project.org/>. From the main page, lower left, Documentation, there are links to the main R FAQ documents (Hornick, 2009) (see <http://cran.r-project.org/doc/FAQ/R-FAQ.html>) and R MAC OS X FAQ and R Windows FAQ. We also recommend, for those new to R, that you download the “R-Intro.pdf” file (“Introduction to R” by Venables et al.

There are several mailing discussion lists including: <http://www.nabble.com/R-f13819.html>, <https://stat.ethz.ch/mailman/listinfo/r-sig-mac>, and the R Manuals edited by the Core R Development team are a great place to start. The R Manuals include topics such as: An Introduction to R, R Data Import/Export, Extending R, etc.

Contributed Documentation also includes such titles as Using R for Data Analysis and Graphics - Introduction, Examples and Commentary (Maindonald), Simple R (Verzani), R for Beginners (Paradis), R for Windows Users (Wang), Fitting Distributions with R (Ricci), R Reference Card (Baron) etc.

Additional Web sites of interest include:

Learn R at the USGS by (Geissler and Philippi)

<http://www.fort.usgs.gov/brdscience/learnR08.htm>

Quick-R (Kabacoff)

<http://www.statmethods.net/index.html>

R Graph Gallery (Francois)

<http://addictedtor.free.fr/graphiques/>

UCLA Stat Computing Resources to help you learn R:

<http://www.ats.ucla.edu/stat/r/>

CRAN Main page – Task Views provide an introduction and guidance based on topic areas of

using R in such areas as Econometrics, Graphics, Spatial data, Survival Analysis, or Time Series.

R tutorials Web sites:

<http://www.cyclismo.org/tutorial/R/>

<http://www.math.ilstu.edu/dhkim/Rstuff/Rtutor.html>

[http://www.stat.pitt.edu/stoffer/tsa2/R\\_time\\_series\\_quick\\_fix.htm](http://www.stat.pitt.edu/stoffer/tsa2/R_time_series_quick_fix.htm)

<http://www.econ.uiuc.edu/~econ472/tutorial2.html>

<http://faculty.washington.edu/tlumley/Rcourse/>

The Casualty Actuarial Society's Web Site also has a number of R resources. Among them is the Glenn Meyers article, "The R Programming Language—My 'Go To' Computational Software" (*Actuarial Review*, November 2006). Since programming in the R language has been featured at several seminars and meetings, such as the November 2008 Annual Meeting, handouts on R are also available on the CAS Web Site.

## **APPENDIX G**

### **Python for Text Mining**

A popular alternative to Perl for text processing is the open source language Python. Many professional programmers prefer Python to Perl. Our primary consideration in using Perl in this paper is that the authors had some experience with Perl. In addition, the recently published book, *Practical Text Mining with Perl*, by Bilisoly was an excellent aid to those new to using this language for text mining. This reference was invaluable. A recently published book now provides a comprehensive reference for text mining in Python (Bird et al., 2009).

Python can be downloaded from [www.python.org](http://www.python.org). According to the Python Web Site, "You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs." Python is reputed to be easier to learn than Perl, yet is also a particularly handy tool for processing text data. According to Python's downloadable documentation "While Python has some dark corners that can lead to obscure code, there are relatively few such corners, and proper design can isolate their use to only a few classes or modules." On the other hand, Perl has some special variables and language constructs that can be considered "obscure."

Those wishing to learn more about using Python for text mining can visit the Python Web Site.

A brief overview of using Python in Natural Language Processing is provided by Madnani (2009). This paper introduces Python's natural language toolkit, a Python library that performs many text mining tasks. Finally, the book *Natural Language Processing with Python* (Bird, Klein & Loper, 2009) provides a more detailed guide to text mining in Python. Note that natural language processing libraries will generally perform many of the tasks described in this paper as text mining and natural language processing are similar disciplines with a significant overlap. However, natural language processing tools can perform functions not addressed in this paper such as part of speech tagging.

Both Perl and Python are considered especially useful for text processing. Thus, we do not recommend one over the other, but wish to make the reader aware of Python.

## 5. REFERENCES

- [1.] Bilisoly, R., *Practical Text Mining with Perl*, Wiley Publishing, 2008.
- [2.] Brender, A., "The Use of Internal Models for Determining Liabilities and Capital Requirements," *North American Actuarial Journal*, 6:2, 2002, pp. 1-10.
- [3.] Bird, Steven, Ewan Klein, Edward Loper, *Natural Language Processing with Python*, O'Reilly Media, 2009.
- [4.] Brieman, L., J. Freidman, R. Olshen, and C. Stone, *Classification and Regression Trees*, Chapman & Hal/CRC Press, 1993.
- [5.] Brieman, L., "Random Forests," *Machine Learning*, 45:1, 2001, pp. 5–32.
- [6.] Braithwaite, Nancy, "CAS Quinquennial Membership Survey Results Show Members Seek Practical Focus," *Actuarial Review*, August 2009, <http://www.casact.org/newsletter/index.cfm?fa=viewart&id=5804>.
- [7.] Braithwaite, Nancy, et al., "Report of the 2008 CAS Quinquennial Membership Survey Task Force," CAS 2009 Summer *E-Forum*, [http://www.casact.org/pubs/forum/09sumforum/04\\_Quin\\_Survey\\_Report.pdf](http://www.casact.org/pubs/forum/09sumforum/04_Quin_Survey_Report.pdf).
- [8.] Crawley, Michael J., *Statistics: An Introduction Using R* Wiley Publishing, 2005.
- [9.] Dalgaard, Peter, *Introductory Statistics with R*, Springer, 2008.
- [10.] Davi, A., et al., "A Review of Two Text Mining Packages: SAS Text Mining and WordStat," *The American Statistician*, Feb. 2005.
- [11.] De Ville, Barry, *Decision Trees for Business Intelligence and Data Mining: Using SAS Enterprise Miner*, SAS Press, 2006.
- [12.] Francis, Louise A., "Taming Text: An Introduction to Text Mining", *Casualty Actuarial Society Forum*, Winter 2006.
- [13.] Feinerer, I., K. Hornik, and D. Meyer, "Text mining infrastructure in R," *Journal of Statistical Software* 25:5, Mar 2008, <http://www.jstatsoft.org/v25/i05>.
- [14.] Frenz, C., *Pro Perl Parsing*, Apress 2005.
- [15.] Himmel, W., U. Reincke, and H. W. Michelmann, "Using Text Mining to Classify Lay Requests to a Medical Expert Forum and to Prepare Semiautomatic Answers," SAS Global Forum, 2008.
- [16.] Hoffman, P., *Perl for Dummies*, 4th edition, For Dummies, 2003.
- [17.] Inmon, W. and A. Nesavich, *Tapping into unstructured data: integrating unstructured data and textual analytics into business intelligence*, First edition, Prentice Hall Press, 2007.
- [18.] Kaufman, L., and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, 9<sup>th</sup> edition, Wiley-Interscience, 1990.
- [19.] Kim, J., *Introduction to Factor Analysis*, Sage Publications, 1978
- [20.] Kolyshkina, I. and M. van Rooyen, "Text Mining For Insurance Claim Cost Prediction," Presented at the XVth General Insurance Seminar Institute of the Actuaries of Australia, October 2005.
- [21.] Manning, C. and H. Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, 1999.

## *Text Mining Handbook*

- [22.] Madnani, Nitin. "Getting Started on Natural Language Processing with Python," <http://www.umiacs.umd.edu/~nmadnani/pdf/crossroads.pdf> (this version revised slightly from the original article in *ACM Crossroads* 13:4, 2007).
- [23.] Maindonald, J. H. and J. Braun, *Data analysis and graphics using R*, Cambridge University Press, 2007.
- [24.] Mildenhall, S., "Minimum Bias and Generalized Linear Models," *Proceedings of the Casualty Actuarial Society* 1999, Vol. LXXVI, 393-487.
- [25.] Weiss, S.M., N. et al., *Predictive Methods for Analyzing Unstructured Information*, Springer, 2005.
- [26.] Venables, W.N., and B.D. Ripley, *Modern Applied Statistics with S*, 4th Ed., Springer, 2002.
- [27.] Wikipedia, "Text Corpus," Accessed Sep 9, 2009.

### **Biography(ies) of the Author(s)**

**Louise Francis** is a consulting principal at Francis Analytics and Actuarial Data Mining, Inc. She is involved in data mining projects as well as conventional actuarial analyses. She has a BA degree from William Smith College and an MS in Health Sciences from SUNY at Stony Brook. She is a Fellow of the CAS and a Member of the American Academy of Actuaries. She is the 2008-2011 CAS Vice President-Research & Development. She is a five-time winner of the CAS Management Data, and Information Prize.

**Matthew Flynn, PhD** is a senior statistician in claim research at Travelers Insurance in Hartford, CT. He is responsible for leading and developing advanced claim analytics at Travelers. He has a degree in finance from Purdue University. He is a frequent presenter at industry symposia and SAS user group meetings.