Text Analyzer for Efficient Information Retrieval

Sellappan Palaniappan, Looi Siang Shing
Department of Information Technology,
Malaysia University of Science and Technology, Block C, Kelana Square, Kelana Jaya,
47301 Petaling Jaya, Malaysia

Abstract: When we retrieve information from a website or web-based system we receive both relevant and unwanted material. Reading unwanted material wastes time and reduces productivity. A well-designed text analyzer can help minimize this problem. This research presents a text analyzer that reduces the amount of unwanted material retrieved. It filters unwanted material by using the knowledge that it had gained previously or acquired by grouping data elements. Results show that the accuracy of information retrieved is proportional to the efficiency of work done and decisions made. Its performance is compared and bench-marked with other text analyzers. Initial investigations based on several sample runs show that this text analyzer is more efficient than many others.

Keywords: Text analysis, text analyzer, information retrieval, web-based systems, short documents, text mining.

1. INTRODUCTION

Because human reading is very slow, much of work done by humans is now transferred to the computer. For example, database management replaces traditional file systems. This conserves time and energy [1]. Although software tools are useful and necessary, we still humans to retrieve and interpret text information.

Text reading tools can convert text to spoken words. Humans can hear and interpret the contents of the text by listening. There are also tools that can read long texts, e.g., research papers or novels [2]. Contents of a text are normally accessed using the highest frequency occurrence of words [3, 4]. However, this doesn't always apply to short texts of a few pages such as job resumes. The lengths of these texts may be short, but they can still take up considerable time if the number of documents is large. A well-designed text analyzer can help minimize this problem, which is the purpose of this research.

2. METHODOLOGY

This study analyzes only short texts consisting of a few pages. It focuses on tools that discover relationships between categories. Specifically, it focuses on the processes that are needed to extract contents, analyze concepts, discover patterns, conduct visualization, and perform interactive analysis [5].

There are two main challenges in developing a text analyzer. The first is to ensure that the text-mining process is not adversely affected by the amount of knowledge stored inside the text analyzer. The knowledge or information repository will grow with the passage of time as new elements would be added. Text mining involves categorizing the text using the knowledge in the repository. So the algorithm used must provide efficient indexing. The second challenge is to ensure that the algorithm matches exact words. It must distinguish between the beginning and the end of a word, e.g., a job resume that contains the skill *JavaScript* and the knowledge that the text analyzer is looking for is *Java*. In this case, the text analyzer must be intelligent to filter out the word *JavaScript* and accept only the word *Java* [6].

This study develops a text analyzer that can assist organizations process their textual information efficiently. The text analyzer is embedded in a previously developed web-based system called Volunteer Management System (VMS).

Figure 1 shows the architecture of the text analyzer. It consists of three development phases: Input, Text Analysis and Decision Support.

Manuscript received November 5, 2008
Manuscript revised November 20, 2008

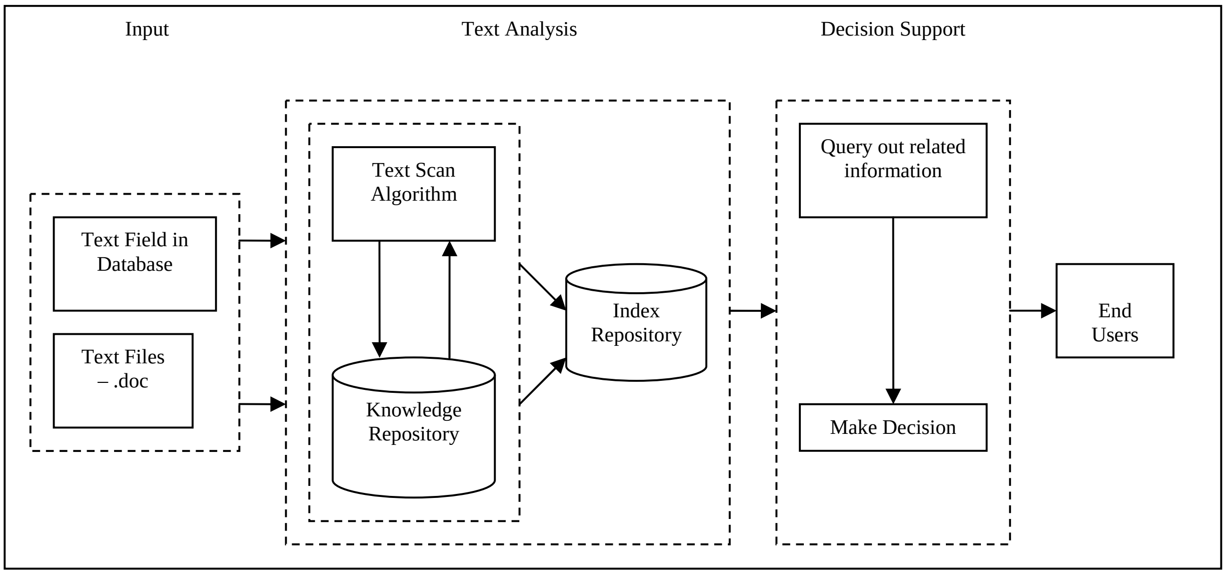

Figure 1 *System Architecture of Text Analyzer*

2.1 Input

The input to the text analyzer is a text, which can take the form of a text field in a database or a text file. For database text fields, the text analyzer must be given the necessary access rights. Currently, the text analyzer supports only two types of input files - Microsoft Word (.doc) and text (.txt). The files' paths must be initialized during the text analyzer is setup.

The text analyzer will read new inputs automatically. When new data arrives into the database or files, the text analyzer will immediately analyze the input and update the knowledge/information repository.

2.2 Text Analysis

Text analysis constitutes the core of the text analyzer. When the text analyzer detects a new input, it reads the whole text and retrieves related information using the knowledge in the repository. The results are categorized and entered into the repository. The repository is indexed to speed up the search during the decision making phase.

2.3 Decision Support

In the decision support phase, the system will query the repository for the information (built from the text analysis phase) it needs. Decision making must be based on user's needs. For example, in a volunteer management system, some activities would require that volunteers possess certain attributes or qualifications such as C programming knowledge. The results of the query are used to determine if the results are accurate.

3. TEXT ANALYZER ALGORITHM

The input to the system are Microsoft Word Documents with the extension .doc. It also uses Microsoft Office Interop dll to read the Word documents.

The text analyzer algorithm used in this study is an improved version of that suggested by Alfred V. Aho and Margaret J. Corasick [7]. The algorithm has two two main parts of the Text Analyzer Algorithm: Built Knowledge Tree and Text Analysis Process.

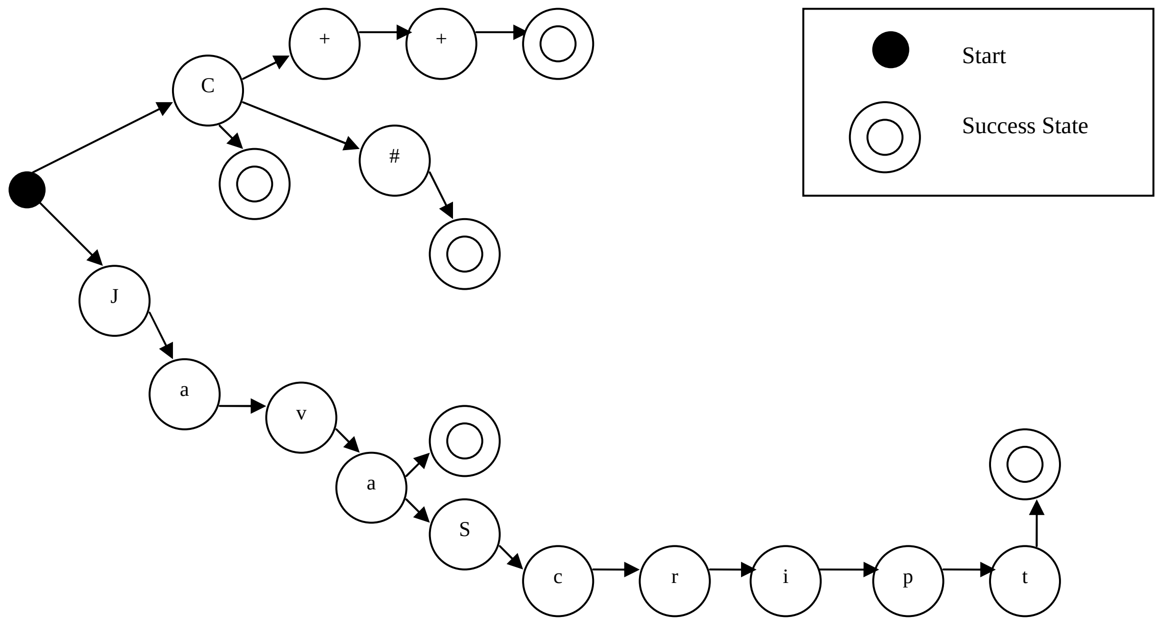

Figure 2 Knowledge Tree Automata

3.1 Build Knowledge Tree

Based on a set of keywords yet to be searched, a knowledge tree is built. Keywords can be a word or a phrase. Figure 2 shows the state diagram of the knowledge tree.

The knowledge tree is formed by a set of finite automatons**.** For example, there are five knowledge keywords in the tree: Java, JavaScript, C, C++ and C#. Every leaf node in the tree is a success state. When a success state is reached, the word or phrase of the input will be accepted and vice versa.

Node

Each character is represented by a node. There is node sharing between knowledge keywords. For example, the “C” node is shared by “C”, “C++” and “C#”. A word like “JavaScript” is formed by ten nodes.

Word Separator

When a text passes through the knowledge tree, the system will accept words or sub words, which matches the character(s) located in the knowledge tree. From the above knowledge tree, we can see that the input string “Communication” passes into the tree. When the character “C” is matched with the “C” node, the system will accept it.

We can solve this problem by adding a word separator at the root and leaf of the tree. A word separator is formed by a group of characters [8]. It consists of the next character that occurs before and after a word. The root and success state are formed by a different set of word separators. A word separator enables the knowledge tree to accept the start of a word. Meanwhile, the success state accepts only the word or phrase that matches the keywords located in the knowledge tree.

At each success state, there are additional word separators to indicate the end of a word. The set of word separators located at each success state are: white space (“ ”), comma (“,”), period (“.”), dash (“-”), question mark (“?”), exclamation mark (“!”), new line (“\n”), tab(“\t”) and carriage return (“\r”).

Word separators located at the base of the knowledge tree consist of new lines and looping of word separators at success state. However, the occurrence of word separators at the beginning of the first word of a file is optional.

Table 1 shows a list of word separators used for recognizing the end of a word.

Word Separator	***ASCII (Dec)***	***Description***
	32	White space
,	44	Comma
.	46	Period
-	45	Dash
?	63	Question mark
!	33	Exclamation point
\n	10	New line
\r	13	Carriage return
\t	9	Tab

Table 1 *Word separator*

Analysis of the knowledge tree shows that the time needed for the construction tree follows a negative exponential function as shown in Figure 3.

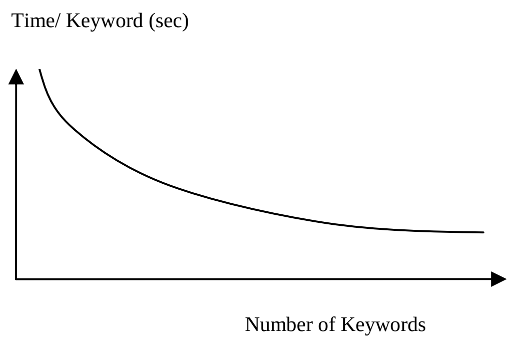

Figure 3 *Performance of build tree algorithm*

3.2 Text Analyzer

Figure 4 shows the steps performed by the Text Analyzer.

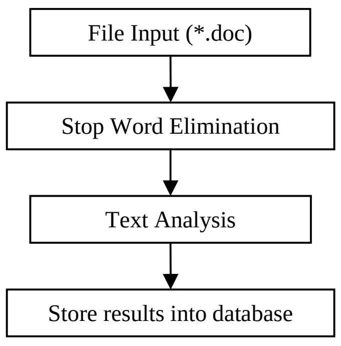

Figure 4 *Process of Documents*

Prior entering text into the knowledge tree the stop words are eliminated.

Stop Word Elimination

Wikipedia defines stop words as words which are so common that they are useless to index or use in search engines. Usually articles and adverbials are stop words. [9]

The first step in text processing is to eliminate stop words from the documents. Stop word elimination enhances the text by saving space and improving searching speeds [10]. In this system, stop word elimination is done by using a multitasking technique which helps to speed up the text processing [11].

Key Matching in Tree Node

The node and the input of character are compared based on their ASCII index number. If both have equal ASCII value the flow will proceed to the next node and the next input character. The searching process is terminated or relinquished for another new word when an unmatched character is found.

Character comparison is based on its ASCII values. The search is case sensitive so as to get the exact word contained in the knowledge tree.

3.3 Input Structure and Word Detection

The input to the system in this study is short text, specifically, job resumes. A resume is a short text, so the main content cannot be determined by the frequency of occurrence of particular keywords [12].

The main challenge in reading text is how to recognize a word that is matched with the keyword that we are looking for. In the tree structure design, the root plays the main role in detecting the beginning of a word. For the first word of a file, there might not be any stop words, so occurrence of stop words is optional. For the rest of the words in the text, there is looping at success states until a new start word is found. This study assumes that the connection between words in the text is linked by one or more stop words.

Detection of beginning of a word

In order to determine if there is a beginning for a word, two steps are performed. First, since the first character of a word occurs after a stop word, i.e., in a sentence, words begins after a white space, may occur after few tabs, new line, or new page. *If s(i) is the first character of a word, then s(i-1) is root stop word.*

Second, the beginning of a word is the character before the first character of the word, which is a non- alphabet. *If s(i) is the first character of a word, then s(i-1) is non-alphabet.*

A word may start after another character such as "*", "@", "^"and "&". These characters are not in the word separator list. The second step is used to double check the result set. The checking of non-alphabet is executed after the text reading is completed. At this stage, if the first alphabet of a word is s(i), then the s(i-1) will be checked. If s(i-1) is an alphabet, it will be removed from the result list.

Detection of end of a word

When a word had reached the success state, the input text is considered as consisting of the knowledge keyword. Similar to the detection of the beginning of a word as explained above, there are two steps to be performed. In the first step, when the success state is reached, the word or phrase is accepted. *If s(i) is the last character of a word, then s(i+1) is leaf stop word.*

The second step is similar to the first step, where its main function is to double check after the whole input text is executed. *If s(i) is the last character of a word, then s(i+1) is non-alphabet.*

At this stage, if the last alphabet of a word is s(i), then the s(i+1) will be checked. For the last character in an input text, there is no checking on the s(i+1) character.

3.4 Tree Traversal Algorithm

Figure 5 shows the tree traversal algorithm which is specially designed for the searching of word or phrase. The searching will start from the first character of the input. It will travel into the tree beginning from the root (n = root). If an unmatched c and n is found, the algorithm will skip the remaining characters of the word. The searching will restart from next start point (nStart) [13].

```
Input: Input string (s)
Tree Node : Node (n), successState
(sState)
Nodes start from root until successState
(keyword)
Next Start (nStart)
Return: List of result
i=0
n=root

While (i<s.Length)
   If s(i) = n then
      NextNode(n)
      i++
      if s(i) = newline or s(i) =
                                WhiteSpace
         nStart = i
      EndIf
      If c = sState then
         result.Add(keyword)
         If sState = WhiteSpace
            nStart = i-1
            i = nStart
         EndIf
      EndIf
   ElseIf c <> n then
      i = nStart
   EndIf
EndWhile
```

Figure 5 *Tree Traversal Algorithm*

If character s(i) travels until the leaf of the tree, it means the success state (sState) is reached. The keyword that had been found will be added to the result list. There will be backtracking of i in the next traversal. The purpose of back tracking is to enable the search to be continued right to the next new word

The performance of text analysis is given in Figure 6. As the length of the input text is increases, the execution time of text analysis decreases.

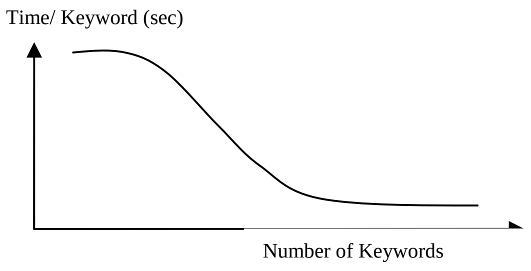

Figure 6 *Performance of Text Analyzer*

4. ACCURACY OF TEXT ANALYZER

In ascertaining the accuracy of the Text Analyzer a collection of 10 short input texts is amassed-in order to reduce repetition. These short texts are retrieved from job description that were posted in JobStreet.com [14].

For the purpose of this experiment, the result generated from the Text Analyzer is compared with the result obtained by human reading. It is assumed that human reading is 100% accurate.

The formula for accuracy is:

$$\frac{No.keyword\ \ found\ - No.unmatched\ [Text\ \ Analyzer\ ]}{No.keyword\ \ found\ [Human\ \ Being\ ]} \times 100\ \%$$

No. of Sample	Keyword Found			Classification Accuracy (%)
	Text Analyzer		Human Being	
	Match	Unmatched		
1	12	0	12	100
2	9	0	10	90
3	9	0	9	100
4	9	0	9	100
5	6	0	6	100
6	14	1	14	92.86
7	5	0	5	100
8	15	0	15	100
9	9	0	9	100
10	7	0	7	100

Table 2 *Text Analysis Result*

From Table 2, it is found that the overall standard of accuracy is good although the results seem to give some false alarms. In sample 2, there is one keyword not found and the reason for this is because there is a "&" symbol occurring at the prefix of the keyword, &[Keyword]. The Text Analyzer which detects the beginning of a word does not encounter the "&" symbol because it does not ordinarily belong to the beginning of a word character. In sample 6, there is an unmatched result found. The Text Analyzer matched the word correctly. However, the original meaning of the input text might differ from what the Text Analyzer had matched. So, sample 6's result is considered as an unmatched keyword.

The overall accuracy of the text analyzer is 98.2%. This value is considered good as most of the time it is able to produce correct outputs.

5. COMPARISIONS

5.1 Performance Comparisons

This section compares the text analyzer's result with the regular expression provided in .Net, namely, *Regex.IsMatch.* This function can replace the build tree and text analysis algorithm [15].

Table 3 shows the results of the text analysis by the text analyzer. It shows the execution times of the text analysis. The build knowledge tree is performed before analyzing the input text.

Input Text (pages)	**Knowledge Keywords Size**			
	20	**50**	**100**	**200**
2	0.0200	0.0160	0.1824	0.0400
5	0.0140	0.0400	0.0840	0.1100
10	0.0360	0.0600	0.1264	0.2042
20	0.0920	0.0882	0.2324	0.2502
40	0.1662	0.1400	0.4528	0.4764

Table 3 *Execution time of text analysis*

Table 4 shows the execution times of text analysis by using the Regular Expression. Since there is no grouping of the list of keywords, the keywords are checked by using the *Regex.match* function.

Input Text (pages)	**Knowledge Keywords Size**			
	20	**50**	**100**	**200**
2	0.0774	0.1876	0.4346	0.8130
5	0.1748	0.4788	1.0880	1.9092
10	0.3538	0.8964	2.1664	3.7208
20	0.6474	1.8112	3.7026	7.2202
40	1.6210	3.7158	7.3696	14.0760

Table 4 *Execution time of Regular Expression*

From the results we can be say that the execution times of the regular expression is higher compared to the execution time of the text analyzer. As the keywords size is increase, the differences become higher.

Additionally, regular expressions suffer when applied to dynamic keywords. For example, some keywords like "C#" or "C++" consist of special characters, "#" and "+". These characters are also part of the syntax of regular expressions. To enable text analysis with special characters, the character "\" needs to be added in front of each special character. This means that regular expression is not so suitable for dynamic keywords. Before performing text analysis using regular expressions, there must be process to manage special characters in the keyword list. In this experiment, the execution time of this process is excluded.

To summarize: the text analyzer result is better than the regular expression result. Besides, regular expression is only applicable to low number of keywords and non dynamic keywords.

5.2 Accuracy Comparisons

The accuracy of the Text Analyzer was determined by comparing the results from three job search websites, namely, JobStreet (http://my.jobstreet.com) [14], JenJobs (http://www.jenjobs.com) [16] and Guardianjobs (http://jobs.guardian.co.uk) [17] and the VolunteerNet website. The same filtering criteria were used for all.

An account was created for each of three websites as well as for the VolunteerNet website. Required details were entered and the same job resume in Word document was uploaded to all these websites.

In this experiment, the results or output is a list of jobs suitable for applicant. The uploaded resume and the website registration forms are needed in the job matching process. As the domain for Text Analyzer is on IT job searching, the filtering process focused only on IT-related jobs. The output from this experiment is a set of reports related to IT skills of applicants. The results are derived from the list of jobs matched by the website. Table 7.4 summarizes the results generated by the four websites.

The value in *No. of Suitable jobs* field is determined by users. Each job description is read and related job is selected. The *No. of Suitable Jobs* is considered to have an accuracy of 100%. Each job supplied to each website is categorized into *No. of Correct Jobs Received* and *No. of Incorrect Jobs Received.*

Attributes	**Job Street**	**Jen Jobs**	**Guardian jobs**	**Volunteer Net**
No. of Suitable Jobs	7	5	9	10
No. of Jobs Received	6	7	7	10
No. of Correct Jobs Received	4	4	7	10
No. of Incorrect Jobs Received	2	3	0	0

Table 5 *Job Filtering Result*

As Table 5 reveals, not all job alerts given by the 3 websites are relevant to the applicant. The alerts by JobStreet focus only on newly posted jobs. Alerts such as Management Training and Service Engineer are also sent to the applicant. It is found that JobStreet also sends out partially relevant jobs to the applicant. For example, the list includes unrelated job titles such as Management Training and Service Engineer. This is because the applicant selected the location field. Besides, the Service Engineer job is sent to the applicant as he had also selected the manufacturing field.

The JobMatcher of JenJobs found 7 jobs suitable for the applicant. Similar to JobStreet, it also sends jobs based on industry. Here, unrelated positions such as Sales Account Manager are sent to the applicant.

GuardianJobs gives better job alerts than JobStreet and JenJobs. This is because GuardianJobs provides more job-related alerts and requires applicants to fill in more information, including checking a check box list consisting IT skills. Thus, most of the jobs sent to the applicants are related jobs. However, the Prolog job has been missed out from the job alert list. This is because the IT skill list is not complete.

The VolunteerNet website which implemented the Text Analyzer for processing applicants' resumes performed well in the job filtering process. It is able to reduce the number of unrelated alerts. The Text Analyzer is flexible as it allows new knowledge (IT skill) to be added from time to time.

From the study of the 3 job websites, applicants to upload resume is an extra feature for company to view applicant's details. They do not perform any analysis work on short text resumes. The solution provided here improves the job filtering process as well as avoids spamming.

8. CONCLUSION

Reading a large number of short texts manually can be quite time-consuming [18]. This research has presented a text analyzer for reading and analyzing short input texts more efficiently than other text analyzers. It retrieves information using a set of predefined keywords. It has three parts: (1) Processing input texts in the form of Microsoft Word documents; this involves converting Word files to text files and eliminating stop words, (2) Building a knowledge tree by grouping keywords, and (3) Processing the knowledge tree.

Several test runs show that this text analyzer is more efficient than many others. The statistics generated by the text analyzer include mean, variance and rate of change. It can be installed on servers to analyze input texts.

9. FUTURE WORK

While the results presented are encouraging, there is still room for improvement.

1. The text analyzer in this research performs exact word searching. However, there is no text stemming. Stemming, for example, ensures that the words "traveling" and "traveled" refer to the same word [18, 20].

2. The exact text matching in this research is case sensitive. That means, the word “JAVA” is different from the word “Java”. Work can be done to determine if case sensitivith is important in text analysis [19].

3. Synonyms such as "sick" and "ill" or words that are used in particular phrases where they denote unique meaning can be combined for indexing. For example, "Microsoft Windows" might be such a phrase, which is a specific reference to the computer operating system, but has nothing to do with the common use of the term "Windows" as descriptions in home improvement projects [20].

REFERENCES

[1] New Approach to Text Analysis (2006). Retrieved by November, 2006 from http://www.analyst.ru/index.php?lang=eng&dir=content/tech/&id=approach&left=content/tech/menu.txt

[2] TextAnalyst(2006). Retrieved by November, 2006 from http://www.megaputer.com/products/ta/index.php3

[3] Text Mining. (2004). Retrieved November 12, 2005, from http://www.statsoft.com/textbook/sttextmin.html#incorporating

[4] Go Beyond Counting Keywords (2006). Retrieved by November, 2006 from http://www.megaputer.com/textanalyst.php

[5] Sergei Ananyan, Michael Kiselev (2006).Automated Analysis of Unstructured Texts (2006) TextAnalyst white paper. Retrieved November, 2006, from http://www.megaputer.com/file/textanalyst_whitepaper.pdf

[6] Fabrizio Sebastiani. (March 2002). Machine Learning in Automated Text Categorization. *Journal of ACM Computing Surveys*, Vol. 34, No. 1, March 2002, pp. 1–47. Retrieved November 12, 2005, from ACM Digital Library Database.

[7] Aho-Corasick Algorithm. Retrieved by November, 2006 from http://en.wikipedia.org/wiki/Aho-Corasick_algorithm

[8] Word Seperator Definition. Retrieved by February, 2007 from http://www.pcmag.com/encyclopedia_term/0,2542,t=word+separator&i=54837,00.asp

[9] Stop Words (2006). Retrieved November 17, 2006 from http://en.wikipedia.org/wiki/Stop_words

[10] PCMag Encyclopedia, Definition of Stop Word (2006). Retrieved November, 2006 from http://www.pcmag.com/encyclopedia_term/0,2542,t=word+separator&i=54837,00.asp

[11] Danny Sullivan, What Are Stop Word (01/01/2003). Retrieved November 1, 2006 from http://searchenginewatch.com/showPage.html?page=2156061

[12] Osmar R.Zaiane, Maria-Luiza Antonie. (2001) *Classifying Text Documents by Associating Terms with Text Catogories*, 215-222. Retrieved November 12, 2005, from ACM Digital Library Database.

14] Tree Traversal. Retrieved by February,2007 from http://en.wikipedia.org/wiki/Tree_traversal

[14] JobStreet. Retrieved by November, 2007 from http://www.jobstreet.com/

[15] How To: Use Regular Expressions to Constrain Input in ASP.NET. Retrieved by February,2007 from http://msdn2.microsoft.com/en-us/library/ms998267.aspx

[16] JenJobs. Retrieved by November, 2007 from http://www.jenjobs.com/

[17] Guardian Jobs. Retrieved by November, 2007 from http://www.guardianjobs.com/

[18] J V Bryar. (2001). Taxonomies The Value of Organized Business Knowledge. *New Edge Corperation.*

[19] M. K. Kowar. (2006, Jan). Automatic Generation of Back-of The Book Index: An Integrated Approach Through Text Mining Operations. *Proceedings of the International Conference on Recent Trendy in Information System (IRIS'06), National Engineering College,Kovilpatti, Tamil, India* , 416-422.

[20] StatSoft (2004). Retrieved by January, 2007 from http://www.statsoft.com/textbook/sttextmin.html

Sellappan Palaniappan obtained his PhD in Interdisciplinary Information Science from University of Pittsburgh and a MSc in Computer Science from University of London. He is an Associate Professor at the Department of Information Technology, Malaysia University of Science and Technology. His research interests include information integration, clinical decision support systems, OLAP and data mining, web services and collaborative CASE tools.

Looi Siang Shing received her MSc in Information Technology from Malaysia University of Science and Technology and a BSc (Hons) from Multimedia University, Malaysia. Her research interests include data/text mining, healthcare decision support systems and web applications and technologies.