TextMOLE: Text Mining Operations Library and Environment

Daniel B. Waegel
Ursinus College
Mathematics and Computer Science
PO Box 1000, 601 Main St.
Collegeville PA 19426
dawaegel@ursinus.edu

April Kontostathis
Ursinus College
Mathematics and Computer Science
PO Box 1000, 601 Main St.
Collegeville PA 19426
akontostathis@ursinus.edu

ABSTRACT

The paper describes the first version of the TextMOLE (Text Mining Operations Library and Environment) system for textual data mining. Currently TextMOLE acts as an advanced indexing and search engine: it parses a data set, extracts relevant terms, and allows the user to run queries against the data. The system design is open-ended, robust, and flexible. The tool is designed to quickly analyze a corpus of documents and determine which parameters will provide maximal retrieval performance. Thus an instructor can use the tool to demonstrate information retrieval concepts in the classroom, or use the tool to encourage hands-on exploration of concepts often covered in an introductory course in information retrieval or artificial intelligence. Reseachers will find the tool useful when a 'quick and dirty' analysis of an unfamiliar collection is required.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Experimentation

Keywords

Information Retrieval, Textual Data Mining, Machine Learning

1. INTRODUCTION

In this paper we describe the TextMOLE (Text Mining Operations Library and Environment) system. This tool has been developed to assist with research and classroom instruction in textual data mining topics. These topics are typically covered in courses such as Machine Learning, Information Retrieval, and Artificial Intelligence. The tool provides a medium to demonstrate the fundamentals in information retrieval, particularly the complicated and ambiguous problem of determining what document(s) are most relevant to a given query. The user can configure the indexing and/or retrieval options, allowing for quick comparisons between different sets of parameters. This will allow easy exploration of the benefits and drawbacks for each set of parameters. Currently the system allows easy configuration of local and global term weighting, stop lists, and the use of stemming. An instructor can develop assignments which encourage students to use the system in order to determine optimal parameter settings for a given collection. For example, it can be used to show how a simple change to the the weighting schema for a given query can result in more relevant results (or can ruin an otherwise excellent query).

The tool has a clear and simple implementation, which sets it apart from the other complicated text-mining tools that exist today. The source code can be easily understood by a student with two semesters of programming experience in C++. The source code will be freely available so that an instructor can choose to develop more challenging assignments that involve changes to the TextMOLE system or use of the library functions.

In the next section we provide a brief introduction to information retrieval. In Section 3 we present an overview of the application. In Section 4 we present our views on why this application is needed and how it can be used in the classroom. In Section 5 we offer our concluding remarks and describe some enhancements we are planning to make to the system.

2. BACKGROUND

In this section we set up the terminology that will be used to describe the application in Section 3.

2.1 Overview of Information Retrieval

In information retrieval applications, we refer to two primary entities: terms and documents. Documents are units of retrieval; for example, a paragraph, a section, a chapter, a web page, an article, or a whole book. An index term (or simply a term) is a word (or group of words) that is used to represent the content of a document.

2.1.1 Traditional Vector Space Retrieval

In traditional vector space retrieval, documents and queries are represented as vectors in t-dimensional space, where t is the number of indexed terms in the collection. Generally the document vectors are formed when the index for the collection is generated (these vectors form a matrix that is often referred to as the term by document matrix), and the query vectors are formed when a search of the collection is performed. In order to determine the relevance of a document to the query, the similarity between the query vector and each document vector is computed. The cosine similarity metric is often used to compare the vectors [10]. The cosine similarity metric provides an ordering of the documents, with higher weight assigned to documents that are considered more relevant to the query. This ordering is used to assign a rank to each document, with the document whose weight is highest assigned rank = 1. Retrieval systems typically return documents to the user in rank order.

2.1.2 Term Weighting

Term weighting schemes are commonly applied to the entries in both the query and the document vectors [5, 10, 4]. The purpose of weighting schemes is to reduce the relative importance of high frequency terms while giving words that distinguish the documents in a collection a higher weight.

2.1.3 Stop List

Terms extracted from a corpus are usually compared to a list of common words - ‘noise’ words that are presumed to be useless for running queries - and any matches against this list are discarded. This is usually called a *stop list* (the words on this list are called *stop words*).

2.1.4 Stemming

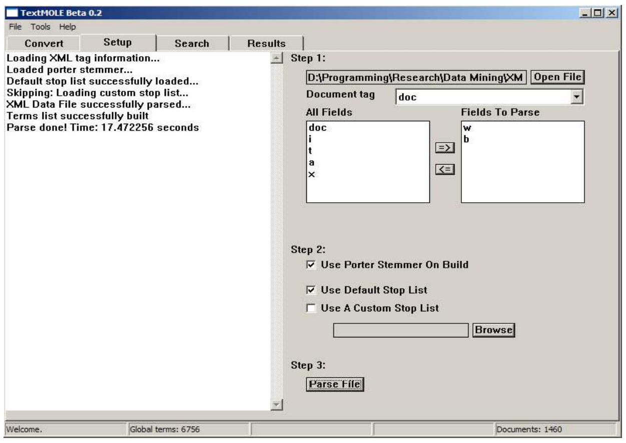
Stemming is another technique that is commonly used to control the list of words indexed for a collection. If stemming is chosen, words are run through an algorithm that attempts to reduce a word to its *root*, although this will often not be the true linguistic root of the word. The stemmer reduces similar words to the same root, and this has two positive effects. First, the number of indexed terms is reduced because similar terms are mapped into one entry. Second, the relevancy of the results is often improved by a significant margin. Intuitively, if you search for a stemmed word, you will get results for all stemmed words similar to it. For example, if you search for ’design’, you will return all results from ’designs’, ’designed’, ’designing’, etc. Of course, just as with the stop lists, this is not always a desirable result. TextMOLE contains an implementation of the Porter Stemmer [6].

3. OVERVIEW OF TEXTMOLE

Currently TextMOLE is an advanced indexing and search engine: it parses a data set, extracts terms, and allows the user to run queries against the data. Future versions will include support for other textual data mining tasks, such as classification, emerging trend detection, filtering, and automatic summarization.

We have designed the system to be as open-ended, robust, and flexible as possible. The program is designed using a 3-tiered architecture, isolating the presentation layer from the rest of the program. The GUI interface for the tool is written for the Windows operating system. However, with the Windows code isolated, future ports to other operating systems will be as painless as possible. The tool is written in C++, and most of the internal storage of data is done using STL (Standard Template Library) vectors and lists.

Figure 1: TextMOLE Indexing Screen



```
Loading XML tag information...
Loaded porter stemmer...
Default stop list successfully loaded...
Skipping: Loading custom stop list...
XML Data File successfully parsed...
Terms list successfully built
Parse done! Time: 17.472256 seconds
```

Step 1:
D:\Programming\Research\Data Mining\XM
Document tag: doc
All Fields: doc, i, t, a, x
Fields To Parse: w, b

Step 2:
☑ Use Porter Stemmer On Build
☑ Use Default Stop List
☐ Use A Custom Stop List

Step 3:

Welcome. | Global terms: 6756 | Documents: 1460

3.1 Indexing Options

The first task required in any text-mining application is reading, analysis, organization, and storage of the text contained in the data set. All of these actions are collectively known as the indexing of the collection. This parsing is often very time-consuming, especially for larger collections; however, we accept a performance degradation in the initial parsing time in order to make queries faster. A sample TextMOLE indexing screen appears in Figure 1.

Documents to be parsed by the TextMOLE tool must be in XML format, with a tag delimiter designating each document as well as each field within a document. A function which converts files containing documents in SMART format to XML is including in the current release. Converters for other document types will be added as time permits. The user specifies the document delimiter as well as the fields which should be extracted and indexed. Performance times for the initial parsing of the data sets range from approximately 30 seconds for 900 documents (2.03 MB) to 80 seconds for 2350 documents (5.39 MB). The performance time for applying weighting schemes to terms and executing queries is subsecond for data sets of these sizes.

3.1.1 Stop List

Our tool provides a default stop list containing 570 of the most common words in the English language. Use of this stop list is optional. In addition, the option to include a second stop list is included. This allows a user to augment the default stop list with an additional set of words, or use just their own stop list and exclude the default, providing a great degree of flexibility.

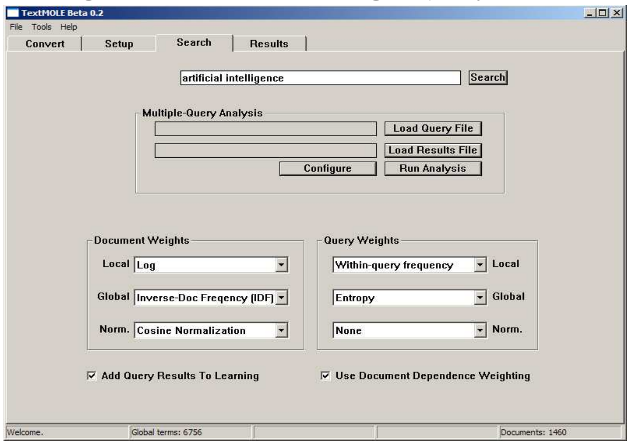
3.1.2 Stemming

When setting up the initial parsing of the data collection, a user has the option to include use of stemming. If stemming is chosen, words that are not thrown out by a stop list are run through the Porter Stemmer algorithm [6]. The Porter Stemmer algorithm reduces the number of indexed terms by 1/3 on average [6]. If stemming is used, the stemming algorithm will be applied automatically to the queries when a search is initiated.

3.2 Query Options

Figures 2 and 3 show the single query input screen along with the results screen. Currently all documents contain-

Figure 2: TextMOLE Single Query Screen



artificial intelligence

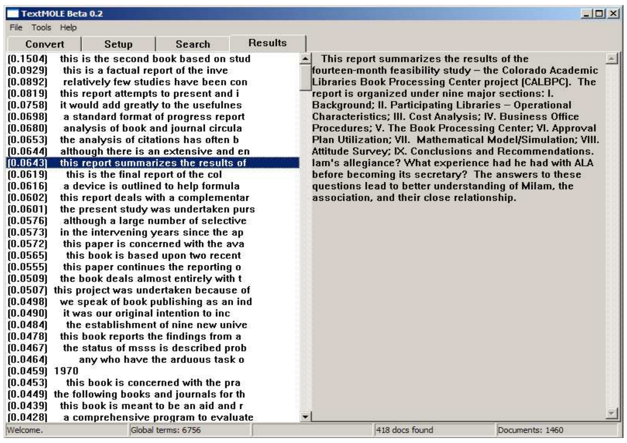
Document Weights
Local: Log
Global: Inverse-Doc Freqency (IDF)
Norm.: Cosine Normalization

Query Weights
Local: Within-query frequency
Global: Entropy
Norm.: None

☑ Add Query Results To Learning
☑ Use Document Dependence Weighting

Global terms: 6756
Documents: 1460

Figure 3: TextMOLE Single Query Results



```
[0.1504]   this is the second book based on stud
[0.0929]    this is a factual report of the inve
[0.0892]    relatively few studies have been con
[0.0819]   this report attempts to present and i
[0.0758]   it would add greatly to the usefulnes
[0.0698]    a standard format of progress report
[0.0680]    analysis of book and journal circula
[0.0653]   the analysis of citations has often b
[0.0644]   although there is an extensive and en
[0.0643]   this report summarizes the results of
[0.0619]     this is the final report of the col
[0.0616]    a device is outlined to help formula
[0.0602]   this report deals with a complementar
[0.0601]   the present study was undertaken purs
[0.0576]    although a large number of selective
[0.0573]   in the intervening years since the ap
[0.0572]    this paper is concerned with the ava
[0.0565]     this book is based upon two recent
[0.0555]    this paper continues the reporting o
[0.0509]   the book deals almost entirely with t
[0.0507] this project was undertaken because of
[0.0498]   we speak of book publishing as an ind
[0.0490]    it was our original intention to inc
[0.0484]     the establishment of nine new unive
[0.0478]   this book reports the findings from a
[0.0467]    the status of msss is described prob
[0.0464]        any who have the arduous task o
[0.0459] 1970
[0.0453]     this book is concerned with the pra
[0.0449] the following books and journals for th
[0.0439]   this book is meant to be an aid and r
[0.0428]    a comprehensive program to evaluate
```

This report summarizes the results of the fourteen-month feasibility study – the Colorado Academic Libraries Book Processing Center project (CALBPC). The report is organized under nine major sections: I. Background; II. Participating Libraries – Operational Characteristics; III. Cost Analysis; IV. Business Office Procedures; V. The Book Processing Center; VI. Approval Plan Utilization; VII. Mathematical Model/Simulation; VIII. Attitude Survey; IX. Conclusions and Recommendations. Iam's allegiance? What experience had he had with ALA before becoming its secretary? The answers to these questions lead to better understanding of Milam, the association, and their close relationship.

Global terms: 6756 | 418 docs found | Documents: 1460

ing at least one query term will be retrieved (default OR operation). Implementation of advanced boolean search is planned.

We have implemented a variety of well-known and effective term weighting schemes [3]. Term weighting can be split into three types: A *local* weight based on the frequency within the document, a *global* weight based on a term's frequency throughout the dataset, and a *normalizing* weight that negates the discrepancies of varying document lengths. The entries in the document vector are computed by multiplying the global weight for each term by the local weight for the document/term pair. Normalization may then be applied to the document vector.

The weighting scheme(s) to be applied to the document and/or query vectors are specified by the user (see Figure 2).

3.2.1 Local Weighting

The local weighting schemes are outlined in Table 1. As a rule of thumb, local weights are designed to make terms that appear more frequently within a document more important. The binary weight is very intuitive: it states that if f_{ij} (the frequency of term i in document j) is zero, then the local weight is zero, and if it is anything else, it is set to 1. The within-document frequency is the default, because it just uses the number of times a term i appears within a document j as the local weight. The log scheme is designed to lessen the importance of frequently-appearing terms within a single document, while still assigning them more value than the binary weighting scheme. The normalized log scheme divides the log scheme by $1 + a_j$, where a_j is the average frequency of all terms in document j.

Binary	1 if $f_{ij} > 0$ 0 otherwise
Term Frequency	f_{ij}
Log	$(1 + \log f_{ij})$ if $f_{ij} > 0$ 0 otherwise
Normalized Log	$(1 + \log f_{ij})/(1 + a_j)$ if $f_{ij} > 0$ 0 otherwise

Table 1: Local Weighting Schemes

Inverse Doc Frequency (IDF)	$log(N/n_i)$
Probabilistic Inverse	$log(N - n_i)/n_i$
Entropy	$1+\sum_{j=1}^{N}(f_{ij}/f_j)(log(f_{ij}/f_i)/log(N))$
Global Frequency IDF	f_i/n_i

Table 2: Global Weighting Schemes

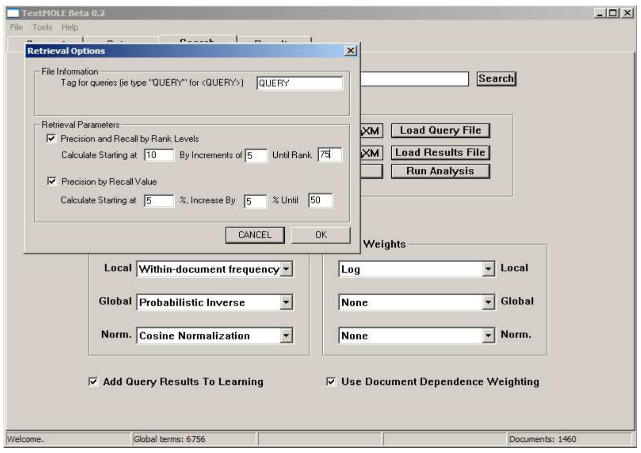
3.2.2 Global Weighting

Global weighting schemes are designed to indicate the relative importance of a term within the entire corpus. The global weighting schemes we have implemented are described in Table 2. The inverse document frequency is probably the most well-known, and it simply the number of documents in the data set, N, divided by the number of documents containing term i. This awards a higher weight to terms that appear in fewer documents and are thus better discriminators. The probabilistic inverse algorithm looks similar, but acts differently because terms that appear too frequently (in more than half the documents in the collection) will actually be awarded a negative weight. The entropy weighting scheme assigns terms a value between 0 and 1; it assigns a term the weight of 0 if it appears in every document, and it assigns a term the weight of 1 if it appears in only one document. Anything in between will be assigned a weight somewhere between 0 and 1. The net effect of this weighting is to award a higher weight to terms that appear less often in a small percentage of documents. The global frequency IDF algorithm awards a high weight to terms that have a higher than expected frequency (in relation to the number of documents containing the term). The minimum weight that the global frequency IDF algorithm can grant is 1, and there is no maximum.

3.2.3 Normalization

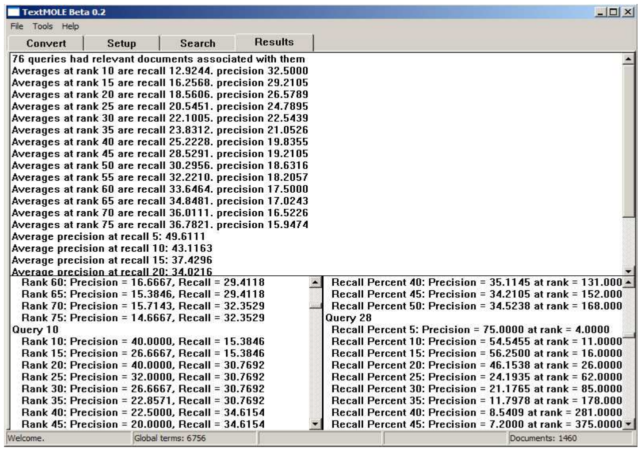
Document normalization, if specified by the user, is applied using the cosine algorithm, which is the most common form of normalization. This algorithm divides each element of the document vector (after local and global weights have been applied) by the magnitude of the vector, thereby forcing the length of each vector to be one. Normalization attempts to fix the inherent advantage that longer documents would get when calculating the cosine similarity between a document and query.

Figure 4: TextMOLE Multiple Query Screen



```
File Information
  Tag for queries (ie type "QUERY" for <QUERY>)  QUERY

Retrieval Parameters
  [✓] Precision and Recall by Rank Levels
      Calculate Starting at 10   By Increments of 5   Until Rank 75
  [✓] Precision by Recall Value
      Calculate Starting at 5   %, Increase By 5   % Until 50

Weights
  Local   Within-document frequency      Log    Local
  Global  Probabilistic Inverse          None   Global
  Norm.   Cosine Normalization           None   Norm.

[✓] Add Query Results To Learning     [✓] Use Document Dependence Weighting

Welcome.   Global terms: 6756   Documents: 1460
```

Figure 5: TextMOLE Multi-Query Results



```
76 queries had relevant documents associated with them
Averages at rank 10 are recall 12.9244. precision 32.5000
Averages at rank 15 are recall 16.2568. precision 29.2105
Averages at rank 20 are recall 18.5606. precision 26.5789
Averages at rank 25 are recall 20.5451. precision 24.7895
Averages at rank 30 are recall 22.1005. precision 22.5439
Averages at rank 35 are recall 23.8312. precision 21.0526
Averages at rank 40 are recall 25.2228. precision 19.8355
Averages at rank 45 are recall 28.5291. precision 19.2105
Averages at rank 50 are recall 30.2956. precision 18.6316
Averages at rank 55 are recall 32.2210. precision 18.2057
Averages at rank 60 are recall 33.6464. precision 17.5000
Averages at rank 65 are recall 34.8481. precision 17.0243
Averages at rank 70 are recall 36.0111. precision 16.5226
Averages at rank 75 are recall 36.7821. precision 15.9474
Average precision at recall 5: 49.6111
Average precision at recall 10: 43.1163
Average precision at recall 15: 37.4296
Average precision at recall 20: 34.0216
```

```
  Rank 60: Precision = 16.6667, Recall = 29.4118
  Rank 65: Precision = 15.3846, Recall = 29.4118
  Rank 70: Precision = 15.7143, Recall = 32.3529
  Rank 75: Precision = 14.6667, Recall = 32.3529
Query 10
  Rank 10: Precision = 40.0000, Recall = 15.3846
  Rank 15: Precision = 26.6667, Recall = 15.3846
  Rank 20: Precision = 40.0000, Recall = 30.7692
  Rank 25: Precision = 32.0000, Recall = 30.7692
  Rank 30: Precision = 26.6667, Recall = 30.7692
  Rank 35: Precision = 22.8571, Recall = 30.7692
  Rank 40: Precision = 22.5000, Recall = 34.6154
  Rank 45: Precision = 20.0000, Recall = 34.6154
```

```
  Recall Percent 40: Precision = 35.1145 at rank = 131.000
  Recall Percent 45: Precision = 34.2105 at rank = 152.000
  Recall Percent 50: Precision = 34.5238 at rank = 168.000
Query 28
  Recall Percent 5: Precision = 75.0000 at rank = 4.0000
  Recall Percent 10: Precision = 54.5455 at rank = 11.0000
  Recall Percent 15: Precision = 56.2500 at rank = 16.0000
  Recall Percent 20: Precision = 46.1538 at rank = 26.0000
  Recall Percent 25: Precision = 24.1935 at rank = 62.0000
  Recall Percent 30: Precision = 21.1765 at rank = 85.0000
  Recall Percent 35: Precision = 11.7978 at rank = 178.000
  Recall Percent 40: Precision = 8.5409 at rank = 281.0000
  Recall Percent 45: Precision = 7.2000 at rank = 375.0000
```

```
Welcome.   Global terms: 6756   Documents: 1460
```

3.2.4 *Measuring Retrieval Performance*

Precision and recall are often used to measure the quality of an information retrieval system. Precision is defined as the percentage of retrieved documents which are relevant to the query. Recall is the percentage of all relevant documents that were retrieved.

Precision and recall require the existence of collections that contain a group of documents, a set of standard queries, and a truth set for each query (a list of truly relevant documents). The TextMOLE includes the capability of running a set of queries for a collection and measuring precision and recall, based on a truth set that is provided. There are several open source collections that contain standard queries and truth sets, such as the SMART collections, which can downloaded from the SMART web site at Cornell University [8], and Text Retrieval Conference (TREC) collections, available from the Linquistic Data Consortium (LDC) [9]. Figures 4 and 5 show the TextMOLE operation using one such collection, CISI, from the SMART web site.

3.3 Zipf Graph

TextMOLE also provides a visualization of the data set by graphing the log of the frequency of each global term against the log of its rank (where the most frequent term is given a rank of 1, etc.). Due to the natural distribution of words in language, when all of a document's terms are plotted on a log scale, they create a near perfect plotting of a negatively-sloped line. The natural tendency of this line to form for any given collection of documents is known as Zipf's Law (see Figure 6).

Zipf's Law is unintuitive and therefore difficult for students to believe when they first encounter it. With the TextMOLE, an instructor can demonstrate Zipf's Law in a few minutes. It is often helpful to follow this demonstration with an assignment asking the students to show that Zipf's Law works for a collection of their own choosing. This assignment can utilize the TextMOLE, or students can be asked to develop a program to count words and use Excel to produce the graph.

4. DISCUSSION

When students first attempt to develop an Information Retrieval application for a first course in Information Retrieval (IR), they immediately run into issues that they have not faced in other courses. First and foremost is the need to parse text with little or no structure. Undergraduate programming assignments require infrequent use of text, and very few assignments force students to deal with unstructured text. A second problem that arises is the issue of size and time. In our experience, students have little patience with waiting for several minutes for a program to parse a moderate-sized file. They tend to assume the program is 'wrong' and/or reduce the size of the input file to a impractical size.

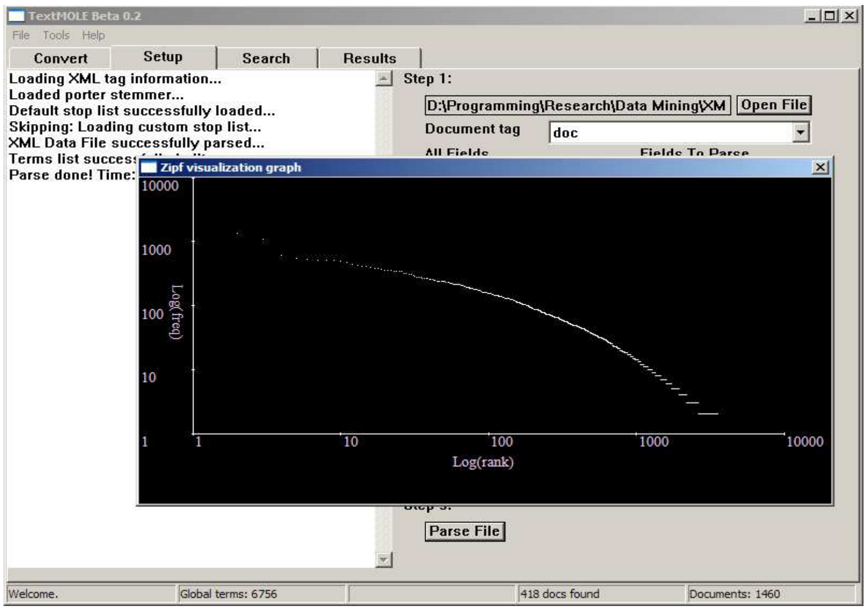
Weeks of a semester can be spent getting students through these issues (and the authors believe that this time is well invested). If an instructor wishes to move into a survey advanced topics in IR or text mining, the IR students are asked to implement a basic search and retrieval system and then the class moves on to other topics. Issues pertaining to weighting schemes or indexing options are mentioned in passing, and the impact these parameters have on retrieval performance tends to be overlooked by students.

TextMOLE can be used to develop an interim assignment which encourages robust exploration of these issues in indexing without requiring a large amount of programming. A week is a reasonable deadline for students to complete an analysis of a dataset using TextMOLE. Some programming will most likely be required unless the instructor provides sample datasets which are already in XML format.

Existing textbooks provide code [2] or refer students to existing retrieval systems, such as the SMART system [1, 7], but these systems do not tightly integrate indexing and retrieval, and are difficult for students to install and run. Furthermore, multiple indexing options and weighting schemes are not implemented (although excellent assignments that involve adding options to these systems can be developed).

Instructors for courses in Machine Learning (ML) or Artificial Intelligence (AI) may wish to include a segment on Data Mining or Textual Data Mining. It is impractical to expect students to overcome the programming issues mentioned above when only a week or two can be dedicated to a topic. The WEKA tool [11] can be used to model data mining tasks, but does not address text mining. TextMOLE can be used to fill this gap and provide students with a solid introduction to IR and Text Mining, which can be leveraged if students will be asked to complete a final project on some advanced topic in ML or AI.

TextMOLE was developed by an undergraduate student for his honors thesis. Adaptions and enhancements to the



```
Loading XML tag information...
Loaded porter stemmer...
Default stop list successfully loaded...
Skipping: Loading custom stop list...
XML Data File successfully parsed...
Terms list succes
Parse done! Time:
```

Step 1:

D:\Programming\Research\Data Mining\XM

Document tag: doc

Welcome. | Global terms: 6756 | 418 docs found | Documents: 1460

Figure 6: TextMOLE Zipf Graph

tool would make excellent semester or year long projects for seniors looking for capstone experiences in computer science.

5. CONCLUSIONS AND FUTURE WORK

Currently, the most practical use for the tool is classroom instruction. The simple and intuitive interface provides a wealth of information about a chosen corpus, and the source code can be easily understood by students. The tool provides an excellent medium to demonstrate the fundamentals of information retrieval. In particular, it shows how varying the indexing and weighting options for a corpus can impact query results.

Thus far only the foundation for indexing collections of documents and running queries has been developed. There are many advanced features available to be implemented. In the next version, we will include options for indexing using noun phrases or n-grams instead of words. The only method of data querying available in the current version is vector space retrieval, but more advanced analyses, such as dimensionality reduction techniques, will be added. Additional preprocessing support for other data formats will also be built into future versions of the program. We also plan to add support for other applications in text mining, such as emerging trend detection, first-story detection and classification. We will maintain our commitment to developing code which can be understood and modified by undergraduate computer science majors.

6. ACKNOWLEDGMENTS

We would like to express our gratitude to Ursinus College for funding support during the Summer of 2005.

7. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley/ACM Press, New York, 1999.

[2] R. K. Belew. *Finding Out About.* Cambridge University Press, 2000.

[3] E. Chisholm and T. G. Kolda. New term weighting formulas for the vector space method in information retrieval. Technical Report ORNL-TM-13756, Oak Ridge National Laboratory, 1999.

[4] D. Harman. An experimental study of factors important in document ranking. In *Proceedings of the Ninth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Pisa, Italy, 1986.

[5] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge, MA, 1999.

[6] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[7] G. Salton. *The SMART Retrieval System–Experiments in Automatic Document Processing.* Prentice Hall, Englewood Cliffs, New Jersey, 1971.

[8] ftp.cs.cornell.edu/pub/smart.

[9] trec.nist.gov/data/docs_eng.html.

[10] C. van Rijsbergen. *Information Retrieval.* Department of Computer Science, University of Glasgow.

[11] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition.* Morgan Kaufmann, 2005.