NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AUTOMATIC TEXT CATEGORIZATION APPLIED TO E-MAIL

by

Scott R. Hall

September 2002

Thesis Advisor: Neil Rowe
Second Reader: Thomas Otani

Approved for public release; distribution is unlimited
**Title:** Automatic Text Categorization Applied to E-Mail

**Authors:** Hall, Scott R.

**Abstract:**

The author developed an automatic text categorization approach and investigated its application upon categorizing emails. The categorization approach is derived from an instance-based learning method that explores conditional probabilities of particular words. The effectiveness of the author’s categorization approach using collections from a set of emails is then evaluated and assigned a numerical score based upon precision and recall. Precision was 65% while recall was 17%. The author’s experiments indicated automatic categorization of incoming emails at the client level can categorize email, but is difficult when not using a standardized corpus. Word frequency is valuable, but should be used in combination with other methods such as phrase extraction for a higher level of performance.

**Subject Terms:** Text Categorization, Automatic Classification, Java Text Processing

**Number of Pages:** 60

**Security Classification:** Unclassified

---

**Report Date:** September 2002

**Performing Organization:** Naval Postgraduate School

**Sponsoring/Monitoring Agency:** N/A

**Distribution Statement:** Approved for public release; distribution is unlimited

**Abstract Limitation:** UL

---

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18
AUTOMATIC TEXT CATEGORIZATION APPLIED TO E-MAIL

Scott R. Hall
Major, United States Marine Corps
B.B.A., University of Oklahoma, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2002

Author: Scott R. Hall

Approved by: Neil Rowe
Thesis Advisor

Thomas Otani
Second Reader

LCDR Chris Eagle
Chairman, Department of Computer Science
THIS PAGE INTENTIONALLY LEFT BLANK
ABSTRACT

The author developed an automatic text categorization approach and investigated its application upon categorizing emails. The categorization approach is derived from an instanced-based learning method that explores conditional probabilities of particular words. The effectiveness of the author’s categorization approach using collections from a set of emails is then evaluated and assigned a numerical score based upon precision and recall. Precision was 65% while recall was 17%. The author’s experiments indicated automatic categorization of incoming emails at the client level can categorize email, but is difficult when not using a standardized corpus. Word frequency is valuable, but should be used in combination with other methods such as phrase extraction for a higher level of performance.
# TABLE OF CONTENTS

## I. INTRODUCTION
- A. BACKGROUND ................................................................. 1
- B. THESIS OBJECTIVE .......................................................... 1
- C. OUTLINE OF THESIS ....................................................... 2

## II. OTHER WORK ON TEXT CATEGORIZATION
- A. INTRODUCTION ............................................................... 3
- B. LITERATURE REVIEW ....................................................... 3

## III. DESCRIPTION OF APPLICATION
- A. DATA ............................................................................... 7
- B. EVALUATION MEASURES ............................................... 11
- C. METHODOLOGY ............................................................... 12

## IV. DESCRIPTION OF TEXT CATEGORIZER PROGRAM
- A. PROGRAM STRUCTURE .................................................... 15
- B. PROGRAM PERFORMANCE ............................................. 15

## V. EXPERIMENT RESULTS
- A. INTRODUCTION ............................................................... 19
- B. DATA ANALYSIS .............................................................. 21

## VI. CONCLUSIONS AND RECOMMENDATIONS
- A. CONCLUSIONS ............................................................... 23
- B. RECOMMENDATIONS ....................................................... 23

### APPENDIX A. CLUEWORDS SAMPLE OUTPUT ................................. 25
### APPENDIX B. CLUEWORDS PROGRAM ......................................... 27
### APPENDIX C. RATEDOCS PROGRAM ......................................... 33
### APPENDIX D. SAMPLE RUN FROM RATE DOCS ............................ 37
### APPENDIX E. PARSER PROGRAM ............................................. 39
### LIST OF REFERENCES .................................................................. 43
### INITIAL DISTRIBUTION LIST .................................................. 45
THIS PAGE INTENTIONALLY LEFT BLANK
LIST OF FIGURES

Figure 1. Architecture for the Program. .................................................................16
LIST OF TABLES

Table 1. Results for Parameter Selection in (Lam, Ruiz, Srinivasan 1999) .................4
Table 2. Training and Testing Data Characteristics.........................................................7
Table 3. Category Descriptions and Possible Keywords Identifying These Categories.................................................................10
Table 4. Number of Texts Per Category in Training and Test Sets. .........................11
Table 5. Evaluation Measures .....................................................................................11
Table 6. Porter Stemming Algorithm Behavior ............................................................12
Table 7. Sample Word Frequency Data: Frequencies from 336,310 Documents in the 1GB TREC Volume 3 Corpus 125,720,891 Total Word Occurrences; 508,209 Unique Words. .................................................................13
Table 8. Processing Time ............................................................................................17
Table 9. Example Cluewords in Category Personal .......................................................19
Table 10. Example Cluewords in Category Fiscal .........................................................19
Table 11. Example Cluewords in Category Requests .....................................................19
Table 12. Results of Categorizing Test Set Documents. With Subject Line Code Included......................................................................................................................................20
Table 13. 19x19 Confusion Matrix and Potential Category Clusters. .........................21
Table 14. Actual Number of Words Identified for Positive Examples Versus Non-Examples........................................................................................................................................22
ACKNOWLEDGMENTS

The author wishes to express a special thanks to Dr. Neil Rowe whose encouragement, intelligent counsel, and practical suggestions were crucial to the successful outcome of this Graduate Thesis. Appreciation is also due to Dr. Thomas Otani for his teaching Java to the author from the ground up.

This statement of acknowledgement would be incomplete without a formal expression of sincere appreciation and gratitude of the author’s friends and family for providing the assistance and encouragement needed to complete the task. A special thanks is due to Steve Simmons for the numerous ad-hoc technical discussions and tackling of the programming problems that helped this authors understanding.

Finally, I would like to say to my wife, Michelle; I love you and thank you for the dedicated support you have given me throughout the writing of this thesis. To my daughter Abigail you have given me the inspiration to finish so we can play together, I love you. As a professing Christian I would be remiss if I did not say thank you Lord.
I. INTRODUCTION

A. BACKGROUND

Email has become the standard for fast, inexpensive and easily accessible communication. The explosive growth of email is affecting everyone in the Department of Defense as well as the civilian work environment. Its largest impact is on management and record-keeping personnel. Typical military users receive between 30 – 70 emails a day depending on their rank and billet (Marsan, 2002). If one attaches spreadsheets, documents, presentations, graphics and executable programs then email gains even more value. Because of its ease of use, email has become an integral part of military-organization daily operations. All of the information in unclassified and classified emails is also a treasure trove of operational data.

B. THESIS OBJECTIVE

This thesis seeks to improve the organization of individual user’s email by implementing an automated categorizer for email. The author seeks to try to eliminate the large amounts of manual email categorization that is currently done by many users. This could be useful to military personnel due to efficiency, privacy, and high-turnover concerns. Turnover of military personnel happens every 2-3 years and frequently old email is simply deleted rather than organized and used to document valuable operational processes and data. Generally speaking within the DoD there are no formal filing or retention policies for email. There are guidelines such as the DoD Directive 5012.2, but these deal with large-scale records management.

Old email messages are required for numerous purposes. These include day-to-day business operations, and requests for, historical financial information, activities, logistics, etc. When each individual decides which email messages to retain or delete, much information is hidden from the rest of the organization. Email storage of this kind is scattered in personal archives. Although servers can store these messages, the number of messages is constantly increasing. If backup tapes are not kept for retrieving old messages, and an individual goes on leave or is unavailable, critical information in his email is unavailable. This suggests organizational-level archiving and categorizing, but what can be done at the user level to help organize this information?
With the average user manually archiving an estimated 300 megabytes (MB) of email a year, categorizing methods can vary dramatically in their effectiveness and overall organization (Ferris, 1999). Our primary goal of this research was to survey existing methods and determine a method or a combination of methods that would work well for email categorization. A secondary goal of this research was to create a tool to accurately and quickly categorize and archive email messages at the local user level.

This thesis will attempt to answer the following questions:

• Can machine-learning programs accurately categorize e-mail?
• What are the strengths and weaknesses of automated categorizers?

C. OUTLINE OF THESIS

Chapter II describes previous attempts at automated categorizing and other similar problems in text retrieval. Chapter III gives a description of the structure and components of the program. A detailed description of the data and corpus is also given. Chapter IV provides a description of a categorization program that we developed. Chapter V discusses the program’s performance and the accuracy of its results. Chapter VI reviews the program’s achievements and major weaknesses.
II. OTHER WORK ON TEXT CATEGORIZATION

A. INTRODUCTION

Text categorization has become a very active research topic over the last few years. Many of the approaches seek to categorize documents of the Internet. In this thesis email is the document and specifically the text within the email. Categories can be summarized using phrases, words, or numerically. Traditionally, a domain expert, usually a librarian, does text categorization manually. Documents are read by the expert and then placed in the appropriate category. To eliminate the large amount of manual effort required, we could use automatic categorization that learns automatically from using training examples. The classic approach is to assign weights to particular words in particular categories; the inferred category of a document is the one with the highest weighted sum (Witten, Frank, 2000).

Two categorization techniques used are instance-based learning and Naïve-Bayes probabilistic classification. Instance-based learning methods begin with a particular example and generalize it to cover other similar examples in the same category. The Naïve-Bayes approach uses the conditional probabilities of categories given a word to estimate the probabilities of categories given an email document; this model assumes word independence. Typically a list of “stop words” to be ignored and some sort of destemming algorithm are used to help normalize the word list.

B. LITERATURE REVIEW

Lam, Ruiz, and Srinivasan investigated whether automatic categorization will have better retrieval performance than that achieved using manual categorization applied to medical documents (Lam, Ruiz, Srinivasan 1999). They analyzed the retrieval performance on test queries to gain insights on the interaction of their categorizer and text retrieval. The first part of their work dealt with automatic categorization including a category-extraction process. For their test documents they use a corpus of medical documents from the MEDLINE database that is referred to as the HERSH corpus.

The authors ran a series of experiments on parameter selection to provide a metric and categorization results. Their results are broken down into category and document
perspectives. The category perspective results are related to sizes of categories ranging from 10 to 60 categories. Three different parameters were tested: C0, C35 and C50. C0 used all manually assigned categories that existed in the training set and test set. C35 and C50 limit the number of categories to those that have a document frequency greater than 35 or 50 per category. The document frequency is the number of documents that a specific category is assigned to. The F1 score is a weighted combination of recall and precision, with the scores being averaged to determine a mean. Their results for parameter selection can be seen in Table 1.

<table>
<thead>
<tr>
<th>Run</th>
<th>Parameter selection based on training set</th>
<th>N</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of categories</td>
<td>F1 score</td>
<td></td>
</tr>
<tr>
<td>C0</td>
<td>641</td>
<td>0.258</td>
<td>5</td>
</tr>
<tr>
<td>C35</td>
<td>58</td>
<td>0.468</td>
<td>5</td>
</tr>
<tr>
<td>C50</td>
<td>43</td>
<td>0.509</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 1. Results for Parameter Selection in (Lam, Ruiz, Srinivasan 1999).

The results indicate that as the frequency threshold on the category set increases, the mean F1 score improves. N represents the number of documents while M was the number of categories.

Yang did a comparative evaluation of statistical approaches to text categorization (Yang, 1998). The author uses several versions of the Reuters newswire corpus of 20,000 documents to evaluate the categorization methods of k-nearest neighbors, simple word matching, decision trees, Naïve-Bayes, inductive-rule learning in disjunctive normal form, neural networks, Rocchio, linear least-squares fit, and “sleeping experts”. The authors found that linear least squares fit performed best.

In addition to experimenting with thresholding techniques, Yang concluded that variability on the performance of classifiers with collection is common. Although the Word approach, which looks at single word frequency, had increased performance when changing from a labeled to an unlabeled corpus, it was still out performed by other methods such as kNN and LLSF.
Moens and Dumortier applied text categorization to magazine articles to study the effects of selection of feature words and proper names (Marie-Francine, Dumortier, 2000). The authors use a standardized approach of stop-word removal and then select keywords by applying statistical weights to the remaining words after stop words are taken out. For proper names, words with capitalization are given a heavier weighting. Terms with a calculated weight above 0.4 conditional probability are selected. The authors also apply the technique of “zoning”, which is the selection of word examples that are in close proximity to other word examples within the document. The results of the Moens and Dumortier compare a Bayesian independence classifier to the Rocchio algorithm and a $X^2$ algorithm. The $X^2$ algorithm is used to test how closely a set of observed frequencies corresponds to a set of expected frequencies. The observed frequencies are the number of texts relevant or non-relevant for the text category that contain the feature word. The authors conclude that the $X^2$ algorithm worked best with a recall of 0.73 and precision of 0.64 versus the Bayesian method recall of 0.58 and precision of 0.61 and the Rocchio algorithm recall of 0.64 and precision of 0.57.

Salton and Buckley propose a similar method to show how similar one document is to a query document by statistically weighting terms within the document (Salton, Buckley, 1988). The authors compare results of eight different term-weighting methods on different collections of documents. They make recommendations on query and document vectors concerning the term-frequency component, the collection frequency and the normalization component. The authors conclude that for short queries each term is important and query-term weights are preferred. When dealing with document vectors the authors conclude that for technical vocabulary, an enhanced frequency-weighting scheme should be used which places terms automatically between 0.5 and 1.0. Our application involves short technical documents and can use this approach by using individual term weighting rather than using a similarity calculation.
III. DESCRIPTION OF APPLICATION

In this section we describe our application, and present an algorithm for categorizing email documents using a probabilistic model. The algorithm uses count data, the frequency of the terms in a document. Our approach relies on keyword clues. A training process identifies categories for new documents from pre-categorized examples. The categorization technique used in the algorithm is a linear numeric prediction model (Witten, Frank, 2000).

A. DATA

A total of 737 emails were used to train and test the categorizer. Table 3 provides the data characteristics of the emails. Some emails were previously saved with an “.html” extension and others were saved with a “.txt” extension. HTML tags were identified and included in the stop-word list. The collection of emails was from the author’s personal work archives. Approximately 20% pertain to the author’s experience as a Supply Officer; the remaining 80% were collected during the author’s experience as a graduate student.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of emails</td>
<td>737</td>
</tr>
<tr>
<td>Total number of words in text corpus</td>
<td>31,593</td>
</tr>
<tr>
<td>Total number of words after destemming and stopword removal</td>
<td>20,115</td>
</tr>
<tr>
<td>Total number of unique words after destemming, stopword removal and extraction of HTML tags and other special characters</td>
<td>~ 12,000</td>
</tr>
<tr>
<td>Number of words per email document</td>
<td>45 -1082</td>
</tr>
</tbody>
</table>

Table 2. Training and Testing Data Characteristics.

Table 3 shows the specific category descriptions and example relevant words which the author thinks the program should choose as keyword clues relating to a category. The author identified these categories by placing them in logical categories.
according to their content. Except for categories #4, #5, #10, #15, and #17, the categories could be applied to other military-service emails as well as civilian-business emails.

<table>
<thead>
<tr>
<th>Category # 1: Classes</th>
<th>Emails with administrative course material.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Key Words</td>
<td>class, info, course info, homework, homework problems, answers</td>
</tr>
<tr>
<td>Category # 2: Grades</td>
<td>Emails with information about grades and transcripts.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>grades, registrar, python, final grade, homework grade, test grade</td>
</tr>
<tr>
<td>Category # 3: Personal</td>
<td>Emails with information received from the author’s family members and other matters he deemed not directly connected to his work environment.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>trip, Michelle, Abby, love, usmc, thanks, dear</td>
</tr>
<tr>
<td>Category # 4: 3270</td>
<td>Emails dealing with connectivity problems to a mainframe computer using 3270 emulation software.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>3270, ACID, password, connectivity, lack, mainframe, service (ACID – access control identification)</td>
</tr>
<tr>
<td>Category # 5: Bwd Mess</td>
<td>Emails that involved the authors job as the Marine Officer in charge of collecting wardroom dues onboard the USS Belleauwood from 11/98-6/99.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>money, payment, dues, mess, receipt, check</td>
</tr>
<tr>
<td>Category # 6: Equipment</td>
<td>Emails that dealt with equipment issues.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>swap, equipment, truck, weapons, parts, lead-time, fix, gear</td>
</tr>
<tr>
<td>Category # 7: Equipment Allowance</td>
<td>Emails that deal with equipment allowances and what a unit was reporting to have.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>OH, on-hand, own, equipment, T/O, temp, shortage, overage (OH and T/O – stand for on hand and table of organization, which is the structuring of a unit)</td>
</tr>
<tr>
<td>Category # 8: Equipment Readiness</td>
<td>Emails that pertain to the physical condition of a piece of equipment and whether it was working properly or not.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>maintenance, parts, deadline, fix, repair, leadtime, running, MIMMS (Marine Corps Integrated Maintenance Mgt. System)</td>
</tr>
<tr>
<td>Category # 9: Exercises</td>
<td>Emails that pertain to military exercises the author partook in and various problems that he resolved or worked on.</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>Cobra-Gold, billeting, exercise, funding, travel, planning, meeting, Y2K</td>
</tr>
<tr>
<td>Category # 10: Expeditor</td>
<td>Emails that pertain to a person whose job involved the explicit tracking of equipment through the transportation pipeline.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>tracking, equipment, package, arrival, Carl, time, where</td>
</tr>
<tr>
<td>Category # 11: Fiscal</td>
<td>Emails concerning payment and disbursal matters, contracts, and equipment receipts.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>payment, due, money, SABRS, JON, financial, authority (SABRS – standard accounting and budgeting requirement system; JON – job order number)</td>
</tr>
<tr>
<td>Category # 12: General</td>
<td>Emails concerning general administrative purposes.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>administrative, requirement, meeting, turn-in, due</td>
</tr>
<tr>
<td>Category # 13: Maintenance</td>
<td>Emails that pertain to the physical condition of a piece of equipment, whether it was working properly or not, and the parts status for a piece of equipment.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>maintenance, parts, deadline, fix, repair, lead-time, running, MIMMS, status, up (MIMMS - Marine Corps Integrated Maintenance Mgt. System)</td>
</tr>
<tr>
<td>Category # 14: Miscellaneous</td>
<td>Emails that were of mixed purposes. They are primarily differentiated from category #12 by the variety within each email and that category #12 had a general administrative theme.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>odd, here, fun, get, read, keep, future, misc</td>
</tr>
<tr>
<td>Category # 15: NBC</td>
<td>Emails that involved Nuclear, Biological or Chemical (NBC) supply issues.</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>mask, parts, atropine, injector, NBC, filter, gas, chamber</td>
</tr>
<tr>
<td>Category # 16: records</td>
<td>Emails concerning supply record administration issues.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>Possible Key Words</td>
<td>CMR, on-hand, drop, add, quantity, description, account, inventory, count (CMR – consolidated memorandum receipt)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category # 17: requests</th>
<th>Emails concerning supply requests for equipment or the purchasing of administration supplies.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Key Words</td>
<td>computer, request, get, date, buy, purchase, money, contract</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category # 18: shipboard billeting</th>
<th>Emails concerning issues involving billeting or berthing for Marine Officers on board the USS Belleauwood. Most emails revolve around room assignments.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Key Words</td>
<td>room, assignment, billeting, berthing, Belleauwood, officer, assignment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category # 19: tech info</th>
<th>Emails that include technical issues focused around the area of computer science. This area was differentiated from category # 1 by its lack of specificity in many cases.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Key Words</td>
<td>networking, computer, artificial, intelligence, software, web, Internet</td>
</tr>
</tbody>
</table>

Table 3. Category Descriptions and Possible Keywords Identifying These Categories.

The exact number of texts used for each category in the training and test sets are given in Table 4 below. The training and test sets were formed by placing 80% of each category into the training set, and the remaining 20% into the test set. A constraint was the limited number of examples in some of the categories. As with much text categorization, a central problem is the lack of standard data collections.
<table>
<thead>
<tr>
<th>Category</th>
<th>Number of email texts for the training set</th>
<th>Number of email texts for the test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shipboard billeting</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>Requests</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>Records</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NBC</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>Maintenance</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>General</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Fiscal</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>Expeditor</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Exercises</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Equipment Readiness</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>Equipment Allowances</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Equipment</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>BWD Mess</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3270</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Personal</td>
<td>242</td>
<td>61</td>
</tr>
<tr>
<td>Grades</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>Classes</td>
<td>104</td>
<td>26</td>
</tr>
<tr>
<td>Tech Info</td>
<td>88</td>
<td>22</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>585</strong></td>
<td><strong>152</strong></td>
</tr>
</tbody>
</table>

(Total Emails: 737)

Table 4. Number of Texts Per Category in Training and Test Sets.

B. EVALUATION MEASURES

The author uses conventional measures of recall and precision to measure categorization accuracy. They are computed by selecting the highest value for the returned email:

\[
\text{recall} = \frac{\text{emails correctly assigned to a category}}{\text{emails in a category}}
\]

\[
\text{precision} = \frac{\text{emails correctly assigned to a category}}{\text{emails assigned to a category}}
\]

Table 5. Evaluation Measures.
The author also displays the results in the form of two 19x19 confusion matrices that show the true categories plotted against actual categories chosen after being run through our categorizer.

C. METHODOLOGY

Given an email document, an independent binary classifier compares values and chooses the single category of highest value. Two methods of preprocessing text are used. First, a word “destemmer” algorithm is used (Rowe, 1998). The algorithm removes suffixes on an English word to regularize its forms. A sample behavior of Porter’s stemming algorithm can be seen in Table 6 below:

<table>
<thead>
<tr>
<th>Word</th>
<th>Porter Stemming Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>believes</td>
<td>believ</td>
</tr>
<tr>
<td>working</td>
<td>work</td>
</tr>
<tr>
<td>starting</td>
<td>start</td>
</tr>
<tr>
<td>playfully</td>
<td>play</td>
</tr>
</tbody>
</table>

Table 6. Porter Stemming Algorithm Behavior.

In addition to destemming words, a “stop word” removal list is used (Rowe, 1998). The stop-word list consists of 700 common words such as “a”, “and”, “the”, “of”, etc., that are generally non-informative and can be removed to improve categorization. Some HTML tags and special characters were also added to the stop-word list to eliminate redundant non-useful characters such as BR, TR, and other commonly used markup tags. Two of the most common words, “of” and “the”, account for 10% of word occurrences in most documents (Mooney, 2002). Sample stop word frequencies can be seen in Table 7.
<table>
<thead>
<tr>
<th>Frequent word</th>
<th>Number of occurrences</th>
<th>Percentage of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>7,398,934</td>
<td>5.9</td>
</tr>
<tr>
<td>of</td>
<td>3,893,790</td>
<td>3.1</td>
</tr>
<tr>
<td>to</td>
<td>3,364,653</td>
<td>2.7</td>
</tr>
<tr>
<td>and</td>
<td>3,320,687</td>
<td>2.6</td>
</tr>
<tr>
<td>in</td>
<td>2,311,785</td>
<td>1.8</td>
</tr>
<tr>
<td>is</td>
<td>1,559,147</td>
<td>1.2</td>
</tr>
<tr>
<td>for</td>
<td>1,313,516</td>
<td>1.0</td>
</tr>
<tr>
<td>The</td>
<td>1,144,860</td>
<td>0.9</td>
</tr>
<tr>
<td>that</td>
<td>1,066,503</td>
<td>0.8</td>
</tr>
<tr>
<td>said</td>
<td>1,027,713</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 7. Sample Word Frequency Data: Frequencies from 336,310 Documents in the 1GB TREC Volume 3 Corpus 125,720,891 Total Word Occurrences; 508,209 Unique Words.
IV. DESCRIPTION OF TEXT CATEGORIZER PROGRAM

A. PROGRAM STRUCTURE

The text categorizer was written in Java. The overall architecture for the program can be seen in Figure 1. The program starts by accepting email documents in text document format. The training set is manually categorized and then both sets are tokenized. The email document is then run through a destemming program and removal of any of the 700 stop words is done. All capitalized letters are made into lower case. The training set is used in the calculation of clue probabilities. If a word remains after stop-word removal it must appear a minimum of 10 times to have its probability calculated. Additionally, probabilities are calculated by viewing two subdirectories labeled, “yes” and “no” and finding the conditional probability of a “yes” given the occurrence of particular. The test set involves the calculation of 19 weighted sums for each document.

The “ClueWords” program was adapted with minor changes from another program, “GetClueProbs” (Rowe, 1998). The program was modified to extract stop words. The RateDocs program was modified to check for the “Subject” line of an email and increment the overall word count. The “RateDocs” program takes the weighted sum of the number of occurrences of each clueword, and then chooses the category of highest total weight.

B. PROGRAM PERFORMANCE

All code was written in Java using JDK1.3.1_02 Java Virtual Machine release. The programs were executed on a Pentium II Processor Intel MMX chip running Windows 98 with 160 MB of RAM. Table 8 shows processing time.
Email training set
Destemming
Stop Words Removed
Calculate Conditional Word Probabilities
Store Conditional Word Probabilities

Email test set
Tokenization
Destemming
Stop Words Removed
Calculate Document Total Weight Per Category
Store Document Weights Per Category

Actions Performed in dotted lined box are performed by ClueProbs program.

Actions Performed in dotted lined box are performed by RateDocs program.

Figure 1. Architecture for the Program.
<table>
<thead>
<tr>
<th>Train/Test</th>
<th>Number of documents</th>
<th>Real total time in seconds</th>
<th>Real time Per Document in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>585</td>
<td>225</td>
<td>4.2</td>
</tr>
<tr>
<td>Test</td>
<td>152</td>
<td>165</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 8. Processing Time.
V. EXPERIMENT RESULTS

A. INTRODUCTION

After destemming and eliminating stop-words, the training set consisted of 585 different email documents containing more than 30,000 words including duplicates. Minimum word counts were set at 5, 10 and 15 words per document and run through our clue-probability program. The number of clue words after destemming, elimination of stop-words, and changing of upper-case letters to lower-case letters ranged from 470 to 1,866 over the different categories. The top five keywords in conditional probability for three sample categories are shown below in Tables 9-11. “Personal” is a very large category with lots of keywords and high probabilities. “Fiscal” is a medium-sized category with some high-ranking clue words, and some important low ones. “Requests” is a small-sized category with low-ranked key words.

<table>
<thead>
<tr>
<th>% Prob</th>
<th># in category (yes)</th>
<th># not in category (no)</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99</td>
<td>515</td>
<td>5</td>
<td>hotmail</td>
</tr>
<tr>
<td>0.98</td>
<td>97</td>
<td>1</td>
<td>resort</td>
</tr>
<tr>
<td>0.98</td>
<td>127</td>
<td>2</td>
<td>sooner90</td>
</tr>
<tr>
<td>0.96</td>
<td>64</td>
<td>2</td>
<td>Love</td>
</tr>
<tr>
<td>0.94</td>
<td>149</td>
<td>9</td>
<td>her</td>
</tr>
</tbody>
</table>

Table 9. Example Cluewords in Category Personal.

<table>
<thead>
<tr>
<th>% Prob</th>
<th># in category (yes)</th>
<th># not in category (no)</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.92</td>
<td>36</td>
<td>3</td>
<td>ABC*</td>
</tr>
<tr>
<td>0.51</td>
<td>23</td>
<td>22</td>
<td>fiscal</td>
</tr>
<tr>
<td>0.27</td>
<td>15</td>
<td>39</td>
<td>money</td>
</tr>
<tr>
<td>0.22</td>
<td>14</td>
<td>47</td>
<td>spend</td>
</tr>
<tr>
<td>0.21</td>
<td>15</td>
<td>54</td>
<td>finance</td>
</tr>
</tbody>
</table>

Table 10. Example Cluewords in Category Fiscal.

<table>
<thead>
<tr>
<th>% Prob</th>
<th># in category (yes)</th>
<th># not in category (no)</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.53</td>
<td>15</td>
<td>13</td>
<td>mimm</td>
</tr>
<tr>
<td>0.50</td>
<td>8</td>
<td>8</td>
<td>laptop</td>
</tr>
<tr>
<td>0.27</td>
<td>5</td>
<td>13</td>
<td>gear</td>
</tr>
<tr>
<td>0.25</td>
<td>4</td>
<td>12</td>
<td>machine</td>
</tr>
<tr>
<td>0.12</td>
<td>34</td>
<td>230</td>
<td>request</td>
</tr>
</tbody>
</table>

Table 11. Example Cluewords in Category Requests.
Our classifiers were tested upon 152 new, previously unseen email texts. Table 12 shows recall and precision for the test set. Table 13 shows a confusion matrix of 19 categories and the 152 test set documents. This shows which categories were “confused” with one another and which categories were clearly identified.

<table>
<thead>
<tr>
<th>Category</th>
<th>Recall</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>0.77</td>
<td>0.18</td>
</tr>
<tr>
<td>Grades</td>
<td>1.00</td>
<td>0.05</td>
</tr>
<tr>
<td>Personal</td>
<td>0.18</td>
<td>1.00</td>
</tr>
<tr>
<td>3270</td>
<td>0.67</td>
<td>0.07</td>
</tr>
<tr>
<td>BWD Mess</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>Equipment</td>
<td>0.50</td>
<td>0.03</td>
</tr>
<tr>
<td>Equipment Allowance</td>
<td>0.67</td>
<td>0.05</td>
</tr>
<tr>
<td>Equipment Readiness</td>
<td>1.00</td>
<td>0.12</td>
</tr>
<tr>
<td>Exercises</td>
<td>1.00</td>
<td>0.06</td>
</tr>
<tr>
<td>Expeditor</td>
<td>0.50</td>
<td>0.06</td>
</tr>
<tr>
<td>Fiscal</td>
<td>0.80</td>
<td>0.10</td>
</tr>
<tr>
<td>General</td>
<td>1.00</td>
<td>0.05</td>
</tr>
<tr>
<td>Maintenance</td>
<td>1.00</td>
<td>0.07</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>0.80</td>
<td>0.18</td>
</tr>
<tr>
<td>NBC</td>
<td>0.50</td>
<td>0.17</td>
</tr>
<tr>
<td>Requests</td>
<td>Not enough records</td>
<td>Not enough records</td>
</tr>
<tr>
<td>Shipboard Billeting</td>
<td>0.80</td>
<td>0.08</td>
</tr>
<tr>
<td>Info_Tech</td>
<td>0.32</td>
<td>0.78</td>
</tr>
<tr>
<td>Average</td>
<td>0.65</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Table 12. Results of Categorizing Test Set Documents. With Subject Line Code Included.
B. DATA ANALYSIS

Results indicate that an average of 65% of all documents were correctly classified into their respective category. Of the 19 categories, 15 had greater than 50% probability of being properly classified. The remaining 4 categories were not properly classified for several reasons. In the case of categories #4 (Personal), and #19 (Info_Tech), these were frequently confused with one another and with category #14 (Miscellaneous). For categories #5 (BWD Mess) and #16 (Records) there were not enough examples to train on. Category #19 (Info Tech) was often confused with category #1 (Classes). Larger categories had better precision; smaller categories demonstrated higher recall. Average recall rates were acceptable, but precision rates were disappointing and can be contributed to categories clustered together.

Due to the unique nature of the corpus, each of the categories seemed to have certain cluewords that only helped it. In some categories stop words could have been good discriminators, such as category #3 (Personal) where the words “can” and “do” frequently show up.

---

Table 13. 19x19 Confusion Matrix and Potential Category Clusters.

* Not enough data to accurately categorize
Results with double weighting of the document “Subject” line show minimal increase in overall success probability. Table 14 shows actual number of words in categories as compared against the number of non-example documents. The table identifies the low number of training examples for smaller categories such as #5 (BWD Mess).

<table>
<thead>
<tr>
<th>Category Name</th>
<th>Actual # words in category (yes – examples)</th>
<th>Actual # of words in category (no – examples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>78,939</td>
<td>562,272</td>
</tr>
<tr>
<td>Grades</td>
<td>25,952</td>
<td>617,491</td>
</tr>
<tr>
<td>Personal</td>
<td>417,982</td>
<td>239,196</td>
</tr>
<tr>
<td>3270</td>
<td>1,634</td>
<td>638,416</td>
</tr>
<tr>
<td>BWD Mess</td>
<td>237</td>
<td>639,813</td>
</tr>
<tr>
<td>Equipment</td>
<td>2126</td>
<td>637,924</td>
</tr>
<tr>
<td>Equipment Allowance</td>
<td>1945</td>
<td>638,105</td>
</tr>
<tr>
<td>Equipment Readiness</td>
<td>1901</td>
<td>638,149</td>
</tr>
<tr>
<td>Exercises</td>
<td>481</td>
<td>639,569</td>
</tr>
<tr>
<td>Expeditor</td>
<td>486</td>
<td>639,564</td>
</tr>
<tr>
<td>Fiscal</td>
<td>3757</td>
<td>636,293</td>
</tr>
<tr>
<td>General</td>
<td>538</td>
<td>639,512</td>
</tr>
<tr>
<td>Maintenance</td>
<td>881</td>
<td>639,169</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>3332</td>
<td>636,718</td>
</tr>
<tr>
<td>NBC</td>
<td>685</td>
<td>639,365</td>
</tr>
<tr>
<td>Records</td>
<td>65</td>
<td>639,985</td>
</tr>
<tr>
<td>Requests</td>
<td>3233</td>
<td>640,050</td>
</tr>
<tr>
<td>Shipboard Billeting</td>
<td>1500</td>
<td>641,783</td>
</tr>
<tr>
<td>Info_Tech</td>
<td>115,961</td>
<td>541,280</td>
</tr>
</tbody>
</table>

Table 14. Actual Number of Words Identified for Positive Examples Versus Non-Examples.
VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis examined automatic text categorization of email documents. The use of keywords and their conditional probabilities was the primary method used. Final recall and precision results were 65% and 17% respectively. A stop-word list and destemmer program proved to be very helpful when dealing with text categorization.

B. RECOMMENDATIONS

This thesis could be extended by incorporating term phrases to improve categorization. If possible, a more standardized corpus of text should be used with approximate equal number of documents per category. A program to accurately strip out all HTML characters and other special characters for non-text would be helpful. Finally, a program or method to deal with personal names might prove very helpful in some categories.
APPENDIX A. CLUEWORDS SAMPLE OUTPUT

0.03878220540933329 overall probability, 18146 yes examples, 449749 no examples,
0.0 0 12 shape
0.0 0 17 shall
0.024 3 122 write
0.0 0 72 friend
0.0 0 37 certificate
0.0 0 14 comfort
0.0 0 17 netscape
0.0 0 43 bwlogu
0.00392156862745098 1 254 usmc
0.0 0 14 considerate
0.05834683954619125 36 581 monterey
0.0 0 13 justin
0.06666666666666667 6 84 here
0.0 0 40 lejeun
0.0 0 34 hell
0.0 0 83 effect
0.0 0 15 comment
0.0 0 51 sans-serif
0.0125 1 79 hear
0.0578206785767235 78 1271 head
0.0 0 17 friday
0.0 0 17 urge
0.06481481481481481 7 101 strategy
0.06484641638225255 57 822 subject
0.17073170731707318 7 34 interact
0.0 0 188 Mike
0.0 0 18 your-account
0.0 0 11 extreme
0.0 0 29 prodigy
0.0 0 63 Fred
0.06097560975609756 40 616 http-equiv
0.0 0 38 script
0.17391304347826086 4 19 before
0.0 0 13 high-spee
0.11111111111111111 2 16 amador
0.0 0 62 Logue
0.0 0 12 accommodate
0.0 0 26 simply
0.0 0 26 upon
0.014492753623188406 1 68 federal
0.0 0 45 false
0.13333333333333333 2 13 adrian
0.0 0 17 hidden

** The list continues for up 38 pages.
import java.io.*;
import java.util.*;

public class ClueWords {
    public static void main (String args[]) throws IOException {
        // Data Member Declarations
        double Mincount = 10;//was 10
        // An integer declaration
        int j;
        // Long integer data types for 4 items that allow better granularity for calculating probabilities.
        long Oldcount, Count, wordyescount, wordnocount;
    }
}

APPENDIX B. CLUEWORDS PROGRAM

/**
 * Title:          ClueWords
 * Description:    This class performs two functions. One is to take out
 *     each non-stop word and then looks at two subdirectories of "yes"
 *     and "no", find the conditional probabilities of "yes"
 *     given the occurrence of a particular word. Initial use is to
 *     test against emails and try to categorize them appropriately.
 *     Elements adapted from Dr. Neil Rowe's programs GetClueProbs
 *     and CountWords.
 * Copyright:    Copyright (c) 2002
 * Company:    USMC NPS
 * @author Scott R. Hall
 * @version 1.0
 */

double Mincount = 10; //was 10
int j;
long Oldcount, Count, wordyescount, wordnocount;
// Integer declarations and assignment values.
int yescount = 0;
int nocount = 0;

// Double real number declarations for 5 items that allow
// Standard Deviation and Probabilities to be displayed properly.
double yesratio, yesprob, Dev, Prob, SD;

// String characters declared to include a string tokenizer
// to help extract tokens from emails.
String Inputline, Word, Stopword;
StringTokenizer st;

// File
File Dir;

// Declaring and creating HashSet. Implements Set using an
// internal hashtable. Allows any type of object or null to
// be a member of the set. There is no guarantee of order
// for the set elements. There are no duplicates in a HashSet.
HashSet rchs = new HashSet();

// Within the Destemmer class call the hashKnownWords method
// and pass it results of the rchs.
Destemmer.hashKnownWords(rchs);

// Declaration and creation of hashmap. Same thing as a Hashtable
// but methods are not synchronized.
HashMap hm = new HashMap(200000);

// Declarations below imported from CountWords program.
HashSet hsstop = new HashSet(1000);
FileReader fr1 = new FileReader("stopwords.txt");
BufferedReader br1 = new BufferedReader(fr1);
while ((Stopword = br1.readLine()) != null) hsstop.add(Stopword);

// Creating a new instance of the Directory object and passing
// its contents of "yes" directory.
Dir = new File("yes");
String Filelist [] = Dir.list();

// A "for" loop to go through "yes/" directory and read in
// files via buffered reader.
for (j=0; j<Filelist.length; j++)
{
    FileReader fr = new FileReader("yes/" + Filelist[j]);
    BufferedReader br = new BufferedReader(fr);
}
// Inner "while" loop while the buffered reader is not empty (null)
// create a new String Tokenizer Object and tokenize based on the
// characters identified.
while ((Inputline = br.readLine()) != null)
{
    st = new StringTokenizer(Inputline," .::^-?!(){}_+=| \"<>/@#&*");
}

// Another inner "while" loop that loops through each token while
// there are more tokens left to tokenize. If the is not a number
// string than increment "yescount" and destem the word.
//
while (st.hasMoreTokens())
{
    Word = st.nextToken();
    if ((Word.length()>1) && (!numberString(Word)) &&
    (!hsstop.contains(Word))
    {
        yescou nt++;
        Word = Destemmer.destem(Word,rchs);
        if (!hm.containsKey(Word))
        {
            hm.put(Word,new Long(1000000));
        }
        else
        {
            Oldcount = (Long)hm.get(Word)).longValue();
            hm.put(Word,new Long(1000000+Oldcount));
        }
    }
}

fr.close(); // close out of file reader

// Same statements except for no category.

Dir = new File("no");
String Filelist2 [] = Dir.list();
for (j=0; j<Filelist2.length; j++)
{
    FileReader fr2 = new FileReader("no" + Filelist2[j]);
    BufferedReader br2 = new BufferedReader(fr2);
    while ((Inputline = br2.readLine()) != null)
    {
        st = new StringTokenizer(Inputline," .::^-?!(){}_+=| \"<>/@#&*");
    }
    while (st.hasMoreTokens())
    {
        Word = st.nextToken();
        if ((Word.length()>1) && (!numberString(Word)) &&
        (!hsstop.contains(Word)))
{nocount++;  
Word = Destemmer.destem(Word,rchs);
if (!hm.containsKey(Word)) 
{hm.put(Word,new Long(1));  
}
else
{
Oldcount = ((Long)hm.get(Word)).longValue();
hm.put(Word,new Long(1+Oldcount));
}
}
fr2.close();
}

PrintWriter fileout = new PrintWriter(new FileWriter("clueprobs.out"));
if (nocount > 0) yesratio = (double)yescount/(double)nocount;
else yesratio = 2.0*(double)yescount;
yesprob = (double)yescount/(double)(yescount+nocount);
fileout.println(yesprob + " overall probability, " + yescount + " yes examples, " + nocount + " no examples, ");
Set set = hm.entrySet();
Iterator i = set.iterator();
while (i.hasNext())
{
Map.Entry me = (Map.Entry)i.next();
Word = (String)me.getKey();
Count = ((Long)me.getValue()).longValue();
wordnocount = Count % 1000000;
wordyescount = (Count -wordnocount)/1000000;
Dev = (double)wordyescount -(yesratio*(double)wordnocount);
Prob = (double)wordyescount/(double)(wordyescount+wordnocount);
SD = Math.sqrt(1.0/((1.0/(double)wordyescount)+(1.0/(double)wordnocount)));
if (((wordyescount+wordnocount)>Mincount) & (Math.abs(Dev) > SD))
fileout.println(Prob + " " + wordyescount + " " + wordnocount + " " + Word);
}
fileout.close();
}

/* Says whether a string of characters represents an integer or decimal */
private static boolean numberString (String S)
{
boolean numberflag = false;
int N = S.length();
if (N > 0)
{
    int i=0;  

if (S.charAt(0) == ' -') i=1;
char C;
numberflag = true;
while ((numberflag) & (i<N))
{
    C = S.charAt(i);
    numberflag = (((C >= '0') & (C <= '9')) | (C == '.'));
i++;
}
}
return numberflag;

// For now I am remarking this call out
/*
  public static void Parser ()
  {
    Parser parser = new Parser ("yessupplyfiscal.txt");
  }
*/
}
THIS PAGE INTENTIONALLY LEFT BLANK
class RateDocs
{
    public static void main (String args[]) throws IOException
    {
        int j, k1, k2, Wordcount, M;
        double yesratio, Dev, Prob, SD, Average, Total;
        Double DProb;
        String Inputline, Word, Probstring;
        StringTokenizer st;
        File Dir;
        HashSet rchs = new HashSet();
        Destemmer.hashKnownWords(rchs);
        HashMap hm = new HashMap(200000);
        FileReader fr;
        BufferedReader br;
        String tempString = new String ();//temporary hold string object for subject line
        String subjectLine = new String ();
        boolean foundSubj = false;// flag set to find subject line
        FileReader frprobs = new FileReader("clueprobs.out");
        BufferedReader brprobs = new BufferedReader(frprobs);
        Inputline = brprobs.readLine();
        k2 = Inputline.lastIndexOf(' ');
        k1 = Inputline.lastIndexOf(' ',k2 -1);
        double Totalprob = Double.valueOf(Inputline.substring(k1+1,k2)).doubleValue();
        while ((Inputline = brprobs.readLine()) != null)
        {
            k1 = Inputline.indexOf(' ')
            k2 = Inputline.lastIndexOf(' '); //find subject line
            M = Inputline.length();
            Probstring = Inputline.substring(0,k1);
            Prob = (Double.valueOf(Probstring).doubleValue()) - Totalprob;
            Word = Inputline.substring(k2+1,Inputline.length());
            hm.put(Word, new Double(Prob));
        }
        frprobs.close();

        Dir = new File("unknown"); // Begin unknown directory here
        String Filelist [] = Dir.list();
        for (j=0; j<Filelist.length; j++)
        {
            Wordcount = 0;
            Total = 0.0;
            fr = new FileReader("unknown/" + Filelist[j]);
            br = new BufferedReader(fr);
            while ((Inputline = br.readLine()) != null)
            {
st = new StringTokenizer(Inputline,".,;~^?{}\+/\"\n\r\"<>@#&*");  
boolean foundSubj = false;  
//use "Subj:" for html  
if (!foundSubj && Inputline.indexOf("Subject:") >= 0)  
{
    subjectLine = Inputline;  
    foundSubj = true; //change flag to true  
    Wordcount++;  
}

PrintWriter fileout = new PrintWriter(new FileWriter("RATEDOCS.out"));

while (st.hasMoreTokens())
{
    Word = st.nextToken();  
    if ((Word.length()>1) && (!numberString(Word)))  
    {
        Word = Destemmer.destem(Word, rchs);  
        Wordcount++;  
        if (hm.containsKey(Word))  
        {
            DProb = (Double)hm.get(Word);  
            System.out.println(DProb + " retrieved for " + Word);  
            Total = Total+(DProb.doubleValue());  
            fileout.println(DProb + " retrieved for " + Word);
        }
    }
}

fileout.close();  
} //outer if statement for subject line find  

//while close  
fr.close();  
Average = Total/(double)Wordcount;  
System.out.println(Average + " strength for document " + Filelist[j]);

}  
}

/* Says whether a string of characters represents an integer or decimal */  
private static boolean numberString (String S)
{
    boolean numberflag = false;
    int N = S.length();  
    if (N > 0)  
    {
        int i=0;
        if (S.charAt(0) == '-') i=1;
        char C;
        numberflag = true;  
        for (int j=i; j<N; j++)
        {
while ((numberflag) & (i<N))
{
    C = S.charAt(i);
    numberflag = (((C >= '0') & (C <= '9')) | (C == '.
    i++;
}
};
return numberflag;
APPENDIX D. SAMPLE RUN FROM RATE DOCS

Test Set - Rate Docs for grades run against clueprobs

0.932639952016222 retrieved for qpr
0.932639952016222 retrieved for qpr
0.932639952016222 retrieved for qpr
0.932639952016222 retrieved for qpr
0.932639952016222 retrieved for qpr
0.932639952016222 retrieved for qpr
0.848555867937538 retrieved for nw3230
0.6519746713563415 retrieved for logistic
0.6172012256239916 retrieved for grade
0.6172012256239916 retrieved for grade
0.6172012256239916 retrieved for grade
0.6172012256239916 retrieved for grade
0.6172012256239916 retrieved for grade
0.6172012256239916 retrieved for grade
0.6172012256239916 retrieved for grade
0.6172012256239916 retrieved for grade
0.6172012256239916 retrieved for grade

* Document continues for up to 38 pages.
THIS PAGE INTENTIONALLY LEFT BLANK
APPENDIX E. PARSER PROGRAM

/**
 * Title:          Parser
 * Description:     (1) Reads a text file (emails saved as *.txt file)
 *                (2) Finds Subject line and parses it to find keywords
 *                (3) Reads entire file and counts the frequency of occurrence of key words in file
 *                (4) Prints subject line keywords & freqs to screen
 *                
 * Some elements adapted from Steve Simmon’s Parser program
 * @version 1.0
 */

import java.io.*;
import java.util.*;

public class Parser {

    //Class Variables (Global)
    String fileName = new String();
    StringBuffer filetext = new StringBuffer();
    String subjectLine = new String();
    String keyClueWord = new String();
    Vector subjKeyWords = new Vector();

    //******************************************************
    // Constructor
    //******************************************************

    public Parser(String fileInput) {
        //get the filename from the commandline argument
        fileName = fileInput;

        ReadFile();

        ParseSubject();

        ParseEmailText();

        //ReadClueProbs();
    }  //end Constructor

    //******************************************************
    // Method:  ReadFile
    //******************************************************
    void ReadFile(){
        String tempString = new String();
        boolean foundSubj = false;

        // Read file text
        try {
            File file = new File(fileName);
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line;
            while ((line = reader.readLine()) != null) {
                tempString = tempString + line + "\n";
                if (line.startsWith("Subject: ")) {
                    foundSubj = true;
                    subjectLine = line.substring("Subject: ".length());
                    break;
                }
            }
            reader.close();

            // Parse subject line
            StringTokenizer tokenizer = new StringTokenizer(tempString,
                                                            "\n", true);
            while (tokenizer.hasMoreTokens()) {
                String token = tokenizer.nextToken();
                if (!token.startsWith("Subject: ")) {
                    subjKeyWords.addElement(token);
                }
            }

            // Count frequencies
            for (int i = 0; i < subjKeyWords.size(); i++) {
                String key = (String) subjKeyWords.elementAt(i);
                int count = 0;
                for (int j = 0; j < subjKeyWords.size(); j++) {
                    if (key.equals((String) subjKeyWords.elementAt(j))) {
                        count++;
                    }
                }
                System.out.println("Key: " + key + " Count: " + count);
            }

            // Print subject line keywords & freqs to screen
            System.out.println("Subject Line:");
            System.out.println(subjectLine);
            System.out.println("Key Words:");
            System.out.println(tempString);
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }  //end ReadFile

    //******************************************************
    // Method:  ParseSubject
    //******************************************************
    void ParseSubject(){
        // Parse subject line
        StringTokenizer tokenizer = new StringTokenizer(subjectLine,
                                                        " ", true);
        while (tokenizer.hasMoreTokens()) {
            String token = tokenizer.nextToken();
            subjKeyWords.addElement(token);
        }
    }  //end ParseSubject

    //******************************************************
    // Method:  ParseEmailText
    //******************************************************
    void ParseEmailText(){
        // Parse email text
        StringTokenizer tokenizer = new StringTokenizer(filetext.toString(),
                                                        " ", true);
        while (tokenizer.hasMoreTokens()) {
            String token = tokenizer.nextToken();
            subjKeyWords.addElement(token);
        }
    }  //end ParseEmailText

    //******************************************************
    // Method:  ReadClueProbs
    //******************************************************
    void ReadClueProbs(){
        // Read clue probabilities
        File clueFile = new File("clue-probs.txt");
        BufferedReader clueReader = new BufferedReader(new FileReader(clueFile));
        String clue;
        while ((clue = clueReader.readLine()) != null) {
            subjKeyWords.addElement(clue);
        }
        clueReader.close();
    }  //end ReadClueProbs

    //******************************************************
    // Method:  Return
    //******************************************************
    String Return(){
        // Return results
        return "Return String";
    }  //end Return

    }  //end class Parser

}  //end class Parser
try{
    BufferedReader fileReader = new BufferedReader(new FileReader(fileName));
    while(fileReader.ready())
    {
        //Read each line of the email text file & store in string buffer
        tempString = fileReader.readLine();

        //make all lowercase
        tempString = tempString.toLowerCase();

        //Find subject line, change it from just "subj"
        If(!foundSubj && tempString.indexOf("subj") >= 0 )
        {
            subjectLine = tempString;
            foundSubj = true;

            //Debug print out
            System.out.println("Subject line: " + subjectLine);
        }
        //add line read to String Buffer, goes to frequency count
        filetext.append(tempString);
    }  //end While
}  //end try stmt

//Opening a file via FileReader object can throw FileNotFound Exception
catch(FileNotFoundException fileEX){
}

//Reading text in from a file can throw an IOException
catch(IOException IOEX){
}
}  //end method ReadFile

//**********************************************************************
// Method:  ParseSubject
//**********************************************************************
void ParseSubject()
{
    //create a String Tokenizer from the string that is the subject line
    //default tokenizing is to break string into words
    StringTokenizer subjectWords = new StringTokenizer(subjectLine);
    String tempString = new String();
while (subjectWords.hasMoreTokens())
{
    tempString = subjectWords.nextToken();

    // Debug print out
    System.out.println("Token: " + tempString);

    // check to see if word id a key word;
    // if the word is a keyword, add to vector
    if(KeyWord(tempString))
    {
        KeyWord temp = new KeyWord(tempString);
        subjKeyWords.add(temp);
    }
}
} // end While

} // end method ParseSubject

//******************************************************************************
// Method:  KeyWord
//******************************************************************************
boolean KeyWord(String text)
{
    boolean IsKeyWord = true;
    String smallWords[] = {"and", "the", "a", "an", "if", "it", "is", "this",
                           "subject", "subj", "re", ".", ":", "?", ":!", ":", "to", "FW:",
                           "fwd:"};
    String tempString = new String();
    for(int i = 0; i < smallWords.length; i++)
    {
        tempString = smallWords[i];
        if(text.startsWith(smallWords[i]))
        {
            IsKeyWord = false;
            break;
        }
    }
} // end for loop

return IsKeyWord;

} // end method KeyWord

//******************************************************************************
// Method:  ParseEmailText
//******************************************************************************
void ParseEmailText()
{
}
Iterator KeyWordITR = subjKeyWords.iterator();
KeyWord tempKeyWord = new KeyWord("txt"); // dummy variable
int counter = 1;
int keyWordFreq = 0;
while (KeyWordITR.hasNext())
{
    // Get keyword from vector keywords in subject
    tempKeyWord = (KeyWord) KeyWordITR.next();
    // ResetFreq
    keyWordFreq = 0;
    // Get Freq for this word
    keyWordFreq = getFrequency(tempKeyWord.keyword, filetext.toString());
    tempKeyWord.frequency = keyWordFreq;
    if (keyWordFreq > 2)
    {
        // Print out result to screen
        System.out.println("Subject Keyword "+ counter++ +":  "+ tempKeyWord.keyword +"  Frequency:  "+ tempKeyWord.frequency);
    }
}

//******************************************************************
// Method:  getFrequency
//**********************************************************************
int getFrequency(String keyWord, String file)
{
    int count = 0;
    int index = -1;
    index = file.indexOf(keyWord);
    // 1st occurrence of keyword found
    if (index >= 0)
    {
        // Increment count and make recursive call to this function with remaining
        // text less all words up to and including the found keyword occurrence
        count = 1 + getFrequency(keyWord, file.substring(index + keyWord.length()));
    }
    return count;
}

//******************************************************************
// Method: main
//**********************************************************************
public static void main(String[] args)
{
    Parser parser1 = new Parser("testparser.txt"); /* was args[0] or "cs4556_8.txt" */
}

LIST OF REFERENCES


Mooney, Raymond J., CS378 Class Notes, University of Texas, April 2002.


INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Prof. Neil Rowe, Code 32
   Department Of Computer Science
   Naval Postgraduate School
   Monterey, California

4. Prof. Thomas Otani, Code 32
   Department Of Computer Science
   Naval Postgraduate School
   Monterey, California

5. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

6. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

7. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

8. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California