



Automatic Term Extraction Using Statistical Techniques

A Comparative In-Depth Study & Applications

"This Thesis was Submitted in partial fulfillment of the requirements for the
Masters Degree in Scientific Computing
from The Faculty of Graduate Studies at Birzeit University – Palestine".

By

Yousef Sabbah

Supervisors:

Dr. Yousef Abuzir

Dr. Haithem Abu-Rub

-August 2005-

To

The Memory of my Mother

ACKNOWLEDGEMENTS

I would like to thank Dr. Hassan Shibly; chairman of Scientific Computing Master Program, for his interest and follow up.

I was very fortunate to be supervised by Dr. Yousef Abuzir from Quds Open University and Dr. Haithem Abu-Rub. Besides encouragement, they took care of the entire review process and problems encountered me while writing this thesis.

I do not forget Prof. Dr. Younis Amro, President of Quds Open University, for his approval to continue my higher education, Eng. Imad Hodali, ICTC Manager, for his help and support, Mr. Mustafa Tamim and Mr. Mohammad Hamarsheh for their help and valuable suggestions during the development of ATEWB, all my colleagues in ICTC, and all friends for their encouragement.

Lastly, all thanks and appreciations to my family for support and patience.

Table of Contents

Chapter	Name	Page
	LIST OF FIGURES.....	H
	LIST OF TABLES	J
	LIST OF ABBREVIATIONS.....	K
	ABSTRACT	P
1	INTRODUCTION.....	1
1.1	OVERVIEW.....	1
1.2	MOTIVATION	5
1.3	ATEWB.....	6
1.4	CONTRIBUTIONS	7
1.5	OUTLINE OF THIS THESIS	7
2	RELATED WORK	9
2.1	LANGUAGE AND INFORMATION	9
2.2	LANGUAGE ENGINEERING	9
2.3	COMMUNICATION OF INFORMATION.....	10
2.4	NATURAL LANGUAGE PROCESSING.....	11
2.4.1	<i>NLP in Information Retrieval.....</i>	<i>13</i>
2.5	INFORMATION RETRIEVAL.....	15
2.5.1	<i>Digital Audio Retrieval</i>	<i>16</i>
2.5.2	<i>Digital Video Retrieval.....</i>	<i>17</i>
2.6	TEXT RETRIEVAL AND AUTOMATIC TERM EXTRACTION (ATE).....	19
2.6.1	<i>Index Terms.....</i>	<i>20</i>
2.6.2	<i>Index terms, topics, and terminological terms</i>	<i>21</i>
2.6.3	<i>Index Term Corpus.....</i>	<i>22</i>
2.6.4	<i>Manual Indexing</i>	<i>23</i>
2.6.5	<i>Comparison of Manual and Automatic Indexing</i>	<i>24</i>
2.7	ATE APPROACHES	26
2.7.1	<i>Statistical Techniques.....</i>	<i>26</i>
2.7.2	<i>Neural Networks and Machine learning</i>	<i>29</i>

2.7.2.1	Learning Process in Neural Networks	31
2.7.2.2	Application of Neural Network Models in IR	31
2.7.2.2.1	The application of Self-Organizing Feature Map (SOFM)	32
2.7.2.2.2	The application of Hopfield Net	35
2.7.2.2.3	The application of MLP and Semantic Networks	36
2.7.3	<i>Probabilistic Models</i>	37
2.7.4	<i>Syntactic Analysis</i>	39
2.7.4.1	Pre-search and Post-search	40
2.7.4.2	Syntactic Analysis	40
2.7.4.3	Phrase Matching	42
2.7.4.4	Syntactic Analysis Evaluation	42
2.8	IR AND ATE EVALUATION	42
2.9	SUMMARY	46
3	AUTOMATIC TERM EXTRACTION (ATE)	47
3.1	ATE STAGES	47
3.2	SINGLE TERM VS. PHRASE EXTRACTION	48
3.3	ATE STATISTICAL TECHNIQUES	50
3.3.1	<i>Term Frequency</i>	53
3.3.2	<i>Inverse Document Frequency</i>	55
3.3.3	<i>Combination of TF and IDF</i>	58
3.3.4	<i>Term Discrimination Value Model</i>	60
3.4	ATE AUXILIARY APPROACHES	63
3.4.1	<i>Porter's Stemming</i>	63
3.4.2	<i>Stop-words Removal</i>	65
3.5	SUMMARY	67
4	AUTOMATIC TERM EXTRACTION WORKBENCH (ATEWB) SYSTEM DESCRIPTION	68
4.1	AN OVERVIEW OF ATEWB	68
4.2	ATEWB PACKAGE	70
4.2.1	<i>Design Issues</i>	73
4.3	ATEWB SYSTEM REQUIREMENTS	74
4.3.1	<i>Hardware Requirements</i>	74
4.3.2	<i>Software Requirements</i>	76
4.4	ATEWB MAIN CLASSES AND ALGORITHMS	76

4.4.1	<i>Driver Class</i>	77
4.4.2	<i>GUI, Navigation and Connection Classes</i>	80
4.4.2.1	Dialog and fileNewDialog Classes	81
4.4.2.2	FileNavigator Class.....	81
4.4.2.3	dbConnection Class	81
4.4.3	<i>Statistical Techniques</i>	82
4.4.3.1	Term Frequency Classes	82
4.4.3.2	IDF Classes	85
4.4.3.3	TFxIDF Classes	87
4.4.3.4	TDVM Classes.....	90
4.4.4	<i>ATE Auxiliary Approaches</i>	94
4.4.4.1	Stop Words and Parsing Class	94
4.4.4.2	Porter's Stemming Algorithm (Stemmer Class).....	97
4.4.5	<i>Evaluation Class</i>	101
4.5	SUMMARY	103
5	A COMPARATIVE STUDY	104
5.1	COMPARISON METHODOLOGY	105
5.1.1	<i>Comparison Criteria</i>	105
5.1.2	<i>Main Factors Affecting Performance and Accuracy</i>	105
5.2	EXPERIMENTS, RESULTS AND DISCUSSION	109
5.2.1	<i>Experiment One</i>	110
5.2.1.1	Objectives	110
5.2.1.2	Setup	110
5.2.1.3	Procedure	110
5.2.1.4	Results.....	114
5.2.1.5	Discussion and Conclusions	114
5.2.2	<i>Experiment Two</i>	116
5.2.2.1	Objectives	116
5.2.2.2	Setup	116
5.2.2.3	Procedure	117
5.2.2.4	Results.....	118
5.2.2.5	Discussion and Conclusions	122
5.2.3	<i>Experiment Three</i>	123

5.2.3.1	Objectives	123
5.2.3.2	Setup	123
5.2.3.3	Procedure	124
5.2.3.4	Results.....	124
5.2.3.5	Discussion and Conclusions	128
5.3	COMPARISON WITH PREVIOUS STUDIES.....	130
5.4	SUMMARY	131
6	CONCLUSIONS AND FUTURE WORK.....	133
6.1	CONCLUSIONS	133
6.2	FUTURE WORK	134
	BIBLIOGRAPHY.....	136
A.	APPENDIX A	I
B.	APPENDIX B.....	IV
B.1	ATEWB SYSTEM INSTALLATION	IV
B.1.1	<i>Microsoft windows environment.....</i>	<i>IV</i>
B.1.2	<i>Linux RedHat environment.....</i>	<i>IV</i>
C.	APPENDIX C	VI
C.1	WORKING WITH ATEWB	VI
C.1.1	<i>Using ATEWB</i>	<i>VI</i>
D.	APPENDIX D	XVIII
D.1	ATEWB JAVA AND MYSQL COMPLETE CODE.....	XVIII

List of Figures

FIGURE 2.1: COMMUNICATION PROCESS – THE DIFFERENT CONCEPTS.	11
FIGURE 2.2: A TYPICAL IR SYSTEM [9]	15
FIGURE 2.3: OVERLAP OF TERMINOLOGICAL TERMS, TOPICS AND INDEX TERMS.....	22
FIGURE 2.4: COMPARISON OF MANUAL WITH AUTOMATIC INDEXING	25
FIGURE 2.5: ATE-TECHNIQUES CLASSIFICATION	27
FIGURE 2.6: A SIMPLE NEURAL NETWORK	30
FIGURE 2.7: KOHONEN MODEL	33
FIGURE 2.8: HEAD-MODIFIER RELATIONS OF "SYNTACTIC ANALYSIS OF INDEX TERMS"	41
FIGURE 3.1: A PLOT OF HYPERBOLIC CURVE RELATING THE FREQUENCY AND RANK.	51
FIGURE 3.2: DOCUMENT VECTOR SPACE.....	51
FIGURE 3.3: PORTER'S STEMMING ALGORITHM	64
FIGURE 4.1: MAIN STAGES OF ATEWB SYSTEM.....	69
FIGURE 4.2: UML DIAGRAM OF ATEWB PACKAGE	72
FIGURE 4.4: PSEUDO CODE FOR MAIN DRIVER.....	79
FIGURE 4.5: CLASS DIAGRAM OF TF CLASSES.	83
FIGURE 4.6: PSEUDO CODE OF TF() METHOD.	85
FIGURE 4.7: CLASS DIAGRAM OF IDF CLASSES.....	86
FIGURE 4.8: PSEUDO CODE OF IDF() METHOD.....	87
FIGURE 4.9: CLASS DIAGRAM OF TFXIDF CLASSES.....	88
FIGURE 4.10: PSEUDO CODE OF TFXIDF() METHOD.....	90
FIGURE 4.11: CLASS DIAGRAM OF TDVM CLASSES.....	91
FIGURE 4.12: PSEUDO CODE OF DISCVALUE() METHOD.	92
FIGURE 4.13: PSEUDO CODE OF AVGSIM() METHOD.	93
FIGURE 4.14: PSEUDO CODE OF COSINECORRFACOR() METHOD.	94
FIGURE 4.15: STOP CLASS DIAGRAM.....	95
FIGURE 4.16: PSEUDO CODE OF STOP WORDS AND PARSING ALGORITHM.....	97
FIGURE 4.17: STEMMER CLASS DIAGRAM.	98
FIGURE 4.18: PSEUDO CODE OF DOSTEMMING() METHOD IN PORTER'S ALGORITHM.	100
FIGURE 4.19: EVALUATION CLASS DIAGRAM.....	102
FIGURE 5.1: COLLECTION SIZE VS. COMPUTATIONS TIME FOR TDVM.	107

FIGURE 5.2: RECALL VS. STATISTICAL TECHNIQUE/ FACTOR	113
FIGURE 5.3: PRECISION VS. STATISTICAL TECHNIQUE/ FACTOR	113
FIGURE 5.4: NOISE VS. STATISTICAL TECHNIQUE/ FACTOR	114
FIGURE 5.5: EFFECT OF INCREASING NUMBER OF RETRIEVED TERMS ON RECALL.....	120
FIGURE 5.6: EFFECT OF INCREASING NUMBER OF RETRIEVED TERMS ON PRECISION.....	120
FIGURE 5.7: EFFECT OF INCREASING NUMBER OF RETRIEVED TERMS ON NOISE.....	121
FIGURE 5.8: AVERAGE RECALL, PRECISION AND NOISE	121
FIGURE 5.9: RELATION BETWEEN R, P AND N FOR TF	126
FIGURE 5.10: RELATION BETWEEN R, P AND N FOR TFXIDF.	127
FIGURE 5.11: R-P RELATIONSHIP FOR TFXIDF.....	127
FIGURE 5.12: AVERAGE RECALL, PRECISION AND NOISE	128
FIGURE C.1: ATEWB MAIN SCREEN.....	VI
FIGURE C.2: ATEWB FILE MENU	VII
FIGURE C.3: ATEWB NEW PROJECT DIALOG BOX	VII
FIGURE C.4: ATEWB NEW PROJECT OPTIONS	VIII
FIGURE C.5: ATEWB BROWSER.....	VIII
FIGURE C.6: CREATE NEW DATABASE DIALOG BOX.....	IX
FIGURE C.7: ATEWB DATABASE OPTIONS DIALOG BOX.	X
FIGURE C.8: ATEWB ATE MENU WITH ITS SUBMENUS	X
FIGURE C.9: CONDITIONS OF SELECTED INDEX TERMS.	XI
FIGURE C.10: ATEWB TERM FREQUENCY RESULT SCREEN.....	XII
FIGURE C.11 ATEWB IDF RESULT SCREEN	XII
FIGURE C.12: ATEWB TFXIDF RESULT SCREEN	XIII
FIGURE C.13: ATEWB TDVM RESULT SCREEN.....	XIII
FIGURE C.14: ATEWB DOCUMENT SUMMERY	XIV
FIGURE C.15: ATEWB TOOLS MENU.....	XV
FIGURE C.16: KEYWORDS LIST RESULT	XVI
FIGURE C.17: ATEWB EVALUATION DIALOG BOX.	XVI
FIGURE C.18: WINDOWS CALCULATOR.....	XVII

List of Tables

TABLE 1.1: CHAPTERS SUMMARY	8
TABLE 2.1: ATE EVALUATION MEASURES	44
TABLE 3.1: STEPS OF PORTER’S STEMMING ALGORITHM	66
TABLE 4.1: PERFORMANCE EXPERIMENTS FOR ATEWB SYSTEM REQUIREMENTS	75
TABLE 4.2: ATEWB CLASSES AND INSTANCES USED TO CALL THEIR METHODS	77
TABLE 4.3: SUMMERY OF PORTER’S STEMMING METHODS	101
TABLE 5.1: SUMMARY OF FACTORS AFFECTING EXTRACTION RESULTS AND EXPECTED EFFECT	106
TABLE 5.2: EXPERIMENT 1 COLLECTION DESCRIPTION AND STATISTICS	111
TABLE 5.3: RECALL, PRECISION AND NOISE OF EXPERIMENT 1 WITH DIFFERENT CONDITIONS	112
TABLE 5.4: EXPERIMENT 2 COLLECTION DESCRIPTION.	117
TABLE 5.5: RECALL, PRECISION AND NOISE OF THREE PARTS OF EXPERIMENT2.	119
TABLE 5.6: AVERAGE RECALL, PRECISION AND NOISE OF EXPERIMENT 2	119
TABLE 5.7: COMPUTATION TIME ESTIMATION AND TERM STATISTICS OF EXPERIMENT3.	125
TABLE 5.8: EXPERIMENT3 COMPUTATIONS OF R, P AND N	125
TABLE 5.9: AVERAGE RECALL, PRECISION AND NOISE OF EXPERIMENT 3	126

List of Abbreviations

Abbreviation	Meaning
ANSI	American National Standards Institutes
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATE	Automatic Term Extraction
ATEWB	Automatic Term Extraction WorkBench
ATN	Augmented Transition Network
AVGSIM	Average Similarity
AWT	Abstract Window Toolkit (JAVA Package)
CPU	Central Processing Unit
DB	DataBase
DISCVALUE	Discrimination Value
DM	Discrimination Model
DMS	Document Management Systems
DOCFREQ	Document Frequency
DV	Discrimination Value
GUI	Graphical User Interface
HCI	Human Computer Interaction
HMM	Hidden Markov Model
HTM	Hyper Text Markup
HTML	Hyper Text Markup Language
I/O	Input/ Output

IDF	Inverse Document Frequency
IP	Internet Protocol
IR	Information Retrieval
JAVA	JAVA Programming Language
JDBC	Java DataBase Connector
JDK	JAVA Development Kit
LINUX	LINUX Operating System
Me	Millennium version of Microsoft windows
MS	Microsoft
MYSQL	Database Engine supports Structured Query Language
NLP	Natural Language Processing
NOVELL	NOVELL Operating System
NT	Network Technology Version of Microsoft windows
OS	Operating System
RAM	Random Access Memory
RTN	Recursive Transition Network
SOM	Self-Organizing Map
SPSS	Statistics Package for the Social Science
SQL	Structured Query Language
SSOM	Scalable Self-Organizing Map
TDV	Term Discrimination Value
TDVM	Term Discrimination Value Model
TF	Term Frequency

TFxIDF	Combined Term Frequency-Inverse Document Frequency
TREC	Text REtrieval Conference
UML	Unified Modeling Language
UNIX	UNIX Operating System
XML	eXtensible Markup Language
XP	Version of Microsoft windows 2002

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

"قُلْ لَوْ كَانَ الْبَحْرُ مَدَادًا لَكَلِمَاتِ رَبِّي لَنَفَذَ الْبَحْرُ قَبْلَ أَنْ تَنفَذَ كَلِمَاتُ رَبِّي وَلَوْ جِئْنَا بِمِثْلِهِ مَدَدًا"

صدق الله العظيم

ملخص

(Information Retrieval)

)

.()

(

Automatic)

.(Term Extraction

(Term Frequency TF)

:

Inverse Document Frequency)

(TFxIDF)

(IDF

(Term Discrimination Value Model)

(ATEWB)

(Database Engines)

(Abstracts)

Abstract

The idea of Information Retrieval (IR) has been generated during the evolutionary change in the way cultural, social or scientific information are stored from ink-on-paper to digital libraries distributed on international networks. Typically this information concerns material such as text, graphic documents (pictures, maps, technical drawings etc.), sound and moving images. In attempt to make such huge amounts of information efficiently retrievable, some techniques for Automatic Term Extraction (ATE) are proposed. This Automatic procedure is considered the cornerstone of a wide range of applications such as search engines, because manual production of keywords is highly labor intensive. To ensure precise information retrieval, the extracted keywords should accurately describe the contents of their documents.

To improve this operation, researchers proposed many techniques for Automatic Term Extraction (ATE) or Automatic Indexing, some used statistical techniques and others used syntactic and probabilistic techniques. This thesis is a comparative study aimed at leading to the use of statistical techniques including four techniques: Term Frequency (TF), Inverse Document Frequency (IDF), combined Term Frequency-Inverse Document Frequency (TFxIDF) and Term Discrimination Value Model (TDVM). We have also developed a computational tool for Automatic Term Extraction (ATEWB) to be used in the comparison; three experiments are used for this purpose to specify the conditions in which each technique is mostly efficient and/or accurate.

On the other hand, this thesis aims at improving statistical techniques efficiency through the utilization of database engines to reduce the computations time of their algorithms. As well as improving documents retrieval by caching them in the database.

We have tested our model on a collection of abstracts of papers in the field of automatic term extraction, containing keywords composed by their authors in the first experiment, and a collection of documents prepared for test available on some web sites concerned with IR.

Key Words¹:

Term, Document, Word, Index, Technique, Class, Computation, Weight, Collection, ATEWB, Step, Result, Extraction, Statistical, Method, Number, Retrieval, Frequency, Database, Information, System, TF, Time, Performance, Text, IDF, ATE, TFxIDF, Stemming, Automatic.

¹ The above key words are extracted using ATEWB, which is developed in this thesis. 30 terms are extracted using TF technique, 28 terms are relevant. They are ordered by weight in descending order.

1 Introduction

1.1 Overview

Information Retrieval (IR) involves retrieving desired information from textual, graphical voice or video data. A typical IR request would be to find all documents related to a particular subject. Before getting started, it is necessary to define some basic concepts:

- **A Term** is a linguistic sign denoting a concept and referring to a reference object [1]. It can also be defined as a word or expression of a specific meaning in a specific field, which is concerned with some definitions, standards or presentation. Terms may be suitable keywords but they are not defined originally for information retrieval.
- **Index term** is a keyword used to describe the contents of a text and to guide a user to the information [2].
- **Index term corpus** is a text collection that is linguistically analyzed, the index terms are marked up manually [2].
- **Automatic Term Extraction (ATE)** is the operation of keywords extraction of a specific domain from corpus using computers [3].
- **Automatic indexing** ATE is some times refereed to as automatic indexing which is the process of producing the keywords (index terms) of a text automatically [2].
- **Retrieval** is locating relevant documents to queries [4].

- **Information Retrieval IR** is the recall of stored information [5].

In IR systems, documents are represented by some data (such as identifiers, titles, authors, dates, abstracts, extracts, reviews, and keywords) called document surrogates [5].

To retrieve documents, there must be an approach in which keywords are extracted. These keywords may be used as the index of those documents, for instance, in a search engine. Term extraction can be defined as the procedure of extracting terms (index terms candidates) from a document or a collection of documents in a specific domain. In the past, this operation was performed manually, which is a labor intensive and time-consuming procedure.

Automatic Term Extraction (ATE) approaches came to solve this problem automatically using computers. Some ATE approaches are referred to as statistical techniques, some of these techniques are simple numerical algorithms such as term frequency which counts how many each word occurs in a document, others are very complicated such as term discrimination value in which millions of vectors each containing millions of elements to be multiplied and summed many times. Such techniques cannot be tested on normal computers, supercomputers are needed for computations even if the simplest technique is used, because of huge amount of textual data to be processed (tokenized, counted, divided by the computed number of terms in their documents to be then weighted and sorted, and the documents containing them to be retrieved). ATE and statistical techniques will be discussed in details in Chapter 3.

This means that statistical techniques need intensive computations and can utilize scientific computing algorithms such as high performance computing and algorithms, numerical simulation and numerical algorithms, and parallel and distributed computing and algorithms to weight the terms and find which has the highest weights (candidate index terms).

It seems to the reader for the first time that IR is not related to scientific computing, even though, one of the main fields Scientific Computing concerns with is Computational Information Science, which includes Data Mining and algorithms, Information Retrieval (IR), Medical Informatics, Bioinformatics, Genomics, and Biometrics, Computational Graphics, Text, Video, Multimedia Mining and High Performance Information Processing and algorithms. Hence, IR is much related to scientific computing.

On the other hand, Index terms are extracted using some techniques referred to as syntactic analysis, which includes new algorithms of content analysis in a new computational information science field called computational linguistics.

The list of index terms generated by ATE system can be used in many real-world applications such as for example:

1. Search Engines: in which you enter a keyword or a sentence to search for a file or files containing it on the web or in a directory on local disks on your computer.
2. Digital Libraries: "libraries in which a significant proportion of the resources are available in digital (machine-readable) format, as opposed to

print or microform"². They replicate many functions of traditional libraries in digital media. They contain selected collection of texts with various means of access. Indexing and abstracting are the first step of digitization process.

3. Archiving Systems: like those used in mail systems to archive old e-mails, keywords are used to find your e-mails quickly by subject, sender or contents.
4. Thesaurus Use and Construction: in thesaurus we need term extraction techniques to classify terms of the same meaning.
5. Content Analysis (Text Filters): content analysis is very important in firewalls like those used in proxy servers to prevent access to some unwanted web sites, anti-spam filter is another application that gets benefit of term extraction, in which the contents of e-mails or received files are analyzed to be checked against some contents, subjects, domain names or senders. If any contains such keywords listed by the anti-spam they are deleted or an alert appears to the recipient to take an action. It is also used in anti-viruses and anti-spy-wares.
6. Document Management Systems DMS: Document management is used to manage the life cycle of a document, from the creation through multiple revisions and finally into long-term storage and record management (archiving). DMS usually feature searching in repositories of documents

² Anderson J. D. & Perez-Carballo J. 2005. *Information Retrieval Design*. Web site: <http://www.scils.rutgers.edu/~carballo/>.

both by externally applied information about the documents (e.g., user who entered it, date of revision, or version relationship) and by content (e.g., search on words contained within the document).

1.2 Motivation

To the best of my knowledge, and after reviewing related work, there was no classification of ATE techniques, there was no comprehensive and clear comparative study between statistical techniques and there was no computational tool developed to evaluate the performance and efficiency of all techniques in one package.

This thesis involves examination of the research issues in ATE general problem, as well as the solutions proposed for several issues. The problem of Automatic Term Extraction can be broken down into the following sub-problems:

1. Term extraction that includes:
 - a. Parsing.
 - b. Removing stop words.
 - c. Stemming.
2. Index Terms selection based on statistical techniques.
3. Performance improvement of statistical techniques

This is done by trying to answer the following questions:

1. What techniques or approaches are used in ATE? And what statistical techniques represent among these techniques?
2. What are the famous statistical techniques used in ATE? What methodology is used in each statistical technique?

3. What are the main points of differences and similarities between these techniques? What are the advantages and disadvantages of each one?
4. Which technique is considered the best in terms of performance and accuracy? When to use each technique?
5. What are the main factors that affect their performance and accuracy?
How to improve performance and accuracy.
6. How to evaluate IR systems, and which measures may be used in evaluation of effectiveness and efficiency?

1.3 ATEWB

A computational tool is required to make the comparison and evaluation of used statistical techniques. There is no comprehensive and integrated tool that can be used for this purpose. This is why we have developed a computational tool named Automatic Term Extraction WorkBench (ATEWB) and used it to help us in our comparison and evaluation. This ATE tool can be used for term extraction using any of the above four statistical techniques for a collection of documents. Two versions of ATEWB tool are developed, one under Microsoft windows and the other under UNIX environment. A database engine is used to do all computations instead of a programming language, which improves performance.

Experts can use ATEWB to help them in generating the index terms. Beginners can use it to learn and experiment the different statistical techniques for automatic term extraction.

1.4 Contributions

The main focus of this thesis is to extract index terms or keywords automatically from different types of electronic documents using different statistical techniques. Due to the complexity of the problem we are tackling, there is a need for us to try to focus on the process of extracting a single index term.

The major contribution of this research pertains the comparison of statistical techniques and the evaluation of these techniques using ATEWB which is a comprehensive computational tool that extracts the keywords and evaluates each technique by computing the evaluation measures such as recall, precision and noise. These measures are discussed in section 2.8.

A database engine is used in ATEWB for computations and vectors multiplication is replaced by joining tables which improved the performance.

1.5 Outline of this Thesis

This thesis introduces different statistical techniques for automatic term extraction and compares between them in six chapters.

The next chapter, Chapter 2, presents a literature review of related work. This is followed by a discussion of Automatic Term Extraction (ATE) and ATE Statistical Techniques in Chapter 3. Chapter 4 presents design issues and a description of ATEWB extraction tool. Chapter 5 is a comparative study that presents a comparison between statistical techniques and an evaluation of ATEWB through three experiments followed by a discussion of results. Finally, Chapter 6 presents conclusions and future work.

Table 1.1 shows chapters' summary. The chapters can be divided into two categories; previous studies and our contribution.

Table 1.1: Chapters summary.

Chapter#	Chapter Name	Description
1	Introduction	An introduction and description of what to be studied in this thesis.
Previous Studies		
2	Related Work	Literature review of related work in three areas: Natural Language Processing, IR and ATE techniques and finally Evaluation.
3	Automatic Term Extraction	ATE concepts, stages and statistical weighting techniques
Our Contribution		
4	ATEWB description	ATEWB system description and design issues using UML notation and pseudo code.
5	A Comparative Study	Three experiments to compare between techniques, results and evaluation of ATEWB
6	Conclusions and Future Work	Summary of conclusions and ATE future work.

2 Related Work

In computational linguistics we have recently witnessed a growth in the interest in automatic treatment of terms, or linguistic units which characterize specialized domains, especially when Natural Language Processing (NLP) systems are moving from the development to the application stage [6].

2.1 Language and Information

Language is used for information communication. Information Retrieval is a sub-discipline of information science. Information science has the following sub-disciplines [2]:

- Informatics: is the quantitative study of information exchange.
- Information management: information or text retrieval systems evaluation and quality.
- Information retrieval systems design.
- Information retrieval interaction.

2.2 Language Engineering

As we mentioned above, we are now in the information age, and information becomes available from a huge amount of resources, this makes it increasingly difficult for recipients to select and get what is useful. Language engineering software provides the facilities to overcome the problems of information overload. The techniques developed within language engineering allow the analysis of the

content of information sources, either in a quick shallow sense, looking for information of potential interest on which to focus, or, within a specific subject area, to perform a complete analysis identifying specific information. In addition, the selected information can then be summarized for presentation to the user who can later decide to request the full information [2].

2.3 Communication of Information

Language is used for information communication. The following definitions have been found in the Concise of Oxford dictionary (1976) and the Macquarie dictionary (1981) [2]:

- *Knowledge* is what I know.
- *Information* is what we know (shared knowledge).
- *Communication* is the interchange or transfer of information by speech, writing or signs.
- *Data* is any fact(s) that is matter of direct observation.
- *A document* is a group of recorded information arranged in physical form.

Figure 2.1 describes the different concepts of the communication process and the overlap between them. In communication we transmit factual information. The sender is the writer who expresses meanings into a text. The content of the text is the subject and the main idea it contains, while the meaning of a word is the idea it refers to and which can be explained by other words [2]. The receiver here is the reader who interprets the meanings of the text and receives information and stores what he understands as a new knowledge.

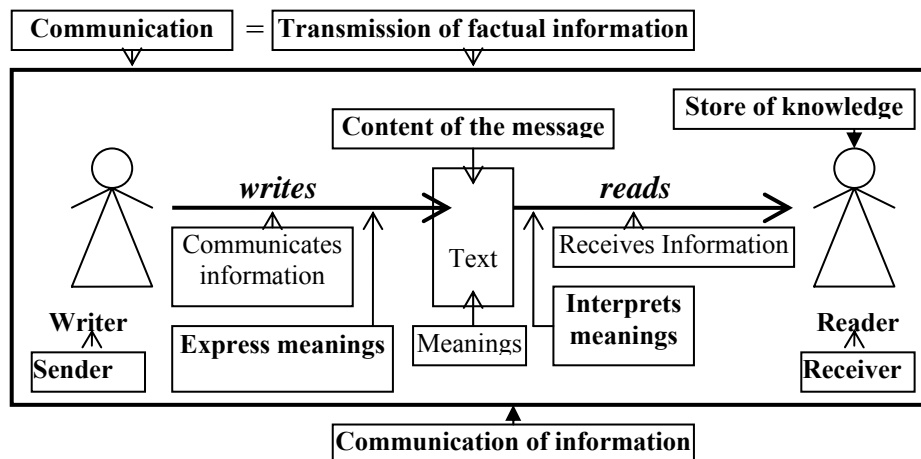


Figure 2.1: Communication process – the different concepts [2].

2.4 Natural Language Processing

Natural Language Processing (NLP) means that if we can define plurals, singulars, verbs, nouns, and other language patterns and describe them to a computer, then we can teach a machine something of how we speak and understand each other. This work is based on researches in linguistics and cognitive science [7].

NLP research pursues the question of how we understand the meaning of a sentence or a document. What are the keys we use to understand who did what to whom, or when something happened, or what is fact and what is supposition or prediction? While words (nouns, verbs, adjectives and adverbs) are the building blocks of meaning, it is their relationship to each other that conveys the true meaning of a text within the structure of a sentence, within a document, and within the context of what we already know about the world [7].

People extract meanings from text or spoken language on at least seven levels, it is not necessary that all NLP systems to use every level. These levels are [7]:

1. **Phonetic Level:** refers to the way the words are pronounced. This level is not important for text retrieval systems. It is crucial to understanding in spoken language and in voice recognition systems.
2. **Morphological Level:** the morpheme is a linguistics term for the smallest piece of a word to carry meaning. Examples are word stems like *child* (the stem for *childlike*, *childish*, *children*) or prefixes and suffixes like *un-*, or *-ation*, or *-s*.
3. **Lexical Level:** can be used either for part-of-speech tagging or for the utilization of lexicons from which the detailed features of individual terms can be accessed. The lexical level is used in the construction of thesauri and other similar resources, which are used as tools for manual indexers and searchers. This way, we ensure that a common vocabulary is used in selecting appropriate keywords.
4. **Syntactic Level:** the structure of a sentence conveys meanings and relationships between words even if we don't know what they mean. Position of a word can determine whether it is the subject or the object of an action.
5. **Semantic Level:** examines words for their dictionary meaning and the meaning derived from the context of the sentence. Most words have more than one meaning but we can identify the best one from the rest of the sentence.

6. **Discourse Level:** uses document structure to extract additional meanings for words. NLP uses this structure to understand the main and exact role of a word or an expression in a document [8].
7. **Pragmatic Level:** in this level we use the information we know about the world from outside the contents of the document to extract more meanings. This is why we have to provide the IR system with this information for higher effectiveness.

All these levels of understanding are combined within the structure of a sentence to narrow the meanings of words as much as possible. Because each of these levels of language understanding follows definable patterns, it is possible to inject some language patterns into a computer system to be utilized in index terms extraction [7].

2.4.1 NLP in Information Retrieval

Here we summarize the main steps of an information retrieval system. Any IR system concerning text may consist of the following steps [7]:

1. **Document Processing:** includes tagging and information extraction from each document and creating a list of words in alphabetical order and removing stop-words. Text retrieval systems also create knowledge bases with internal lexicons, semantic networks, or lists of phrases, synonyms, and personal pronouns.

At this stage additional operations may be performed on words like stemming, identification of part of speech, and the relationship of a word

to the others within the sentence, the paragraph, or the document. Term weights are computed at this stage and assigned to index terms.

2. **Query Processing:** when a query is entered, a statistical system identifies the terms to search for and it may look for stems and singular and plural forms. It may also assign weights to each term. A full NLP system tags all the parts of speech, identifies objects, subjects, agents, verbs, and creates an unambiguous representation of the query for the system to match against its knowledge base.
3. **Query Matching:** the entered query is matched against the extracted index terms in the knowledge base. Traditional systems match the query in the same sequence it was entered by the searcher. Statistical systems use Boolean combination, if your query is *automatic indexing* for example, you get *automatic indexing*, *automatic AND indexing*, and *automatic OR indexing*. A full NLP system may expand the query and add some synonyms from its knowledge base to get for instance *automation of index terms*.
4. **Ranking and Sorting:** the list of documents matching the query are sorted by date, field or by how relevant the document is to the query.

NLP can be added at any or all stages using any or all of the seven levels of understanding. NLP is mostly used on the lower levels of understanding, and for query interpretation only, parsing the query sentences (syntactic level) and stemming (morphological level). NLP semantic level is rarely used in IR systems, while phonetic level is necessary for audio and video retrieval systems [7].

2.5 Information Retrieval

Information Retrieval (IR) is the process of retrieving the requested information or data from a very huge amount of data. IR is a new field of scientific computing that needs high performance or say supercomputers to be done automatically. This enables you to get any piece of information you want from a large collection of documents as fast as possible. Figure 2.2 illustrates a typical IR system by means of a black box [9].

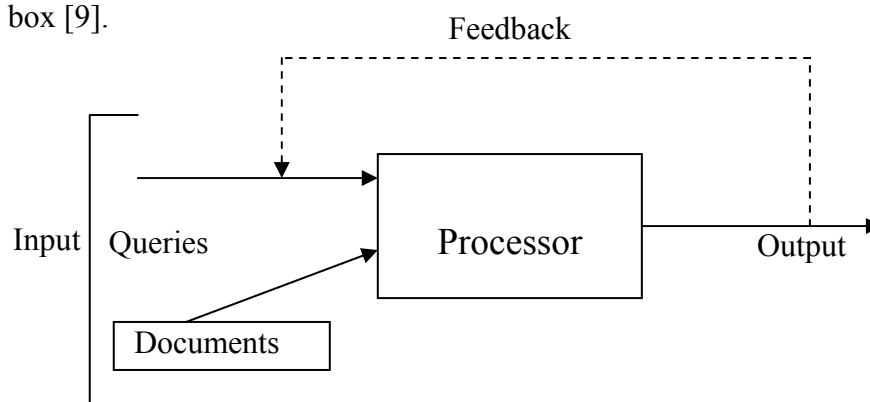


Figure 2.2: A Typical IR System [9]

The purpose of an information retrieval system is to find the most relevant materials for the user while it eliminates the least relevant. We measure the ability of the system to accomplish this feat as its precision and its recall [7]. Precision and recall are measures of IR systems' performance, and they are better discussed in section 2.8.

IR includes retrieval of text, voice, video, image, structured data, animation, etc. Text retrieval is the most widespread among these types (text retrieval is presented in section 2.6 in details), while the others are limited to some researches. This is because it is too difficult to retrieve other types like images, voice and video, in which retrieval is described as processing of these types so

that a query in search engines becomes for example (all image files containing a dog). Till now there is no IR system that can effectively retrieve such information, it just retrieves files containing the text (dog) in its name or contents. An example of voice retrieval is to find (all audio files containing 'moon' in the speech). It is obvious that this process is very complicated and needs voice or speech recognition, analysis and then extraction of voiced keywords.

One of the latest researches in digital video and digital audio management was proposed by Alan F. Smeaton [10]. Browsing audio and video contents is more essential than it with text or image retrieval. This means that we cannot simply take text-based information retrieval and apply it to video but we must rebuild the whole system and integrate browsing and searching operations as well as possible.

2.5.1 Digital Audio Retrieval

Four approaches may be used in digital audio retrieval [10]:

- **Word Spotting:** given a pre-defined vocabulary of a number of words, process the spoken audio to look for these words only, and use them as indexing terms. By this restriction we can reduce complexity of speech recognition to become manageable [11].
- **Speaker Recognition:** speaker-independent continuous speech recognition to the audio recording of a dialog like that used in the Jabber project at the University of Waterloo [10]. This shows that it is possible to achieve effective retrieval from moderate processing of the audio signal.
- **Phone-based Retrieval:** instead of recognizing the whole words, it is possible to recognize sub-words (phones). This approach has been

implemented in indexing radio news in German language at ETH-Zurich [12]. In this approach, the candidate phones are related to words through some defined paths.

- **Word-based Audio Retrieval:** this approach applies continuous speech recognition to spoken audio. This recognition must be speaker independent to get the best results. It is used to support subsequent IR.

In spoken document track of the fifth Text REtrieval Conference (TREC-5), they tested a collection of 100 hours of news broadcasts to find the relevant broadcasts to a query. The tested systems of all groups in TREC-5 used speaker recognition approach [10].

2.5.2 Digital Video Retrieval

To discuss video retrieval, it is necessary to define some video concepts. "Video is basically a sequence of images displayed at a constant speed, normally 25 to 30 frames per second, with a synchronized audio track" [10]. It requires 720KB of storage to display a single frame of TV quality video with 25 frames per second for smooth motion. This means that it needs 18MB of storage for one second of a video without compression. It is obvious that video utilizes a huge amount of storage size and needs a high data transfer rate. To solve this problem it is necessary to compress the video data using some compression algorithms. In this way we can improve TV quality video manipulation to be displayed smoothly on computer screens [10].

One of the most important video formats that use compression and video encoding standards is MPEG family (MPEG-1 through MPEG-7). Motion compensation is

a basic and necessary step in all video compression techniques, in which the motion of adjacent video frames is recognized and only the differences between adjacent frames are transmitted [10].

To understand the process of video retrieval, it is necessary to understand the definition of a video and the used encoding algorithms. Video can be defined as a group of successive variable-length single shots combined together in some way to be played into a 2-D window (e.g. screen) continuously. The 2-dimensions are the positions (x, y) and a third can be added which is time (t) [10].

Automatic video segmentation is a fundamental operation in video retrieval, in which video is segmented into a list of shots using automatic Shot Boundary Detection (SBD). Initial attempts at video segmentation were based on processing the primitives in a video which are similar to the primitives in a single-frame image, but with time added, namely color and associated histograms and their within-frame distribution, texture, intensity/brightness, etc [10].

An experiment on video retrieval was run by Smeaton in Dublin City University on 8 hours of broadcast TV including different program types. All shot boundaries were marked up manually. From these 720,000 frames he took 5380 shot cuts and 779 fades or resolves as a ground truth to compare between different techniques. Next step was to use an indexing approach to find a representative frame from each shot that establish a list of keyframes. The simplest approach was to take the middle or the first frame of a shot. Another approach is to represent a video stream by a list of images where each image has an associated length of the clip from which it was taken [10, 13].

In Informedia project (A TV new application) [14], the approach was to incorporate several techniques for content retrieval including:

- Speech recognition on the audio of the video using Sphinx recognizer.
- Segmenting the video.
- Object detection looking for VIP faces and text captions in the image.
- Caption and text extraction from frames using OCR.
- The user's text query is matched against text obtained from frames and against a series of representative frames associated with keywords extracted from dialog to be used as a query to find video clip like it.

2.6 Text Retrieval and Automatic Term Extraction (ATE)

A comparable research topic in the field of Information Retrieval (IR) is text retrieval and its related approaches for automatic indexing or Automatic Term (keyword) Extraction (ATE) [15]. Research in automatic indexing can be traced back to the late 1950's, when Luhn published a paper on extracting meaningful elements from texts to be used in information retrieval [16].

Efficient text retrieval system depends on the efficiency of the techniques used to extract index terms automatically from a collection of documents of a specific domain that will be the base of IR systems. This operation is called automatic term extraction . ATE and its techniques will be discussed in details in chapter 3.

"An automatic text retrieval system is designed to search a file of natural language documents and retrieve certain stored items in response to queries submitted by a

user" [17]. Typically, each document is described using certain words (keywords) contained in the documents [17].

Document and text retrieval methods have been proved theoretically and tested on many commercial retrieval systems [18]. Experiments on these methods show that they work very well with full texts and large files, not only titles and abstracts. The user's request can be a word list, phrases, sentences or extended text [18]. In this thesis the user's request is confined on word list.

ATE is the most important stage of text retrieval and its applications like: Document Management Systems (DMS) [19], Automatic Summarization, Automatic Abstracting, Thesauri Construction [20, 21], Text Filters, Firewall Systems and E-mail Management Systems [22], etc.

For instance DMS, which is "the automated control of electronic documents, spread sheets and word processing documents within an organization from creation to final archiving" [23], focuses on indexing and text retrieval as important functions [23].

Indexing refers to the process of choosing the surrogates (keywords) which represent the topic or content of documents [24]. A great number of studies have appeared since 1950 on the quantitative approaches to single term extraction. Some of them use linguistic information such as stop-word lists consisting of function words, and many use quantitative measures [15].

2.6.1 Index Terms

Index terms can be considered as meta-information pointing to potential information in documents. According to ANSI 1968 we can define Indexing as

"the process of analyzing the informational content of records of knowledge and expressing the informational content in the language of indexing system" [2, 25].

It involves [2, 25]:

1. Selecting indexable concepts in a document.
2. Expressing these concepts in the language of the indexing system (as index entries), and an ordered list.

An indexing system is: "The set of prescribed procedures (manual and/or machine) for organizing the contents of records of knowledge for purposes of retrieval and dissemination" [2].

An index term points to and describes the contents of a document. For example the main subjects and key contents of a book can be described and referred to by using an index. Indexing can be done automatically or manually, and index terms can be expressions obtained from the text or defined independently. These expressions indicate what is being written about, and guide users or readers to relevant information.

2.6.2 Index terms, topics, and terminological terms

Three kinds of index terms may be distinguished: topics, subtopics and passing concepts and proper names. A topic is defined as 'what is being written about in the course of discourse'. A topic and an index term both describe the contents of a document, but the point of view is different. For example a proper name which may be an index may not be a topic of the text. While indexing, choose terms that the reader might be interested in [2].

The difference between terminology and index term is that terminology is concerned with definition, standardization and presentation of terms, it may be suitable index term, but it is not defined specially for IR. Terminological terms must be as exact and universal as possible, while index terms describe the contents of a certain document. Indexing languages also include passing concepts and proper names; that is we can use proper names and concepts as index terms. Figure 2.3 shows the overlap of terminological terms, topics and index terms [2]. It is also necessary to distinguish between words, terms and concepts [27].

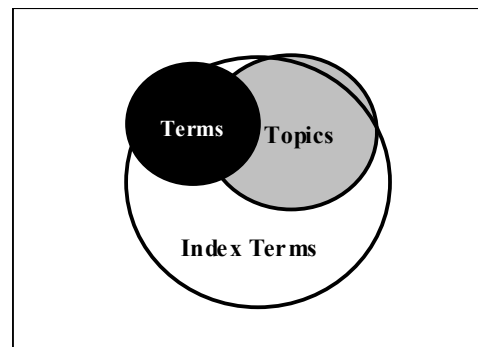


Figure 2.3: Overlap of terminological terms, topics and index terms [2]

2.6.3 Index Term Corpus

Index term corpus is a specific domain collection of documents or text for test purpose in automatic indexing. A corpus is a linguistically analyzed text collection, where index terms are manually marked up for each document page using previously generated book indexes [2].

It can also be defined as "a large body of natural language text used for accumulating statistics on natural language text" [2]. Corpora often include extra information such as a tag for each word to indicate its part-of-speech, and perhaps

the parse tree for each sentence. Part-of-speech tagging is labeling each word in each sentence with its part-of-speech [2].

2.6.4 Manual Indexing

In manual indexing, indexers specify which keywords are suitable as index terms in a document based on controlled vocabulary "an established list of standardized terminology for use in indexing and retrieval of information to ensure that a subject will be described using the same term each time it is indexed" [28], this procedure is long and of significant cost. When an indexer starts the indexing procedure he firstly asks himself what is the document about and in what is the reader interested. The procedure of indexing starts with choosing suitable keywords via content analysis, assigning content indicators, adding location indicators, assembling the resulting entries, and finally, choosing the way of displaying the final index [2]:

Content indicators like titles, subtitles, and abstracts, in addition to first and last sentences give good indicators of subject contents. Then the list of derived concepts is converted into the controlled vocabulary of the indexing language and a thesaurus is used to get the final index terms. There are no universal indexing rules and there are many indexing guides, but a good indexer follows one guide for consistency [2].

It is necessary to define an appropriate depth of indexing, that is, the optimal number of topics covered in the index. Users may miss something if too few topics are covered and they may have to read irrelevant material if too many topics are covered [2].

2.6.5 Comparison of Manual and Automatic Indexing

The main steps of automatic indexing are the same as manual indexing, except the use of computers to be performed automatically instead of manual indexers. Using a NASA database consisting of documents from Scientific and Technical Aerospace Reports (STAR) and International Aerospace Abstracts (IAA), automatic indexing was compared with manual indexing in the mid-nineteen seventies of the last century using different automatic and manual systems. Recall and precision, which are the evaluation measures for IR systems, were used in the comparison; these measures are discussed in details in section 2.8. The following indexing systems were compared [17]:

1. Automatic indexing system 1: using natural language with machine search of titles and abstracts.
2. Automatic indexing system 2: using natural language and thesaurus.
3. Manual indexing system 1: indexing documents with controlled language.
4. Manual indexing system 2: controlled indexing using natural language term extraction.

The results of this comparison show that the automatic indexing system 1 produced the best average recall and a high precision. The manual indexing system 1 produced a better precision than the automatic system 1, but a worse recall. Figure 2.4 illustrates the points of NASA system evaluation and the curves representing the performance of Medlars manual indexing test, and the STAIRS system indication [17]. We conclude that automatic indexing proved at least equal or we can say better results than manual indexing [17].

Another comparison of the performance of the Medlars manual indexing system with the automatic indexing system 2 based on abstract searching with thesaurus has shown that the two systems give approximately the same results [30].

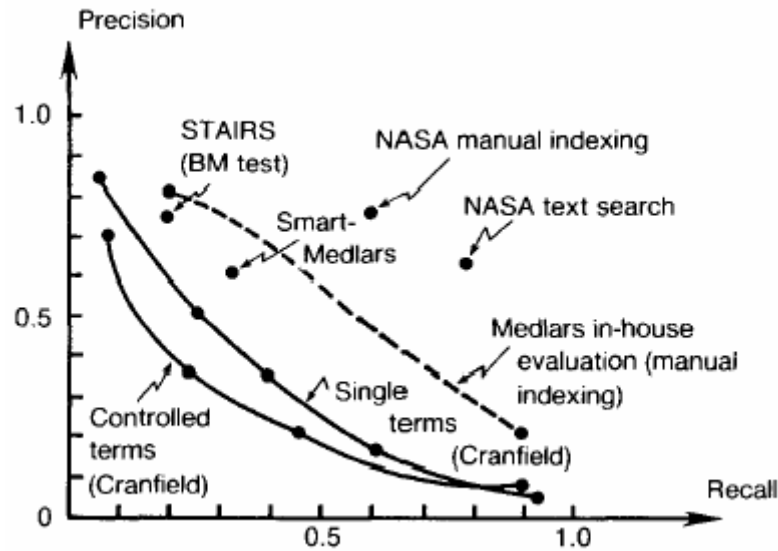


Figure 2.4: Comparison of Manual with Automatic Indexing [17]

In a study of automatic (single term) indexing systems with natural language and abstract search by Aslib and Cranfield to evaluate their performance, a comparison between automatic (single term) indexing systems and different manual (controlled term) indexing systems was conducted [29]. The two curves of Cranfield study are illustrated figure 2.4. From the curves it is clear that the automatic (single-term) indexing gave almost better results than the manual (controlled-term) indexing [17].

From this comparison we conclude that automatic indexing provide almost the same results if not better than manual indexing. On the other hand, automatic indexing dramatically reduces time and efforts needed in manual indexing.

2.7 ATE approaches

In ATE, a weighting approach is needed to weight the extracted words, to judge, which may be an index term. Many studies have been published since 1950's proposing different approaches, each has its advantages and disadvantages.

As shown in Figure 2.5, ATE approaches may be categorized into four main categories: Statistical Techniques, Neural Networks and Machine Learning, Probabilistic Techniques, and Syntactic Analysis.

2.7.1 Statistical Techniques

Automatic term extraction statistical techniques are an example of these approaches. They depend on the use of simple terms to index both request and document texts; on term weighting utilizing statistical information about occurrences of a term; on scoring for request-document matching, and then using these weights to obtain a ranked search output; and lastly, on relevance feedback in iterative searching to modify request weights or term sets [18].

Luhn (1957) was the first who used this approach [16]. Some other researchers proposed approaches based on the existence and non-existence of a word in a document, while some proposed approaches based on frequency [15].

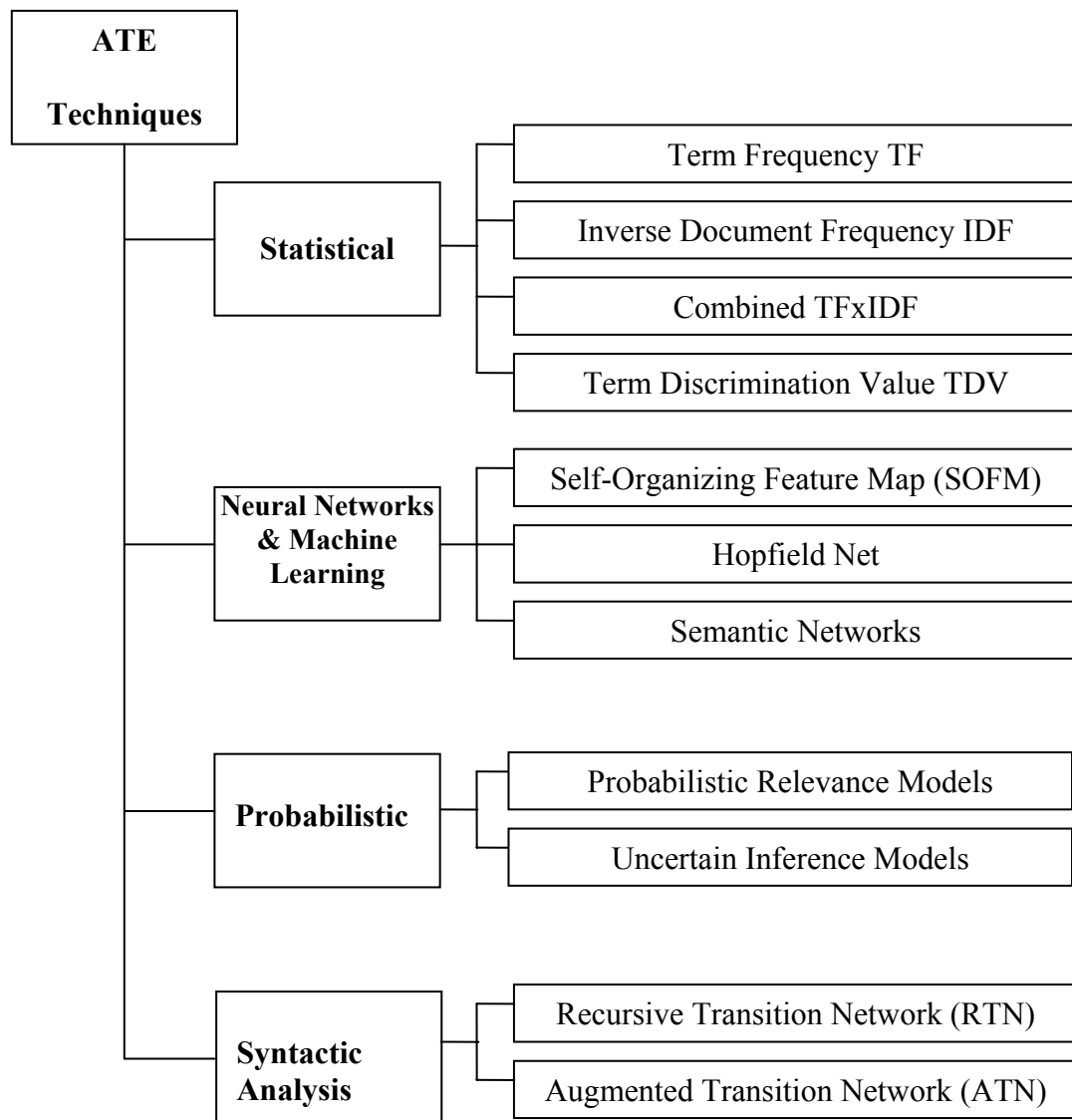


Figure 2.5: ATE-Techniques Classification

The two most straight forward measures for the weights of a word in a document are: term existence and term frequency. The former gives 1 to all the words that appear in the document, while the latter gives the number of occurrences of the word in the document as its weight [15].

A famous weighting measure called Inverse Document Frequency (IDF) is based on word existence or non-existence in the documents [31]. Salton and Yang

(1973) present the measure, in which IDF weight is multiplied by the frequency of the word in the document [32]. The result weight is a combination of TF and IDF to generate a new approach called TFxIDF.

On 1975 Salton Proposed Term Discrimination Value (TDV) as a new term-weighting technique and document similarity measure. The Theory is essentially that the best terms for indexing are those that highlight differences among documents in a collection [33, 15]. Salton defined the discrimination value of an index term as an estimate of an individual term's contribution to the density of a document space in systems based on the vector-processing model [6].

On 1980 M. F. Porter proposed an algorithm for suffix stripping claiming that it improves text retrieval [34]. He said "Removing suffixes by automatic means is an operation which is useful in the field of information retrieval. We can say that, a document may be represented by a vector of words or terms. Terms with a common stem will usually have similar meanings." For example:

Connect, Connected, Connecting, Connection, and Connections.

Frequently, the performance of an IR system is improved if term groups such as this are unified into a single term. This is done by removal of suffixes such as -ed, -ing, -ion, -ions to leave the single term Connect. In addition, suffix stripping reduces the total number of terms in the IR system, and hence reduces the size and complexity of the data in the system, which is always useful [34].

This thesis establishes a comparison between the most frequently used quantitative statistical techniques proposed above to find which is the most effective and efficient approach. These techniques are:

1. Term Frequency (TF).
2. Inverse Document Frequency (IDF).
3. Term Frequency combined with Inverse Document Frequency (TFxIDF).
4. Term Discrimination Value Model (TDV or TDVM).

Different techniques are discussed along with different examples. It is shown how an index term can be extracted using different techniques. Chapter 3 will describe these techniques in more details.

2.7.2 Neural Networks and Machine learning

Neural Network is an important component in Artificial Intelligence (AI). It is used in many fields like speech and image recognition, machine learning and IR. "A neural network model (or neural model) refers to a connectionist model simulating the biophysical information processing which occurs in the nervous system" [37].

If viewed as an adaptive machine, a neural network is defined as a parallel distributed processor consists of simple processing units, which has a natural bias to store knowledge of previous experiments to make it available for use later [37]. It is similar to the brain in learning and storing knowledge by interneuron connection strength.

Figure 2.4 illustrates a simple neural network [35]. Haykin refers that the neuronal model consists of three basic elements [37]:

- A set of connecting links (synapses), each link has its own weight or strength.

- An adder for summing the input signals with their respective weights of synapses of the neuron.
- An activation function that limits the amplitude of the output of a neuron.

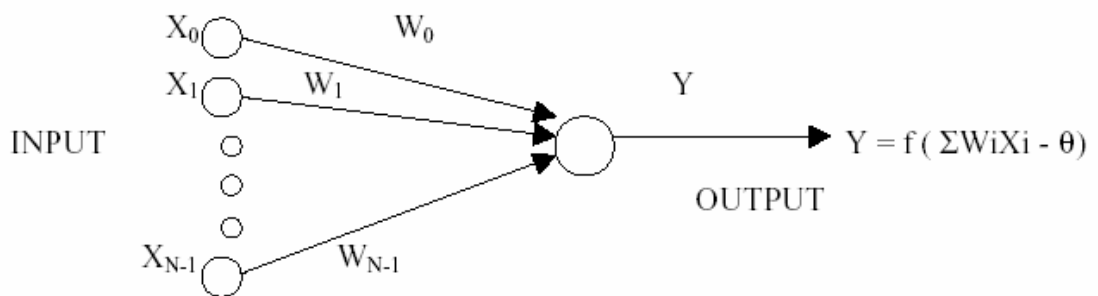


Figure 2.6: A simple Neural Network [35]

Three types of activation functions are distinguished: 1) threshold function; 2) Piecewise-linear function, and 3) sigmoid function. The sigmoid function, whose graph is s-shaped graph, is the most common activation function.

Neural network models have the following advantages over traditional IR models [36, 37]:

1. They are self-processing. The nodes and links in a neural network have intelligent behavior. No external active agent operates on them.
2. Neural network models exhibit global system behaviors. Concurrent local interactions on different components made them potential for parallel processing.
3. Resistance to noise.

Neural network architectures (topologies) can be divided into three classes [37]:

1. Single-layer perceptrons (feed forward networks)
2. Multi-layer perceptrons (feed forward networks)

3. Recurrent networks with at least one feedback loop.

2.7.2.1 Learning Process in Neural Networks

The ability of neural nets to learn from input data with or without a teacher has been a central issue for researchers developing neural networks. Learning rules used in nets learning can be categorized into five groups [37]:

1. Error-correction learning
2. Memory-base learning
3. Hebbian learning
4. Competitive learning
5. Boltzmann memory-based learning

On the other hand, learning processes are divided into three groups [36, 37]:

1. Supervised learning: weights are adjusted by a teacher.
2. Unsupervised learning (self-organized learning): no external indication is given to the network such as the correct responses and what of the generated responses are right or wrong.
3. Reinforcement learning: it is somewhere between supervised learning, in which the desired output is provided, and unsupervised learning, in which the system gets no feedback.

2.7.2.2 Application of Neural Network Models in IR

Neural network models can be used for clustering and keyword classification in different domains for more powerful IR systems. Applying connectionist approaches in IR might produce IR systems that will be able to [37]:

- Recall memories in spite of failures in individual memory units.
- Modify stored information when new inputs are entered by the user.
- Retrieve nearest data when no exact data match exists.
- Associatively recall information in spite of input missing or noise.
- Categorize information by their associative patterns.

Let's review the application of some neural network models in IR systems.

2.7.2.2.1 The application of Self-Organizing Feature Map (SOFM)

SOFM was introduced by Kohonen 1988 [38]. A simple Kohonen net consists of two layers, an input layer and a Kohonen (output) layer. These two layers are fully connected. Each input layer neuron has a feed-forward connection to each output layer neuron as illustrated in figure 2.7. From this simple net a complicated SOFM can be constructed [37].

SOFM uses competitive (unsupervised) learning and works in 2 steps. First, it selects the unit whose connection weight vector is closest to the current input vector as the winning unit. After a winning neighborhood is selected, the connection vectors to the units whose output values are positive are rotated toward the input vector. Inputs to the Kohonen (output) layer can be calculated by the dot product between the neuron weight vector and the input vector. The neuron with the biggest dot product is the winning output layer neuron [37].

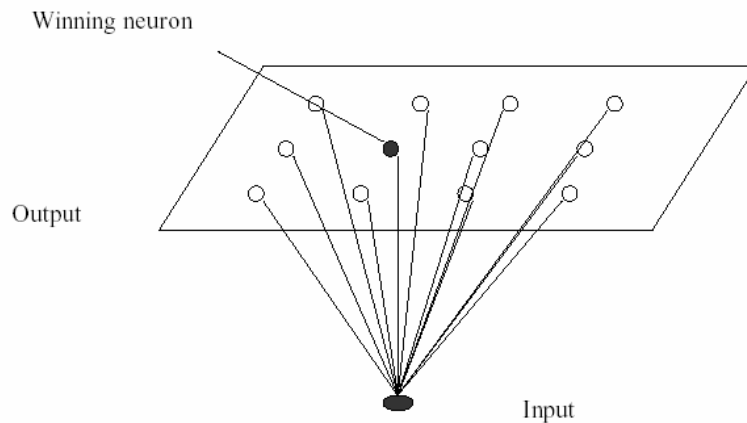


Figure 2.7: Kohonen Model [37]

Kohonen has shown that his self-organizing feature map "is able to represent rather complicated hierarchical relations of high-dimensional space in a two-dimensional display [38]." He used his feature map and conventional hierarchical clustering analysis to process a matrix of 32 documents and five indexing terms of artificial data. He obtained similar minimal spanning trees in both methods. In another example he showed that the self-organizing mapping could also display, two dimensionally, topological and similarity relations between phonemes from continuous speech. He had concluded that "the self-organized mappings might be used to visualize topologies and hierarchical structures of high-dimensional pattern spaces" [38]. Document space is such a high-dimensional space.

On 1997, Hanato used SOFM in an information organizer for effective clustering and similarity-based retrieval of text and video. He showed that word-frequency based algorithms of SOFM seem suitable to cluster documents and generate a global overview map, while algorithms based on Salton's measurements seem more effective to perceive document's distinction, which is useful to IR [37].

The digital library initiative (DLI) project at Illinois University uses SOFM to classify and map the category of text documents and to map the category of the texture of images [39].

The SOFM algorithm for textual classification is summarized below [40]:

1. Initialization: use the most frequently occurring N terms as the input vector and create a 2-dimensional map of M output nodes. Initialize the weights w_{ij} to small random values.
2. Present the documents in order: each document is described as an input vector of N coordinates. If the document has the corresponding term set a coordinate to 1, if there is no such term set the coordinate to 0. Each document presentation is repeated several times.
3. Compute distance to all nodes: the Euclidean distance d_j between the input vector and each output node j is Computed using the formula 2.1:

$$d_j = \sum_{i=0}^{N-1} (x_i(t) - w_{ij}(t))^2 \quad (2.1)$$

where $x_i(t)$ may be 1 or 0 depending on the presence of the i^{th} term in the document presented at time t . w_{ij} is the vector which represents position of the map node j in the document vector space, or the weight from input node i to output node j

4. Select winning node j^* and update weights to node j^* and its neighbors:
Select winning node j^* , which produces minimum distance d_j . Update

weights to nodes j^* and its neighbors to reduce the distances between them and the input vector $x_i(t)$ as in formula 2.2:

$$w_{ij}(t+1) = w_{ij}(t) + h(t)(x_i(t+1) - w_{ij}(t)) \quad (2.2)$$

After such updates, nodes in the neighborhood of j^* become more similar to the input vector $x_i(t)$.

$h(t)$ is an error-adjusting coefficient ($0 < h(t) < 1$) that decreases over time.

5. Label regions in map: After training the network through repeated presentations of all documents, each output node is assigned a term by choosing the one corresponding to the winning term (the term with largest weight). Merge neighboring nodes, which contain the same winning terms, to form a concept/topic region (group). Again, each document is submitted as input to the trained network and assigned a particular concept in the map. We get a map that represents regions of important terms/concepts with the documents assigned to them. The conceptually similar concept regions appear in the same neighborhood. On the other hand, similar documents are assigned into the same or similar concepts [40].

2.7.2.2.2 The application of Hopfield Net

Hopfield net was proposed as a content addressable memory and used for various classification tasks and global optimization.

Chen and his colleagues 1993 proposed a variant Hopfield net to create a network of related keywords. Asymmetric similarity function was used to produce thesauri for different domain-specific databases automatically. That is to be integrated with thesauri produced manually [37].

The Hopfield has been adapted by Chung and her colleagues for IR concept assigner. The main steps of the adapted Hopfield are [41]:

1. Assigning weights of the links: the concepts in the Concept space represent the net nodes, and synaptic weights between nodes are the similarities computed based on the co-occurrence analysis.
2. Initialization: an initial set of concepts (noun phrases) extracted from a document are the input patterns.
3. Activation: activate nodes in parallel and combine activated values from neighboring nodes for each single node.
4. Convergence: the process is repeated until reaching a stable state, in which there is no significant change in output state between two time steps.

2.7.2.2.3 The application of MLP and Semantic Networks

It is difficult to distinguish between the applications of MLP and the applications of semantic networks with spreading activation models in IR. That is because they have similar feed-forward structure. MLP was used in developing a method for term association by Wong and his colleagues on 1993 [37]. The results indicate the usefulness of neural networks in adaptive IR systems design.

Kwok represented an IR system using 3-layer MLP network (queries connected to index terms connected to documents) [37]. He attempted to reformulate the

probabilistic model of IR with single terms as document components. This system optimally ranks the collection documents with respect to a query. The discrimination function of this 3-layer network is based on IDF with learning algorithms exist. It is proved that activation using network provides better results than that in traditional IDF weighting [37].

Peece (1981) and Cohen & Kheldsen (1987) used semantic networks with spreading activation models [37].

There are many other systems using neural networks for IR but they can not be mentioned here, this is just a review of some neural network based IR systems.

2.7.3 Probabilistic Models

Probabilistic modeling is "the use of a model that ranks documents in decreasing order of their evaluated probability of relevance to a user's information need or query" [42], or the ratio of the documents' probability of being relevant to a query to the probability of their being irrelevant [43], refer to formula 2.3.

$$w_i = \log \frac{P(x_i | rel) [1 - P(x_i | irrel)]}{P(x_i | irrel) [1 - P(x_i | rel)]} \quad (2.3)$$

All documents are ranked in decreasing order according to the weight w_i in equation (2.3), assuming that x_i are individual terms that occur independently of each other in the relevant and irrelevant documents in the collection. $P(x_i | rel)$ is the probability of a document to be relevant and $P(x_i | irrel)$ is the probability of a document to be irrelevant [43].

Many researchers have made use of formal theories of probability and statistics in order to evaluate or estimate those probabilities of relevance. One of these attempts was the vector space model proposed by Salton [42], in which documents are ranked according to similarity with queries. A measure of similarity can not be considered as probability. On the other hand, similarity models lack the theoretical soundness of probabilistic models.

The first attempt to develop a probabilistic theory of retrieval was introduced by Maron and Kuhns on 1960 and then by Miller on 1971 [42]. There are already several operational probabilistic or semi-probabilistic IR systems. The obstacle in these models is to find a method to evaluate the probability of relevance, which is theoretically sound and computationally efficient. To simplify the problem, some assumptions like event independence are made in some models. For example "binary independence indexing model" and "binary independence retrieval model" used such assumptions [42].

When dependencies are taken into account, the results are worst than when simplifying assumptions were used. Moreover, higher computations are needed with dependencies [42].

Some approaches involved statistical techniques with pattern recognition and regression analysis [42]. These approaches do not make use of independence assumptions. The only probabilistic assumptions involved are those implicit in the statistical regression theory itself.

Other models are proposed, some based on Bayesian inference theory taking into account conditional dependencies. Some models are based upon non-classical

logic, or integrating logic with NLP. All these models are under development and validity is to be confirmed.

After this review we can distinguish two classes of probabilistic models:

1. Probabilistic Relevance Models.
2. Uncertain Inference Models.

2.7.4 Syntactic Analysis

Syntactic analysis uses NLP techniques and linguistic algorithms. It is concerned with term phrases instead of single terms and the relation between words in each sentence. Inverted file is efficiently used in syntactic analysis. Inverted file inverts free text search terms into narrower units than documents or even paragraphs. Boolean operators like AND are used in inversion on the sentence level to link terms within sentence boundaries [44].

Work in the linguistic approach for IR started on 1966 in SMART system by Salton [45], followed by LEADER system by Hillman and Kasarda [44]. After that, NLP techniques have been used in developing other systems like: SPIRIT system by Andreewsky 1977, then on 1983 Dillon and McDonald proposed FASIT system [44], another system was developed by Doszkocs on 1982 named CITE, then SIRE and CONTEXT systems by Koll et al. on 1984 and Contahal on 1985 respectively [44]. On 1986 and 1987 the interest in syntactic analysis increased dramatically. This includes Belgonov, Berrut & Palmer, Chiamarella with an information retrieval system that combines linguistics and statistics, Defude works on query analysis, Jones/Bell, Smeaton, Walker with techniques for large scale applications which need some sort of syntactic processing all on 1986,

then Biswas with a system using natural language interface, Das-Gupta used natural language conjunction analysis, Liddy used anaphoric problems, and stem dictionary expert system by Hyams all on 1987 [44]. Till now for example, Thinking Machines Corporation and Brattle Research Corporation are offering NLP tools and sometimes combined with other techniques for IR.

Syntactic analysis concentrates on noun phrases which are compound terms or a group of syntactically related terms combined together to give a new meaning different from the meaning of each term alone. Special algorithms are used for noun phrase recognition to isolate the terms [44]. In the following sub-sections (2.7.4.1- 2.7.4.4), we are going to review the stages of syntactic analysis systems.

2.7.4.1 Pre-search and Post-search

Pre-search is used instead of inverted file, in which the system combines the meaningful terms for an OR search. The result may be documents containing one or more search terms. Post-search ranks the retrieved documents which contain more than one search term according to the phrase or sub-phrase correspondence of the query based on the number of retrieved terms and the hierarchical structure of the links between them [44].

2.7.4.2 Syntactic Analysis

Syntactic analysis involves using some syntactic rules for free text analysis called dependency structure rules. After isolation of noun phrases, one or more of 45 syntactic rules may be applied to generate the dependency structure of noun phrases. For example, head-modifier relation between adjectives and prepositional

phrases of the sentence "syntactic analysis of index terms in documents" is shown in figure 2.8 [44].

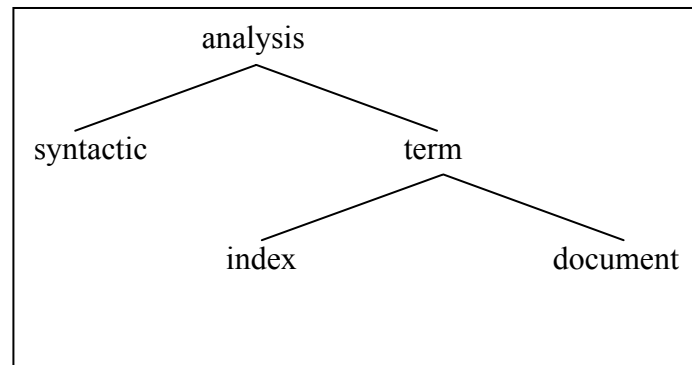


Figure 2.8: Head-modifier relations of "syntactic analysis of index terms"

The inverted file may contain all combinations, which may seem longer like "syntactic" "analysis" "term" "index" "document" "syntactic analysis" "analysis of term" "index term" "term in document" "syntactic analysis of term" "analysis of index term" "analysis of term in document" "index term in document" "syntactic analysis of index term" "syntactic analysis of term in document" and finally "syntactic analysis of index term in document".

Categorical information is needed for syntactic analysis, which is a group of word lists containing negative or positive representatives for word categories. These lists include function words, verbs, negative lists for gerund recognition, positive and negative lists for adjective recognition, and list of words ending with s in the stem. Categorical information effectively reduces the number of inverted file entries, which speeds up the system [44].

2.7.4.3 Phrase Matching

If the query for example is "index term" it searches in the inverted file for the combinations "index" "term" "index term". In other words, it goes down paths through the dependency trees, thus yielding tree or sub-tree or tree and sub-tree overlaps of structures between query and documents. Documents containing "index term" sub-tree in the above example have higher rank [44].

2.7.4.4 Syntactic Analysis Evaluation

Evaluation includes ranking correctness, which was proved that there is practically no means to arrive an error rate under 5% using syntactical methods only. Ranking correctness depends to a high degree on how good syntactic analysis is. On the other hand, recall and precision measures for IR evaluation may be used in syntactic analysis. Evaluation of IR systems will be discussed in the next section 2.8 [44]. The most popular techniques that use syntactic analysis are:

1. Augmented Transition Network (ATN).
2. Recursive Transition Network (RTN).

2.8 IR and ATE Evaluation

The criteria of IR systems evaluation include retrieval accuracy, processing time, cost-effectiveness, user satisfaction, effort required to learn the system, system utility, etc [47]. Although some researches use multiple criteria [46], majority of evaluation is conducted based on retrieval accuracy [47]. It is accepted to get moderate accuracy in current IR systems [7].

Kent et al [98] proposed the criterion of relevance and the measures of precision and recall to evaluation IR systems. "Precision is the ratio of the number of relevant documents retrieved to all retrieved documents. Recall is the ratio of the number of relevant documents retrieved to the number of all relevant documents in the search space" [47]. Precision and recall are the most common measures used to evaluate the effectiveness of IR systems [47].

In other words, Recall is the proportion of relevant documents retrieved from the total number of relevant documents in the database, and precision is proportion of relevant documents retrieved from the total number of retrieved documents [6]. They are closely correlated with how well a keyword represents documents in which it is a surrogate, and how well it can discriminate the documents from others. Hence the degree of representation and discrimination are the most important characteristics of index terms [15]. In principle, a search should achieve high recall by retrieving almost everything that is relevant, while at the same time maintaining high precision by rejecting a large proportion of extraneous items [17]. In this thesis we measure two types of performance:

1. Processing Performance is measured by utilization of CPU time and memory size during the process, and by time it takes from the beginning of parsing till getting the table of results for each technique, including stop words removal, stemming, computations and updating database.
2. Retrieval performance and accuracy of systems which is measured by five main measures; all of these measures are applied on document retrieval [6]. These measures can be adapted to be applied on index terms in term

extraction. Table 2.1 can be used to compute them. These measures include [48]:

Table 2.1: ATE evaluation measures.

No. of Terms	Relevant (Rel)	Irrelevant (Nrel)	Total
Retrieved (Ret)	A=(RetRel)	B=(RetNrel)	A+B (all ret. terms)
Not Retrieved (Nret)	C=(NretRel)	D=(NretNrel)	C+D (all not ret. terms)
Total	A+C (all relevant terms)	B+D (all not rel. terms)	A+B+C+D (all terms.)

- Recall (R): measures the ability of the user/system to extract the available relevant index terms.

$$R = \frac{A}{A + C} \quad (2.4)$$

- Precision (P): measures the ability of the user/system to extract only relevant index terms.

$$P = \frac{A}{A + B} \quad (2.5)$$

- Noise (N): is the proportion of irrelevant index terms found in the set of retrieved terms.

$$N = \frac{B}{A + B} \quad (2.6)$$

- Fallout (F): measures the probability that an irrelevant index term was retrieved.

$$F = \frac{B}{B + D} \quad (2.7)$$

- Generality (G): is the proportion of the index terms in the system that the query addresses.

$$G = \frac{A + C}{(A + C) + (B + D)} \quad (2.8)$$

On the other hand, Effectiveness (E) compares between Recall and Precision to get the best performance. The following formula was proposed by Van Rijsbergen to measure IR systems performance [9, 47].

$$E = 1 - \frac{(1 + \beta^2)PR}{\beta^2P + R} \quad (2.9)$$

Where P and R are precision and recall, and β is a parameter reflecting the relative importance of recall to precision defined by the user. In this thesis we used mainly the first three measures; recall, precision, and noise, which are sufficient IR measures for evaluation.

Statistical tools like SPSS also may be used with its famous tests to compare the results of different techniques. It measures weight average, standard deviation, correlation coefficient between different variables affecting the result. This type of analysis can be used to check to which extent the results obtained from each statistical technique are related or different. The problem of such tools that they are not comprehensive tools, you need to get the results of ATE system and enter them manually to the tool, which is not efficient. This is why we have built our own evaluation module in ATEWB.

2.9 Summary

In this chapter, we have discussed the literature review of information retrieval, automatic term extraction and related work. We have discussed language information and communication, natural language processing in IR, information retrieval fields including text, audio and video and then automatic term extraction (ATE). In ATE we have discussed different techniques or approaches that have been used for keywords extraction, which includes statistical techniques, probabilistic models, neural networks models such as self organizing feature map (SOFM), and syntactic analysis. Finally we have discussed IR evaluation metrics that are used to compare between different IR techniques in terms of effectiveness and performance.

In chapter 3 we discuss the different statistical techniques used for term weighting and how these weights are computed to decide which term could be used as an index term (keyword).

3 Automatic Term Extraction (ATE)

Several methods have been proposed for the extraction and constitution of a list of index terms from a specific domain. To build a monolingual terminology bank, different software packages have been implemented, which provide a list of likely terminological units, based on linguistic specifications. Different statistical methods, making effective use of recently available large-scale machine-readable corpora, have been tested to extract collocations. Another approach combines both linguistic and statistical knowledge in order to establish terms.

Automatic Term Extraction (ATE) is the extraction of index terms from special language corpora with the use of computer. Its applications include specialized dictionary construction, human and machine translation, indexing in books and digital libraries, hypertext linking, text categorization and indexing.

ATE and its related computations can be applied using one or more algorithms or techniques. In this thesis, we discuss statistical analysis techniques.

3.1 ATE Stages

ATE is the process of extracting words or phrases of a document, which are candidates to be index terms computationally. ATE is sometimes referred to as automatic indexing. Basic ATE the same as Automatic Indexing process consists of four stages:

- **Text parsing and tokenizing:** reading the words of text files or documents in a collection or a corpus.

- **Stop words removal:** removal of high frequency words that don't affect the meaning of a sentence, like prepositions, pronouns, conjunctions and other function words that do not convey much meaning.
- **Stemming:** It is used in IR to conflate morphological variants. It consists of rules and/or dictionaries used to remove prefixes and suffixes of words in a collection of documents.
- **Weighting of words:** using an algorithm to weight the remaining words, this weight will be used to find if a word is suitable to be an index term.

Indexing or ATE program decides which words, phrases or other features to use from the text of documents. This increases the speed of doing that manually thousands of times. The basic issue of ATE is to answer the question “Which terms should be used to index (describe) a document?”.

3.2 Single Term vs. Phrase Extraction

During term extraction process, two types of index terms arise: single terms and compound terms (phrases). A single term consists of one word like car, door, and approve etc., whereas a compound term or a phrase consists of two or more words like Dead Sea, Smart Board, and Information Retrieval etc.

As for single terms, statistical methods have been used to extract phrases; this includes finding all word pairs that occur more than n times in a collection or using part-of-speech tagger to identify simple noun phrases. Phrases have impact on both efficiency and effectiveness. For example, finding documents containing

the phrase “Dead Sea” is better than finding both words. Effectiveness depends on retrieval model in phrase extraction.

Phrase extraction process goes through three main steps [49]:

- **Tokenization:** The goal of the tokenization process is to determine sentence boundaries, and to separate the text into a sequence of single tokens (words) by removing extraneous punctuation. Single spaces may be used to delimit tokens.
- **Part-of-Speech Tagging:** The tagger is divided into two main phases:
 - Lexical Analysis: involves specifying part of speech (noun, verb, or adjective) of each word in using lexicon. Each word is marked up with its part of speech listed for it in the lexicon. If a word does not appear in the lexicon, the tagger will mark it as an unknown noun.
 - Contextual Analysis: further text processing to ensure that the part-of-speech tags are disambiguated, using several contextual rules.
- **Noun Phrase Identification:** the tagger rules are applied to the tagged words using a sliding window of maximum number of words that form a phrase. As the window slides over the words of the text, the noun phrase patterns are applied to the window contents. Then the longest matching rule is used to obtain the best noun phrase. Once a noun phrase is located, the window slides to the next word following the phrase and starts reading the contents of a new window.

3.3 ATE Statistical Techniques

In chapter two, the definition and steps of ATE process have been discussed. Four statistical techniques have been mentioned, by which the weight of a term is computed to judge if it is suitable to be index term candidate. This chapter describes the weight computations of each technique.

Each technique has its own algorithm. Each has its own advantages and disadvantages in terms of efficiency, effectiveness, and cost. In addition to statistical techniques algorithms, Porter's stemming algorithm for suffix stripping is described here.

Statistical techniques that are used in this thesis for automatic text analysis depend on frequency and rank of words or documents. We may use frequency data to extract words and sentences to represent a document, as Luhn assumed [50].

If we plot the relation between the frequency of words in a given text f and their rank order r as shown in figure 3.1 [18]. The result is a hyperbolic function which represents Zipf's law. This law considers the product of the frequency of words in a given text and their rank order almost constant [51].

As illustrated in figure 3.1, there will be an upper cut-off and a lower cut-off, significant words are the words between the two cut-offs. The words outside the cut-offs are considered less important and can not be used as index terms. Those words below the lower cut-off are considered rare, while those above the upper cut-off are considered familiar. The most significant words (content discriminators) are located in the center of the natural distribution curve between the two cut-offs to reach the peak value and fall down in both directions to reach

approximately zero at the cut-off points as shown in figure 3.1. Depending on some experiments and by trial and error, the cut-off points are created [50].

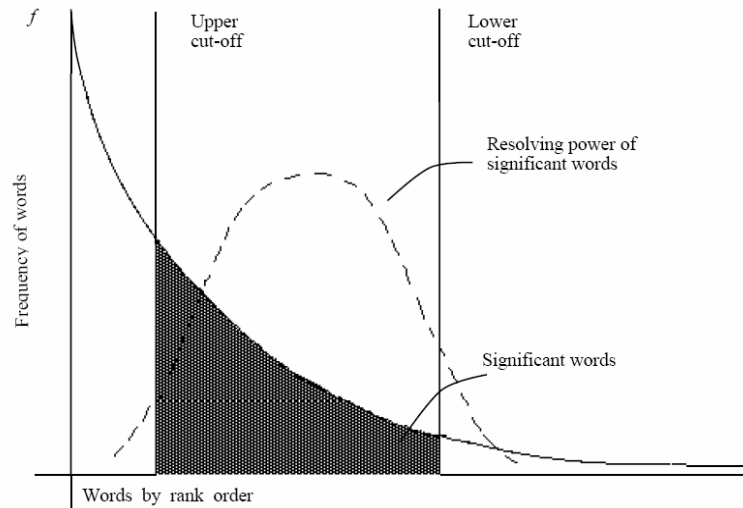


Figure 3.1: A plot of hyperbolic curve relating the frequency and rank [52].

For easier computations, Salton represented each document as a vector in an N -dimensional vector space (collection of documents) as illustrated in figure 3.2. The dimensions of the vector space (N) equals to the total number of distinct terms of the collection. Each document is represented by a vector of weights of the terms it consists of.

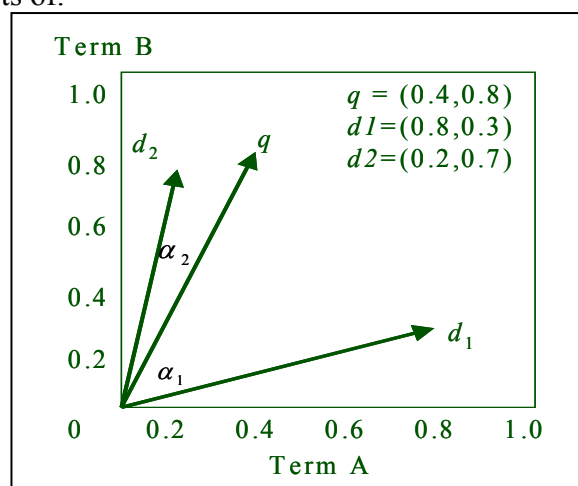


Figure 3.2: Document Vector Space

Figure 3.2 shows a 2-dimensional vector space, term A and Term B. In document $d1$ for instance, the weight of Term A is 0.8 and Term B is 0.3 which produces the vector (0.8, 0.3). The query also is presented as a document in the vector space to be compared with documents. The similarity is checked by measuring the cosine of the angle between the query and other documents. To check the similarity between the query q and the document $d1$ for instance, find the cosine of the angle $\alpha1$, as it approaches to 1 as they are more similar. If the cosine is 1, then the query and the document are exactly the same.

Let's define some notations that are used in the computations of this thesis:

Document space: $D = \{d_1, d_2, \dots, d_j, \dots\}$

D is the document space that contains all documents in the collection, and each document is denoted d_j .

The vocabulary or the set of all the different words or word types w_i appearing in D , is denoted by W_D . Thus: $W_D = \{w_1, w_2, \dots, w_i, \dots\}$

W_{d_j} is the vocabulary of a document d_j , a word w_i with respect to a specific document d_j is denoted by w_{ij} .

The following three functions are defined:

$n(S)$, is the number of elements of a set S .

$f(w_{ij})$, is the frequency of word w_i in document d_j .

$df(w_i)$, the document frequency of word w_i .

Lastly, the weight of a word w_i in a document d_j is denoted by $Weight_{ij}$. When the document is not specified use $Weight_i$, which means the weight of a word in the whole collection.

3.3.1 Term Frequency

In term frequency (TF) technique, the weight depends on the term frequency to judge its importance. After parsing, stemming, and stop words removal, if needed, we find how many times each word occurs in each document in the corpus. Then we find the summation of all word frequencies in document d_j . Then we divide each word frequency by the summation of frequencies. The result is the weight of each word. The words of highest weights are index terms candidates.

The weight in TF can be computed using the formula [53, 54]:

$$Weight_{ij} = \frac{f(w_{ij})}{\sum_i f(w_{ij})} \quad (3.1)$$

In order to get a weight in the range $[0, 1]$, obtain the word of maximum frequency in each document and divide each word frequency by the maximum word frequency in that document. To do this, equation (3.1) could be written as:

$$Weight_{ij} = \frac{f(w_{ij})}{Max(f(w_{ij}))} \quad (3.2)$$

Where $Max(f(w_{ij}))$: The maximum word frequency in document d_j .

Example:

Suppose that you have 3 documents. After parsing they can be written as:

$d1 = [\text{Information, Retrieval, Automatic, Extraction, Retrieval}]$

$d2 = [\text{Term, Extraction, Extraction, Extraction}]$

$d3 = [\text{Yousef, Sabbah, Information, Extraction}]$

$D = \{d1, d2, d3\}, n(D) = 3$

Or we can write:

$w11$ = Information, $w12$ = Retrieval, $w13$ = Automatic, $w14$ = Extraction, $w14$ = Retrieval

$w21$ = Term, $w22$ = Extraction, $w23$ = Extraction, $w24$ = Extraction

$w31$ = Yousef, $w32$ = Sabbah, $w33$ = Information, $w34$ = Extraction

$W_D = \{\text{Information, Retrieval, Automatic, Extraction, Term, Yousef, Sabbah}\}$

$n(W_D) = 7$

We need a search algorithm to search for similar words in these vectors and count the number of occurrences of each word in each vector and divide by vector length, refer to TF algorithm represented by pseudo code in figure 4.6.

To be represented in the vector space, each document will be represented by a 7 elements vector of weights (using formula 3.2) as follows:

To compute the vector $d1$ for example, start with the word "Information" as shown above in the set W_D , count how many times it occurs in document $d1$ and continue with other words until you finish the set. Find the word with maximum frequency which is "Retrieval" in $d1$ with frequency equals to 2, then divide the frequency of each word by this maximum frequency. Each word occurred once in document $d1$ except the word "Retrieval" which occurred twice. Hence, the weights of the words of $d1$ are 0.5, 1.0, 0.5, and 0.5 respectively. The weight of all words of the set W_D that didn't appear in $d1$ is 0. The result is shown below:

$d1 = [0.5, 1.0, 0.5, 0.5, 0.0, 0.0, 0.0]$

$d2 = [0.0, 0.0, 0.0, 1.0, 0.33, 0.0, 0.0]$

$d3 = [1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0]$

The above vectors can be written using a 7x3 term-by-document matrix in which each vector represents a column and each word-weight represents a row. Each entry W_{Dij} represents the weight of the word w_{ij} as follows:

$$W_D = \begin{pmatrix} 0.5 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.0 \\ 0.5 & 1.0 & 1.0 \\ 0.0 & 0.33 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad (3.3)$$

These documents can be presented by a 7-Dimension vector space. The dimension of the vector space depends on the number of unique words or W_D .

In this example, the terms with highest weights are the index terms. As discussed above, in the first document $d1$, the term "Retrieval" has the highest weight, so it can be used as a keyword. In $d2$, the term "Extraction" may be used as a keyword. Lastly, in document $d3$, any of the terms may be used as a keyword, because they have the same weight. It is obvious that this process needs much computations time for a huge collection with long documents.

3.3.2 Inverse Document Frequency

Inverse document frequency (IDF) is commonly used in Information Retrieval. IDF is defined as: $-\log_2 df(w_i)/n(D)$, where $n(D)$ is the number of documents in the collection and $df(w_i)$ is the document frequency or the number of documents that contain w_i [55]. For instance, if we have a collection of abstracts, each starts

with the word "Abstract" while it doesn't give us a meaning although it appears in all documents. So, the word which appears in fewer documents seems to be more important.

Attempts have been made to apply weighting based on the way the index terms are distributed in the entire collection. The index term vocabulary of a document collection often has a Zipfian distribution, that is, if we count the number of documents in which each index term occurs and plot them according to rank order, then we obtain the usual hyperbolic shape. Sparck-Jones [9] showed experimentally that if there are N documents and there is an index term, which occurs in n of them then a weight of $\log (N/n)$ leads to more effective retrieval than if the term were used not weighted. If indexing specificity is assumed to be inversely proportional to the number of documents in which an index term occurs then the weighting can be seen to be attaching more importance to the more specific terms [9].

IDF weight is based on the assumption that term importance is inversely proportional to the total number of documents to which the term is assigned. That is, the smaller the number of documents containing the term, the greater the importance of the term for discriminating between the documents. A measure of the inverse document frequency for a term i can be written as [9]:

$$Weight_i = \frac{1}{df(w_i)/n(D)} \quad (3.4)$$

In general, we can compute the weight of a term in this technique using the formula [31]:

$$Weight_i = \log_2 \frac{n(D)}{df(w_i)} \quad (3.5)$$

If we distribute the log, the equation (3.5) becomes:

$$Weight_i = \log_2 n(D) - \log_2 df(w_i) \quad (3.6)$$

For the weight to be in the range $[0, 1]$, divide by $\log_2 n(D)$ to get:

$$Weight_i = 1 - \frac{\log_2 df(w_i)}{\log_2 n(D)} \quad (3.7)$$

If we compute the weight of terms in the previous example using IDF technique (using equation 3.7), we get the following vector space, term-by-document matrix:

$$W_D = \begin{pmatrix} 0.37 \\ 1.0 \\ 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix} \quad (3.8)$$

To compute the vector W_D for example, start with the word "Information" as shown in the set W_D in the previous example, count how many documents contain it and continue with other words until you finish the set. "Information", for instance, occurred in 2 documents out of 3. Use equation 3.7 to get:

$$Weight(Information) = 1 - \frac{\log_2(2)}{\log_2(3)} = 0.37$$

Repeat for the rest in the same procedure to establish a vector 3.8 for the collection. Hence, the weights are: {(Information, 0.37), (Retrieval ,1.0) , (Automatic, 1.0) , (Extraction 0.0), (Term, 0.0), (Yousef, 0.0) , (Sabbah, 0.0)}.

The candidate index terms of the collection are (Retrieval and Automatic), because they occurred in only 1 document and their weights are both 1.

Notice that, although "Extraction" frequently appeared in d_2 , it might not be used as a keyword, because it appears in all 3 documents and its weight is zero.

3.3.3 Combination of TF and IDF

TF weighting is not always efficient, especially if most of the terms have the same frequency or have small deviation, and hence the same importance. That does not give us a good result. The same happens with IDF weighting when most of terms have approximately the same document frequency, which means that we have to find a new technique. If we combine TF and IDF by multiplication, we derive a new technique, which is TFXIDF technique. This technique leads to the result that if TF and IDF weights of a term are high, this term has a very good opportunity to be an index term. In other words the most important terms of a collection are those, which occur more frequently in one document, and less frequently occurs in other documents of the collection. For instance, if the words 'parallelism' and 'computer' both occur in 100 documents, they have the same IDF weight even in a document in which 'parallelism' occurs 20 times and 'computer' only once. For this reason in widely used TFXIDF weighting, IDF is multiplied by the number of occurrences of a term i in a document d_j .

To compute the weight of TFXIDF, multiply the term frequency $f(w_{ij})$ by IDF weight in equation (3.6) to get [32]:

$$Weight_{ij} = f(w_{ij}) (\log_2 n(D) - \log_2 df(w_i)) \quad (3.9)$$

To force all weight values of TFxIDF to fall within the range 0 and 1, it is necessary to normalize the weights. By this operation we increase retrieval accuracy that longer documents are not unfairly given more weights. Normalized weights may be obtained by formula (3.10).

$$Weight_{ij} = \frac{f(w_{ij})(\log_2 n(D) - \log_2 df(w_i))}{\sqrt{\sum_{i=1}^t [f(w_{ij})(\log_2 n(D) - \log_2 df(w_i))]^2}} \quad (3.10)$$

Again for the previous example (using equation 3.10), the term-by-document matrix becomes:

$$W_D = \begin{pmatrix} 0.163 & 0.0 & 0.577 \\ 0.882 & 0.0 & 0.0 \\ 0.441 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.577 \\ 0.0 & 0.0 & 0.577 \end{pmatrix} \quad (3.11)$$

It is hard to compute the above matrix 3.11 manually, it was computed using our developed tool. If this result is compared with 3.3 and 3.8, we note that the result is the same as in 3.3 with TF except that "Extraction" is removed from the keywords list by the effect of IDF.

To improve weighting, Spark-Jones [18] added a third weighting input, which is document length. A term of the same frequency in a short document and in a long one is likely to be more valuable for the former [18].

$$DL_j = \sum_i f(w_{ij}) \quad (3.12)$$

The use of document length described below actually normalizes the measure by the length of an average document [18].

$$NDL_j = DL_j / AVGDL \quad (3.13)$$

Where:

DL_j : is the document length or the total of term occurrences in document d_j

NDL_j : is the normalized length of document d_j

$AVGDL$: is the length of an average document.

If it is combined with equation (3.9) we get the formula [18]:

$$CW_{ij} = \frac{f(w_{ij}) \cdot (\log_2 n(D) - \log_2 df(w_i)) \cdot (K1 + 1)}{f(w_{ij}) + K1 \cdot ((1 - b) + b \cdot NDL_j)} \quad (3.14)$$

Where CW_{ij} is the combined weight of the word i in document d_j , $K1$ and b are tuning constants between 0 and 1 to reduce or increase the effect of term frequency. In tests $K1=2$ and $b=0.75$ are found to be effective. This formula has proved effectiveness in trials during the TREC program [56].

Thus, TFxIDF weighting assigns a high degree of importance to terms occurring frequently only in few documents of a collection.

3.3.4 Term Discrimination Value Model

Salton and Yang [32] proposed the Term Discrimination Value (TDV) on 1975 based on vector space, vector space refers that documents and queries are vectors in an n -dimensional space for n terms. TDV measures the degree to which the use of a term will help to distinguish the documents from each other. The discrimination value of a term i can be computed by comparing $AVGSIM$ (the

average document-pair similarity calculated by comparing the words of documents) with $(AVGSIM)_i$ (the average document-pair similarity if the term i is removed from all the documents) [33]. To compute average similarity or density of document space, we use the following formula:

$$AVGSIM = K \sum_{j=1}^n \sum_{k=1, k \neq j}^n similar(d_j, d_k) \quad (3.15)$$

Where

K is a normalizing constant and $similar()$ is a similarity function such as cosine correlation factor, which was proposed by Salton to measure similarity of documents as follows [57]:

$$similar(d_j, d_k) = \frac{\sum_{i=1}^t (Weight_{ij}) \cdot (Weight_{ik})}{\sqrt{\sum_{i=1}^t (Weight_{ij})^2 \cdot \sum_{i=1}^t (Weight_{ik})^2}} \quad (3.16)$$

Where: $similar(d_j, d_k)$: similarity between document d_j and document d_k .

$Weight_{ij}$: weight of term i in document d_j .

$Weight_{ik}$: weight of term i in document d_k .

t : number of similar terms in documents d_j and d_k .

The range of similarity between two documents is (0-1, 0 for no similarity and 1 for identical).

Average similarity can be computed more efficiently using an average document or centroid. Frequencies in the centroid vector are average of frequencies in document vectors. We get:

$$AVGSIM = K \sum_{i=1}^n \text{similar}(\overline{D}, D_i) \quad (3.17)$$

Let $(AVGSIM)_i$ be the density with term i removed from documents, we can compute discrimination value for term i :

$$DISCVALUE_i = (AVGSIM)_i - AVGSIM \quad (3.18)$$

Index terms may be placed then into three rough categories according to their discrimination values:

1. The good discriminators with positive $DISCVALUE_i$ whose introduction for indexing purposes decreases the space density.
2. The indifferent discriminators with a $DISCVALUE_i$ close to zero whose removal or addition leaves the similarity among documents unchanged.
3. The poor discriminators whose utilization renders the documents more similar, producing a negative $DISCVALUE_i$.

Once again, $DISCVALUE_i$ computes the weight of each word with respect to the collection not to each document. Some propose the multiplication of these measures by document dependent measures such as term frequency $f(w_{ij})$. So it can be computed to each term in each document by combining the term frequency factor with the discrimination measure:

$$Weight_{ij} = f(w_{ij}) \bullet DISCVALUE_i \quad (3.19)$$

Another formula also can be used to measure the distance between documents in a vector space [57]:

$$D_{jk} = \sqrt{\sum_{i=1}^t (Weight_{ij} - Weight_{ik})^2} \quad (3.20)$$

Where D_{jk} is the distance between document j and document k that measures how much document j is different from document k .

3.4 ATE Auxiliary Approaches

The following NLP approaches may be used to increase extraction efficiency and effectiveness. They involve stemming and stop-words removal. These approaches are sometimes combined in the parsing stage to reduce the number of words for which weights will be computed which improve performance. On the other hand, we get rid of some meaningless words from the text to improve accuracy.

3.4.1 Porter's Stemming

Porter proposed an algorithm for stemming words. It strips terms of common suffixes that indicate plurality and verb tense [26]. For instance, the words (compute, computes, computing, computed, computer, computerize, computerized, computation) all are stemmed from the word (compute) and give the same meaning. It will more efficient if the eight words are treated as one word in reducing computations and improving accuracy.

Figure 3.3 illustrates the flow chart of Porter's algorithm. When a new word is entered, its length is checked. If it is less than three letters, it is passed and moves to the weighting stage of ATE. If its length is greater or equals to three letters, it passes through six steps shown in table 3.1.

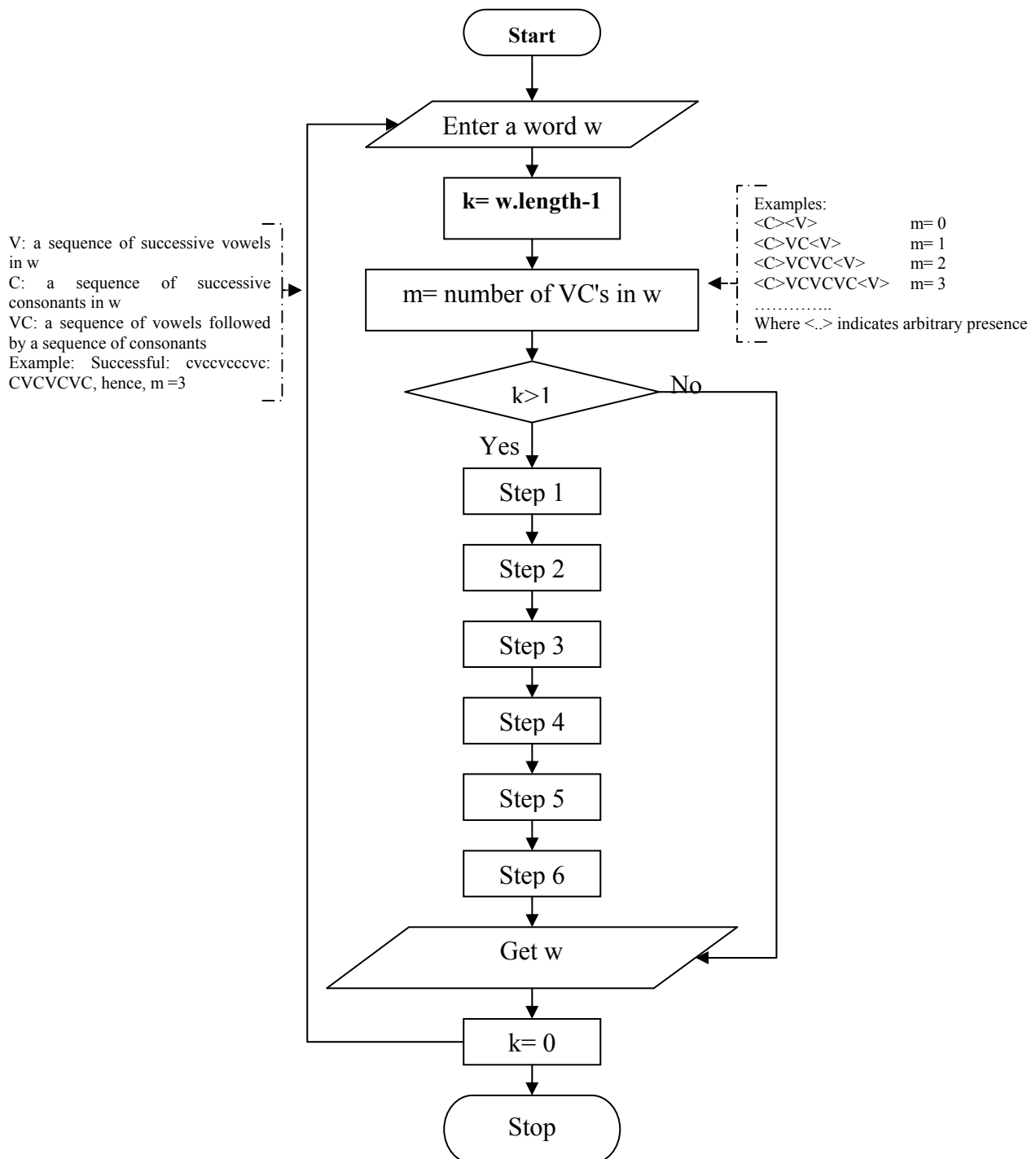


Figure 3.3: Porter's stemming algorithm

As shown in table 3.1, Porter's algorithm works through a string in a series of steps. In the first step, it removes pluralizations, including special cases. In the

second step, Porter's converts *y* into *i* if the rest of the term contains a vowel, see table 3.1 (step2). In step three, Porter's does pattern matches on some common suffixes, such as those in table 3.1 (step3). These transformations remove suffixes and replace them with their roots. For some special words, more transformations are needed, and in step four, these cases are handled further transformations as those in table 3.1 (step4).

In step five, the stripped word is checked against more suffixes in case the word is compounded, including suffixes in table 3.1 (step5). The sixth and final step checks if the stripped word ends in a vowel and fixes it appropriately, see table 3.1 (step6) [58]. After the word is stemmed, it will be ready to move to the weighting stage using statistical techniques.

With Porter's stemming algorithm, word extraction is made much more powerful, as variants of words (such as walk, walking, and walked) can be looked up as the algorithm proceeds, which ensures that a search can locate appropriate terms that would otherwise be missed [58].

3.4.2 Stop-words Removal

An optional operation in text analysis is stop-words removal. Stop- words are function words that do not add meaningful information to the text like: conjunctions (and, or, because, as, but, so), transitions (although, therefore, however, furthermore), modifiers (few, little, much), prepositions (in, on, of, off) and others.

Table 3.1: Steps of Porter's Stemming Algorithm [57]

Step	Stemming Operation	Step	Stemming Operation
1	sses -> ss ies -> i ss -> s s -> (m > 0) eed -> ee (*v*) ed -> (*v*) ing -> at -> ate bl -> ble iz -> ize	2	(*v*) y -> i
3	(m > 0) ational -> ate (m > 0) tional -> tion (m > 0) enci -> ence (m > 0) anci -> ance (m > 0) izer -> ize (m > 0) abli -> able (m > 0) entli -> ent (m > 0) eli -> e (m > 0) ousli -> ous (m > 0) ization -> ize (m > 0) ization -> ize (m > 0) ation -> ate (m > 0) ator -> ate (m > 0) alism -> al (m > 0) iveness -> ive (m > 0) fulness -> ful (m > 0) ousness -> ous (m > 0) alitii -> al (m > 0) iviti -> ive (m > 0) biliti -> ble	4	(m > 0) icate -> ic (m > 0) ative -> (m > 0) alize -> al (m > 0) iciti -> ic (m > 0) ical -> ic (m > 0) ful -> (m > 0) ness ->
5	(m > 1) ance -> (m > 1) ence -> (m > 1) er -> (m > 1) ic -> (m > 1) able -> (m > 1) ible -> (m > 1) ant -> (m > 1) ement -> (m > 1) ment -> (m > 1) ent -> (m > 1) and (*s or *t) ion -> (m > 1) ou -> (m > 1) ism -> (m > 1) ate -> (m > 1) iti -> (m > 1) ous -> (m > 1) ive -> (m > 1) ize ->	6	(m > 1 and *d and *l) ->

When these words are removed from computations and analysis, performance increases as well as accuracy. Because they always have the highest frequency in documents, which means that they will be the index terms while they are not and in fact, they have no meaning, they are used to connect words together to form sentences.

3.5 Summary

In this chapter, we have focused on automatic term extraction that includes extracting the keywords to be used as an index to search engines. We have explained the main stages of a term extraction system. We have used four statistical techniques including term frequency, inverse document frequency, combined term frequency-inverse document frequency, and term discrimination value model. These statistical techniques are used for term weighting to decide on which terms to be the best index terms that describe a document or a collection of documents. We have focused on how to compute the weights in each technique with some examples to clarify the operation. Finally, we have discussed related natural language processing techniques that may be used to improve ATE performance and effectiveness, such as Porter's stemming algorithm and stop words removal.

Next chapter describes the developed tool used to evaluate different techniques used in our work. It is named automatic term extraction workbench (ATEWB).

4 Automatic Term Extraction WorkBench

(ATEWB) System Description

ATE plays a main role in information retrieval; it is the first and the main stage of any IR system. Many techniques and tools have been developed to improve this process. However, the cost of manual indexing is very high. This fact urged attempts to automate the process. In addition, it is interesting to develop a customized computational tool that performs this stage to build the comparison between different techniques depending on some experiments and their results. To the best of my knowledge, there is no comprehensive tool that covers our work. Most of them can be used in a specific part. This is why ATEWB is developed.

This chapter describes our developed tool which is an ATE System referred to as Automatic Term Extraction WorkBench (ATEWB). It goes through the main idea and function of the system and system design; this includes the main classes and methods using class diagram and UML notation, which may help to understand the whole system. Then all algorithms of ATEWB system (including all used statistical techniques) are explained and described using pseudo code representation.

4.1 An Overview of ATEWB

This thesis discusses Automatic Term Extraction WorkBench (ATEWB), a tool developed for content analysis of a text file or a collection of text files or

(electronic documents). The ATEWB extracts the index terms or the terms that can be used as keywords to describe the contents of those electronic documents.

The overall architecture of our system is described in figure 4.1. The ATEWB contains different modules that can be used to extract index terms.

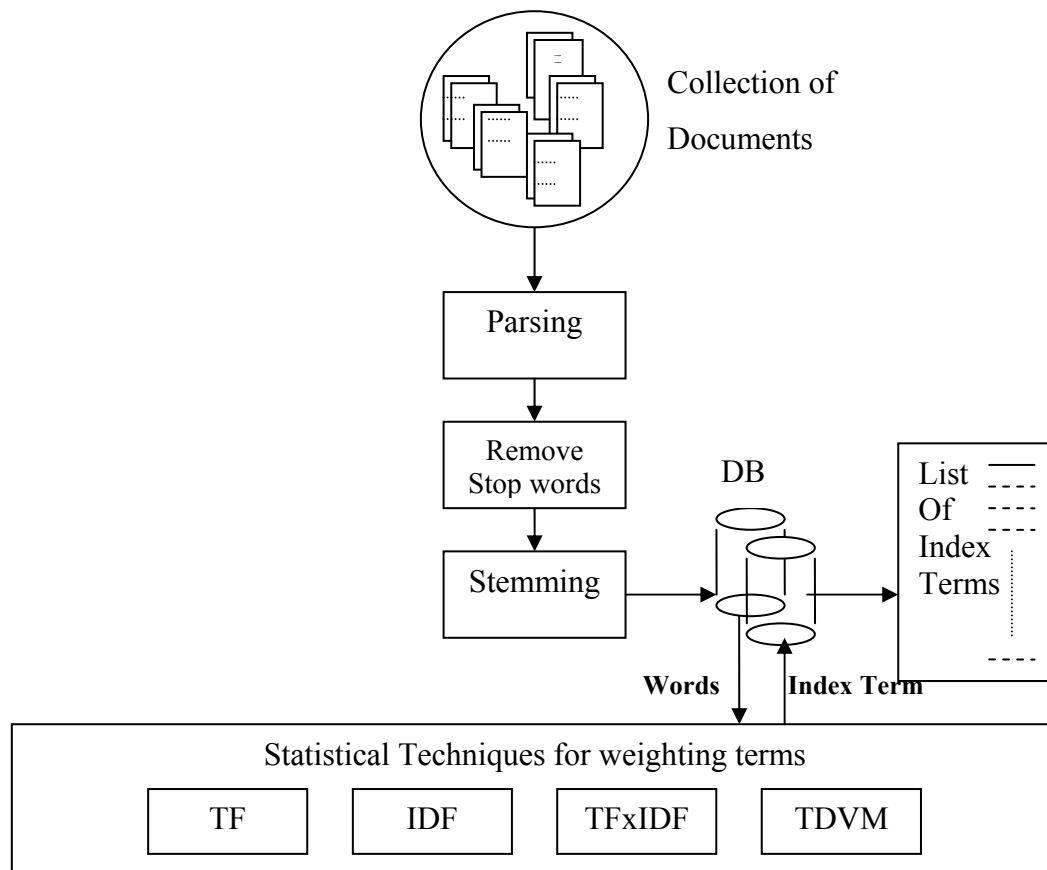


Figure 4.1: Main stages of ATEWB system.

ATEWB main modules are:

- **Text Parsing:** distinguishes between each single word using a tokenizer with space as a delimiter between words, because any other delimiters are cleaned from the text by removing punctuation marks.

- **Stop-word removal:** removing stop-words using a stop-word list to get rid of any function word like on, of, over, this, at, .etc.
- **Stemmer:** stemming of words using any stemming algorithm, in ATEWB Porter's algorithm is used; this is done by stripping suffixes from words to make it easier to find similar-meaning words.
- **Term Weighting:** four statistical techniques were used for weighting terms. Depending on the weight of each term, candidate index terms could be extracted.
- **Evaluation:** computing Recall, Precision and Noise to measure the effectiveness of the used statistical techniques and ATEWB Efficiency.

4.2 ATEWB package

The system ATEWB is implemented using JAVA. ATEWB package got benefit of already built-in packages in JAVA. Figure 4.2 illustrates the UML diagram of ATEWB package with main classes and how it is connected to the other built-in packages. UML stands for Unified Modeling Language, which is a standard notation for modeling object-oriented systems, it graphically describes a set of elements connected together to build a system. It uses some UML relations like associations and dependencies. Association notation is a dashed arrow (--->), which represents the ability of one instance to send a message to another. Dependency notation is an arrowed line (—>) from one class to another, which represents that the first class depends on the result of the second class. Inheritance notation is a closed arrow (—>). As illustrated, ATEWB is the main package with

Driver as the main class of the package. Other classes of ATEWB that are connected to Driver class are: Dialog, fileNewDialog, Stemmer, computeTF, computeIDF, computeTFxIDF and computeTDVM. ATEWB package performs all algorithms used in the system.

The other used built-in packages are java.lang, java.awt, javax.swing, java.sql, java.util and java.io.

- The language package java.lang is automatically imported into every Java program; it contains the language's main support classes which are fundamental to the design of the Java programming language.
- The Abstract Window Toolkit (AWT) package java.awt, provides support for Graphical User Interface (GUI) programming and includes such features as user interface components, event-handling models, layout managers, graphics and imaging tools, and data transfer classes for cut and paste.
- The javax.swing Provides a set of "lightweight" (all-Java language) components that work the same on all platforms, swing components are Pure Java versions of the existing AWT component set, such as button, scrollbar, and label, with an additional set of components, such as tree view, table, and tabbed pane.
- The SQL package java.sql contains classes that provide the API for accessing and processing data in a data source, it allows the Java platform to connect with almost any database, even those written in other languages such as Structured Query Language (SQL), the database package used in this thesis is MYSQL 5.0.

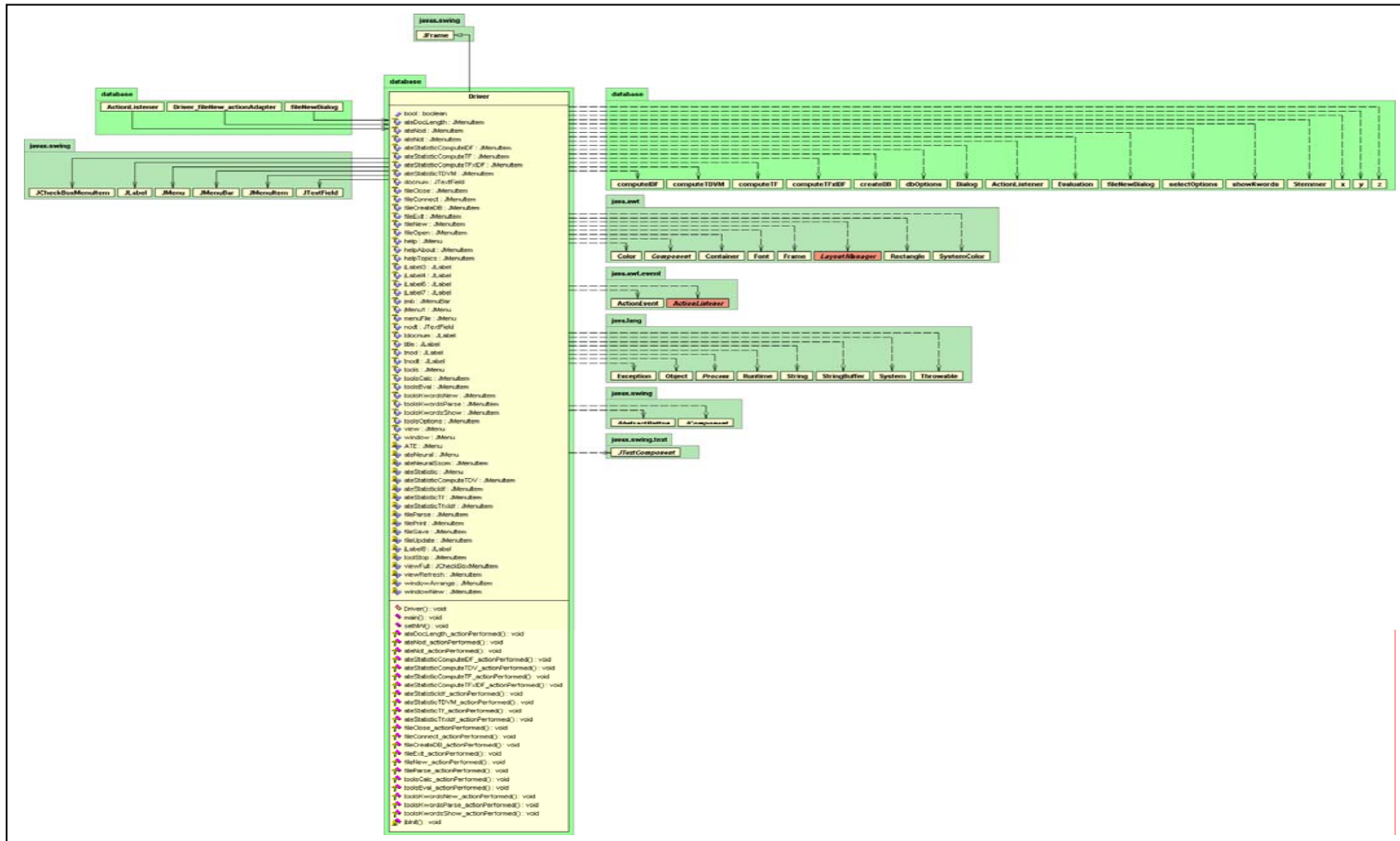


Figure 4.2: UML diagram of ATEWB package

- The java.util package contains various utility classes and interfaces that are crucial for Java development such as date and time facilities.
- The I/O package java.io provides support for reading and writing data to and from different devices, which includes input stream classes, output stream classes, file classes, and the StreamTokenizer class used for text parsing, which is the first phase in ATEWB system.

4.2.1 Design Issues

It may be asked, why to use database engine? And why we choose to use MYSQL? In this context, it is necessary to mention that instead of a programming language, a database engine (MYSQL here) is used as a data store and to do all computations including term and document frequencies, inverted file creation, correlation factors, similarity function, recall, precision, noise, and efficiency. This is one of the most important design issues that speed up data processing and heavy computations. It uses table indexing, multi-threading and applies high performance algorithms in computations using small pieces of SQL commands instead of tens of java code lines.

MYSQL database engine has suitable features that exactly fit this research. As described in MYSQL manual on its web site,³ it has the following main features:

- **Speed and Portability:**

³ MYSQL web site: <http://www.mysql.com>

1. Works on many different platforms, such as Windows 9x, Me, NT, 2000 and XP, different versions of UNIX and Linux and others.
 2. Fully multi-threaded using kernel threads. This means it can easily use multiple CPUs if they are available.
 3. A very fast thread-based memory allocation system.
 4. Very fast joins using an optimized one-sweep multi-join.
- **Scalability:**
 1. Handles large databases. MYSQL Server has been used with databases that contain 50 million records. MYSQL claims that some users used MYSQL Server with 60,000 tables and about 5,000,000,000 rows.
 2. Up to 32 indexes per table are allowed. Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 500 bytes.

In addition, the Connector/J interface provides MYSQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available by SUN company. All these features made MYSQL suitable in the field of study and research.

4.3 ATEWB System Requirements

4.3.1 Hardware Requirements

Hardware requirements depend on size of the collection; it was tested on P2 400MHz processor with 64MB RAM for a collection size of 3.19MB of html files which consists of 206863 word, 10991 unique terms after removing stop words. The performance suffered from a big delay in the first three techniques, while

hanging with no response in the fourth; see Table 4.1, which illustrates 3 experiments. The second experiment was done on P3 800MHz processor with 128MB RAM, the performance was somewhat acceptable, see Table 4.1. The same collection was tested on P4 2.4GHz with 512MB RAM, the performance was good, refer to table 4.1. The test was repeated on other systems, from which the minimum system requirements to run well can be deduced:

1. P3 800MHz processor and more.
2. 128MB RAM and more.
3. 100MB minimum free hard disk space to install the package and its software requirements.

But if the collection is greater in size such as search engines, it will need computers with high specifications, suitable for huge computations. Multiprocessor servers with parallel computing, or clusters of servers that form together a supercomputer should perform this type of work.

Table 4.1: Performance Experiments for ATEWB system requirements

ATE Statistical Technique	Description					
	Exp1 400MHz, 64MB		Exp2 800MHz, 128MB		Exp3 2400MHz, 512MB	
	Time (minuets)	Table records	Time (minuets)	Table records	Time (minuets)	Table records
TF	12:17:99	11,053	06:00:40	11,053	02:29:33	11,053
IDF	15:13:22	10,991	07:24:12	10,991	02:59:43	10,991
TFxIDF	25:13:75	11,053	13:31:04	11,053	03:47:27	11,053
TDVM	>500 Min	10,991	312:18:00	10,991	43:18:77	10,991
Notes	Collection size: 3.19MB. Document type: html. No. of words: 206,863. No. of documents: 2. No. of unique terms: 10,991					

4.3.2 Software Requirements

It was tested on different versions of MS Windows and Linux RedHat, and it was operational, with higher performance on Linux.

1. Microsoft Windows 98, 2000 or XP, Linux RedHat7.0 or more.
2. MYSQL server 5.0, or connected to a MYSQL server on the network.
3. Java runtime virtual machine.

4.4 ATEWB Main Classes and Algorithms

This subsection describes ATEWB package and developed classes, their UML Diagram, and algorithms in pseudo code. These classes together produce the operational system with MYSQL as the database engine to store and process generated textual data. ATEWB package has been tested on both Microsoft Windows and UNIX operating systems and it operates well, without any change in the classes. Test has been also done on the network and the performance was very good, in this case the database server is installed on another machine than the client's. If another database like Oracle has been used, nothing would have been done except using the Oracle connector provided by SUN instead of MYSQL connector, this can be done by changing one line in the dbConnector class.

The declarations in table 4.2 will be used in the rest of this chapter; they are instances of the most important classes that may be needed to call methods from other classes.

Table 4.2: ATEWB classes and instances used to call their methods

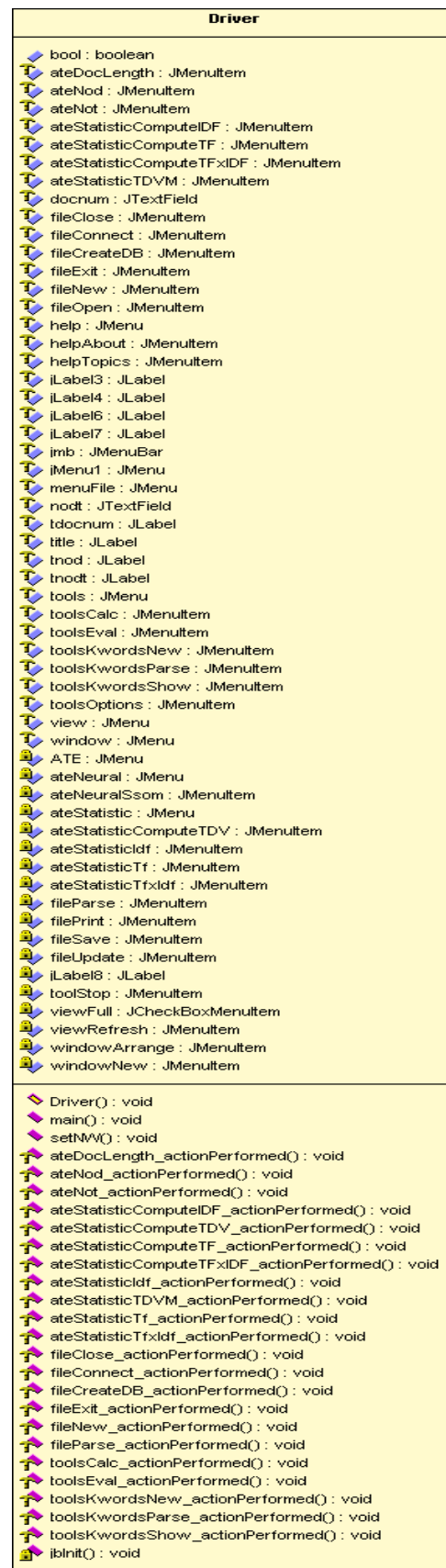
Class Name	Instance	Description
Driver	drv	The main class
Stemmer	smr	Porter's stemming algorithm.
Stop	stp	Stop word removal before parsing
dbConnection	cn	Connect to database
dbOptions	dbo	Database options (user, password, schema name).
FileNavigator	fn	Search for files of specific type
fileNewDialog	fnd	Start new ATE project
computeTF	tf	Term Frequency
computeIDF	idf	Inverse Document Frequency
computeTFxIDF	tfidf	Combination of TF and IDF
computeTDVM	dv	Term Discrimination Value Model
errorDialog	ed	Error dialog box
Evaluation	eval	Efficiency evaluation of ATEWB
showTF	shtf	Show TF result
showIDF	shidf	Show IDF result
showTFxIDF	shtfxidf	Show TFxIDF result
showTDVM	shtdv	Show TDVM result

4.4.1 Driver Class

Driver is the main class of ATEWB that controls and manages all other classes. When executed the main screen appears in GUI. The main screen contains the main menu; from file submenu choose "New", this will execute the fileNewDialog class, which extends Dialog class, and a dialog box appears to the user to start a new term extraction operation, this dialog contains the following options:

- Choose the full path where the text collection to be extracted is stored.
- Choose the full path where the result will be stored.
- Choose file type in the collection to be extracted (text, htm or html) file.
- Specify if you want to use stemming or not.
- Choose the stop word list you want to use.

Figure 4.3: Driver class diagram



Pseudo Code for Main Driver

Step1: Declare the class and its public and private variables

Step2: Get instances from other needed classes

Step3: Show the main user interface with main menu using **main()** method

Step4: Start events and take suitable actions by calling methods of other classes

Step5: To start a new extraction, call **fnd.setOptions()** method and set options; collection path, use stop word list or not, document type, stem or not, all in GUI.

Step6: To connect to database, call **cn.dbConnection()** after setting database options needed for connection, which includes user name, password, schema name and database server. Use the following methods: **dbo.setUser(), dbo.setPWD(), dbo.getDBname(), dbo.getDBserver()**.

Step7: To parse the documents in the collection, stem words, remove stop words and store words in the database, call **smr.doStemming()** method

Step8: To get the number of documents in the collection, call **fn.getNod()**

Step9: To get the number of terms in the collection, call **idf.getNODT()** method

Step10: Depending on the statistical technique chosen in statistical techniques submenu in ATE menu, call the following methods: **tf.tf(), idf.idf(), tfidf.tfxidf(), dv.discValue()**

Step11: To close the main Driver window and other open windows, call the output method **System.exit(0)**

Figure 4.4: Pseudo Code for Main Driver

If you choose "New ATE" from file submenu, this will execute `fileNewDialog` class that sets new ATE options. "Connect to Database Server" in file submenu executes `dbConnection` class to establish connection between ATEWB and the database. If "Parse" is chosen from file submenu it will execute `FileNavigator` class that navigates the collection to find all documents of chosen type, it also executes `Stop` and `Stemmer` classes depending on the chosen options, then text parsing and removing stop words operations start using a list stored in the database, finally, stemming operation is performed. The result is a table containing two fields: stemmed terms and their document ID.

`Driver` class is connected to most of the classes as reverse associations, which allow receiving messages from them, refer to figure 4.2. `Driver` class consists of 25 methods, one of them is the constructor and the others are used to build and show different user interfaces and menus of ATEWB. Figure 4.3 illustrates `Driver` UML class diagram showing all class properties and methods. On the other hand, figure 4.4 illustrates the pseudo code that describes the main steps of the main class named `Driver.class`.

4.4.2 GUI, Navigation and Connection Classes

The following classes are used to generate the GUI's such as dialog boxes (`Dialog`, `fileNewDialog`), document navigation (`FileNavigator`), and connection to database (`dbConnection`).

4.4.2.1 Dialog and fileNewDialog Classes

As illustrated in figure 4.2 Dialog and fileNewDialog classes are connected to Driver class as reverse associations, fileNewDialog inherits Dialog and is connected with reverse dependency to Driver. fileNewDialog class is executed when "New ATE" is chosen from file submenu in the main menu. When executed, the dialog box described in Driver above appears to set options of the new ATE operation.

4.4.2.2 FileNavigator Class

This class is executed when "Parse" is chosen from "File" submenu in the main menu. When executed it starts navigating the specified collection path to find all documents of the selected type. It is connected with all classes that need navigation of files and folders with dependencies.

4.4.2.3 dbConnection Class

This class is one of the most important classes in ATEWB, it is the class used to establish connection with MYSQL database. MYSQL is the data store that stores all terms after processing the contents of free text in documents. All classes of statistical techniques connect to database and use SQL statements for computing frequencies and weights of the terms on which the ATEWB depends to extract index terms. The last result of any technique is stored in the database via this class with Java connector.

4.4.3 Statistical Techniques

In this subsection, four statistical techniques are described. A class diagram for each technique is developed. Class diagram gives a general description of the class and its methods. Pseudo code of the statistical techniques algorithms is presented. For more details about the class implementation refer to appendix D.

4.4.3.1 Term Frequency Classes

This technique has two classes, computeTF class computes the weights, and showTF class shows the table of terms and their weights. Choose "Compute TF" from "Statistical Techniques" submenu in "ATE" menu in the main menu to execute the former and "TF Result" to execute the latter. When "Compute TF" is executed, it connects to ATE table containing the terms in the database through dbConnection class. Term frequencies and weights are computed using SQL statements. The result is stored in a table named 'result' in the database; this inverted file contains term ID, term, term frequency, term weight and document ID. By an SQL statement the range of weight 'needed for a term to be candidate index term' is specified. Then connection to database is ended, and the result appears in a GUI table by executing showTF class. Figure 4.5 illustrates the main methods and properties of TF classes.

The method **tf()** is used to compute weights in computeTF class depending on term frequency. Figure 4.6 illustrates the pseudo code of term frequency algorithm. For more details on the implementation of TF classes, refer to appendix D.

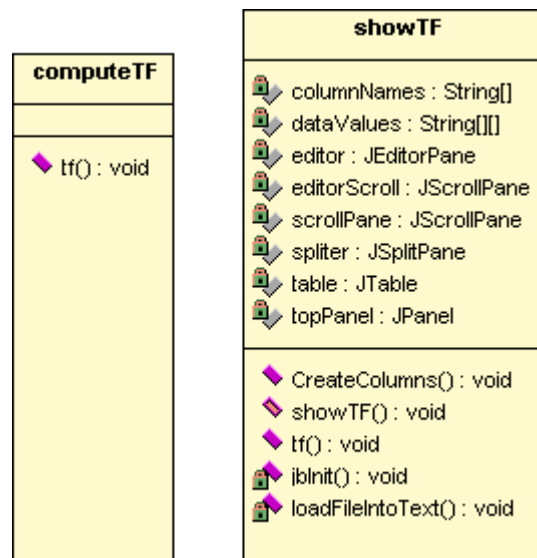


Figure 4.5: Class Diagram of TF classes.

Input:	
	A table in the database containing terms of the collection and their document IDs.
Output:	
	A table updated with each term and its frequency in each document, and term weight after computations.
Pseudo code of tf() method. Step1: Get instance of needed classes Step2: Establish connection with ATE database Step3: Delete the contents of term frequency table in database by SQL statement, if it contains old data	

- Step4:** Use SQL statement to get the number of documents (the same number of documents in the collection) in the terms table that created after parsing, using document ID field
- Step5:** Set numDocs to the obtained number of documents
- Step6:** For ii going from 0 to numDocs
- Execute SQL statement to calculate the number of times a term occurs (term frequency) in document ii
- Update the term frequency table with the term (term), frequency (freq) and document ID (doc_id)
- Step7:** For ii going from 0 to numDocs
- Set sumFreq to the sum of term frequencies in document ii by executing SQL statement
- Set weight to $freq/sumFreq$ for each term and update weight field in term frequency table with its value
- Step8:** Call **shtf.jbInit()** and **shtf.createColumns()** methods to create a friendly user interface consists of a table to contain the term frequency table, and editable text area to contain a preview of the document containing the selected term in the term column in the table. The table has five columns, term ID, term, term frequency, term weight and document ID
- Step9:** Set GUI table cells to the data obtained from database by SQL statement that selects specific data from term frequency table
- Step10:** Add mouse listener in **shtf.jbInit()** method that if any term is clicked, the method **shtf.loadFileIntoText(docId, term)** will be called. This sets text

area to the document containing the term, with term is marked in different color

Step11: Shut down database connection

Figure 4.6: Pseudo code of tf() method.

4.4.3.2 IDF Classes

This technique has also two classes, computeIDF class computes the weights, and showIDF class shows the table of terms and their weights. Choose "Compute IDF" from "Statistical Techniques" submenu in "ATE" menu in the main menu to execute the former, and "IDF Result" to execute the latter. When computeIDF class is executed, it connects to ATE table containing the terms in the database through dbConnection class. Document frequencies and weights are computed using SQL statements. The result is stored in a table named 'idfresult' in the database; this table contains term ID, term, document frequency and term weight. By an SQL statement the range of weight 'needed for a term to be candidate index term' is specified. Then connection to database is ended, and the result appears in a GUI table by executing showIDF. Figure 4.7 illustrates the main methods and properties of IDF classes.

computeIDF class computes weights depending on IDF using **idf()** method. showIDF is used to design the GUI for the output. Figure 4.8 illustrates the pseudo code of IDF algorithm.

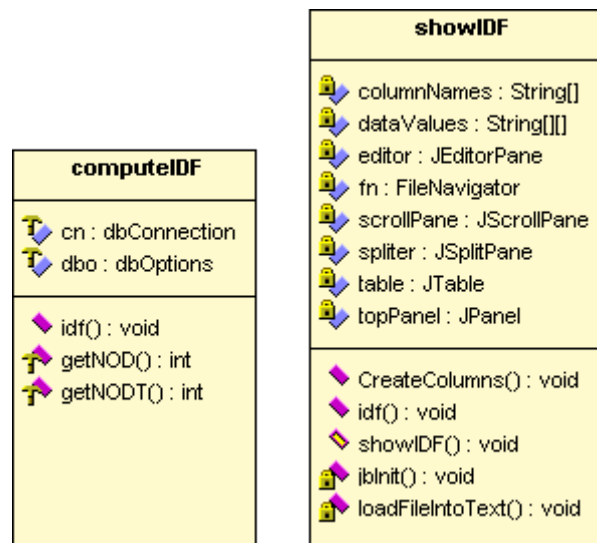


Figure 4.7: Class diagram of IDF classes.

Input:	
	A table in the database containing terms of the collection and their document IDs.
Output:	
	A table updated with each term and its document frequency, and term weight after computations.
Pseudo code of idf() method. Step1: Get instance of needed classes Step2: Establish connection with ATE database Step3: Delete the contents of IDF table in database if it contains old data Step4: Use SQL statement to get the number of documents (the same number of documents in the collection) from the terms table that created after parsing, using document ID field	

- Step5:** Set **tnod** to the obtained number of documents
- Step6:** Execute SQL statement to calculate the number documents containing a term (document frequency) from the terms table created after parsing
- Step7:** Update the inverse document frequency table with the term (**term**), document frequency (**doc_freq**)
- Step8:** Set weight to $1 - \log_2(\text{doc_freq}) / \log_2(\text{tnod})$ for each term and update weight field in inverse document frequency table with its value by executing SQL statements
- Step9:** Call **shidf.jbInit()** and **shidf.createColumns()** methods to create a friendly user interface consists of a table to contain the inverse document frequency table. The table has four columns, term ID, term, document frequency and term weight
- Step10:** Add mouse listener in **shidf.jbInit()** method that if any term is clicked, the method **shidf.loadFileIntoText(term)** will be called. This sets text area to the full paths of all documents containing the term
- Step11:** Set GUI table cells to the data obtained from database by SQL statement that selects specific data from inverse document frequency table

Figure 4.8: Pseudo code of idf() method.

4.4.3.3 TFxIDF Classes

This technique has two classes, computeTFxIDF class computes the weights, and showTFxIDF class shows the table of terms and their weights. Choose "Compute TFxIDF" from "Statistical Techniques" submenu in "ATE" menu in the main

menu to execute the former, and "TFxIDF Result" to execute the latter. When compute TFxIDF is executed it connects to ATE table containing the terms in the database through dbConnection class. Term frequencies, document frequencies, and IDF weights are computed using SQL statements in two separate tables, then TFxIDF weights are computed by joining the two tables and multiplying TF by IDF weight for each term. The result is stored in a table named 'tf_idf' in the database; this table consists of four fields; term ID, term, term weight and document ID. By an SQL statement the range of weight 'needed for a term to be candidate index term' is specified. Then connection to database is ended, and the result appears in a GUI table by executing showTFxIDF class. Figure 4.9 illustrates the main methods and properties of TF classes.

computeTFxIDF computes the weights depending on TFxIDF using **tfidf()** method. Figure 4.10 illustrates the TFxIDF algorithm using pseudo code.

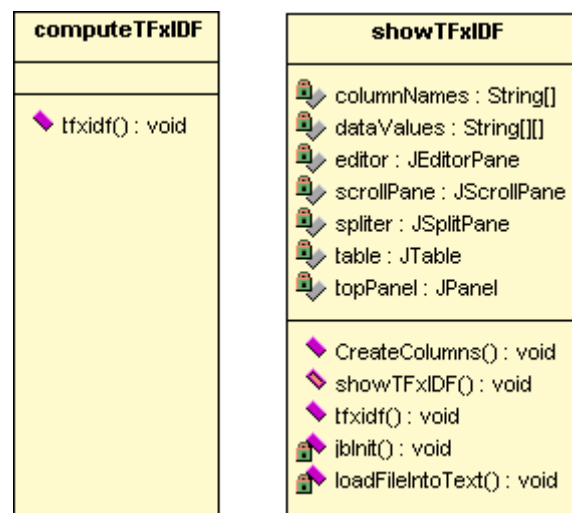


Figure 4.9: Class diagram of TFxIDF classes.

Input:	
	<p>A table containing each term and its frequency in each document, and term weight after computations (TF technique output)</p> <p>A table containing each term and its document frequency, and term weight after computations (IDF technique output)</p>
Output:	
	<p>A table updated with each term and its TFxIDF weight, and term weight after computations that multiply TF weight by IDF weight, in addition to document ID</p>
<p>Pseudo code of tfxidf() method.</p> <p>Step1: Get instance of each needed class</p> <p>Step2: Establish connection with ATE database</p> <p>Step3: Delete the contents of TFxIDF table in database if it contains old data using SQL statement</p> <p>Step4: Use SQL statement to get the number of documents (the same number of documents in the collection) from the terms table that created after parsing, using document ID field</p> <p>Step5: Set tnod to the obtained number of documents</p> <p>Step6: Execute SQL statement to calculate TFxIDF weight by joining the TF table and IDF table and multiplying TF weight by IDF weight for each term</p> <p>Step7: Update the TFxIDF table with the term (term), TFxIDF weight (weight) and document ID (doc_id)</p>	

- Step8:** Call **shtfidf.jbInit()** and **shtfidf.createColumns()** methods to create a friendly user interface consists of a table to contain the TFxIDF table, and editable text area to contain a preview of the document containing the selected term in the term column in the table. The table has four columns, term ID, term, term weight and document ID
- Step9:** Set GUI table cells to the data obtained from database by SQL statement that selects specific data from TFxIDF table
- Step10:** Add mouse listener in **shtfidf.jbInit()** method that if any term is clicked, the method **shtfidf.loadFileIntoText(docId, term)** will be called. This sets text area to the document containing the term, with term is marked in different color
- Step11:** Shut down database connection

Figure 4.10: Pseudo code of tfxidf() method.

4.4.3.4 TDVM Classes

Two classes are also used for discrimination value model TDVM, computeTDVM class to do computations, and showTDV class to get result. TDVM is developed as an improvement on the previous techniques. It uses the weight obtained by one of the previous techniques to check similarity between documents and find the terms that discriminate between documents when they are removed from the collection. The high discriminators have the highest weights. Computation class is executed when you choose "Compute TDVM" from "Statistical Techniques" submenu in "ATE" menu. If it is considered that TDVM uses weights obtained by term frequency technique, when computeTDVM class is executed it will connect

to ATE database and use term frequency table that contains terms and their weights in each document. Then computes average similarity between documents and subtracts it from average similarity after removing each term from the collection one at a time. The result is the discrimination value of each term. Figure 4.11 illustrates all methods and properties of TDVM classes. To show the result, choose "TDVM Result" from the same menu, which executes showTDV class and the GUI table appears.

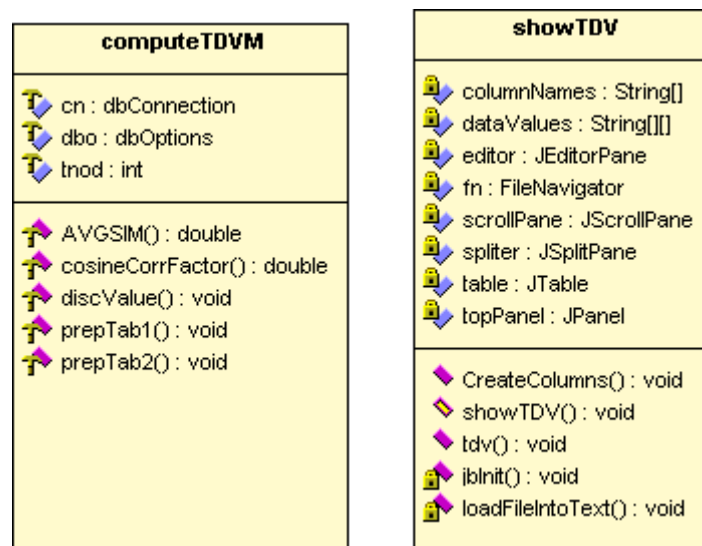


Figure 4.11: Class diagram of TDVM classes.

computeTDVM class performs heavy computations. It consists of six methods. showTDV class is used to build the user interface. TDVM technique is presented here with its most important methods using pseudo code. Figure 4.12 illustrates pseudo code of discValue() method, figure 4.13 illustrates pseudo code of AVGSIM() method and figure 4.14 illustrates pseudo code of cosineCorrFactor() method.

Input:	
	TF technique output table, term weight is extracted from this table.
	IDF technique output table, term is extracted from this table.
Output:	
	A table updated with each term and its discrimination value after heavy computations to find documents similarities

<p>Pseudo code of discValue() method.</p> <p>Step1: Establish connection with ATE database, to do so call the method: cn.dbConnection(user, password)</p> <p>Step2: Call the method AVGSIM(“”) and set AVGSIM to AVGSIM(“”) obtained, which represents the average similarity between documents before removing any term from the collection</p> <p>Step3: Delete the contents of TDVM table in database if it contains old data using SQL statement</p> <p>Step4: Use a cursor in SQL statement to get all collection terms with no repetition (distinct terms) from IDF table</p> <p>Step5: While (cursor points to a new term) DO</p> <p style="padding-left: 40px;">Set DISCVALUE_k to AVGSIM(term)-AVGSIM</p> <p style="padding-left: 40px;">Insert each term and its discrimination value into the output table prepared for that</p> <p>Step6: Any exception, trace the exception and report it</p> <p>Step7: Shutdown database, use cn.shutdown() method</p>
--

Figure 4.12: Pseudo code of discValue() method.

Pseudo code of AVGSIM(term) method.

Step1: Set **avgsim** to 0

Step2: IF **term** = ""

 Call the method **prepRes()** to prepare the input TF table

 For **i** going from 0 to **fn.getNod()**

 Call **prepTab1(i)** to initialize temporary table1

 For **j** going from 0 to **fn.getNod()**

 Call **prepTab2(j)** to initialize temporary table2

 Set **avgsim** to (**avgsim** + **cosineCorrFactor()**) to

 get sum of correlation factors between all documents

Step3: ELSE

 Set a flag to 1 in the TF input table for all terms equal to **term**

 Call the method **prepRes()** to prepare the input TF table

 For **i** going from 0 to **fn.getNod()**

 Call **prepTab1(i)** to initialize temporary table1

 For **j** going from 0 to **fn.getNod()**

 Call **prepTab2(j)** to initialize temporary table2

 Set **avgsim** to (**avgsim** + **cosineCorrFactor()**) to

 get sum of correlation factors between all documents

 Reset the flag in TF table

Step4: Return the average similarity (**avgsim**) between all documents with the term (**term**) exists or with the term removed from computations

Figure 4.13: Pseudo code of AVGSIM() method.

Pseudo code of cosineCorrFactor() method.**Step1:** Set **ccf** to 0**Step2:** Compute cosine correlation factor between every pair of documents using the following SQL statement:

```
SELECT SUM (r1.weight*r2.weight)/ SQRT (SUM (POW (r1.weight, 2))*SUM (POW (r2.weight, 2)))
from r1 join r2 using (term)
```

Step3: Return the Cosine Correlation factor (**ccf**) between pairs of documents

Figure 4.14: Pseudo code of cosineCorrFactor() method.

Note that some simple methods were excluded from pseudo code here. `showTDV` class is a simple class to build the GUI for the result exactly the same as for TF and IDF algorithms. Hence it was also excluded. For more details about them refer to the complete code available in Appendix D.

4.4.4 ATE Auxiliary Approaches

This subsection describes two related NLP approaches used in this work. Stop and Stemmer classes implement these algorithms.

4.4.4.1 Stop Words and Parsing Class

Stop class is executed during parsing to remove all stop words that do not affect the meaning of the text. Stop word lists are stored in the database. A session is established with database to compare any token parsed with words in the list. If it exists in the list it is removed from the text, if not it is not affected. It gets all files found in the specified collection path during parsing process, removes stop words and store all remaining terms in a 2-dimentional vector containing a vector for

each parsed file, each contains the terms of that file. Figure 4.15 shows all properties and methods of Stop class.

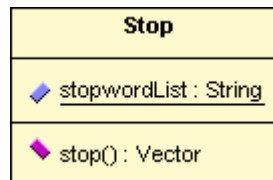


Figure 4.15: Stop class diagram.

In addition to parsing words in the collection documents, it removes stop words if any. Stop class stores all documents before processing into the database for further use as a cache. As illustrated in figure 4.15, Stop class consists of one method named stop(). Figure 4.16 illustrates Pseudo code of stop words and parsing algorithm.

Input:	
	A collection of documents with text, htm, or html format.
Output:	
	<p>A vector of vectors, each vector in the container vector contains the terms of a document in the collection after being parsed (stop words are removed or not depending on user options).</p> <p>A table containing each document in the collection and its ID, as it was before any processing, stored in the database.</p>

Pseudo code of stop() method

Step1: Declare private and public variables, three instances of **Vector** class **stop**, **textFiles**, **allFileTokens** and an instance of each **FileReader**, **BufferedReader** and **StringBuffer** as **x**, **z**, **fileBody** respectively to be used to read files and an instance **tt** of **StringTokenizer** for text tokens parsing

Step2: Call **cn.dbConnection(dbo.getUser, dbo.getPassword)** to establish connection to ATE database

Step3: Set **stop** to **stopwordList** from stop table in ATE database

Step4: Set **textFiles** to **fn.getCollection()** to get collection documents

Step5: For **I** going from 1 to **textFiles.size()**

Set **coll** to **textFiles.elementAt(I)**, to get file name number **I**

Set **x** to **FileReader(new File(fn.getPath(), coll))**, to read the file
coll exists at collection path

While (file is ready **z.ready()**) DO

Read a line from the file **z.readLine()**

Call **fileBody.append(s+" ")** to add line to the string **s**

Set **tt** to **StringTokenizer(s, " ")**, to parse words from **s**

While (still words not parsed **tt.hasMoreTokens()**) DO

Set **sng** to **tt.nextToken().toLowerCase()**, this
converts the parsed word to lowercase

IF (the word is not in stop list **!stop.contains(sng)**)

Call **fileTokens.addElement(sng)** to add

the word **sng** to **fileTokens** vector

Catch exceptions if occurred and report them

Store document ID (**I**), name (**coll**) and contents **fileBody** after converting to string in lowercase in ATE database

Call the method **z.close()** to close the document

Call **allFilesTokens.add(fileTokens)** method to add the vector of words (**fileTokens**) in the vector of documents (**allFilesTokens**)

Step6: Shut down database session using **cn.shutdown()** method

Step7: Return the vector of documents **allFilesTokens** that contains the vectors of words in each document **fileTokens**

Step8: Report Exceptions when occurred by the method **ed.setError("")**

Figure 4.16: Pseudo code of stop words and parsing algorithm.

4.4.4.2 Porter's Stemming Algorithm (Stemmer Class)

This class is very important because it assists to find all words of the same meaning even if there are some suffixes that make the word to seem different while it is of the same stem. Many words containing the stem of one word are presented by that word, which increases parsing performance. It uses Porter's algorithm, which was discussed in details in chapter three. Stemmer class gets the elements of the vectors produced after parsing in Stop class and strips suffixes from all terms. Then a connection is established to the database saving the

stemmed terms in a table named ATE. After execution of Stemmer class, the terms will be ready for applying any of the proposed statistical techniques. Figure 4.17 illustrates the main methods and properties of the class.

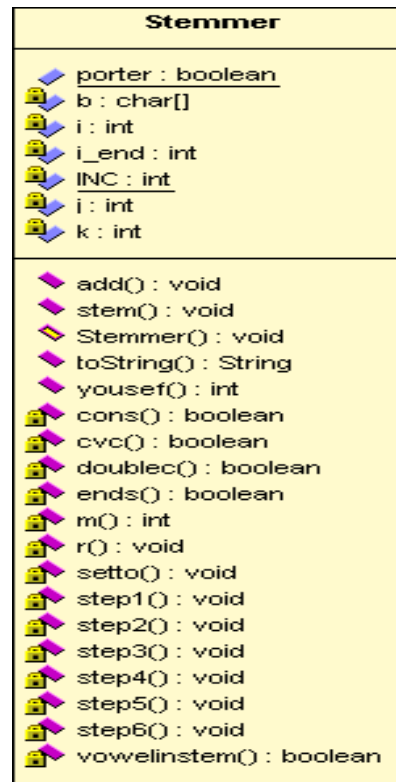


Figure 4.17: Stemmer class diagram.

Again, this algorithm is used to stem words by removing suffixes that indicate plurality and verb tense. Refer to table 3.1 in chapter three for steps of stemming. As illustrated in figure 4.17, Stemmer class consists of 20 methods; each performs a specific task in stemming process. Figure 4.18 illustrates the pseudo code of the main method **doStemming()**. The six steps of `stem()` method are described in table 3.1. It describes the function of each of `step1()`, `step2()`, `step3()`, `step4()`, `step5()` and `step6()` methods.

Input:	
	A vector of vectors, each vector in the container vector contains the terms of a document in the collection (part of Stop class output).
Output:	
	<p>A table containing the collection's terms (stemmed or not and repetition is allowed) and their document IDs.</p> <p>A table containing distinct terms. Both are stored in the database.</p>
<p>Pseudo code of doStemming() method in Porter's stemming algorithm</p> <p>Step1: Get instance from needed classes</p> <p>Step2: Set vec1 to stp.stop()</p> <p>Step3: Set x1 to 0</p> <p>Step4: Call cn.dbConnection(dbo.getUser(),dbo.getPassword()) method to connect to ATE database</p> <p>Step5: Delete the old data from terms table in ATE database</p> <p>Step6: For ii going from 0 to vec1.size()</p> <p style="padding-left: 40px;">Set the vector vec to vec1.elementAt(ii)</p> <p style="padding-left: 40px;">For r going from 1 to vec.size()-1</p> <p style="padding-left: 80px;">Set term to vec.elementAt(r-1)</p> <p style="padding-left: 80px;">IF (porter is set to true)</p> <p style="padding-left: 120px;">Clean the word from noisy punctuation marks</p> <p style="padding-left: 120px;">For g going from 0 to term.length, fill the array of stemming b by calling add(term.charAt(g))</p>	

```
Call stem() method which performs steps 1-6 in table 3.1

Set u to toString(), this method retrieves the stemmed word
as a string

Increment the ID of stemmed term x1 by 1

Execute SQL statement cn.execute(insert the term ID x1,
stemmed term u, document ID ii into the terms table)

Execute SQL statement cn.execute(Compute frequencies
and weights of terms) and update ATE database

ELSE

Clean the word from noisy symbols or punctuation marks

Set u to String.valueOf(term).trim(), by calling this
method the term is converted to clean string

Increment the ID of stemmed term x1 by 1

Execute SQL statement cn.execute(insert the term ID x1,
stemmed term u, document ID ii into the terms table)

Execute SQL statement cn.execute(Compute frequencies
and weights of terms) and update ATE database

Step7: Shutdown database session using cn.shutdown() method

Step8: Report Exceptions when occurred by the method ed.setError("")

Step9: Return the total number of stemmed documents vec1.size()
```

Figure 4.18: Pseudo code of doStemming() method in Porter's algorithm.

The other methods of Porter's algorithm and their main function are summarized in table 4.3. For the more details refer to the complete code in Appendix D.

Table 4.3: Summary of Porter's stemming methods

Method name	Main Function
Stemmer()	Class constructor to initialize private variables.
add(char ch)	Adds a word to an array, character by character after checking if it is a letter. This array will be the term to be stemmed.
toString()	After a word has been stemmed, it can be retrieved by toString() method.
cons(int i)	Checks if a letter in the word is consonant.
cvc(int i)	This method is used when trying to restore an <i>e</i> at the end of a short word, e.g. cav(e), lov(e), hop(e), crim(e). It returns true if i-2,i-1,i has the form consonant- vowel- consonant and also if the second c is not a glide like <i>w</i> , <i>x</i> or <i>y</i> .
doublec(int j)	It is used to check if the word contains double successive consonants. It returns true if j,(j-1) contain a double consonant.
ends(String s)	Checks if a word ends with a group of letters to be cut, e.g. sses, ful, fully or full.
m()	It measures the number of consonant sequences between 0 and j. The number of VCs, vowel sequence of letters (V) followed by a sequence of consonant letters(C). For instance the word (working) is distributed as <VCVC. It means that it contains two VCs or m() returns 2.
r(String s)	If m()>0, it sets (j+1),...k to the characters in the string s, readjusting k, using setto(s) method.
setto(String s)	It sets (j+1),...k to the characters in the string s, readjusting k.
vowelinstem()	The method vowelinstem() returns true if a part of a word from letters 0 to j contains a vowel.

4.4.5 Evaluation Class

This class measures the efficiency, accuracy and performance of ATEWB by computing Recall, Precision, Noise and Efficiency. In Chapter two, section 2.8 we described how these measures can be computed. Evaluation class is executed when "Evaluation" is selected from "Tools" menu in the main menu. Figure 4.19 illustrates all properties and methods of Evaluation class.

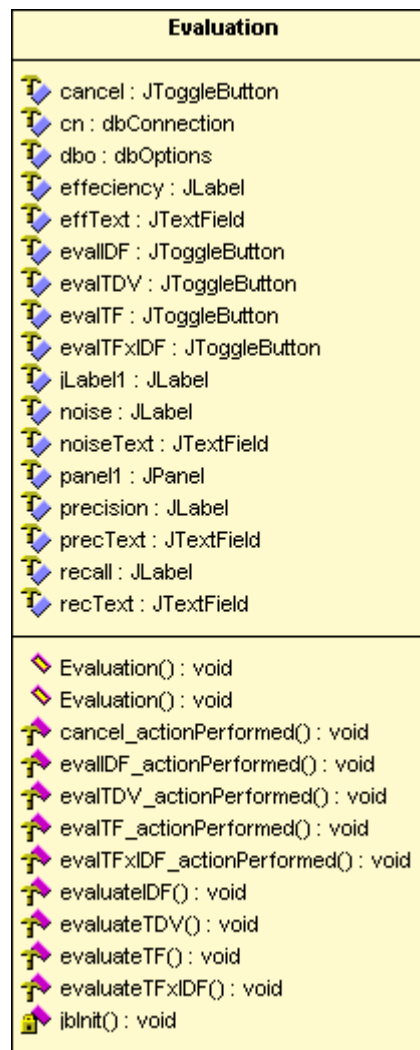


Figure 4.19: Evaluation class diagram.

The classes described above represent a case study which includes the four statistical techniques discussed in chapter three. In addition, it includes some important classes. Implementation details are presented in appendix D using java complete code.

4.5 Summary

In this chapter, we have described the main algorithms of our developed tool (ATEWB). It describes the main function of each class and the main methods using UML notation. On the other hand, the algorithms are described using pseudo code.

The next chapter (chapter 5) aims at measuring the effectiveness and performance of ATEWB. This evaluation is used to compare the used statistical techniques, which is the main theme of this thesis.

5 A Comparative study

This chapter is a comparative study, which represents the main theme of this thesis. It includes a comparison between the four statistical techniques discussed in chapter 3.

The comparison of ATE approaches may be conducted using mathematical models and mathematical analysis such as the vector space model proposed by Salton or experimentally by running some experiments and statistical analysis of results using some evaluation measures to measure the effectiveness of each technique.

The mathematical models of statistical techniques are already established, more over, they are theoretically proved. Our contribution in this thesis is a comparative study that compares between these mathematical models and we don't go through modifying these models or proving them mathematically.

Most of recent studies use the well known evaluation measures (recall and precision, see section 2.8) to evaluate ATE and IR systems. Even the tests of IR systems in the international Text Retrieval Conference (TREC) use these measures. These measures proved that they are sufficient in the evaluation process of ATE and IR systems. This is why we use our developed tool for term extraction, experiments, and evaluation of statistical techniques using these measures. Our task is to find which technique to use and when or which conditions give us the best results in each technique by comparing the efficiency and effectiveness of each approach and the factors affecting them.

5.1 Comparison methodology

5.1.1 Comparison Criteria

The criteria used here concentrate on performance and accuracy of each approach. On the other hand, the effect of different conditions or factors ‘related to input data and processing algorithms’ on the output is discussed. These conditions and their effect are listed in table 5.1.

5.1.2 Main Factors Affecting Performance and Accuracy

Table 5.1 illustrates the main factors affecting the performance and accuracy of our techniques and the expected effect depending on some statistical hypotheses, these factors are explained below:

1. **Collection size:** it means the number of documents, if number of documents increases, it means that number of words increases, hence increase in computations time in all techniques, in summary:
 - Computations time is linearly proportional to collection size for all proposed techniques except for TDVM in which computations time is dramatically affected by collection size. For N documents, correlation factor is to be computed for $N(N+1)/2$ times. See figure 5.1.
 - Accuracy of TF is not affected by collection size, because weight is computed within each document separately and not affected by other documents.
 - In IDF, when the number of documents increases, the opportunity for a word to occur in more documents increases, hence, increased accuracy

and visa versa. For example if collection size=5, the opportunity for a word to occur in more than one document is low. The result is; most of terms have document frequency of 1 and weight of 1, hence all words are important and can be index terms, which is not applicable. Even though it depends on the relation between documents in the collection.

Table 5.1: Summary of factors affecting extraction results and expected effect

Factor	Effect on ATE Statistical Technique			
	TF	IDF	TFxIDF	TDVM
Collection Size	Heavy computations to search for similar terms, accuracy not affected.	More time for computations, Accuracy depends on the collection.	Depends on TF and IDF. More computations and accuracy depends on the collection.	Very huge computations, nonlinear increase in computations time.
Document Length	Affects accuracy and time.	Affects accuracy and time.	Affects accuracy and time.	Affects accuracy and time.
Combination of different Domains	Not affected.	Decreases performance and accuracy.	Decreases performance and accuracy.	Decreases performance and affects accuracy
Stop word removal	Increases parsing time a little bit but decreases time of computations well. Accuracy increases	Increases parsing time a little bit but decreases time of computations well. Accuracy increases	Increases parsing time a little bit but decreases time of computations well. Accuracy increases	Increases parsing time a little bit but decreases time of computations well.
Stemming	Increases parsing time but decreases computations time	Increases parsing time but decreases computations time	Increases parsing time but decreases computations time	Increases parsing time but decreases computations time
Updating data	When new terms are added or document modified	When documents are added or collection is modified	When documents or collection are modified	It is time consuming, so cannot be updated frequently
Cost and Complexity (coding, hardware, time)	Low	Low	Medium	High

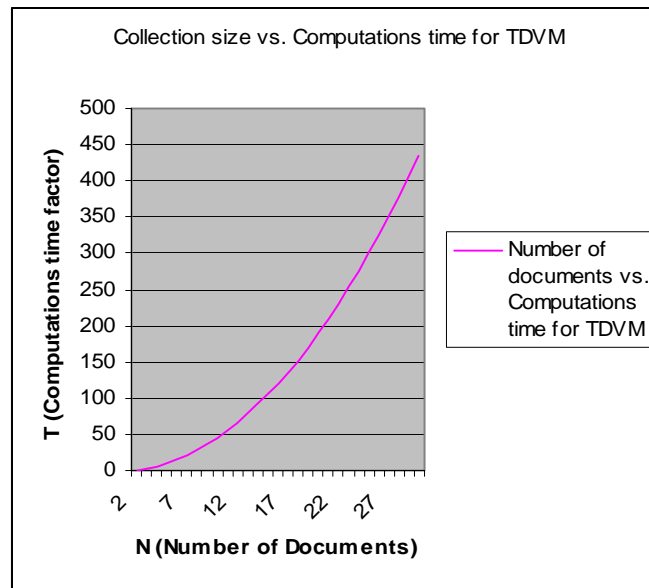


Figure 5.1: Collection size vs. computations time for TDVM.

- $TF \times IDF$ is affected by both TF and IDF, so accuracy is proportional to collection size.
 - If the weights used in TDVM computations are TF weights, accuracy is not affected by collection size.
2. **Document length:** it is the number of words within documents in the collection.
- Computations time is proportional to document length for all proposed techniques.
 - According to accuracy, when document length is low, the opportunity to words to occur more than one time within documents or within the whole collection is low, hence term frequency or document frequency will be the same for most of words if not all. This means that all words

have the same weight and can not be judged which to be index term.

Hence, accuracy is proportional to document length.

3. **Domain:** it is the subject of documents in the collection. Documents in the collection may be in one or more domains. When the collection contains documents in different domains:

- Computations time is increased for all techniques except for TF because the weight is computed within each document in TF, while in the other techniques, the number of distinct words for which weight should be computed may be increased. Hence computations time is increased.
- The same for accuracy, all techniques are affected except TF. Because the opportunity for a word to occur in more documents is lower. Hence, lower accuracy.

4. The effect of **stop-words** and **stemmer** are discussed in details in subsection 5.2.1 in experiment one.

5. **Data update:** when documents are modified as well as the collection, weights should be recomputed and modified in the engine. It is not applicable to repeat computations very frequently, especially for techniques that need huge computations like TDVM.

- The computations time may be increased or decreased depending on modification type. A flag can be used that set to 1 for those modified collections or documents. Accordingly, computations are repeated only for those modified documents. Hence, improving performance and reducing computations time.

- If weights are not updated for a long time, many things may be modified while index terms do not describe the documents, hence, low accuracy.
6. **Cost and Complexity:** it includes hardware, software and coding. The simplest are TF, IDF and TFxIDF, computing weights is also simple, just TFxIDF weighting needs an additional vector multiplication but reduced one division operation for each document, and stay the same. Even though, they do need high specifications for hardware and software when talking about very large collections in different domains, which is true in real world applications such as search engines. The most complex is TDV, which needs huge computations that can not be performed unless on very high specifications of hardware and software.
- Computations time may be very long for all techniques if the hardware and software specifications are low, and if the program wasn't well coded by professionals.
 - Accuracy also may be low if specifications are low, because techniques that need higher specifications may be excluded even if they improve accuracy.

5.2 Experiments, Results and Discussion

In this section, three experiments have been run. At each experiment, performance and accuracy have been checked for each of the four statistical techniques, and a comparison has taken place by computing Recall, Precision and Noise for each as mentioned in section 2.8. For each technique, the computations may be done in

different situations; when stemmer and stop word list applied, stemmer only, stop list only applied and finally, when neither stemmer nor is stop word list applied. Discussion of these results takes place in this section.

5.2.1 Experiment One

5.2.1.1 Objectives

This experiment is to measure the effect of stemming and stop-words removal on the performance of statistical techniques.

5.2.1.2 Setup

In this experiment, the input is a small collection consists of **(16)** documents. The domain of the collection is ‘Abstracts of some published papers in ATE and IR’. The total number of parsed words of all documents in the collection is 2321 words. 38 Key words that may be reduced to 36, 34, or 32 depending on the condition, prepared manually or by the authors of published papers. Table 5.2 illustrates collection description and some statistics of the input.

5.2.1.3 Procedure

For each statistical technique, 55 index terms are extracted (selected by the system) each time, and compared with ($T_{Rel} = A+C = 38$) relevant keywords prepared manually or by the authors of published papers (refer to appendix A). This number (38) may be reduced down to 32 depending on the conditions applied like stemming and stop word list, because these conditions remove some words by unifying some different words of the same stem, or considering them as

stop words. In addition, it is necessary to apply the conditions in both term extraction and query.

Table 5.2: Experiment 1 collection description and statistics

Condition Applied	Technique	Number of terms		A	B	A+C
		Before computations	After computations			
No conditions applied	TF	2321	1402	15	40	38
	IDF		662	8	47	
	TFxIDF		1402	14	41	
	TDVM		662	17	38	
Stop words list	TF	1266	930	18	37	36
	IDF		544	12	43	
	TFxIDF		930	19	36	
	TDVM		544	20	35	
Porter's Stemming	TF	2251	1290	21	34	34
	IDF		540	10	45	
	TFxIDF		1290	17	38	
	TDVM		540	20	35	
Both Stop list and Porter's Stemming	TF	1263	852	24	31	32
	IDF		442	15	40	
	TFxIDF		852	21	34	
	TDVM		442	25	30	

For each technique, four conditions were applied: in the first no condition applied, in the second stop word removal, in the third stemming algorithm is applied, and lastly both stop word removal and stemming are applied. At each time recall,

precision and noise evaluation measures are recorded using the developed Evaluation class. Then the comparison of these measures takes place using graphical charts. Refer to tables (5.2, 5.3) and figures (5.2-5.4). Finally, a discussion takes place upon the obtained results.

Table 5.3: Recall, Precision and Noise of Experiment 1 with different conditions.

Condition Applied	Technique	Recall	Precision	Noise
No conditions applied	TF	0.395	0.273	0.727
	IDF	0.21	0.145	0.855
	TFxIDF	0.368	0.255	0.745
	TDVM	0.447	0.309	0.691
Stop words list	TF	0.5	0.327	0.673
	IDF	0.333	0.218	0.782
	TFxIDF	0.528	0.345	0.655
	TDVM	0.555	0.364	0.636
Porter's Stemming	TF	0.618	0.382	0.618
	IDF	0.294	0.182	0.818
	TFxIDF	0.5	0.309	0.691
	TDVM	0.588	0.364	0.636
Both Stop list and Porter's Stemming	TF	0.75	0.436	0.564
	IDF	0.468	0.273	0.727
	TFxIDF	0.656	0.382	0.618
	TDVM	0.781	0.455	0.545

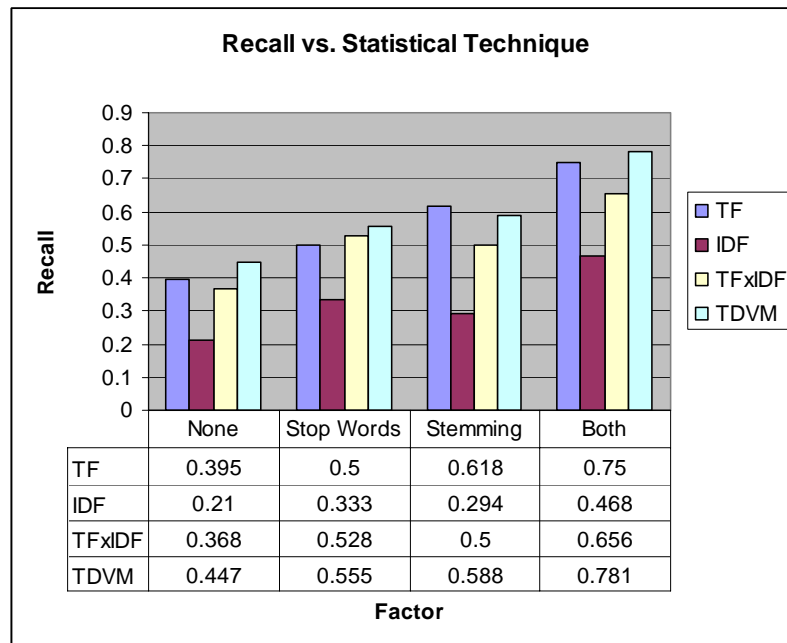


Figure 5.2: Recall vs. Statistical Technique/ Factor

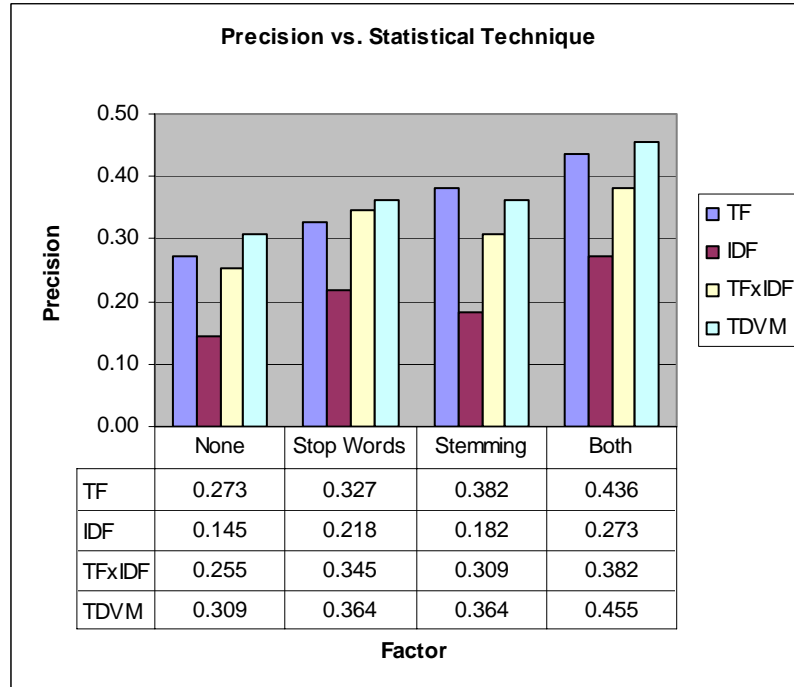


Figure 5.3: Precision vs. Statistical Technique/ Factor

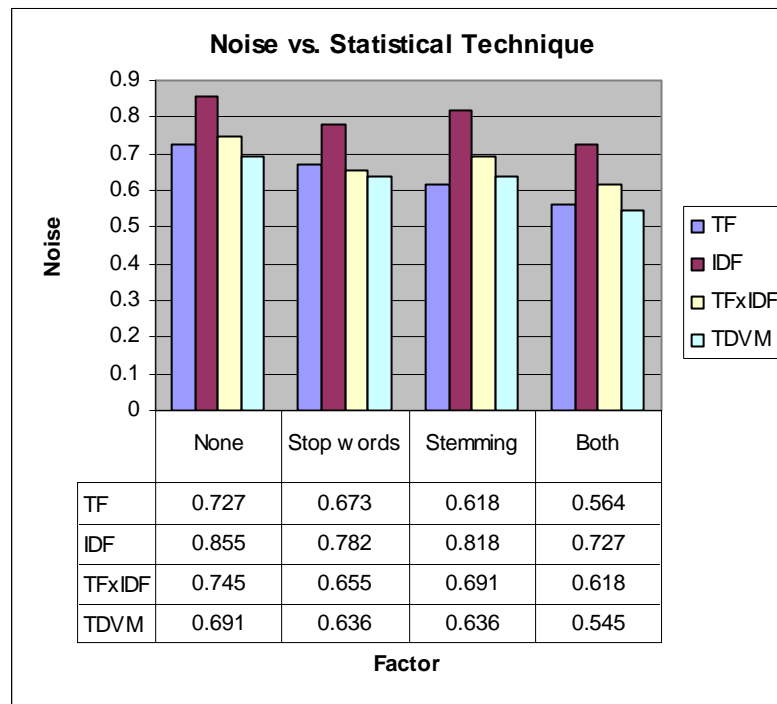


Figure 5.4: Noise vs. Statistical Technique/ Factor

5.2.1.4 Results

In this experiment we measure Recall, Precision and Noise of different techniques with different conditions applied, see section 2.8. Refer to table 5.3 and figures (5.2-5.4) for more details.

5.2.1.5 Discussion and Conclusions

After reviewing results illustrated in table 5.3 and figures (5.2-5.4) the following conclusions are deduced:

1. Recall is improved by stop-words removal as well as by stemming. The effect of stemming is more towards improvement in TF and TDVM, but in IDF and TFxIDF stop-list affects the result more towards improvement. When both stemming and stop-list are applied we get the best recall for all techniques, *see*

figure 5.2. By removing unwanted stop words, computation time is reduced by 45% (1055 words removed from computations out of 2321 words); while stemmer reduces computation time by 3% (71 words out of 2321 words are removed from computations). Refer to table 5.2.

2. Unless with stemming, TDVM technique has the best Recall, *see* figure 5.2. TF and TDVM are competing, their Recall is approximately the same in all situations and TF is the best with stemming, taking into consideration that TDVM computation time is multiple times than TF.
3. TFxIDF technique comes the third in Recall except with stop-words it is the second; it always competes with TF and TDVM, refer to figure 5.2.
4. The worst recall or it may be not recommended to use IDF alone as a weighting technique, *see* figure 5.2. IDF is needed to improve the performance by combination with other techniques depending on documents rather than on the collection such as TF.
5. Precision has nearly the same behavior like Recall, with better Precision for TDVM except with stemmer for TF, refer to figure 5.3.
6. TFxIDF is the third except with stop-words it is the second and IDF has the worst Precision in all cases, refer to figure 5.3.
7. TF has better precision with stemmer. IDF and TFxIDF have better precision with stop-list. TDVM has the same precision with stemmer or with stop-list, refer to table 5.3 and figure 5.3.
8. Noise is the complement of Precision, which means that ($N=1-P$). Refer to

figure 5.4. **Proof:** $P + N = \frac{A}{A+B} + \frac{B}{A+B} = \frac{A+B}{A+B} = 1$, hence $N=1-P$.

9. The best value of R and P was for TDVM with both stemmer and stop-word removal applied with 0.781 and 0.455 respectively, but the highest computations time. Refer to table 5.3.

5.2.2 Experiment Two

5.2.2.1 Objectives

The main objectives of this experiment are:

- To check the effect of collection size (number of documents) on the result. It will be compared with experiment3, which uses the same collection but with higher number of documents. This comparison takes place in subsection 5.2.3.5.
- To get results of TDVM for small collection size, because it cannot be done on large collections unless with supercomputer which is not available in Palestine.
- It also aims at comparing ATEWB on different Operating Systems.

5.2.2.2 Setup

The collection used in this experiment is obtained as a test collection from IR resources. NPL collection (also known as the VASWANI) is a collection of around 10,000 documents. It is available for research purposes in the IR group, University of Glasgow [59].

This experiment is done on the first **(100)** documents to extract index terms and measure ATEWB system's performance. The domain of the collection is 'The use

of digital computer in scientific fields'. The suggested keywords list is also available. It consists of 52 terms. Refer to Appendix A.

Table 5.4 describes the collection used in this experiment using each technique, after computations are done. The number of words of the whole 100 documents is 3546 words. After stemming and removing stop-words, the number is reduced to 1906, refer to table 5.4.

Table 5.4: Experiment 2 collection description.

Technique	Conditions	Statistics	
		# words after condition	# words after computations
TF	Stemming and Stop words removal	1906	1607
IDF		1906	814
TFxIDF		1906	1607
TDVM		1906	814 319 computed & 495 by default = 0

5.2.2.3 Procedure

This experiment applies both stemming and stop-word removal approaches. After extracting the terms, three measures have been computed; Precision, Recall and Noise. This experiment is divided into three sub-experiments with total number of relevant terms $(A+C) = 52$ (refer to appendix A):

Exp 2.1 In this case, total number of retrieved terms $(A+B) = 65$.

Exp 2.2 In this case, total number of retrieved terms $(A+B) = 97$.

Exp 2.3 In this case, total number of retrieved terms $(A+B) = 132$.

Then, average recall, precision and noise are computed and a comparison has taken place with different values of retrieved terms ($A+B$). The four statistical techniques are used.

The procedure in which TDVM results are obtained is an interesting issue. By this procedure we have improved the performance dramatically as follows:

1. All words whose document frequency equals to 1 are excluded from computations, because correlation factor is measured between two documents, each containing at least 1 similar word as in the other. For these words (more than half the words of the collection), DV equals to zero without any computations.
2. The same words are also excluded from TF table containing the weights used with TDV computations (also 495 words out of 1607 excluded).
3. MYSQL database was installed on Linux Enterprise 3.0 instead of Windows 2000.
4. Two other instances of ATEWB were installed on other 2 PCs. The three computers start computations, each for a specified range of words, and all updating the same TDV table in the database with terms and their DV.

5.2.2.4 Results

Table 5.5 describes the computations of R, P and N, while table 5.6 illustrates average R, P, and N that measure ATEWB performance.

Table 5.5: Recall, Precision and Noise of three parts of Experiment2.

Experiment	Technique	A	B	C	Recall	Precision	Noise
Exp 2.1	TF	21	44	31	0.404	0.323	0.677
	IDF	6	59	46	0.115	0.092	0.908
	TFxIDF	10	55	42	0.192	0.154	0.846
	TDVM	23	42	29	0.442	0.354	0.646
Exp 2.2	TF	25	72	27	0.481	0.258	0.742
	IDF	6	91	46	0.115	0.062	0.938
	TFxIDF	11	86	41	0.212	0.113	0.887
	TDVM	25	72	27	0.481	0.258	0.742
Exp 2.3	TF	27	105	25	0.519	0.205	0.795
	IDF	6	126	46	0.115	0.045	0.955
	TFxIDF	21	111	31	0.404	0.159	0.841
	TDVM	25	107	27	0.481	0.189	0.811

Table 5.6: Average Recall, Precision and Noise of Experiment 2

Technique	Avg. R	Avg. P	Avg. N
TF	0.468	0.262	0.738
IDF	0.115	0.066333	0.933667
TFxIDF	0.269333	0.142	0.858
TDVM	0.468	0.267	0.733

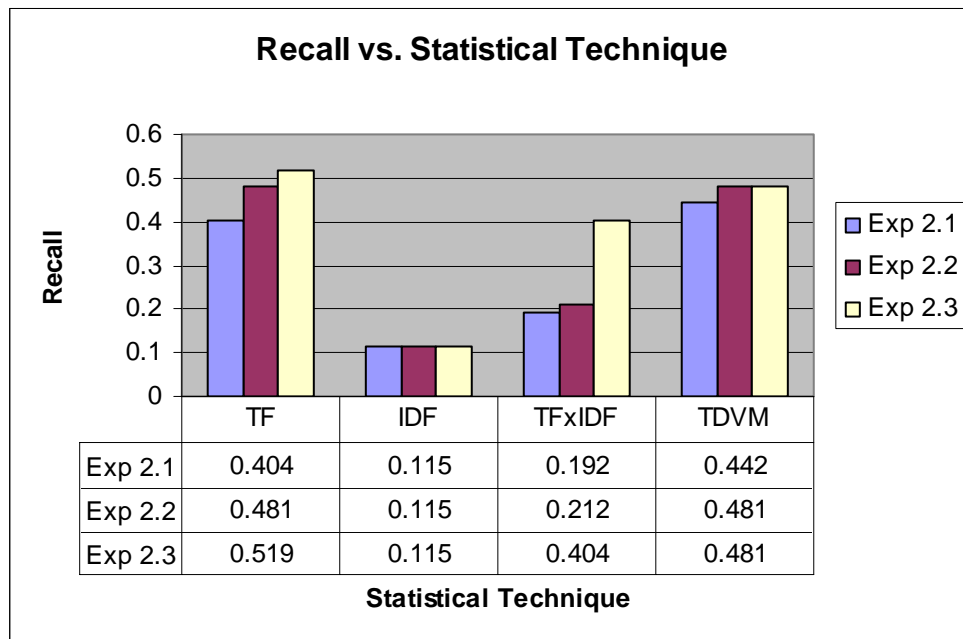


Figure 5.5: Effect of increasing number of retrieved terms on Recall.

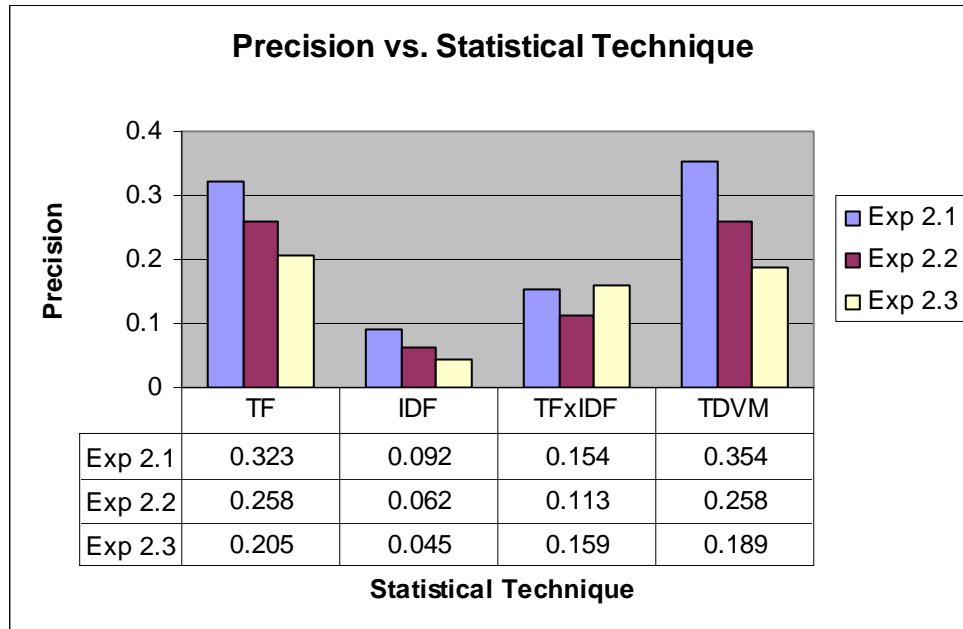


Figure 5.6: Effect of increasing number of retrieved terms on Precision.

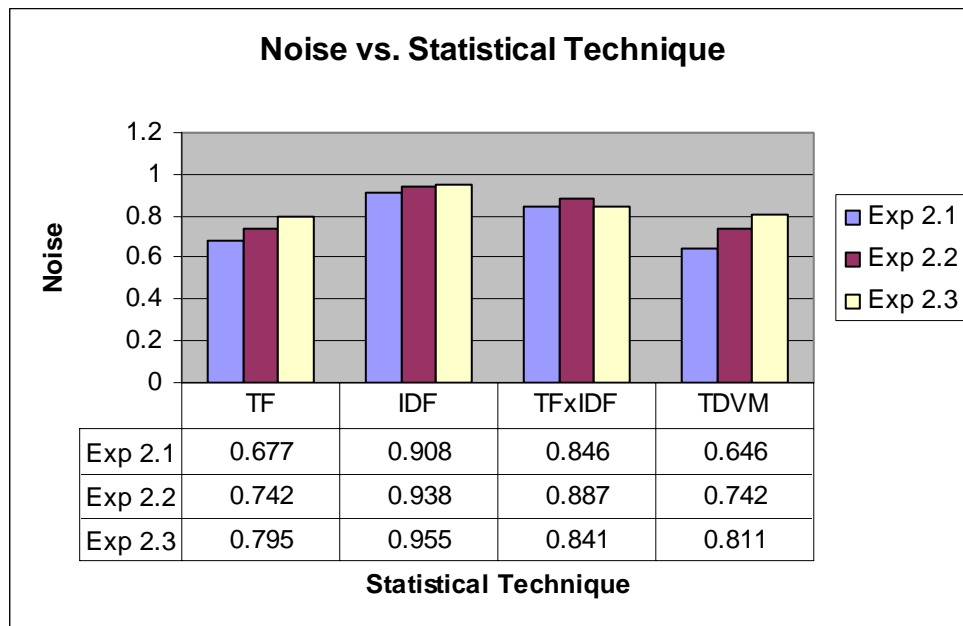


Figure 5.7: Effect of increasing number of retrieved terms on Noise.

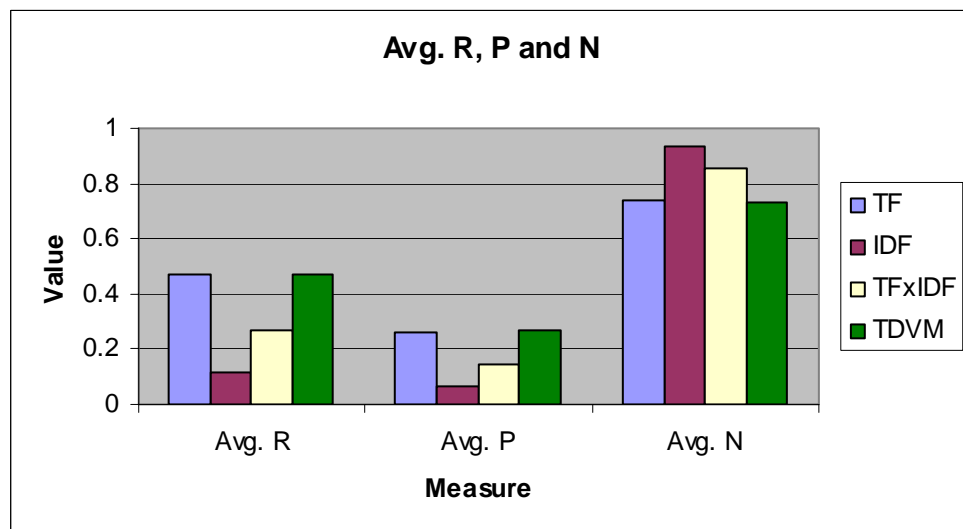


Figure 5.8: Average Recall, Precision and Noise

Figures (5.5- 5.7) illustrate the results shown in table 5.5, which measures the effect of increasing retrieved terms (index terms selected by the system) on system performance. Figure 5.8 illustrates average R, P and N using charts.

5.2.2.5 Discussion and Conclusions

The results above will be discussed again in experiment 3. There will be a comparison between the results of experiments 2 and 3 to measure the effect of collection size.

1. After optimization in TDVM as described in the procedure, the performance is improved as follows:
 - Step 1 improves the performance in this experiment by 60.8% (495 words out of 814 words are excluded from computations). Refer to table 5.4.
 - Step 2 improves the obtained performance again by about 10% (nonlinear). For accumulative performance improvement of 64.72%.
 - Step 3 improves the performance by other 28.56% (computations time needed to extract each term is reduced from 32.14 seconds to 22.96 seconds to compute DV for one term).
 - Step 4 again improves the performance by other 66%. TDV table is obtained by around 1hr instead of 3hrs. Accumulative improvement is about 91%. Or it can be said that computation time is reduced to 9% of original time, (all terms extracted in about 1hr instead of 11 hrs).
2. By reviewing table 5.5 and figures (5.5-5.7), it is obvious that as the total number of retrieved terms (A+B) is proportional with recall and reversely proportional with precision in all techniques.

3. A compromise should be taken between P and R to get the best effectiveness, the relationship between recall and precision looks like figure 5.11.
4. From table 5.6 and figure 5.8, we get the best results with average recall precision pair 0.468 and 0.267 when TDVM is used.

5.2.3 Experiment Three

5.2.3.1 Objectives

The main objective of this experiment is to decide which technique is the best in terms of efficiency and effectiveness. It also aims at measuring the effect of the variance of Total Retrieved Terms ($T_{Ret} = A + B$) on Recall and Precision. In addition, it measures the effect of increasing number of document on the performance in comparison with results obtained in experiment 2.

5.2.3.2 Setup

This experiment is done on the same NPL collection with **(1000)** documents; it needs more and more time to be computed. Number of words before stemming and removing stop-list is (34,000), after this operation they become (19,178) words. Table 5.7 represents computation time to complete the first three techniques (TF, IDF and TFxIDF) with some statistics. It also represents estimation for computations time of the fourth technique (TDVM), because it needs very long time, and we couldn't get results on P4 PC. The total number of relevant keywords used (A+C) is 57 (refer to appendix A), and the total number of

retrieved terms (A+B) varies from 60 in exp3.1, 80 in exp3.2, 100 in exp3.3, to 120 in exp3.4.

5.2.3.3 Procedure

For each value of (A+B), we compute R, P, and N and plot them together in one graph for each technique alone. We then draw the R-P relation as shown in figure 5.11. On the other hand, we compute the average R, P, and N for each technique. The results of experiment 2 are then compared with the results of this experiment to measure the effect of collection size on ATEWB performance. After results are obtained, these results are discussed and conclusions are summarized.

5.2.3.4 Results

As shown in table 5.7, the highest computations time is with TDVM then with TF. TDVM computation time can be estimated in a simple way. Suppose that the time needed to compute cosine correlation factor once equals to the time needed to compute it in experiment2, which is about 0.002235 seconds after optimization described in the discussion in sub-section 5.2.2.

Total time consumed can be written as: $T = K.T_0.\sum_{i=1}^{N-1} i$ where K is number of terms to be extracted, N is number of documents and T_0 is time estimated to compute correlation factor once for two documents.

$$T = 1499(0.002235)\sum_{i=1}^{999} i = 1,617,425.6 \text{ seconds} \approx 18.7 \text{ days.}$$

TDVM was computed with 200 documents, it consumed 24 hrs to extract 6 terms and still waiting the next 450 terms. This leads to the conclusion that it can not be computed unless a supercomputer is available.

Table 5.7: Computation time estimation and term statistics of Experiment3.

Technique	Extracted terms	Computation time (s)	Notes
Parsing	19,178	11 seconds	1743 word/second
TF	16,063	190.32 seconds	84 term/second
IDF	2608	16.61 seconds	157 term/second
TFxIDF	16,063	17.40 seconds	923 term/second
TDVM	1449 out of 2608	1,617,425.6 seconds or about 18.7 days	77 term/day

Table 5.8: Experiment3 computations of R, P and N

Experiment	Technique	A	B	C	Recall	Precision	Noise
Exp 3.1	<i>TF</i>	18	42	39	0.316	0.300	0.700
	<i>IDF</i>	4	56	53	0.070	0.067	0.933
	<i>TFxIDF</i>	38	22	19	0.667	0.633	0.367
Exp 3.2	<i>TF</i>	24	56	33	0.421	0.300	0.700
	<i>IDF</i>	4	76	53	0.070	0.050	0.950
	<i>TFxIDF</i>	46	34	11	0.807	0.575	0.425
Exp 3.3	<i>TF</i>	27	73	30	0.474	0.270	0.730
	<i>IDF</i>	4	96	53	0.070	0.040	0.960
	<i>TFxIDF</i>	46	54	11	0.807	0.460	0.540
Exp 3.4	<i>TF</i>	30	90	27	0.526	0.250	0.750
	<i>IDF</i>	4	116	53	0.070	0.033	0.967
	<i>TFxIDF</i>	46	74	11	0.807	0.383	0.617

Table 5.9: Average Recall, Precision and Noise of Experiment 3

Technique	Avg. R	Avg. P	Avg. N
TF	0.43425	0.28	0.72
IDF	0.07	0.0475	0.9525
TFxIDF	0.772	0.51275	0.48725

Table 5.8 illustrates four parts of experiment3 excluding TDVM. Figures 5.9 and 5.10 illustrate the variation of R, P and N with respect to variation in the number of total retrieved terms (A+B) selected by ATEWB system from 60 to 120 for both TF and TFxIDF techniques. Because of bad performance IDF is excluded.

Figure 5.11 illustrates the R-P relationship for TFxIDF technique, which is similar in other statistical techniques. Average recall, precision and noise are illustrated in figure 5.12.

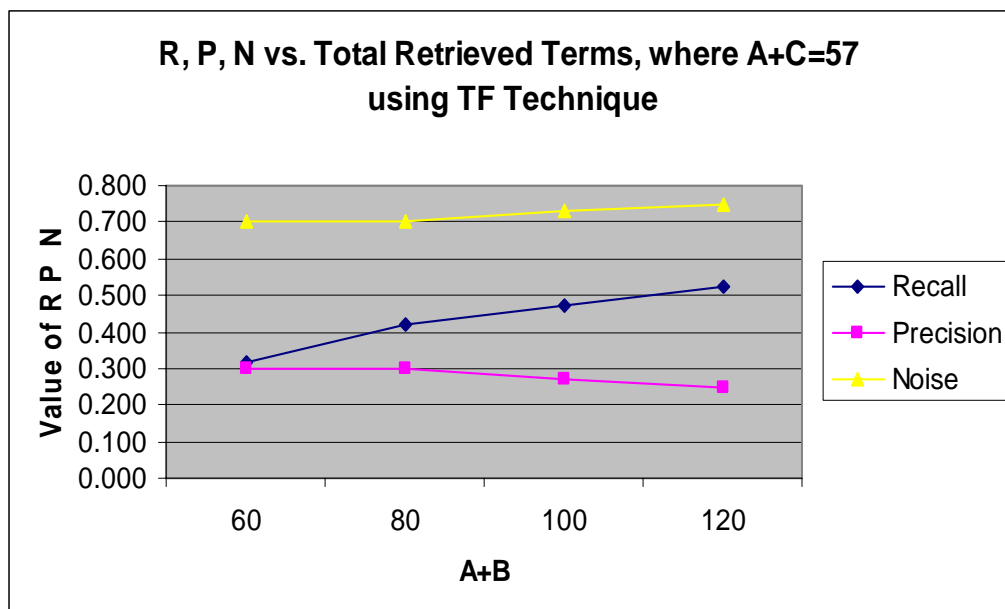


Figure 5.9: Relation between R, P and N for TF.

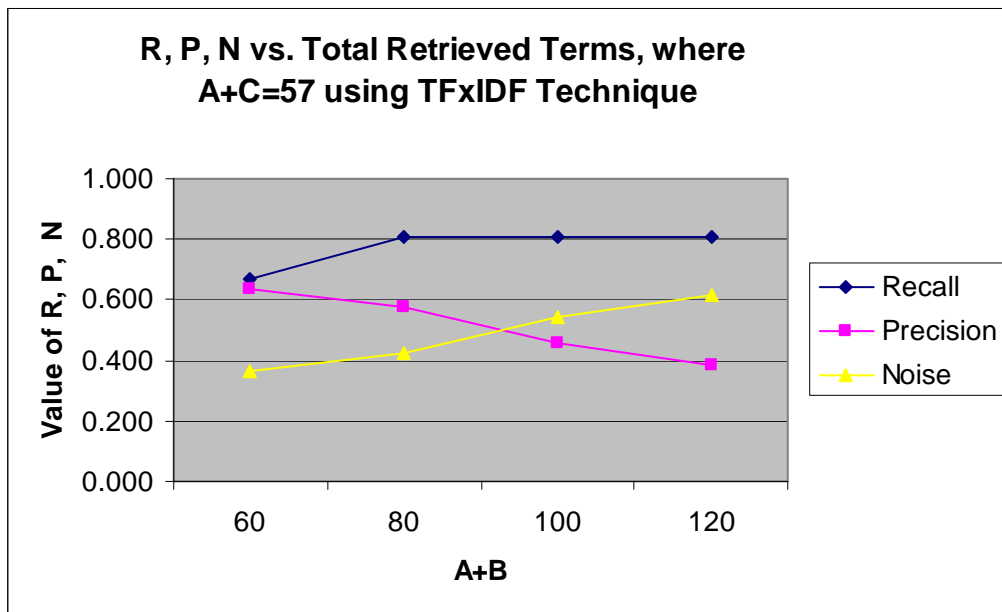


Figure 5.10: Relation between R, P and N for TFxIDF.

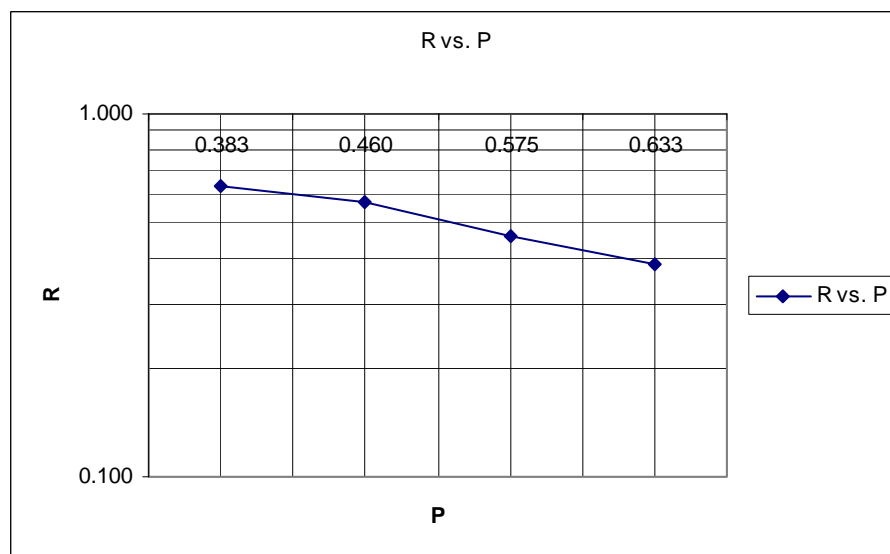


Figure 5.11: R-P relationship for TFxIDF.

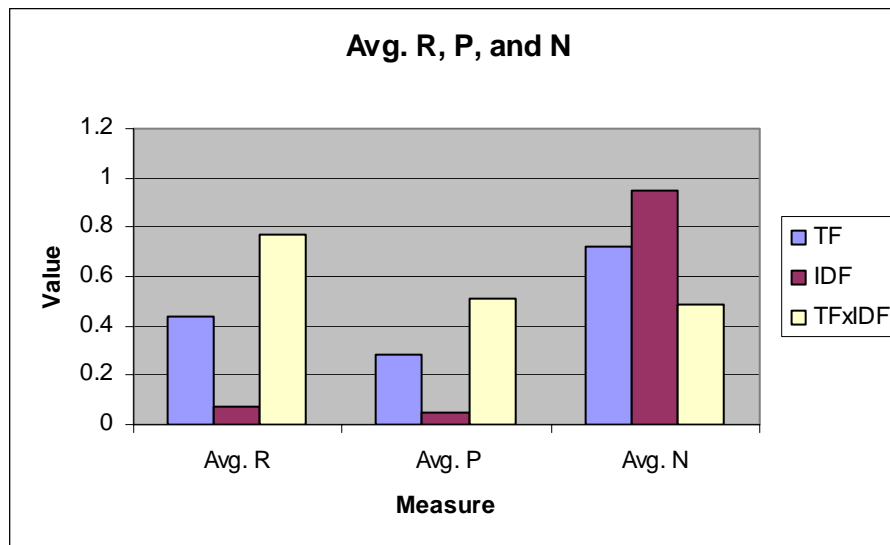


Figure 5.12: Average Recall, Precision and Noise

5.2.3.5 Discussion and Conclusions

1. In this experiment stemmer together with stop-list removal improved the performance by 43.6% (The number of parsed words to be used in computations is reduced from 34,000 to 19,178). Refer to table 5.7.
2. TFXIDF technique gives the best results in this experiment (computation time, Recall, Precision and Noise). Table 5.7 shows the time estimation of computations for each technique on P4 2.4GHz PC with 512MB RAM. The best is TFXIDF with 923 terms/second extraction rate. It has also the highest average (R, P) values of (0.772, 0.51) and the lowest average N value of 0.49, refer to table 5.9.
3. As the total relevant terms (A+C) is constant, if (A) is increased (C) will decrease, hence increasing R. Figures (5.9- 5.11) prove this result.
4. If total retrieved terms (A+B) is increased, it means that the probability of retrieving more relevant terms (A) would be increased. If (A) is kept

constant, (B) will increase, hence reduced P and increased N. Refer to figures (5.9- 5.11).

If results here are compared with results of experiment2:

1. TF in experiment2 is nearly the same as in experiment3, which means that TF is not affected by collection size.
2. Also, when the number of documents was increased, IDF results were badly reduced. This is normal because when the number of documents is increased, the probability that a term appears only in one document increases; this means that the number of terms having the weight (1) will be very large, not because of importance but because of large number of documents, accuracy depends on the collection characteristics. This result again emphasizes that IDF can not be used alone for weighting, refer to tables (5.5, 5.6, 5.8, and 5.9).
3. It is clear that R, P, and N of TFxIDF are improved in experiment3. This means that when the collection size was increased, the result of TFxIDF was improved. But this is not always true, because TFxIDF depends on TF and IDF. Here TF is not affected and IDF is worse than in experiment2, so the results of TFxIDF have to behave like them with some improvement. But why we have got better results for TFxIDF in experiment3? It may be because we got 1000 documents randomly from the NPL collection, and it was very hard to specify the relevant keywords used in computing R, P, and N measures.

4. The highest average (R, P) was for TDVM and TF in experiment2. If compared with highest average (R, P) values in experiment3 which was for TFxIDF, (0.468, 0.267) in experiment2 for TDVM and TF vs. (0.772, 0.513) in experiment3 for TFxIDF, refer to tables (5.6, 5.9) and figures (5.8, 5.12).

5.3 Comparison with Previous Studies

Most of previous studies concentrate on evaluation of IR systems as complete IR systems. They measure recall and precision depending on relevance of retrieved documents like that used in TREC. In ATE as a stage of information retrieval, we concentrate on reflecting these measures on keywords depending on relevance of retrieved terms. We can not compare our results with the results obtained by IR systems or TREC results, because of different point of view. Just a small portion of researches used our method. For example, Zechner's system achieves recall/precision values of 0.55/0.46 for extraction of keywords from abstracts of six sentences and values of 0.74/0.41 from abstracts of ten sentences [60].

One of the most recent ATE systems was proposed by Kerner on 2003 to extract keywords from abstracts and titles [61]. He used two features; term frequency and importance of sentences depending on their position in the text and analysis of text using syntactic relations. This model was applied on a set of abstracts of academic papers containing keywords composed by their authors as we did in experiment 1. He used another method of evaluation using full match, partial match, and failures. To be compared with our results, let the tested 332 keywords be the total number of relevant keywords (A+C), and total matches (full or partial)

be the number of retrieved relevant 205 keywords (A), then recall equals to $(205/332=0.617)$. On the other hand, failures are 0.383.

If both results are compared with our best results (average recall/precision of 0.781/0.455 for TDVM in experiment 1 and 0.772/0.513 for TFxIDF in experiment 3), our results are slightly better. Refer to tables 5.3 and 5.9. However, the systems might not be comparable, because they deal with different tasks, and with different techniques and different kinds of domains or documents.

5.4 Summary

In our comparison we concentrated on retrieval and relevancy of the extracted keywords to calculate recall and precision evaluation measures. These measures were calculated in the three experiments for all techniques in different conditions. Results of experiment 1 have shown that the best results for all techniques are obtained when both stemming and stop-words removal are applied. TDVM got the best recall/precision pair with the values 0.781/0.455, then TF with 0.75/0.436, then TFxIDF with 0.656/0.382, and finally, IDF with 0.468/0.273.

Results of experiment 2 have shown that recall is proportional with the number of retrieved terms extracted by ATEWB, while precision is inversely proportional with the number of retrieved terms. The best recall and worst precision are obtained in exp 2.3 for all techniques when 132 terms are retrieved. The worst recall and best precision are obtained in exp 2.1 when 65 terms are retrieved.

Results of experiment 3 have shown that the fastest technique is TFxIDF which weights about 923 term/second, while the slowest is TDVM which weights about 77 term/day by estimation. Results have also shown that recall, unlike precision,

is proportional with the number of retrieved terms. The best recall/precision value for TF is 0.421/0.30 and for TFxIDF is 0.807/0.575 when the number of retrieved terms is 80. It is clear that TFxIDF got the highest scores in this experiment.

Results of experiments 2 and 3 have shown that TF and IDF was not affected by collection size (nearly the same recall/precision in both experiments). TFxIDF was clearly improved by increasing the collection size; average recall/precision is changed from 0.269/0.142 to 0.772/0.513.

In general we can say that the best result can be obtained if we use TFxIDF technique with stemmer and stop words are applied. In addition, TFxIDF gives the best results as the size of the collection is increased, and when the total number of relevant keywords approximately equals to the total number of retrieved terms. IDF may be excluded; it just improves TF if combined with it. TDVM also may be excluded because of complexity and high specifications of needed hardware.

6 Conclusions and Future Work

6.1 Conclusions

There is a huge amount of information distributed all over the world through Internet, intranets and electronic libraries. All are thinking how to get the information as fast as possible. When your collection becomes large, you lose your time searching for a piece of information.

Term extraction is the first step to get your information as fast as possible, it is the base of search engines and content filters. It builds the keywords or index terms of documents to describe their contents. These index terms may be used for search engines; that is when you enter a query to a search engine, your query is parsed and compared with the index terms produced by the term extractor. If any is matched, it refers to the document/documents containing it. This is what ATEWB was developed to do.

The main issues concluded by the current work are:

1. Stemming and Stop words removal improve the effectiveness and efficiency of all statistical techniques and gives more retrieval efficiency in reducing computations time and increasing accuracy.
2. Results indicate that TFxIDF is the fastest and sufficiently accurate Statistical Technique. TF and TFxIDF are nearly equivalent in performance and accuracy. They are faster than TDVM which needs very huge computations while its accuracy is not much better. It slightly improves precision that is used to measure accuracy. On the other hand, IDF is not recommended as a

stand alone technique. It needs to be combined with other techniques like TF to improve its results.

3. If our best results (average recall/precision of 0.781/0.455 for TDVM in the first experiment and 0.772/0.513 for TFxIDF in the third experiment) are compared with previous recent studies (Zechner's system with recall/precision values of 0.55/0.46 in one experiment and 0.74/0.41 in another, and Kerner system with recall value of 0.617), our results are slightly better. However, the systems might not be comparable, because they deal with different criteria and with different techniques and different kinds of domains or documents.
4. Increasing the collection size improves accuracy. On the other hand, the use of database engines and SQL for computations may improve performance and reduce time and efforts of programming statistical techniques.

6.2 Future Work

The following are suggestions for future work:

1. Applying the same statistical techniques for multilingual like Arabic or European languages, and building a stemming algorithm for Arabic language. In fact a sample Arabic Stemmer was implemented in this research to get the root of any Arabic verb and its derivatives.
2. Implementing other techniques especially neural networks and Kohonen's Self-organizing Map SOM which has been proposed by Teuvo Kohonen for textual classification and thesauri building.

3. Improving ATEWB to build my own search engine for both English and Arabic languages and using parallel and distributed computing to improve the performance of different algorithms used.

Obstacles:

- One of the big troubles facing researchers in Palestinian universities in this field is the unavailability of Labs and supercomputers needed for huge computations. Such computations may need a few days to get a result on the available P4 computers if they continue working without interrupts. This limits the experiments to be applied on small collections which can not give the clear picture.
- The limited number of resources in the university libraries, and on the internet. Most of digital libraries with good references need paid registration, and the university doesn't register with these libraries.

Lastly, information retrieval is an interesting subject and it deserves to be studied and improved by researchers, which makes our life easy.

Bibliography

- [1] Horecky, Jan. 1997. *Intension and Extension of a Term*. Human Affairs, 7, 1997, 2, 134-140.
- [2] Lahtinen, T. 2000. *Automatic Indexing: an approach using an index term corpus and combining linguistic and statistical methods*. University of Helsinki. Ph.D. thesis. Finland.
- [3] Daille, B. 1994. *Study and Implementation of Combined Techniques for Automatic Extraction of Terminology*. University Paris 7. France.
- [4] Salton, G. and Schneider, H. 1982. *Lecture Notes in Computer Science. Research and Development in Information Retrieval*. Springer-Verlag, Berlin Heidelberg New York.
- [5] Dunham, M. H. 2003. *Data Mining: Introductory and Advanced Topics*. Prentice Hall by Pearson Education, Inc. pg 26.
- [6] Ananiadou, S. 1994. *A Methodology for Automatic Term Recognition*. Proceedings of COLING94.
- [7] Feldman, S. 1999. *NLP Meets the Jabberwocky: Natural Language Processing in Information Retrieval*. Information Today.
Online, <http://www.onlinemag.net/OL1999/feldman5.html>.
- [8] Liddy, E. 1998. *Enhanced Text Retrieval Using Natural Language Processing*. ASIS Bulletin.
On the Web at <http://www.asis.org/Bulletin/Apr-98/liddy.html>.

- [9] VanRijsbergen, C.J. 1999. *Information Retrieval*. University of Glasgow. Scotland.
- [10] Smeaton, A. F. 2000. *Indexing, Browsing and Searching of Digital Video and Digital Audio Information*. The third European Summer School Information in Retrieval (ESSIR 2000).
- [11] Jones, G.J.F., Foote, J.T., Sparck Jones, K. and Young, S.J. 1996. *Retrieving spoken documents by combining multiple index sources*. In Proceedings of SIGIR 96, Research and Development in Information Retrieval, 30-38, Zürich, ACM Press.
- [12] Schäuble, P. 1997. *Multimedia Information Retrieval*. Kluwer Academic Publishers.
- [13] Lee, H., Smeaton, A. F., O'Toole, C., Murphy, N., Marlow, S. and O'Connor, N. E. 2000. *The Físchlár Digital Video Recording, Analysis, and Browsing System*. In Proceedings of RIAO '2000: Content-Based Multimedia Information Access, Paris.
- [14] Perry, B., Chang, S. K., Dinsmore, J., Doermann, D., Rosenfeld, A. and Stevens, S. 2000. *Content-Based Access to Multimedia Information*. From Technology Trends to State of the Art. Kluwer Academic Publishers.
- [15] Kageura, K., and Umino, B. 1996. *Methods of Automatic Term Recognition*. Japan.
- [16] Luhn H. P. 1957. *A Statistical Approach to Mechanized Encoding and searching of Literary Information*. IBM Journal of Research and development 2(2).

- [17] Salton, G. 1986. *Another Look at Automatic Text-Retrieval Systems*. Cornell University. Communications of the ACM, Volume 29, Issue 7. USA.
- [18] Robertson, S. E. & Spark Jones, K. 1997. *Simple, proven approaches to text retrieval*. Technical report 356, Computer Laboratory, University of Cambridge.
- [19] Abuzir, Y. and Kaczmariski, P. *The use of ADM for Project Development and Control: A Quantitative approach*. CCAI-The journal for the integrated study of Artificial Intelligence, Cognitive Science and Applied Epistemology -Communication & Cognition, Vol. 18 n° 3-4, Gent-Belgium.
- [20] Abuzir, Y. 2004. *Deriving Concepts Hierarchies*. Proceedings CLUK2004 conference. University of Birmingham. England. January 2004.
- [21] Abuzir, Y., Van Vosselen, N. Gierts, Kaczmariski, S. P., and Vandamme, F. 2002. *MARIND Thesaurus for Indexing and Classifying Documents in the Field of Marine Transportation*. Proceedings MEET/ MARIND 2002 Oct. 6-1. Varna Bulgaria. 2002.
- [22] Abuzir, Y. *E-mail Classification based on Thesaurus*. CCAI-The journal for the integrated study of Artificial Intelligence, Cognitive Science and Applied Epistemology -Communication & Cognition, Vol. 18 n° 3-4, Gent-Belgium.
- [23] Cleveland, G. 1995. *Overview of Document Management Technology*. Information Technology Services. National Library of Canada.
- [24] Boyce, B. R., Meadow, C. T. and Kraft, C. T. 1995. *Measurement in Information Science*. New York Academic Press.

- [25] American National Standards Institutes. 1968. *Basic Criteria for Indexes*. ANSI Z39.4, New York.
- [26] Borko, H., and Bernier, C. L. 1978. *Indexing concepts and methods*. Academic Press Inc., New York.
- [27] Allan, J. Fall 2003. *Slide show in IR lectures. Text Indexing (indexing model)*. University of Massachusetts Amherst.
- [28] Wilson, B. 1998, 2004. *The Natural Language Processing Dictionary*. The University of New South Wales. Australia. Bill Wilson's home page: <http://www.cse.unsw.edu.au/~billw/nlpdict.html#corpus>, last visited on Sept. 2004.
- [29] Cleverdon, C. W., and Keen, E. M. 1966. *Aslib-Cranfield Research Project*. Vol. 2. Test Results. Cranfield Institute of Technology. England.
- [30] Salton, G. 1973. *Recent Studies in Automatic Text Analysis and Document Retrieval*. ACM 20, 2, 258-278. An evaluation of various automatic text-analysis and indexing methods.
- [31] Spark-Jones, K. 1972. *A Statistical Interpretation of Term Specificity and its Application in Retrieval*. Journal of Documentation 28(1), 11-21.
- [32] Salton, G. and Yang, C. S. 1973. *On the Specification of Term Values in Automatic Indexing*. Journal of Documentation 29(4) 351-372.
- [33] Salton, G. 1975. *A Theory of Indexing*. Vol. 18 of Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia.
- [34] Porter, M. F. 1980. *An Algorithm for Suffix Stripping*. Originally published

- in Program, 14 no. 3, pp 130-137, July 1980.
- [35] Lippmann, R. P. 1987. *An introduction to computing with neural nets*. IEEE ASSP Magazine, 4-22.
 - [36] Doszkocs, T. E., Reggia, J., and Lin, X. 1990. *Connectionist models and information retrieval*. Annual Review of Information Science and Technology (ARIST), 25.
 - [37] He, Q. 1999. *Neural Network and Its Application in IR*. University of Illinois.
 - [38] Kohonen, T. 1988. *Self-Organization and Associative Memory*. 2nd Edition. Berlin: Springer-Verlag.
 - [39] Pape, D. X. 1998. <http://www.canis.uiuc.edu/interspace/showcase.html>. last visited on April 1, 2005.
 - [40] Orwig, R. E., Chen, H., and Nunamaker, J. F. 1997. *A graphical, self-organizing approach to classifying electronic meeting output*. Journal of the American Society for Information Science, 48(2), 157-170.
 - [41] Chung, Y. M., Potteringer, W. M., and Schatz, B. R. 1998. *Automatic Subject Indexing Using an Associative Neural Network*. Digital Libraries 98. The 3rd ACM conference on digital libraries.
 - [42] Crestani, F., Lalmas M., van Rijsbergen, C. J., and Campbell, I. 1998. *A Survey of Probabilistic Models in Information Retrieval*. ACM Computing Surveys.
 - [43] Salton, G. 1986. *Recent Trends in Automatic Information retrieval*. Cornell University.

- [44] Schwarz, C. 1990. *Automatic Syntactic Analysis of Free Text*. Journal of the American Society for Information Science, 41(6): 408-417.
- [45] Salton, G. 1966. *Automatic Phrase Matching*. In Hays (Ed.) Readings in Automatic Language Processing, New York.
- [46] Hersh, W.R. and Molnar, A. 1995. *Towards New Measures of Information Retrieval Evaluation*. Proceedings of the 18th ACM-SIGIR International Conference on Research and Development in Information Retrieval, Seattle, WA, USA, 164-170.
- [47] Gao X., Murugesan, S. and Lo, B. 1998. *Multi-dimensional Evaluation of Information Retrieval Results*. Computer Society, Proceedings of IEEE/WIC/ACM international conference.
- [48] Pollit, A. S. 1989. *Information Storage and Retrieval Systems: Origin, Development and Applications*. Toronto, John Wiley & Sons, 164-165.
- [49] Bennett, N. A. H., Qin. Powell, K., and Schatz, B. R. 1999. *Extracting Noun Phrases for all of MEDLINE*. University of Illinois at Urbana-Champaign. Proc AMIA Symposium.
- [50] Belnab, N. D. and Steel T.B. 1996. *The Logic of Questions and Answers*. Yale University Press, New Haven and London.
- [51] Chang, C.L. and Lee, R.T.C. 1973. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York.
- [52] Good, I.J. 1967. *The decision-theory approach to the evaluation of information retrieval systems*, *Information Storage and Retrieval*, 3, 31-34.
- [53] Spark-Jones, K. 1973. *Index Term Weighting*. Information Storage and

Retrieval 9(11).

- [54] Noreault, T., McGill, M., and Koll, M. B. 1977. *A Performance Evaluation of Similarity Measure, Document Term Weighting Schemes and Representation in a Boolean Environment*. In Oddy, R. N. (ed.), Information Retrieval Research. London.
- [55] Church, K. W. and Gale, W. A. *Inverse Document Frequency (IDF), A Measure of Deviations from Poisson*. Gale ATT Bell Laboratories Murray Hill, NJ, USA.
- [56] Robertson, S. E. et al. 1995. *Okapi at TREC-3*, in Harman, D. (Ed) *Overview(TREC-3)*. NIST SP 500-225, 1995, 109-126.
- [57] Dubian, David S. 1998. *Further Cautions for the Calculation of Discrimination Values*. Graduate School of Library and Information Science. University of Illinois at Urbana-Champaign. 1-2.
- [58] The web site <http://home.od.ua/~relayer/algo/text/stem.html>, last time visited Nov. 28 2004.
- [59] NPL collection. IR resources. IR Group Web site. University of Glasgow. Led by Professor Keith van Rijsbergen. http://ir.dcs.gla.ac.uk/ir_resources.
- [60] Zechner, K. 1996. *Fast Generation of Abstracts from General Domain Text Corpora by Extracting Relevant Sentences*. Proceedings of Computational Linguistics Conference.
- [61] Kerner, Y. H. 2003. *Automatic Extraction of Keywords from Abstracts*. Knowledge-Based Intelligent Information and Engineering Systems: Proceedings of 7th International Conference, KES, UK.

A. Appendix A

Here are the key words or index terms and stop-words used for experiments 1-3.

Experiment1 Keywords		Experiment2 Keywords		Experiment3 Keywords	
algorithm	statistical	cascad	pressure	acoust	skin
ambiguity	summarization	circuits	principle	admitt	solar
analysis	tagging	computations	probablistic	ammonia	spark
classification	term	direct	processing	bai	sphere
cluster	terminology	discharg	progress	balanc	stabil
clustering	terms	electronics	properties	caviti	theori
compound	values	field	propose	circuit	time
content	weighting	filter	proton	cloud	triod
discrimination		flare	pump	coat	turbul
discriminative		geomagnetic	reaches	common	variat
distribution		linear	resistance	comput	vehicl
domain		mode	shift	conductor	absorb
extracting		model	signal	conjug	anod
extraction		module	silver	divid	area
index		multiple	skin	eclip	code
information		natural	solar	effect	colour
language		near	stabilization	electrod	content
lead		number	subharmonic	electron	cosmic
model		numerical	switches	field	deton
network		observ	system	gap	fade
neural		orbits	transfer	irregular	faradai
none		origin	transistor	kmc	geomagnet
noun		oscilation		layer	hamiltonian
paragraphs		parameters		matric	height
parser		particles		matrix	helic
phrases		plane		meteor	highaltitud
regularity		plasma		morpholog	index
retrieval		polar		optimum	
sentence		portion		outer	
similarity		power		plasma	

Stop-Words List					
calculate	consequence	do	explained	handling	just
calculated	consequences	does	explaining	harmfull	keep
called	consequently	domain	extra	harmfully	kept
can	consider	domains	fairly	has	know
cannot	considerable	done	favourable	have	knowing
careful	considerably	due	few	having	known
carefully	consideration	during	fewer	he	knows
carried	considerations	each	fig	held	lead
carry	contain	early	figure	hence	leading
catch	containing	easier	figures	here	leads
certain	contrary	easily	find	highly	like
change	could	efficiency	finds	his	made
clear	data	efficient	follow	holding	make
clearly	define	either	followed	holds	making
collaborate	defined	en	following	hour(s)	many
collaborated	delay	end	follows	however	may
collaborates	delayed	enough	for	I	maybe
collaboration	demand	ensure	found	if	me
commonly	department	entering	from	illustrates	mean
commonly	depending	especially	further	illustrating	meaning
completely	describe	essential	furthermore	in	means
concerned	described	essentially	general	include	might
concerning	describes	estimated	generally	including	more
conclude	details	et	give	indicating	moreover
concluded	difference	etc	given	into	most
concludes	different	even	gives	introduced	move
conclusion	difficult	every	giving	introduction	much
conduct	discuss	ex	good	is	must
conducted	discussed	example	guess	it	name
conducts	discussing	explain	had	its	Named
namely	possibly	serious	summary	u	why
need	previous	several	table	under	will
needed	previously	shall	take	until	with
next	probably	should	taken	up	within
no	proceed	show	akes	upon	without
not	proceeded	showed	taking	upper	would
now	proceedings	shown	talking	use	yes
observe	rather	shows	tells	used	you
observed	recent	similar	than	useful	
obtain	regardless	simulate	that	uses	
occur	report	since	the	using	
of	required	small	their	usual	
offer	requirement	so	them	usually	
offers	requirements	some	then	various	

often	requires	sometimes	theory	vary	
on	resulting	somewhat	there	varying	
only	rise	soon	thereby	versus	
or	rises	specific	therefore	very	
order	s	still	these	was	
other	same	studied	they	we	
others	sample	study	this	well	
our	samples	success	thorough	were	
out	say	successfull	those	when	
over	section	successfully	thought	where	
overview	see	such	through	whereas	
paragraph	seem	suggest	thus	whether	
photohograph	seems	suggested	to	which	
photographs	seen	suggesting	too	while	
possibility	seldom	suggests	troublesome	whole	
possible	sender	suitable	tryout	whose	

B. Appendix B

B.1 ATEWB System Installation

B.1.1 Microsoft windows environment

To install ATEWB package for windows, follow the instructions below:

1. Install MYSQL version 5.0 or later for windows from the CD attached with this thesis on the path `X:\Win2KXP\mysql` where X is drive letter, or you can download it from the website <http://www.mysql.com>.
2. Install The Java Runtime Environment from `X:\Win2KXP\JRE` or download it from web at <http://java.sun.com/j2se/1.4.2/download.html>.
3. Copy ATEWB executable JAR from `X:\Win2KXP\ATEWB` any where in your hard drive and create a shortcut on the desktop.
4. Start MYSQL server from command prompt by the command line `Y:\MYSQL\bin\mysqld.exe --max_allowed_packet=160000`, where Y is the drive letter where you have installed MYSQL. Then double click ATEWB shortcut to start.

B.1.2 Linux RedHat environment

To install ATEWB package for Linux, follow the instructions below:

1. Install MYSQL5.0 or later for Linux, from `cdrom/linux/mysql` from the attached CD, or download it from the website <http://www.mysql.com>.

2. Install The Java Runtime Environment from `cdrom/linux/jre` or download it from the website <http://java.sun.com/j2se/1.4.2/download.html>.
3. Copy ATEWB executable JAR from `cdrom/linux/atewb` to your hard drive and create a shortcut on the desktop with X-windows.
4. Start MYSQL server by the command `mysql/bin#./mysqld --max_allowed_packet=160000` , and double click ATEWB or use the command `atewb path# ./atewb` to start ATEWB. You are ready to extract index terms.

C. Appendix C

C.1 Working with ATEWB

This section describes how to use ATEWB; this includes installation and using the package through screenshots for each step. Refer to Section 5.5 for minimum system requirements to install and run ATEWB. To install ATEWB on both Microsoft windows and/or UNIX, refer to Appendix B.

C.1.1 Using ATEWB

To start ATEWB use the following instructions:

1. After installation, Double click on the ATEWB executable jar; the main screen will appear, as shown in figure C.1.

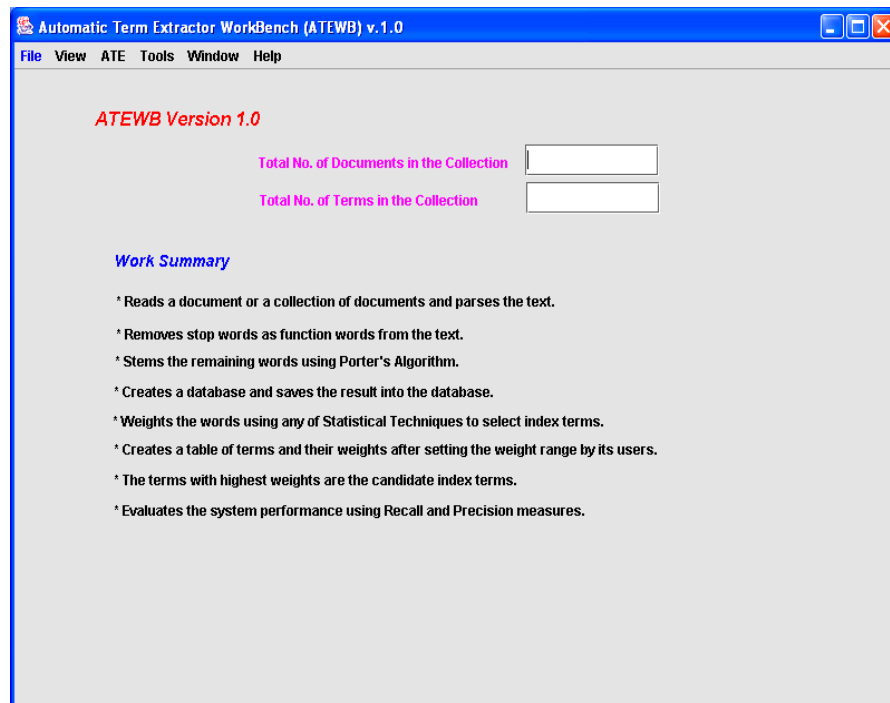


Figure C.1: ATEWB main screen

2. To start a new project, select "New ATE" from "File" menu, *see* figure C.2, a dialog box appears with the title New Project. This dialog is used to set project options. *See* figure C.3.
3. To change project options, use New Project dialog box. *See* figure C.3. To select file type in the collection, use "File Type" pull down menu and select txt, htm or html to search for text, htm or html files respectively. If stemming is to be used select "Porter's Stemming" from "Options" tab, if not select "No Stemming". *See* figure C.3.

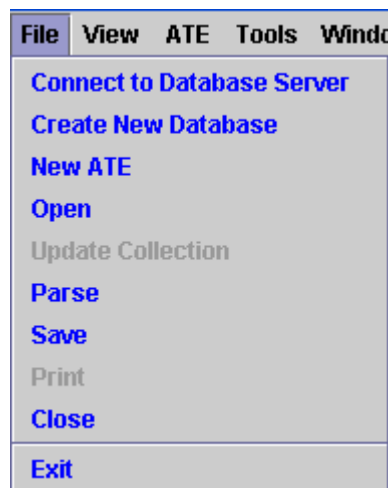


Figure C.2: ATEWB File menu

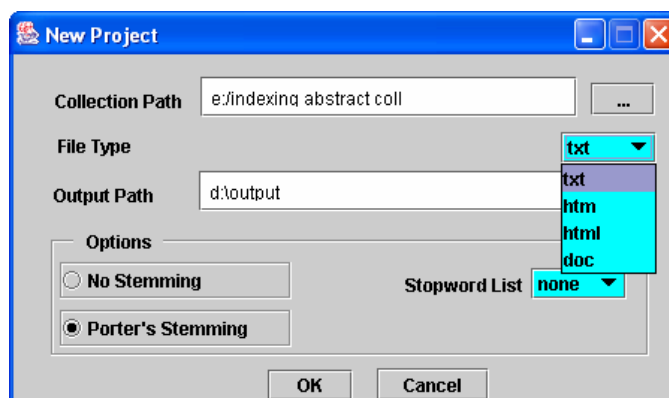


Figure C.3: ATEWB New Project dialog box

4. To select a stop word list use "Stopword List" pull down menu in the "Options" tab, and select the list to be used or none for no list. *See* figure C.4. When options done click "OK" to continue or "Cancel" to cancel.

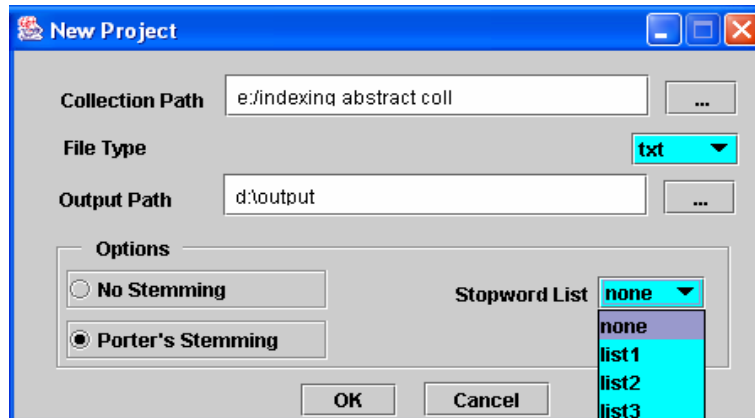


Figure C.4: ATEWB New Project options

5. To browse the document collection folder, click on ellipsis (...) button in figure C.4, "Open" dialog box appears. *See* figure C.5. Navigate the hard drive for the collection folder, select and click "Open" to continue or "Cancel" to stop.

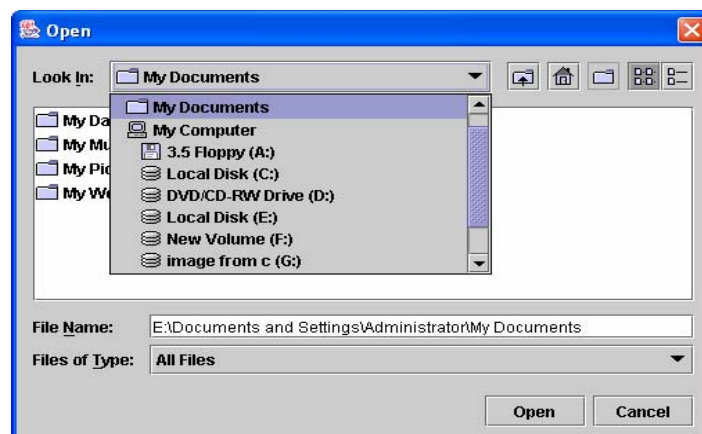


Figure C.5: ATEWB browser

6. To create a new database with its tables and initialize them, select "Create New Database" submenu in "File" menu. *See* figure C.2. "Create New Database" dialog box appears. *See* figure C.6. Enter database name and click "OK" to continue, "Cancel" to exit.

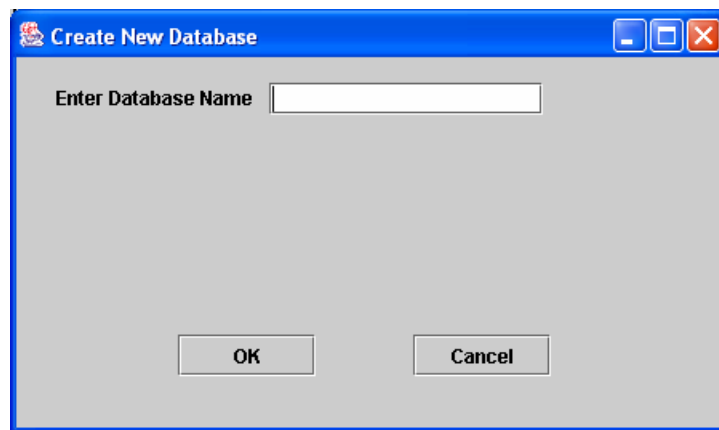


Figure C.6: Create New Database dialog box.

7. Next step is to connect to MYSQL database server, to do so click on "Connect to Database Server" submenu in "File" menu, "Database Options" dialog box appears. Click "OK" to use default information or enter user name, password, schema (database name) and database server name or IP address. Click "OK" to continue, "Cancel" to stop. *See* figure C.7.
8. To start reading the documents in the collection and parsing words, click "Parse" submenu in "File" menu. *See* figure C.2. While parsing, project options like stemmer and stop list are taken into consideration.
9. To choose which statistical technique for ATE to weight the terms, click one of the four techniques in "Statistical Technique" submenu in "ATE"

main menu. *See* figure C.8. Select "Compute TF" for TF weighting, "Compute IDF" for IDF weighting, "Compute TFxIDF" for TFxIDF weighting or "Compute TDV" for TDV weighting.

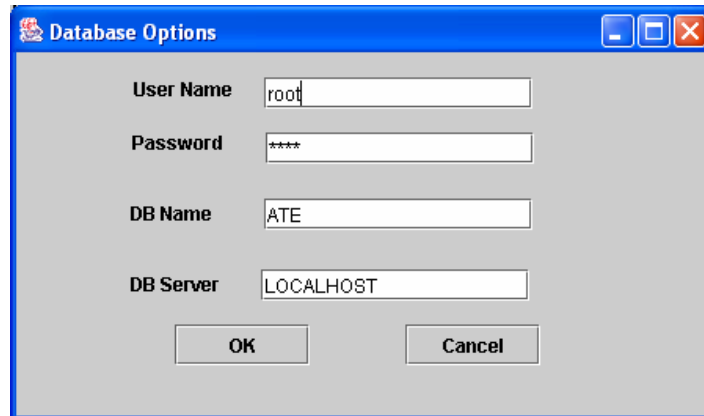


Figure C.7: ATEWB Database Options dialog box.

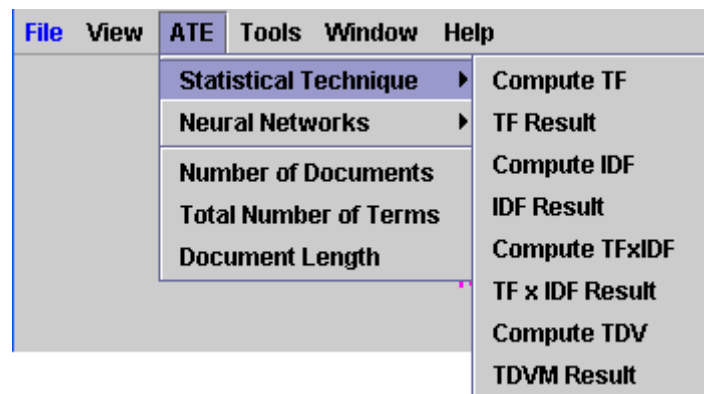


Figure C.8: ATEWB ATE menu with its submenus

10. Select "TF Result" to view term frequency technique result, "IDF Result" to view inverse document frequency technique result, "TFxIDF Result" to view TFxIDF result and "TDVM Result" to view TDVM result. Each time a dialog box appears to set the condition and the order in which the result (index term selected by the system) will appear, *see* figure C.9. To use default settings just click "Show Result" button to view the result, or set

the condition (s) and the name of the field by which the result will be ordered and click "Done" button then "Show Result" button to view the result. Click "Cancel" button to cancel the operation.

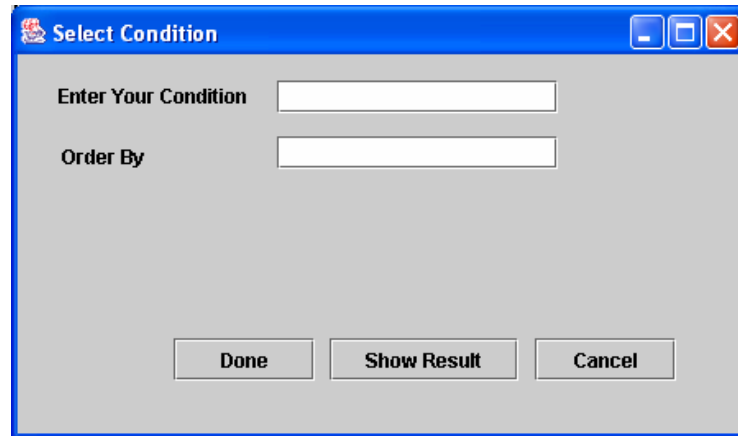


Figure C.9: Conditions of selected index terms.

11. TF result appears as a screen divided by a splitter into two areas. The first is a table of terms and their weights, the second is a preview text area containing the document. If a term in the table is clicked, the document containing it appears with the term distribution in the document marked using a different color. *See* figure C.10.
12. Select "IDF Result" to view the result of inverse document frequency technique, a screen divided into two areas appears. The first is a table of terms and their weights computed by IDF, the second is a text area containing the document full path. If a term in the table is clicked, a list of documents containing it appears with their full path (collection path). *See* figure C.11.
13. Select "TFxIDF Result" to view the result of combined TF-IDF technique, a screen divided into two areas appears. The first is a table of terms and

Term Frequency Table					
ID#	Term (VWord)	Te...	T...	...	
733	algorithm	4	0...	3	abstract this thesis discusses the problems and the methods of finding relevant information in large collections of documents. the contribution of this thesis to this problem is to develop better content analysis methods which can be used to describe document content with index terms. index terms can be used as meta-information that describes documents, and that is used for seeking information. the main point of this thesis is to illustrate the process of developing an automatic indexer which analyses the content of documents by combining evidence from word frequencies and evidence from linguistic analysis provided by a syntactic parser. the indexer weights the expressions of a text according to their estimated importance for describing the content of a given document on the basis of the content analysis. the typical linguistic features of index terms were explored using a linguistically analysed text collection where the index terms are manually marked up. this text collection is referred to as an index term corpus. specific features of the index terms provided the basis for a linguistic term-weighting scheme, which was then combined with a frequency-based term-weighting scheme. the use of an index term corpus like this as training material is a new method of developing an automatic indexer . the results of the experiments were promising.
734	linguist	4	0.4	11	
735	noun	4	1.0	13	
736	extract	4	0.4	9	
737	extract	4	0.4	4	
738	discrimin	4	1.0	8	
739	proporti	4	1.0	1	
740	languag	4	0...	7	
741	cluster	4	1.0	10	
742	inform	5	0...	12	
743	document	5	0.5	11	
744	document	5	0.5	5	
745	sentenc	5	0...	2	
746	content	5	0.5	5	
747	content	5	0.5	11	
748	noun	6	0.6	9	
749	parser	6	0.6	4	
750	noun	6	0.6	4	
751	neural	6	0...	0	
752	parser	6	0.6	9	
753	term	7	0.7	11	
754	term	7	0.7	5	
755	network	7	1.0	0	
756	cluster	7	1.0	3	
757	model	9	1.0	7	
758	term	9	1.0	12	
759	index	10	1.0	5	
760	phrase	10	1.0	4	
761	index	10	1.0	11	
762	phrase	10	1.0	9	
763	method	12	1.0	2	

Figure C.10: ATEWB term frequency result screen

Inverse Document Frequency Table					
ID#	Term (VWord)	Document Frequency	Weight		
276	reason	2	0.698970004336019	The term found in: e:/indexing abstract coll/ab1.txt e:/indexing abstract coll/ab4.txt e:/indexing abstract coll/ab5.txt e:/indexing abstract coll/ab6.txt e:/indexing abstract coll/ab7.txt e:/indexing abstract coll/ab9.txt	
277	new	3	0.522878745280338		
278	content	3	0.522878745280338		
279	discrimin	3	0.522878745280338		
280	index	3	0.522878745280338		
281	automat	3	0.522878745280338		
282	collect	3	0.522878745280338		
283	measur	3	0.522878745280338		
284	main	3	0.522878745280338		
285	applic	3	0.522878745280338		
286	statist	3	0.522878745280338		
287	languag	3	0.522878745280338		
288	relev	3	0.522878745280338		
289	text	3	0.522878745280338		
290	rank	3	0.522878745280338		
291	weight	4	0.397940008672038		
292	base	4	0.397940008672038		
293	system	4	0.397940008672038		
294	result	4	0.397940008672038		
295	problem	4	0.397940008672038		
296	ir	4	0.397940008672038		
297	propos	4	0.397940008672038		
298	model	4	0.397940008672038		
299	method	4	0.397940008672038		
300	word	5	0.301029995663981		
301	document	5	0.301029995663981		
302	term	5	0.301029995663981		
303	paper	5	0.301029995663981		
304	inform	6	0.221848749616356		
305	retriev	6	0.221848749616356		
306	abstract	10	0.0		

Figure C.11 ATEWB IDF result screen

ID#	Term (Word)	TFxIDF Weight	Document ID#
735	global	0.372020987595092	1
736	index	0.405683871082213	11
737	index	0.405683871082213	5
738	sentenc	0.416666666666667	2
739	algorithm	0.42516684296582	3
740	classic	0.428571428571429	3
741	tagger	0.496027983460123	6
742	hmm	0.496027983460123	6
743	tagger	0.496027983460123	14
744	hmm	0.496027983460123	14
745	attribut	0.5	8
746	comput	0.5	8
747	compound	0.5	13
748	subclust	0.5	10
749	similar	0.5	10
750	idea	0.558031481392638	13
751	cluster	0.594316128917787	3
752	noun	0.594316128917787	13
753	phrase	0.594316128917787	4
754	phrase	0.594316128917787	9
755	cluster	0.594316128917787	10
756	discrimin	0.594316128917787	8
757	properti	0.744041975190185	1
758	ambigu	0.744041975190185	6
759	ambigu	0.744041975190185	14
760	igr	0.75	10
761	neural	0.857142857142857	0
762	paragraph	1.0	1
763	network	1.0	0

abstract this paper proposes a new term weighting method for summarizing documents retrieved by ir systems. unlike query-biased summarization methods, our method utilizes not the information of query, but the similarity information among original documents by hierarchical **clustering**. in order to map the similarity structure of the **clusters** into the weight of each word, we adopt the information gain ratio (igr) of probabilistic distribution of each word as a term weight. if the amount of information of a word in a **cluster** increases after the **cluster** is partitioned into sub-**clusters**, we may consider that the word contributes to determine the structure of the sub-**clusters**. the igr is a measure to express the degree of such contribution. we will show the effectiveness of our method based on the igr by comparison with other systems.

Figure C.12: ATEWB TFxIDF result screen

ID#	Term (Word)	Discrimination Value
276	thesi	0.0182441
277	support	0.0203129
278	entropi	0.0203129
279	vector	0.0203129
280	maximum	0.0203129
281	prefer	0.0203129
282	model	0.0206772
283	million	0.0241204
284	subclust	0.0252092
285	similar	0.0252092
286	summar	0.0270291
287	phrase	0.03258
288	content	0.0404226
289	idea	0.0416887
290	machin	0.0466608
291	comput	0.0489722
292	attribut	0.0489722
293	parser	0.0500537
294	igr	0.0583064
295	base	0.0593459
296	algorithm	0.063852
297	local	0.0675927
298	global	0.0675927
299	theoret	0.0796483
300	valu	0.0812969
301	properti	0.0836511
302	problem	0.0997435
303	cluster	0.114886
304	neural	0.294592
305	paragraph	0.310777
306	network	0.433911

The term found in:
e:/indexing abstract coll/ab1.txt
e:/indexing abstract coll/ab2.txt
e:/indexing abstract coll/ab6.txt
e:/indexing abstract coll/ab9.txt

Figure C.13: ATEWB TDVM result screen

their weights, the second is a preview text area containing the document. If a term in the table is clicked, the document containing it appears with the term marked using different color. *See* figure C.12.

14. Select "TDVM Result" to view the result of TDVM technique, a screen divided into two areas appears. The first is a table of terms and their discrimination values, the second is a text area containing the document full path. If a term in the table is clicked, a list of documents containing it appears with their full path (collection path). *See* figure C.13.
15. To get the total number of documents in the collection, click "Number of Documents" submenu in "ATE" main menu. Finally, to get the total number of terms in the collection, click "Total Number of Terms" in "ATE" main menu, *see* figure C.8. The result appears in two text fields in the main screen. *See* figure C.14.

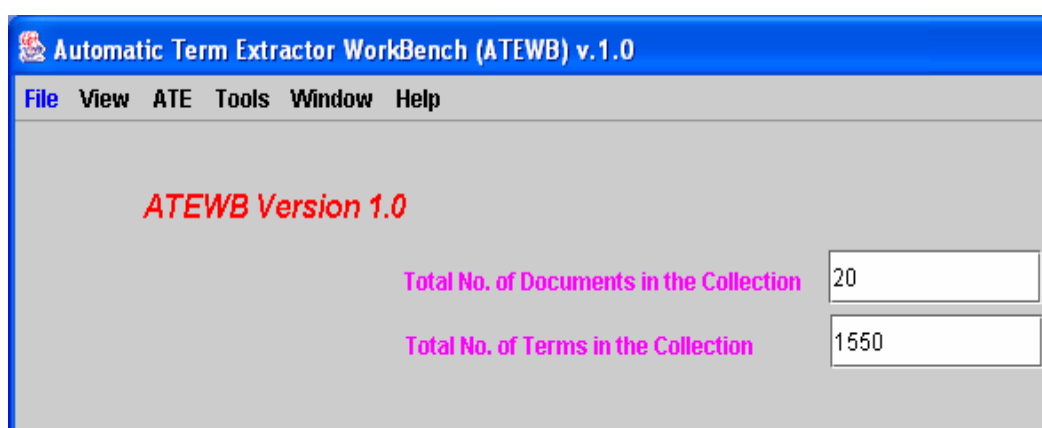


Figure C.14: ATEWB document summery

16. To show the original relevant index terms or keywords that to be compared with the index terms extracted by ATEWB, save them in a document and click "Create Keywords" in "Tools" menu, *see* figure C.15.

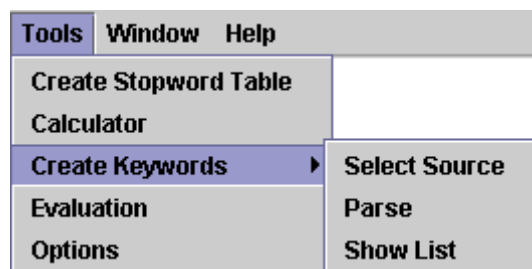
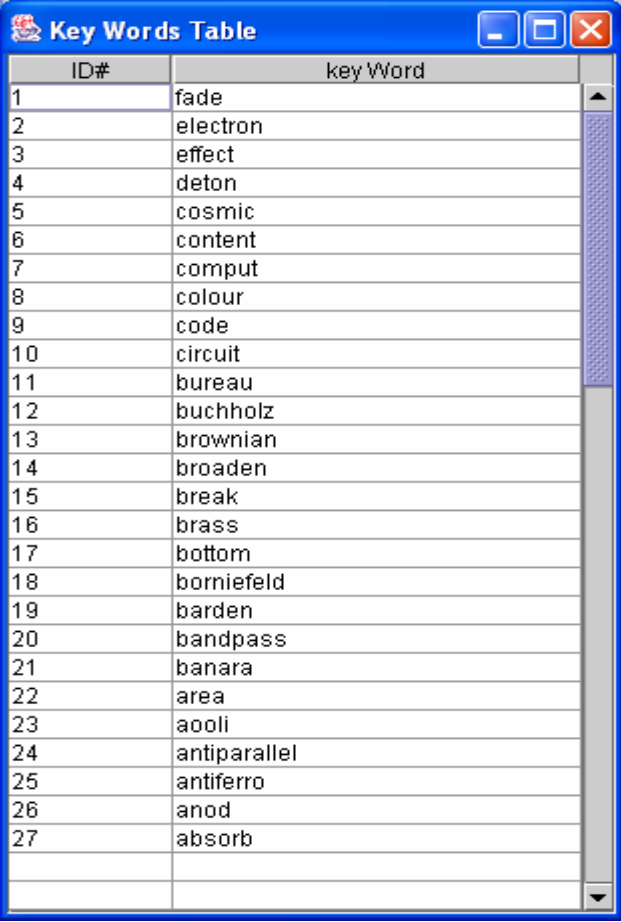


Figure C.15: ATEWB tools menu

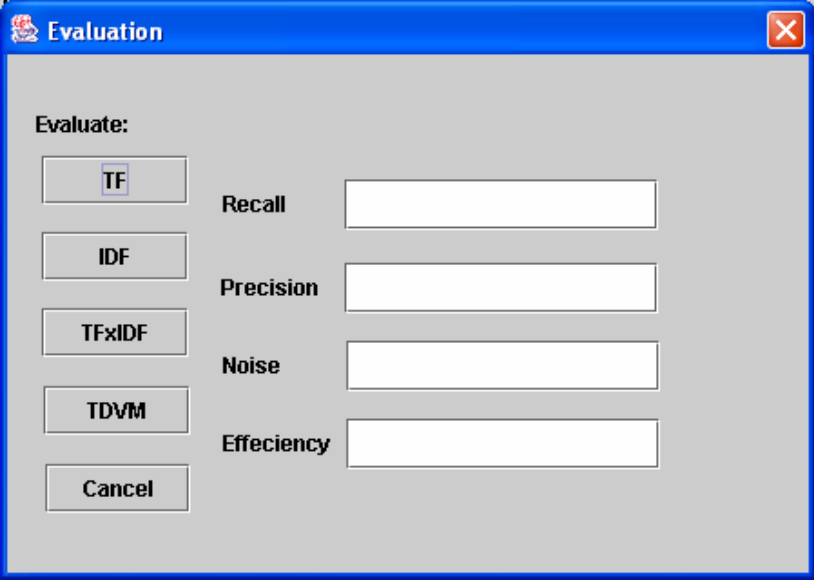
17. Click "Select Source" in "Create Keywords" submenu to browse the folder containing the keywords document, Parse to parse the keywords and "Show List" to view the keywords, *see* figure C.16. The result is a table containing the keywords and their IDs. These keywords are used to evaluate the system by computing Recall, Precision, Noise and Efficiency that measure ATEWB performance.
18. To evaluate the index terms selected by the system, click "Evaluation" in "Tools" menu. "Evaluation" dialog box appears with four buttons, one for each statistical technique, *see* figure C.17.



The 'Key Words Table' dialog box displays a list of 27 keywords, each assigned a unique ID number from 1 to 27. The keywords are listed in a single column, and the corresponding ID numbers are in the adjacent column. The list includes terms like 'fade', 'electron', 'effect', 'deton', 'cosmic', 'content', 'comput', 'colour', 'code', 'circuit', 'bureau', 'buchholz', 'brownian', 'broaden', 'break', 'brass', 'bottom', 'borniefeld', 'barden', 'bandpass', 'banara', 'area', 'aoli', 'antiparallel', 'antiferro', 'anod', and 'absorb'.

ID#	key Word
1	fade
2	electron
3	effect
4	deton
5	cosmic
6	content
7	comput
8	colour
9	code
10	circuit
11	bureau
12	buchholz
13	brownian
14	broaden
15	break
16	brass
17	bottom
18	borniefeld
19	barden
20	bandpass
21	banara
22	area
23	aoli
24	antiparallel
25	antiferro
26	anod
27	absorb

Figure C.16: Keywords list result



The 'Evaluation' dialog box is used to select evaluation metrics and input values. On the left, under the 'Evaluate:' label, there are five buttons: 'TF' (selected), 'IDF', 'TFxIDF', 'TDVM', and 'Cancel'. On the right, there are four input fields labeled 'Recall', 'Precision', 'Noise', and 'Effeciency' (note the spelling). Each input field is currently empty.

Figure C.17: ATEWB Evaluation dialog box.

19. Click "TF" button to evaluate TF technique, "IDF" to evaluate IDF technique, "TFxIDF" to evaluate TFxIDF technique and "TDVM" to evaluate TDVM technique. Click "Cancel" to exit.
20. To use a calculator, click "Calculator" in "Tools" menu, which calls the calculator tool in the OS if exists. *See figure C.18.*

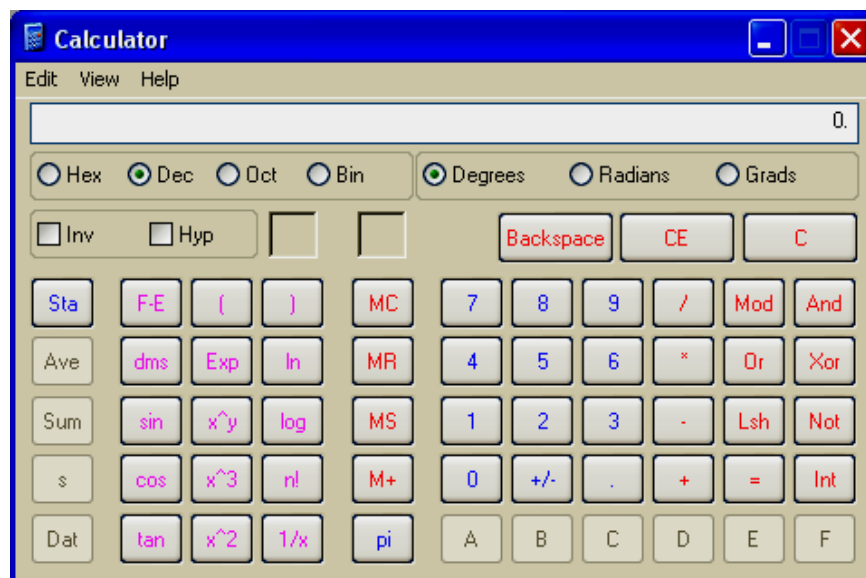


Figure C.18: Windows Calculator.

D. Appendix D

D.1 ATEWB JAVA and MYSQL Complete Code

Here we present the implementation of our developed tool (ATEWB). It includes the complete Java and MYSQL code of all statistical techniques, Porter's stemming, stop words removal, and the evaluation module. The soft copy of this code and executable JAR file are also included in the attached CD-ROM.