2007

REQUIREMENTS TRACING USING INFORMATION RETRIEVAL

Senthil Karthikeyan Sundaram
University of Kentucky, skart2@email.uky.edu

ABSTRACT OF DISSERTATION

Senthil Karthikeyan Sundaram

The Graduate School

University of Kentucky

2007

REQUIREMENTS TRACING USING INFORMATION RETRIEVAL

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Senthil Karthikeyan Sundaram

Lexington, Kentucky

Director: Dr. Jane Huffman Hayes, Associate Professor of Computer Science

Lexington, Kentucky

2007

ABSTRACT OF DISSERTATION

REQUIREMENTS TRACING USING INFORMATION RETRIEVAL

It is important to track how a requirement changes throughout the software lifecycle. Each requirement should be validated during and at the end of each phase of the software lifecycle. It is common to build traceability matrices to demonstrate that requirements are satisfied by the design. Traceability matrices are needed in various tasks in the software development process. Unfortunately, developers and designers do not always build traceability matrices or maintain traceability matrices to the proper level of detail. Therefore, traceability matrices are often built "after-the-fact."

The generation of traceability matrices is a time consuming, error prone, and mundane process. Most of the times, the traceability matrices are built manually. Consider the case where an analyst is tasked to trace a high level requirement document to a lower level requirement specification. The analyst may have to look through M x N elements, where M and N are the number of high and low level requirements, respectively. There are not many tools available to assist the analysts in tracing unstructured textual artifacts and the very few tools that are available require enormous pre-processing.

The prime objective of this work was to dynamically generate traceability links for unstructured textual artifacts using information retrieval (IR) methods. Given a user query and a document collection, IR methods identify all the documents that match the query. A closer observation of the requirements tracing process reveals the fact that it can be stated

as a recursive IR problem.

The main goals of this work were to solve the requirements traceability problem using IR methods and to improve the accuracy of the traceability links generated while best utilizing the analyst's time. This work looked into adopting different IR methods and using user feedback to improve the traceability links generated. It also applied wrinkles such as filtering to the original IR methods. It also analyzed using a voting mechanism to select the traceability links identified by different IR methods. Finally, the IR methods were evaluated using six datasets. The results showed that automating requirements tracing process using IR methods helped save analyst's time and generate good quality traceability matrices.

KEYWORDS: Requirements, Requirements Tracing, Requirements Traceability Matrix (RTM), Traceability Links Generation, Information Retrieval.

Senthil Karthikeyan Sundaram

June 07, 2007

REQUIREMENTS TRACING USING

INFORMATION RETRIEVAL

By

Senthil Karthikeyan Sundaram

Dr. Jane Huffman Hayes

Director of Dissertation

Dr. Grzegorz W. Wasilkowski

Director of Graduate Studies

June 07, 2007

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part requires also the consent of the Dean of The Graduate School of the University of Kentucky.

A library which borrows this dissertation for use by its patrons is expected to secure the signature of each user.

Name	Date

DISSERTATION

Senthil Karthikeyan Sundaram

The Graduate School

University of Kentucky

2007

REQUIREMENTS TRACING USING INFORMATION RETRIEVAL

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Senthil Karthikeyan Sundaram

Lexington, Kentucky

Director: Dr. Jane Huffman Hayes, Associate Professor of Computer Science

Lexington, Kentucky

2007

Acknowledgments

I would like to dedicate this work to my parents Mr. K. M. Sundaram and Mrs. S. Ponnammal. I am greateful to them for giving me all the opportunities and encouragement to pursue my dreams. I thank them for their love and support.

I would like to thank my advisor, Dr. Hayes for supporting me throughout my doctoral studies. She has been an inspiration and I greatly appreciate her help and guidance through my research, exams, and career choices. Especially, I am greateful to her for trusting me through the tough times during my doctoral studies.

I would like to thank Dr. Dekhtyar for sharing his knowledge and thoughts on Information Retrieval methods. His comments on my research approach and ideas have been extremely valuable. This work would not have been possible without the advice and support of both Dr. Hayes and Dr. Dekhtyar.

My thanks and appreciation also go out to my committee members, Dr. Singhal and Dr. Donohue for their support.

And to my friends Suzanne Anandappa and David Crossen, thanks for reviewing this dissertation report. I also want to thank my friends Jhonniya Johnson, Soundarraj Santhanaraman, and Sharmila Nageswaran for their encouragement. Special thanks to my boss, Mr. Christian Ecker for his persistent support.

Table of Contents

Acknowledgments . . . iii
List of Tables . . . vi
List of Figures . . . viii

Chapter 1 Introduction . . . 1

1.1 Problem Statement . . . 3
1.2 Research Thesis . . . 3
1.3 Overview of Dissertation . . . 4

Chapter 2 Background . . . 5

2.1 Requirements Tracing . . . 5
2.2 Information Retrieval (IR) . . . 11
2.2.1 Boolean Model . . . 12
2.2.2 Vector Model . . . 12
2.2.3 Probabilistic IR . . . 15
2.2.4 Latent Semantic Indexing . . . 19

Chapter 3 Related Work . . . 20

3.1 Requirements Tracing . . . 20
3.2 Information Retrieval in Requirements Tracing . . . 26

Chapter 4 Research Approach . . . 30

4.1 Keyword Extraction by χ^2 . . . 30
4.2 Keyword Extraction by idf . . . 32
4.3 Initial estimates for Probabilistic IR . . . 33
4.4 Vocabulary Base . . . 35
4.5 Relevance Feedback . . . 35
4.6 Voting Tool . . . 36
4.7 Automating Requirements Tracing . . . 37
4.8 RETRO . . . 41

Chapter 5 Experimental Setup and Validation . . . 50

5.1 Datasets . . . 50
5.2 Input Format and Candidate List Format . . . 52
5.3 Measures . . . 54
5.3.1 Overall Precision and Overall Recall . . . 54

5.3.2 Selectivity . . . 57
5.3.3 Average Precision and Average Recall . . . 57
5.3.4 Average Expected Precision . . . 57
5.3.5 F-measure . . . 58
5.3.6 Analysis Tool . . . 60
5.4 Experimental Setup . . . 60
5.4.1 Feedback . . . 62
5.4.2 Filter . . . 62
5.4.3 Thesaurus . . . 62
5.4.4 Vocabulary Base . . . 64
5.4.5 Weight Method . . . 64
5.4.6 Voting Tool . . . 64
5.4.7 Empirical Assessment . . . 65
5.4.8 Objectives . . . 68

Chapter 6 Results . . . 69

6.1 Analysis of IR methods . . . 71
6.1.1 Observations . . . 72
6.2 Feedback Analysis . . . 72
6.2.1 Statistical Analysis . . . 80
6.2.2 Observations . . . 81
6.3 Filter Analysis . . . 83
6.3.1 Statistical Analysis . . . 99
6.3.2 Observations . . . 102
6.4 Selectivity . . . 102
6.5 Weight Analysis . . . 102
6.5.1 Statistical Analysis . . . 113
6.6 Thesaurus Analysis . . . 113
6.7 Vocabulary Base Analysis . . . 118
6.8 Voting Tool Analysis . . . 130

Chapter 7 Conclusion and Future Work . . . 134

7.1 Conclusion . . . 134
7.2 Contributions . . . 135
7.3 Future Work . . . 136

Appendix A . . . 137

References . . . 153
Vita . . . 159

List of Tables

Table 2.1 RTM 6
Table 2.2 Requirements Tracing Process 10

Table 5.1 Datasets 52
Table 5.2 Information Retrieval Methods 61
Table 5.3 Empirical Study List 65

Table 6.1 Results Obtained by the Tf-idf, Tf-idf with Thesaurus, Keyword Extraction, IDF, LSI and Probabilistic Methods on Modis, CM1, and CM1 Subset1 73
Table 6.2 Results Obtained by the Tf-idf, Tf-idf with Thesaurus, Keyword Extraction, IDF, LSI, and Probabilistic IR Methods on CM1 Subset2, CM1 Subset3, and CM1 Subset4 74
Table 6.3 Modis - No Feedback vs. Feedback - Paired T-test 81
Table 6.4 CM1 - No Feedback vs. Feedback - Paired T-test 82
Table 6.5 Feedback Analysis, Tf-idf - Modis, CM1, and CM1 Subset1 84
Table 6.6 Results Obtained by the Tf-idf, Tf-idf with thesaurus, Keyword Extraction, IDF, LSI and Probabilistic IR Methods on MODIS, CM1, and CM1 Subset1 at Iteration 8 85
Table 6.7 Results Obtained by the Tf-idf, Tf-idf with thesaurus, Keyword Extraction, IDF, LSI, and Probabilistic IR Methods on CM1 Subset2, CM1 Subset3, and CM1 Subset4 at Iteration 8 86
Table 6.8 Modis - No Feedback vs. Feedback + Filter - Paired T-test 100
Table 6.9 CM1 - No Feedback vs. Feedback + Filter - Paired T-test 101
Table 6.10 Filter Analysis - Modis, CM1, and CM1 Subset1 103
Table 6.11 Filter Analysis - CM1 Subset2, CM1 Subset3, and CM1 Subset4 104
Table 6.12 Filter Analysis, Filter 0.1 - Modis, CM1, and CM1 Subset1 105
Table 6.13 Filter Analysis, Filter 0.1 - CM1 Subset2, CM1 Subset3, and CM1 Subset4 106
Table 6.14 CM1 - Recall - Weight Methods Comparison - Kruskal-Wallis 114
Table 6.15 CM1 - Precision - Weight Methods Comparison - Kruskal Wallis 115
Table 6.16 Weight Analysis - CM1 Subset2 Filter 0.1, CM1 Subset3 Filter 0.1, and CM1 Subset4 Filter 0.1 119
Table 6.17 Vocabulary Base Analysis - Modis, CM1, and CM1 Subset1 120
Table 6.18 Vocabulary Base Analysis - CM1 Subset2, CM1 Subset3, and CM1 Subset4 121

Table 6.19 Voting Tool Analysis - CM1 - Comparison of Recall/Precision values obtained by different methods and rules 131
Table 6.20 Voting Tool Analysis - CM1 Subset4 - Comparison of Recall/Precision values obtained by different methods and rules 131
Table 6.21 Voting Tool Analysis - CM1 Subset3 - Comparison of Recall/Precision values obtained by different methods and rules 132

List of Figures

Figure 2.1 Project Document Hierarchy 6
Figure 2.2 Unsatisfied Requirements 7
Figure 2.3 Unintended Functions 8
Figure 2.4 Forward Tracing 9
Figure 2.5 Backward Tracing 9
Figure 2.6 Thesaurus 15

Figure 4.1 Automated Tracing using IR 39
Figure 4.2 Forward Tracing as an IR Problem 40
Figure 4.3 Backward Tracing as an IR Problem 41
Figure 4.4 Retro 42
Figure 4.5 Retro - Starting a new Project 43
Figure 4.6 Retro View Mode 45
Figure 4.7 Retro Trace Mode 46
Figure 4.8 Retro Browse Mode 47
Figure 4.9 Retro Architecture 48

Figure 5.1 Modis 51
Figure 5.2 CM1 51
Figure 5.3 Input Format 53
Figure 5.4 DTD 54
Figure 5.5 Candidate Link List 55
Figure 5.6 Candidate Link List with Feedback Information 56
Figure 5.7 Average Expected Precision Calculation 59
Figure 5.8 Filter Example 63

Figure 6.1 Recall vs Precision - Modis 70
Figure 6.2 Recall vs Precision - CM1 70
Figure 6.3 Recall vs Precision - CM1 Subset1 71
Figure 6.4 Recall vs Precision - CM1 Subset2 72
Figure 6.5 Recall vs Precision - CM1 Subset 3 75
Figure 6.6 Recall vs Precision - CM1 Subset 4 75
Figure 6.7 Average Recall vs Average Precision - Modis 76
Figure 6.8 Average Recall vs Average Precision - CM1 76
Figure 6.9 Average Recall vs Average Precision - CM 1 Subset1 77

Figure 6.10 Average Recall vs Average Precision - CM 1 Subset2 77
Figure 6.11 Average Recall vs Average Precision - CM 1 Subset3 78
Figure 6.12 Average Recall vs Average Precision - CM 1 Subset4 78
Figure 6.13 Feedback Analysis - Recall/Precision - Modis, CM1 and CM1 Subset1 83
Figure 6.14 Feedback Analysis - Recall/Precision - CM1 Subset2, CM1 Subset3 and CM1 Subset4 87
Figure 6.15 Feedback Analysis - F-measure - Modis, CM1 and CM1 Subset1 . . . 87
Figure 6.16 Feedback Analysis - F-measure - CM1 Subset2, CM1 Subset3 and CM1 Subset4 88
Figure 6.17 Feedback Analysis - Average Recall/Average Precision - Modis, CM1 and CM1 Subset1 88
Figure 6.18 Feedback Analysis - Average Recall/Average Precision - CM1 Subset2, CM1 Subset3 and CM1 Subset4 89
Figure 6.19 Feedback Analysis - Average Expected Precision - Modis, CM1 and CM1 Subset1 90
Figure 6.20 Feedback Analysis - Average Expected Precision - CM1 Subset2, CM1 Subset3 and CM1 Subset4 91
Figure 6.21 Feedback Analysis - F-Measure vs. Selectivity - Modis, CM1, and CM1 Subset1 92
Figure 6.22 Feedback Analysis - F-Measure vs. Selectivity - CM1 Subset2, CM1 Subset3 and CM1 Subset4 93
Figure 6.23 Filter Analysis - Recall/Precision (a) Modis and (b) CM1 95
Figure 6.24 Filter Analysis - Recall/Precision (a) CM1 Subset1 and (b) CM1 Subset2 96
Figure 6.25 Filter Analysis - Recall/Precision (a) CM1 Subset3 and (b) CM1 Subset4 97
Figure 6.26 Filter Analysis - F-measure (a) Modis and (b) CM1 98
Figure 6.27 Filter Analysis - F-measure (a) CM1 Subset1 and (b) CM1 Subset2 . 107
Figure 6.28 Filter Analysis - F-measure (a) CM1 Subset3 and (b) CM1 Subset4 . 108
Figure 6.29 Filter Analysis - Average Recall/Average Precision (a) Modis and (d) CM1 109
Figure 6.30 Filter Analysis - Average Recall/Average Precision (a) CM1 Subset1 and (d) CM1 Subset2 110
Figure 6.31 Filter Analysis - Average Recall/Average Precision (a) CM1 Subset3 and (d) CM1 Subset4 111
Figure 6.32 Filter Analysis - Average Expected Precision (a) Modis and (b) CM1 112
Figure 6.33 Filter Analysis - Average Expected Precision (a) CM1 Subset1 and (b) CM1 Subset2 116
Figure 6.34 Filter Analysis - Average Expected Precision (a) CM1 Subset3 and (b) CM1 Subset4 117
Figure 6.35 Selectivity 118
Figure 6.36 Weight Analysis 122
Figure 6.37 Thesaurus Analysis 123
Figure 6.38 Vocabulary Base (a) Modis and (b) CM1 124
Figure 6.39 Vocabulary Base (a) CM1 Subset1 and (b) CM1 Subset2 125
Figure 6.40 Vocabulary Base (a) CM1 Subset3 and (b) CM1 Subset4 126
Figure 6.41 Voting Method - CM1 (a) With IDF and (b) Without IDF 127
Figure 6.42 Voting Method - CM1 Subset4 (a) With IDF and (b) Without IDF . 128
Figure 6.43 Voting Method - CM1 Subset3 (a) With IDF and (b) Without IDF . 129

Figure A.1 Recall vs. Precision - Tf-idf - Modis, CM1, and CM1 Subset1. 137
Figure A.2 Recall vs. Precision - Tf-idf - CM1 Subset2, CM1 Subset3 and CM1 Subset4. 138
Figure A.3 Average Recall vs. Average Precision - Tf-idf - Modis, CM1, and CM1 Subset1. 139
Figure A.4 Average Recall vs. Average Precision - Tf-idf - CM1 Subset2, CM1 Subset3 and CM1 Subset4. 140
Figure A.5 Filter Analysis - Modis - Recall vs. Precision (Tf-idf) 141
Figure A.6 Filter Analysis - CM1 - Recall vs. Precision (Tf-idf) 142
Figure A.7 Filter Analysis - CM1 Subset1 - Recall vs. Precision (Tf-idf) 143
Figure A.8 Filter Analysis - CM1 Subset2 - Recall vs. Precision (Tf-idf) 144
Figure A.9 Filter Analysis - CM1 Subset3 - Recall vs. Precision (Tf-idf) 145
Figure A.10 Filter Analysis - CM1 Subset4 - Recall vs. Precision (Tf-idf) 146
Figure A.11 Filter Analysis - Modis - Average Recall vs. Average Precision (Tf-idf) 147
Figure A.12 Filter Analysis - CM1 - Average Recall vs. Average Precision (Tf-idf) 148
Figure A.13 Filter Analysis - CM1 Subset1 - Average Recall vs. Average Precision (Tf-idf) 149
Figure A.14 Filter Analysis - CM1 Subset2 - Average Recall vs. Average Precision (Tf-idf) 150
Figure A.15 Filter Analysis - CM1 Subset3 - Average Recall vs. Average Precision (Tf-idf) 151
Figure A.16 Filter Analysis - CM1 Subset4 - Average Recall vs. Average Precision (Tf-idf) 152

Chapter 1

Introduction

It is important to track how a requirement changes throughout the software lifecycle. Each requirement should not only be validated during each phase of the software lifecycle, but also at the end of the software lifecycle. It is common to build traceability matrices to demonstrate that requirements are satisfied by the design. Traceability matrices are also used to verify that design and code satisfy the requirements. They are also used in risk analysis, impact analysis on proposed changes, criticality assessment, and test coverage analysis. Unfortunately, developers and designers do not always build traceability matrices or maintain traceability matrices to the proper level of detail. Therefore, traceability matrices are often built "after-the-fact" by Verification and Validation (V&V) analysts and Independent Verification and Validation (IV&V) analysts.

As mentioned in [30], the objective of Verification and Validation (V&V) and Independent Verification and Validation (IV&V) is to ensure that the right processes have been used to build the right system. Once built, traceability matrices should be maintained to keep track of all the changes made to their elements throughout the software lifecycle.

The most common tracing method currently used by V&V or IV&V analysts works as follows. Assume that the objective is to trace a high level requirement document (such as a System Specification) to a lower level requirement specification (such as a Software Requirement Specification). At first, the requirements are extracted from the documents. An analyst selects a high level requirement and compares it with every low level requirement. If the analyst finds them to be similar, the high-low level pair being compared is added to the link list. This process is repeated for all the high level requirements. The analyst has to look through M x N elements, where M is the number of high level requirements and N is the number of low level requirements. Even if there are only 25 high level requirements and 30 low level requirements, the analyst still has to make 750 comparisons. This process

can be tedious, mundane, and error-prone.

It is also common to use a keyword matching algorithm instead of manually going through each high-low level requirement pair. In this case, an analyst goes through each high level requirement and low level element and assigns keywords, respectively. At times, analysts may use word processing tools to assign keywords. Using a keyword matching technique, a list is built of low level elements that may potentially satisfy a given high level requirement. These are called candidate links.

Next, the candidate links generated manually or by the keyword matching technique are reviewed by the analyst to determine if they are true links or false positives. In order to evaluate candidate links, the analyst goes through each link and checks if the high level and the low level requirement under consideration are related. The evaluation is completely based on the analyst's judgment. Thus, it's prone to human error. Once the candidate links are evaluated, a report listing the high level requirements without a child and the low level requirements without a parent is generated.

From the previous paragraphs, It is clear that the current approaches to after-the-fact tracing have many disadvantages. In the case of manual tracing, the analyst has to go through every high-low level requirement pair. It is a time-consuming and error-prone process. If keyword matching techniques are used, the analyst does not have to compare the requirement elements, but the analyst needs to assign keywords to each high level and low level requirement. In either case, in order to evaluate candidate links, the analyst might have to perform interactive searches and use word processing tools. The keyword matching algorithms are not efficient in returning all the correct links and they tend to return many false positives. They do not provide a means to keep track of the changes made to the requirement elements. In order to ensure requirement completion and to facilitate change impact assessment, risk analysis, and criticality assessment, a method for easy after-the-fact requirements tracing is needed.

It can be seen that the generation of candidate links is the time consuming, but critical, part of the tracing process. If candidate links can be generated automatically without returning a significant number of false positives, much time and effort can be saved. Even with automated candidate links generation, the analyst still has certain critical responsibilities such as candidate link evaluation, making decisions on whether or not to

look for more links, identifying if a requirement has been completely satisfied and deciding if the tracing process is complete. Hayes et al. [27] mention that "the key to successful automation lies not in removing the human decision-maker from the loop, but rather, in introducing an automated agent that takes care of the mundane, time-consuming parts of the process and allows the analyst to concentrate on the parts that really require human decision-making."

A close observation of the candidate link generation process reveals that it can be stated as an Information Retrieval (IR) problem. In a nutshell, Information Retrieval methods identify all the documents in a given document collection that match a given query. If each high level requirement is considered as a query and the set of all low level requirements is considered as a document collection, candidate link generation becomes an Information Retrieval problem. There are already well-established methods to solve Information Retrieval problems. There are well-established measures to evaluate the performance of the IR methods and the quality of the results generated. Hence, Information Retrieval methods can be used to automate and improve the quality of the tracing process. However, these methods may not be used directly to generate candidate links due to the characteristics of the tracing process and the size of the document collection used in tracing. Even with automated candidate link generation, in order to complete the trace, the analyst needs to go through each link generated to determine if it is indeed a link. This process is called candidate link evaluation.

1.1 Problem Statement

The generation of traceability links is a time consuming, error prone, and mundane process. There is a lack of tools for generating traceability matrices for unstructured textual artifacts. There is also a lack of standard measures to assess the quality of the traceability matrices. The very few tools that are available require enormous pre-processing.

1.2 Research Thesis

The problem of dynamic generation of traceability links for unstructured textual artifacts can be addressed using information retrieval methods with an emphasis on improving the accuracy of the traceability links generated while best utilizing the analyst's time.

The objective of this work was to automate the generation of candidate link lists resulting in an insignificant number of false positives by adapting various IR algorithms. We compared different IR methods to identify which ones perform better under a particular scenario. Sometimes, the performance of certain IR methods varied depending on the size of the datasets. We also analyzed the traditional measures used in the IR world in assessing the quality of the candidate links generated. We incorporated feedback methods to further refine the candidate links generated using as minimal input from the analyst as possible.

1.3 Overview of Dissertation

The report is organized as follows. Chapter 2 discusses requirements tracing and Information Retrieval methods in detail. Related work in requirements tracing is presented in Chapter 3. Chapter 4 explains the research approach. Chapter 5 discusses the validation methods and datasets to be used in assessing this work. Chapter 6 discusses all the results obtained. Chapter 7 presents the conclusion and future work.

Chapter 2

Background

2.1 Requirements Tracing

Figure 2.1 shows a project document hierarchy for a typical large software project. In the given hierarchy, how do we make sure that the Software Design Specification satisfies all the Product Requirements? The Software Design Specification can be traced to the Software Requirement Specification to ensure that the Software Design Specification satisfies the Software Requirement Specification. In turn, the Software Requirement Specification can be traced to the Product Requirement. The relation "satisfies" is transitive, and, hence, both the traces mentioned above will determine if the Software Design Specification satisfies all the product requirements.

IEEE standard for software verification and validation plans [1] defines verification as "confirmation by examination and provisions of objective evidence that specified requirements have been fulfilled." Further, it defines validation as "confirmation by examination and provisions of objective evidence that the particular requirements for a specific intended use are fulfilled." Hence, V&V and IV&V analysts ensure that the right product is built using the right process. Building the requirements traceability matrix (RTM) is the backbone of IV&V and V&V.

A sample RTM is shown in Table 2.1. An RTM is a table mapping any pair of software artifacts.

Requirements tracing is defined as "the ability to describe and follow the life of a requirement, in both a forward and a backward direction, through the whole systems life cycle [35]." The prime motive of requirements tracing is to ensure that all the requirements are being satisfied by the system being built and also to perform impact analysis on proposed

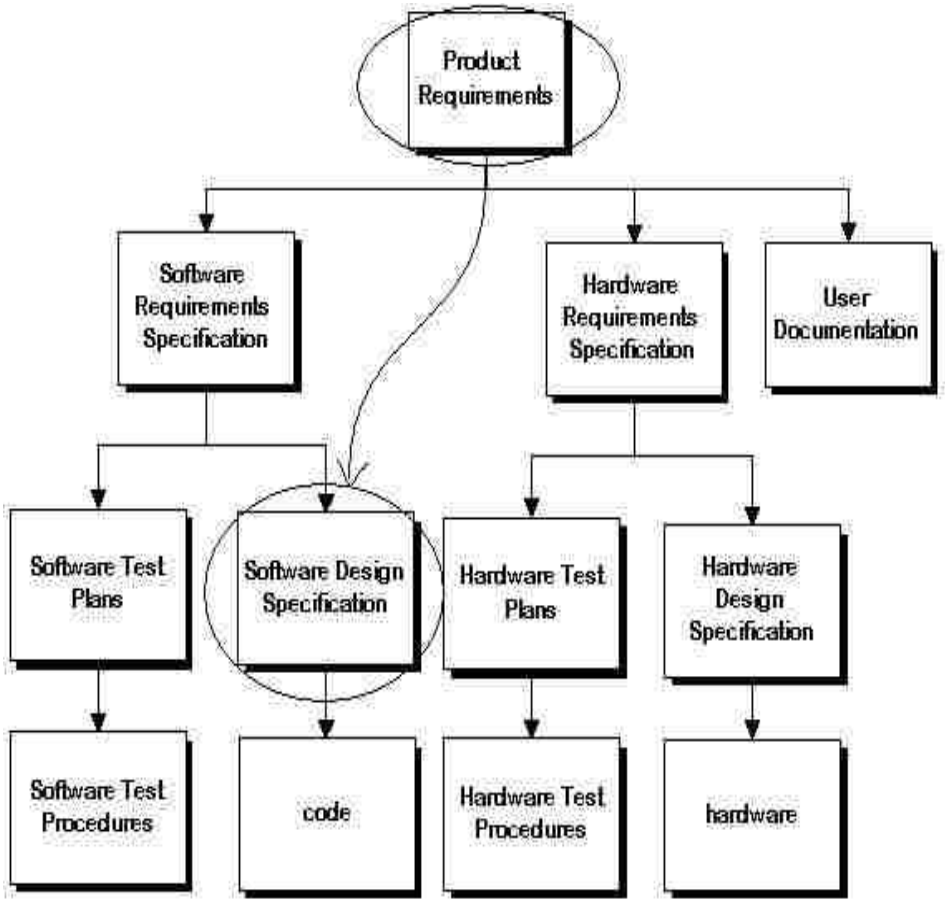


Figure 2.1: Project Document Hierarchy

Design Elements	**Requirement Elements**				
	R1	R2	R3	R4	R5
D1	X				
D2			X	X	
D3				X	
D4					X
D5	X				X

Table 2.1: RTM

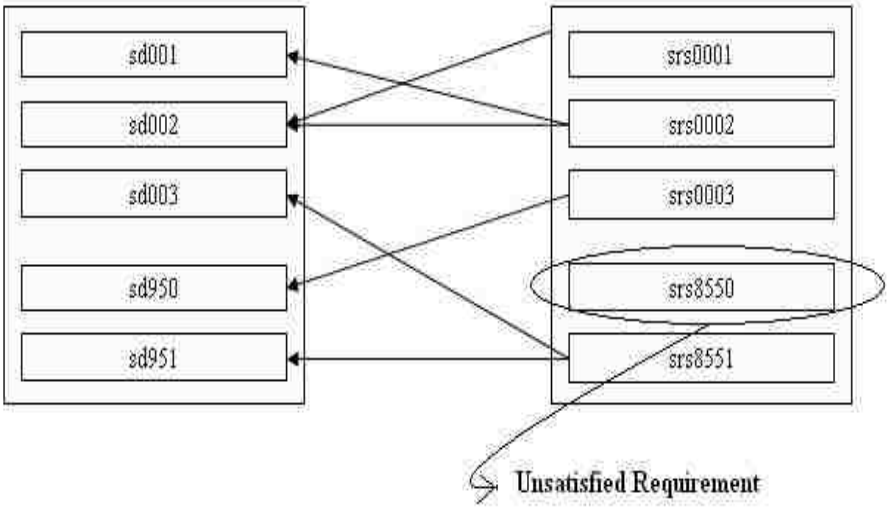


Figure 2.2: Unsatisfied Requirements

changes. At the end of a requirements tracing process, a requirements traceability matrix (RTM) is generated. It is necessary that the RTM reflect all the changes made to the elements represented in it and the traceability information is updated accordingly. Hence, the RTM evolves throughout the software lifecycle.

Many people in the software industry need to generate traceability links. For example, an IV&V analyst should generate an RTM for a set of artifacts such as the Software Design Specification and the Software Requirement Specification. A V&V analyst might trace two API specifications to each other to make sure that they are consistent. A test engineer might have to identify traceability links between test cases and requirements. A software designer might have to make sure that the design is consistent with proposed code changes. Having an efficient requirement traceability process will make life easier for many software professionals.

At every stage in the software lifecycle, each element in the high level artifact must have at least one child element in the low level artifact and each element in the low level artifact must have at least one parent in the high level artifact. Figure 2.2 shows the scenario where one of the requirements is not satisfied by any design element. This might result in the system being built incompletely. Whereas, Figure 2.3 shows the scenario where one of the design elements does not correspond to any requirement. In this case, effort and time

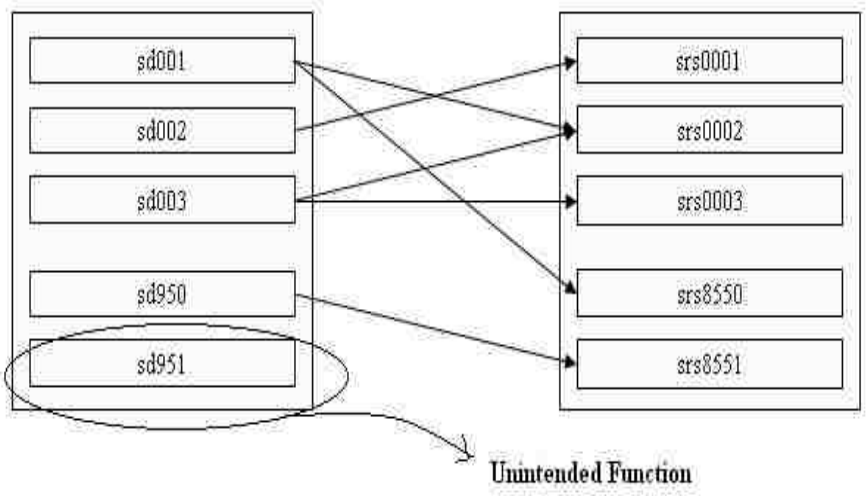


Figure 2.3: Unintended Functions

might be wasted on implementing an unintended functionality. In order to identify the above mentioned scenarios, tracing should be performed in both directions, forward and backward.

Forward tracing can be defined as tracing high level project artifacts to low level project artifacts. Whereas, backward tracing can be defined as tracing low level project artifacts to high level project artifacts. Figure 2.4 and Figure 2.5 show examples of forward and backward tracing, respectively. In the forward tracing example, requirements are traced to design specifications. In Figure 2.5, design specifications are traced to requirements.

In order to understand the requirements tracing process better, let us look at the responsibilities of an analyst who has been tasked to perform a requirements trace. Let us assume that the analyst has been asked to perform a trace between two sets of requirements documents. Without loss of generality, let us call one set of requirements high level and the other low level. Consider the case that high level requirements should be traced to low level requirements.

The process of requirements tracing is described in Table 2.2 [30]. The analyst first goes through both high and low level documents to identify the requirement elements and assign identifiers to them. The analyst goes through each high level requirement and

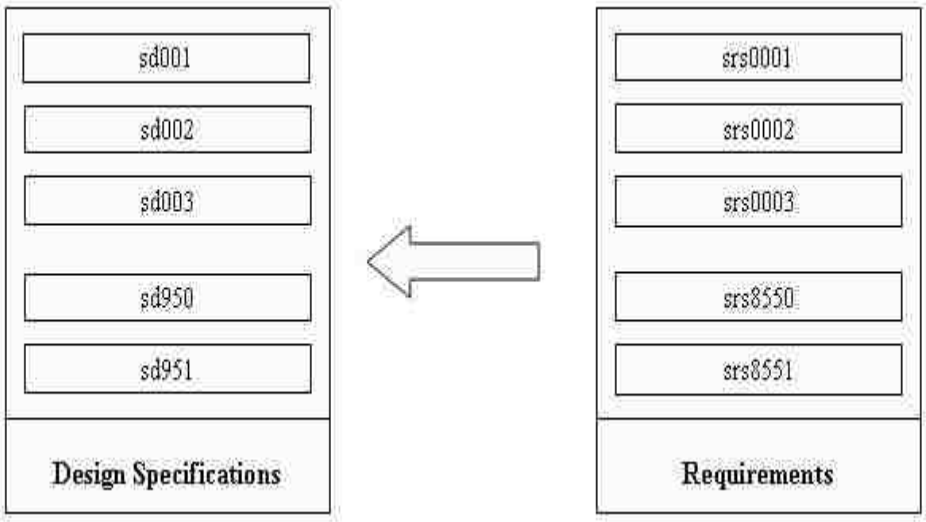


Figure 2.4: Forward Tracing

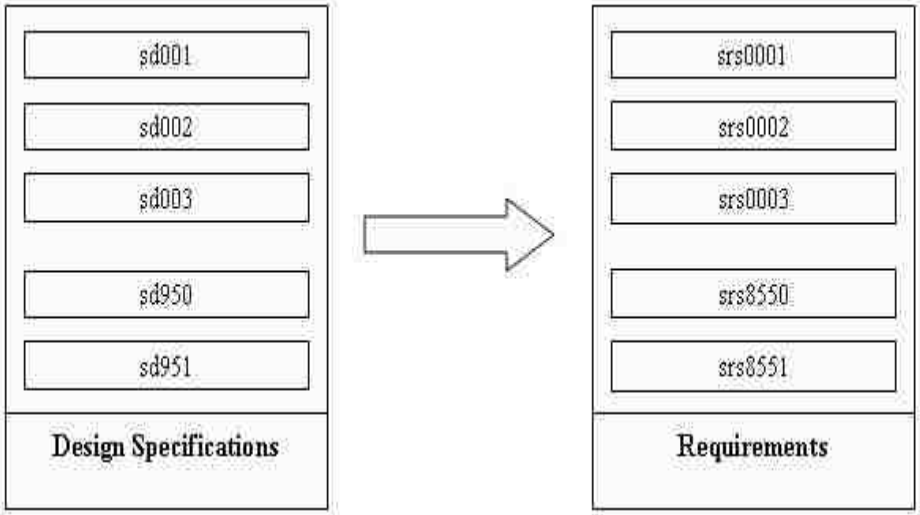


Figure 2.5: Backward Tracing

Step	Task
1	identify each requirement
2	assign a unique identifier to each requirement
3	for each high level requirement, locate all matching low level requirements
4	for each low level requirement, locate a parent element in the collection of high level requirements
5	determine if each high level requirement has been completely satisfied
6	prepare a report that presents the traceability matrix
7	prepare a summary report that expresses the level of traceability of the document pair

Table 2.2: Requirements Tracing Process

identifies all the matching low level requirements. Similarly, for each low level requirement, a parent element should be found in the collection of high level requirements. A requirements traceability matrix, along with a summary report, is built to summarize the traceability links.

Let us examine the above mentioned steps and see if any of those can be automated. A tool can be used to identify requirements and assign identifiers to them. Similarly, generation of a requirements traceability matrix and traceability summary reports can also be automated. There are a number of tools available, such as SuperTrace Plus (STP) [23, 37] that address these activities. That leaves steps 3, 4, and 5 in Table 2.2 to be addressed. In fact, the remaining steps are of greater importance to researchers and practitioners as these steps require the analyst to compare each high level requirement with every low level requirement. As the number of high level and low level requirements grow, the number of traceability links that an analyst should consider increases significantly.

The primary goal of this work is to study the ways in which requirements tracing can be automated. We can see that steps 3, 4, and 5 are the primary candidates for automation. However, it should be noted that even with automation, the analyst still has to evaluate the candidate links; make decisions on whether or not all the candidate links have been found; make decisions on whether or not a requirement has been completely satisfied by the links found; and decide if the traceability matrix generated is acceptable. So, it is obvious that the requirements tracing process cannot be completely automated and that analysts still have critical responsibilities.

In this work, we focus on the dynamic generation of candidate links using an automated process. We analyze various Information Retrieval methods and adopt them to generate candidate links. An ideal automated process should only generate true candidate links and no false positives. Even though we strive to attain perfection with automation, it is highly likely that there will be some false positives among the candidate links generated. It is also possible that the automated process might not find all the true links.

This work also focuses on using the analyst's feedback on the candidate links generated in order to improve the result. In other words, the analyst will go thorough some of the candidate links generated and mark them as "Link" or "Not A Link" based on his/her judgment. This information will be used to look for more candidate links that are similar to the ones marked as "Link" and to discard the candidate links similar to the ones marked as "Not A Link." The feedback process can be repeated until the analyst feels that all the possible candidate links have been found.

2.2 Information Retrieval (IR)

Information Retrieval is a field that studies the problem of identifying relevant documents in a document collection that match given queries. The exponential growth of the World Wide Web (WWW) lead to enormous growth of information and document repositories and made the obtaining of desired information from the WWW has become tougher. Baeza-Yates et al. [7] and Frakes et al. [19] give an excellent introduction to information retrieval and explain how to develop an IR system.

The most-studied and commonly used keyword-based IR methods identify important keywords for each document in the document collection. Similarly, keywords are identified for the queries and these keywords are compared with the ones assigned for the documents to identify matches. In most keyword-based methods, the relevance of a document to a query is expressed in terms of a similarity measure. The result is list of documents in descending order of their relevance to the query. The quality of an IR method depends on the number of possible matches found and the percentage of false positives in the result. *Precision* and *recall* are commonly used to measure the quality of the result. Recall is defined as the ratio of the number of links returned by the IR method to the total number of possible links.

$$Recall = \frac{Number\,of\,matches\,found\,by\,the\,IR\,method}{Total\,number\,of\,possible\,matches} \tag{2.1}$$

Precision is computed as the fraction of the relevant documents in the list of all documents returned by the IR method.

$$Precision = \frac{Number\,of\,true\,links\,found}{Total\,number\,of\,links\,returned\,by\,the\,IR\,method} \tag{2.2}$$

For a query, given the candidate links, it is easy to measure precision. However, in order to measure recall, one needs knowledge of the entire document collection. For example, measuring recall for the result of a web search is a daunting task. An ideal IR method should produce as high precision and recall as possible.

There is a wide array of keyword-based IR models. The ones listed below are very famous:

- Boolean model
- Vector space model
- Vector space model with thesaurus
- Probabilistic model, and
- Latent Semantic Indexing.

Let us discuss these methods in detail.

2.2.1 Boolean Model

The Boolean model is one of the simplest models. It is based on set theory and Boolean algebra. Each document D in a document collection contains a set of keywords. Each keyword has a weight of 1 if it is present in the document or 0 if it is not present. Hence, all the keywords are equally important. A query Q is represented as a Boolean expression containing terms and Boolean operators. A document D will be termed as relevant to Q only if it satisfies Q's Boolean expression. Though the Boolean model is easy to implement, it only identifies exact matches. The Boolean model will not identify partial matches.

2.2.2 Vector Model

The vector model is an extension of the Boolean model. The vector model assigns a *weight* to each keyword that signifies its importance to the document collection. Equation 2.3 shows how the weight is calculated. At first, important keywords are extracted from the

documents. Each keyword is assigned a weight factor and, finally, documents are ranked with respect to the query based on their similarity.

Tf-idf

We now consider a standard vector space model in detail. Each document **d** in the document collection **D** is represented as a vector d=$\{w_1,w_2...w_N\}$, where N is the total number of terms in the vocabulary of the document collection and w_i is the weight of the ith term. The weight is calculated as follows:

$$w_i = tf_i(d) \cdot idf_i \tag{2.3}$$

where tf_i(d) is the *term frequency* of the ith keyword in document d and idf_i is the *inverse document frequency* of the ith term in the document collection. Term frequency is the normalized number of occurrences of the keyword in the document collection. Inverse document frequency of the ith term is calculated as:

$$idf_i = log_2 \left(\frac{n}{df_i} \right) \tag{2.4}$$

where n is the total number of documents in the document collection and df_i is the document frequency of term k_i, i.e., the number of documents in which k_i is found. The importance of a keyword is decided based on how often the keyword occurs in a document and how many documents contain that particular keyword. If a term occurs more often in a document, it is likely to be more important to the document. Also, if a term occurs more commonly in the document collection, it is less discriminating. Similarly, the user query is also converted into a vector $q = (q_1, \ldots, q_N)$. The same formula is used to compute the weight for keywords in the given query.

Given a query vector $q = (q_1, \ldots, q_N)$ and a document vector $d = (w_1, \ldots, w_N)$, the similarity between them is computed as the cosine of the angle between q and d in N-dimensional space:

$$sim(q,d) = cos(q,d) = \frac{\sum_{i=1}^{N} q_i \cdot w_i}{\sqrt{\sum_{i=1}^{N} q_i^2 \cdot \sum_{i=1}^{N} w_i^2}} \tag{2.5}$$

The vector space model can be extended by modifying the computation of term weights in the document and query and the computation of similarity between the document and query vectors. In this work, we used two more weighting schemes in our experiments. One

of the weighting schemes used was proposed within the *Okapi* [46] information retrieval system. The following is the formula used in the Okapi system:

$$w_i = \frac{tf_i(d)}{0.5 + 1.5\frac{dl}{avg\_dl} + tf} log(\frac{n - df_i + 0.5}{df_i + 0.5}) \tag{2.6}$$

where $tf_i(d)$ and df_i are term frequency and document frequency of the ith term, n is the total number of elements, dl is the length of current element, and avg_dl is the average length of an element in the collection.

The other weighting scheme used is linear threshold unit (LTU) [33]. The LTU formula is:

$$w_i = \frac{(log(tf_i(d)) + 1)log(\frac{n}{df_i})}{0.8 + 0.2\frac{dl}{avg\_dl}}. \tag{2.7}$$

Tf-idf with thesaurus

The classical vector space model compares only the individual occurrences of keywords in the documents and the queries. However, it is quite possible that a query may not have an exact matching term to a keyword in the document, but it might have a synonymous term. In this case, the classical vector space model will fail to find the match. For example, the keyword *error* will not match the term *fault* but these terms may be very relevant. A thesaurus can be used to provide information about the relationship between different terms.

The vector space model can be extended by using a thesaurus to account for the presence of synonyms, antonyms, abbreviations, subcategories, etc. The thesaurus can be incorporated in the vector space model in many different ways, based on its type. As shown in Figure 2.6, a simple thesaurus T is a set of triplets <t_i,t_j,s>, where t_i and t_j are matching thesaurus terms and s is the similarity coefficient between them (e.g., <error, fault, 0.5>). Thesaurus terms can either be single keywords or key phrases (i.e., sequences of two or more keywords). The similarity (relevance) formula is modified to account for the thesaurus matches as follows:

$$sim(q,d) = cos(q,d) = \frac{\sum_{i=1}^{N} q_i \cdot w_i + \sum_{<k_i,k_j,\alpha_i j>\in T}(w_i \cdot q_j + w_j \cdot q_i)}{\sqrt{\sum_{i=1}^{N} q_i^2 \cdot \sum_{i=1}^{N} w_i^2}}. \tag{2.8}$$

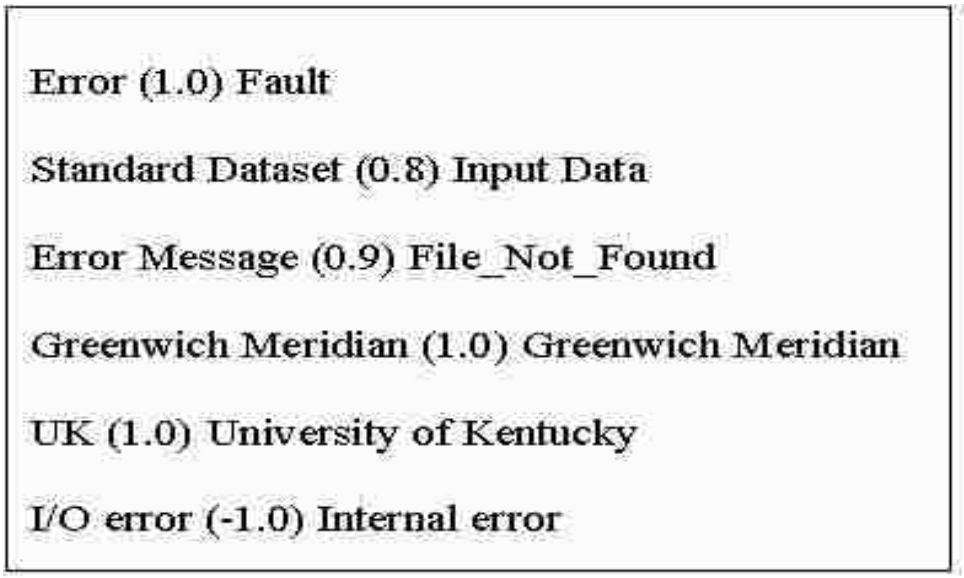


Figure 2.6: Thesaurus

2.2.3 Probabilistic IR

The probabilistic IR model matches documents and queries based on their probability of relevance, which is measured using the number of occurrences of terms in documents and queries. The Probability Rank Principle (PRP) [45] states that the probabilistic methods are effective when an estimate of the probability of their relevance to the given query is used to rank the matching documents. Similar to the Vector Space Model, the probabilistic methods assign a weight to each query term which corresponds to the probability of that term retrieving a matching document for the given query. Hence, the probabilities of all the query terms contribute to the probability of retrieving a matching document.

Binary Independence Model

The Binary Independence Model (BIM) [47, 44] estimates the odds ratio of a document and a query being a link versus not being a link. The quality of the results returned by this method for small datasets largely depends on the initial estimates of two quantities: the probability of a true link containing a specific keyword, and the probability of a false positive link containing a specific keyword.

Each document **d** in the document collection **D** is represented as a binary vector d = $\{d_1,d_2,\ldots,d_N\}$, where N is the total number of terms in the vocabulary of the document

collection, $d_i = 1$ if term i is present in the vector, and $d_i = 0$ if term i is not present in the vector. Similarly, the user query is also converted into a binary vector $q = (q_1, \ldots, q_N)$.

BIM will assign to each document **d** in the document collection **D** a probability of relevance (similarity) value which is computed as follows:

$$sim(q, d) = \frac{P(d\, is\, relevant\, to\, q)}{P(d\, is\, not\, relevant\, to\, q)}. \tag{2.9}$$

Let q be the given query, R be the set of documents that are relevant to the given query, and NR be the set of documents that are not relevant. Let $P(R|d)$ be the probability that the document d is relevant to q and $P(NR|d)$ be the probability that the document d is not relevant to the query q. Now, the similarity value can be calculated as follows:

$$sim(q, d) = \frac{P(R|d)}{P(NR|d)}. \tag{2.10}$$

The conditional probability for two events X and Y can be obtained by the Bayes' theorem as follows:

$$P(X|Y) = \frac{P(Y|X) \cdot P(X)}{P(Y)} \tag{2.11}$$

where P(X) is the prior probability of X, P(Y) is the prior probability of Y, $P(X|Y)$ is the conditional probability of X given Y, and $P(Y|X)$ is the conditional probability of Y given X.

Applying Bayes' theorem to the equation (2.10),

$$sim(q, d) = \frac{P(d|R) \cdot P(R)}{P(d|NR) \cdot P(NR)}, \tag{2.12}$$

where $P(d|R)$ is the probability of randomly selecting a document d from the set R of relevant documents, $P(d|NR)$ is the probability of randomly selecting a document d from the set NR of non-relevant documents, P(R) is the probability of randomly selecting a relevant document from R and NR, and P(NR) is the probability of randomly selecting a non-relevant document from R and NR. It can be seen that P(R) and P(NR) are the same for all the documents in the document collection. Considering P(R) and P(NR) as contstants, we can re-write the similarity formula as follows:

$$sim(q, d) \sim \frac{P(d|R)}{P(d|NR)}. \tag{2.13}$$

In the equation (2.13), the right hand side can be estimated as follows:

$$\frac{P(d|R)}{P(d|NR)} = \prod_{i=1}^{N} \frac{P(d_i|R)}{P(d_i|NR)}. \tag{2.14}$$

$$sim(q,d) \sim \prod_{i=1}^{N} \frac{P(d_i|R)}{P(d_i|NR)}. \tag{2.15}$$

Since $d_i = 0$ or 1,

$$sim(q,d) \sim \prod_{i:d_i=1}^{N} \frac{P(d_i=1|R)}{P(d_i=1|NR)} \cdot \prod_{i:d_i=0}^{N} \frac{P(d_i=0|R)}{P(d_i=0|NR)}. \tag{2.16}$$

Let $p_i = \mathrm{P}(d_i = 1|R)$ and $u_i = \mathrm{P}(d_i = 1|NR)$. Considering the fact that $p_i + u_i = 1$, the equation (2.16) can be written as:

$$sim(q,d) \sim \prod_{i:d_i=1}^{N} \frac{p_i}{u_i} \cdot \prod_{i:d_i=0}^{N} \frac{1-pi}{1-u_i}. \tag{2.17}$$

$$sim(q,d) \sim \prod_{i:d_i=1,q_i=1}^{N} \frac{p_i \cdot (1-u_i)}{(1-p_i) \cdot u_i}. \tag{2.18}$$

Taking the logarithm:

$$sim(q,d) \sim \sum_{i:d_i=1,q_i=1}^{N} \log \frac{p_i}{1-p_i} + \log \frac{1-u_i}{u_i}. \tag{2.19}$$

R and NR are not known in the beginning. Hence, there should be a way to compute p_i and u_i. One such approach is stated in [7]. The approach assumes that p_i is a constant for all the index terms. It also assumes that "the distribution of index terms among non-relevant documents can be approximated by the distribution of index terms among all the documents in the collection [7]." Hence,

$$p_i = 0.5 \tag{2.20}$$

and

$$u_i = \frac{n_i}{N} \tag{2.21}$$

where n is the number of documents containing the index term d_i and N is the total number of doucments in the collection.

Imroving on the initial estimates

The above mentioned process can be used to generate the list of documents that match a given query. The initial list generated can be improved by adjusting the initial estimates of p_i and u_i. Let us look into the approach mentioned in [7] for improving the initial estimates.

Let M be the set of documents initially identified as matches. Let M_i be the subset of M that includes all the documents that contain the term index d_i. p_i can be approximated by the distibution of index term d_i among the documents retrieved initially and u_i can be approximated by considering that all the non-retrieved documents are not relevant [7]. Hence, p_i and u_i can be recalculated as follows:

$$p_i = \frac{M_i}{M}, \tag{2.22}$$

and

$$u_i = \frac{n_i - M_i}{N - M}. \tag{2.23}$$

The above mentinoed process can be applied without any human assistance. Instead, we can also use analyst feedback to prune the initial list. In this case, the pruned list will be considered as M. The analyst feedback will be explained in later chapters. This process can be repeated as many times as needed.

Note that when Mi and M have very small values, computing p_i and u_i can be problematic. This problem can be overcome in many ways. A constant adjustment factor can be added to the equations (2.22) and (2.23) [7] as follows:

$$p_i = \frac{M_i + 0.5}{M + 1}, \tag{2.24}$$

and

$$u_i = \frac{n_i - M_i + 0.5}{N - M + 1}. \tag{2.25}$$

Another way to avoid this problem is to use the fraction n_i/N as the adjustment [7]:

$$p_i = \frac{M_i + \frac{n_i}{N}}{M + 1} \tag{2.26}$$

and

$$u_i = \frac{n_i - M_i + \frac{n_i}{N}}{N - M + 1}. \tag{2.27}$$

2.2.4 Latent Semantic Indexing

Vector space model and probabilistic model match documents and queries based on the occurence of keywords. It could be possible that a pair of documents may be discussing the same concept, but still not share any common keywords. Conversely, a pair of unrelated documents may be sharing many keywords. In the first case, the matching documents may not be retrieved. In the later case, false positives will be generated.

Latent Semantic Indexing (LSI) [20] matches the documents based on their concepts rather than keywords. Similar to vector space model, LSI also builds a term-by-document matrix to represent each document in the document collection. LSI uses a dimension-reduction technique called Singular Value Decomposition (SVD) to reduce the term-by-document matrix into a lower dimension space of concepts. Hence, LSI is capable of identifying matching documents that do not share any common keywords, provided that they both share the same concepts.

The original matrix is transformed into a product of two orthogonal matrices and a diagonal matrix of Eigenvalues. It can be mathematically proved that any matrix can be decomposed as mentioned above. A smaller matrix is obtained as an approximation of the original matrix by considering only the top few Eigenvalues. Rows of the matrix can be compared to each other using the cosine similarity described above.

For example, if L is a document-by-term weight matrix of dimension $A \times B$, its SVD is written as $L = TSD'$, where T is a matrix with orthogonal rows, D' is a matrix with orthogonal columns, and S is a diagonal matrix of Eigenvalues of L. We can trim the list of Eigenvalues of L from $rank(L)$ to a smaller number k and obtain a decomposition $L_k = TS_kD'$, where S_k is the diagonal matrix of size kXk with the k largest Eigenvalues of L on the diagonal. Rows of the matrix TS_k^2D' can be compared to each other using the cosine similarity as defined above. Use of the matrix TS_k^2D' instead of the original matrix L reduces the dimensionality of the document vectors from B to k. Identifying the ideal value for k is of research interest. In this work, we will experiment with different values for k to obtain the best possible results.

Chapter 3

Related Work

This section is divided into two sub-sections. The first section looks into some very early research in requirements tracing. The second section discusses the application of information retrieval methods to solve requirements tracing problems.

3.1 Requirements Tracing

This section provides an overview of the research on requirements analysis and traceability in chronological order. Some of the research discussed here was conducted over a decade ago and pre-dates the application of information retrieval methods to tracing and the measures currently used to assess the accuracy of tracing methods. This section presents traceability research in historical context in order to give an idea of how research on automating traceability emerged.

In 1978, Pierce [38] designed a tool to facilitate requirements analysis. The main purpose of the tool was to simplify system verification, however, it also helped in recording changes in software requirements and estimating schedule and cost impact of those changes. The tool used a requirements database to store and maintain requirements. The database contained all the requirements and the documents produced throughout the software life-cycle. It supported three types of matching, namely matching high level requirements with their low level requirements, matching requirements of the same level, and matching requirements to code. Thus, it provided a detailed mapping of each requirement to design and code. The tool was originally written as a batch program in ANSI Standard FORTRAN.

Hayes et al. [23] built a front-end for a requirements tracing tool called the Software Automated Verification and Validation Analysis System (SAVVAS). The name of their front-end tool was SFEP, for SAVVAS Front-End Processor. SAVVAS used an Ingres relational database to store the requirements and SFEP was written in Pascal. SFEP extracted

the text from the requirements stored in the database of SAVVAS. It assigned keywords to each requirement using link words such as "shall," "will," etc. It matched the requirements based on keyword similarities. SFEP considered two requirements to be relevant only when their keyword similarity was above a threshold limit. Later, the tool was re-written using Microsoft Access and Visual Basic. The SFEP tool was continually enhanced and it is still being used in several projects. Mundie and Hallsworth [37] describe the SFEP tool in detail.

Brouse [9] examined current requirement engineering approaches and developed a process to address the shortcomings in the commonly used approaches. Brouse came up with a domain-independent multimedia requirements traceability to support a requirements elicitation and identification process. She also performed a case study in the transportation domain and demonstrated the applicability of the method. Brouse's model supported the assignment of non-domain specific and domain specific attributes for all the requirements.

Ramesh and Dhar [42] suggest that capturing the history of the design decisions in a structured manner might help designers, maintenance personnel, project managers, and executives. They came up with a conceptual model called REMAP (representation and maintenance of process knowledge) to capture the process knowledge and its effects on the decisions made during the requirements engineering process.

Casotto [10] examined the run-time tracing design activity using requirement cards organized into linear hierarchical stacks. Her approach also supported retracing.

Gotel and Finkelstein [21] analyzed the requirements traceability problem based on empirical studies. The empirical exercises involved more than 100 practitioners whose work areas covered various aspects of software engineering. The authors surveyed the literature and reviewed various commercial and research tools available for requirements tracing. They also used focus groups and questionnaires in understanding the problem and its nature.

They discussed their findings on the support available for requirements traceability. According to the authors, the lack of common definition of "requirements traceability" was one of the major issues. The authors believed that everyone seemed to have their own understanding of requirements traceability. They also believed that there was no consensus among the researchers on the main cause of the requirements traceability problem. The authors provided a framework explaining the multi-faceted nature of the requirements traceability problem. They proposed a definition for requirements traceability that reflected

most of the commonly used definitions for requirements traceability. They also introduced the concept of pre-requirements specification (pre-RS) traceability and post-requirements specification (post-RS) traceability.

"Pre-RS traceability is concerned with those aspects of a requirement's life prior to its inclusion in the RS (requirement production) [21]."

"Post-RS traceability is concerned with those aspects of a requirement's life that result from its inclusion in the RS (requirement deployment) [21]."

The authors argued that most of the existing support for requirements traceability covered post-RS traceability and there was hardly any support for pre-RS traceability. They emphasized the need for improved pre-RS traceability. They listed the problems confronting pre-RS traceability from the providers as well as the end users' view point. They explained how increasing awareness of information, obtaining and recording information, organizing and maintaining information, and accessing and presenting information can all improve pre-RS traceability. In summary, Gotel and Finkelstein strongly believed that the solution to the requirements traceability problem existed in improving the pre-RS traceability.

Watkins and Neal [54] explained the why and how of requirements tracing. They strongly emphasized the importance of requirements tracing. They explained in detail about the basic concepts of requirements tracing such as forward tracing, backward tracing, vertical tracing, and horizontal tracing. Bohner [8] analyzed change impact analysis using graphing techniques that could be used in maintaining an RTM.

Anezin [4] proposed a process and the methods to assist forward and backward tracing. She also developed a prototype incorporating the forward tracing mechanisms that she identified in her research. Among the mechanisms used for forward tracing, mechanisms that can be used for backward tracing were identified and incorporated into the prototype. Generally, high/system level requirements were decomposed into detailed low level requirements. The changes made to high level requirements had to be reflected in the low level requirements and vice-versa. Anezin also tried to analyze mechanisms to support the successful decomposition and composition of requirements. The main hypothesis of Anezin's dissertation was to identify what an analyst should know about requirements in order to trace them. An experiment was conducted to review existing requirements in order to understand the aspects of the requirements. Based on the similarities found, the author came

up with a strategy to compare and match requirements. A knowledge base was established and used in identifying similarities between requirements.

Pohl [40] came up with an approach to trace requirements to their origins called pre-traceability. He built a requirements engineering environment called PRO-ART that supported pre-traceability. PRO-ART was based on a three-dimensional framework for requirements engineering. PRO-ART also enabled selective trace retrieval and automated trace capture.

Pinheiro and Goguen [39] built an object-oriented tool for tracing requirements. TOOR (for Traceability of Object Oriented Requirements) was based on principles from hyper-programming and hyper-requirements. TOOR operated in three trace modes, namely selective tracing, interactive tracing, and non-guided tracing. Selective tracing focused on selected patterns or objects and relations, while interactive tracing enabled interactive browsing through the objects in both forward and backward directions. Non-guided tracing was used to go from any object to any other object. The authors designed TOOR by keeping in mind that requirements issues occur throughout the software lifecycle and that requirements continue to evolve during the software lifecycle. TOOR also supported semantic information about links between objects.

Ramesh [41] analyzed the factors influencing the practice of requirements traceability. His work mainly focused on environmental and organizational factors. It also captured the trends and practices in requirements tracing. The author reported the results from the empirical studies conducted and aimed to develop reference models to guide better practice. Based on his analysis, the author classified the participants of the study into two distinct groups, namely high-end and low-end traceability users, with respect to requirements traceability practice. The author demonstrated that there was a significant difference in the traceability practice of the two groups. He also suggested that a transition from a low-end practice to a high-end practice would bring significant benefits such as increased process maturity and lower life cycle costs.

Tsumaki and Morisawa [53] proposed a framework for requirements tracing using UML. They examined how to trace use-cases, class diagrams, and sequence diagrams from the business model to the analysis model and to the design model [53]. Our research does not focus on tracing non-textual artifacts. Hence, it is not possible to directly compare our work with that of Tsumaki et al.

Cleland-Huang et al. [11] proposed an event-based traceability technique supporting impact analysis of performance requirements. They used a performance model to determine the impacts by the proposed changes. They believed that the performance models that were affected by the change needed to be identified and re-designed to reflect the proposed changes. In order to do that, traceability links needed to be maintained between the performance models and the requirements specification.

Huang et al. [11] proposed a dynamic traceability scheme that was capable of speculatively driving the performance models that were affected by a proposed change. Their framework had a central requirements repository containing key data from within the individual performance models. The links were established and maintained between the data in the repository and those in the performance models. Whenever a change was proposed, the values of the corresponding requirements were modified accordingly in the repository and the change was propagated into the related performance models by following the links. The performance of the models was assessed using the modified data-values.

Based on the case study, Huang et al. stated that event-based traceability had the ability to identify the performance models impacted by a proposed change, propagate the data into those models, and re-run the models to reflect the changes. They believed that the scarcity of truly executable modeling environments was one of the main constraints for their approach.

Egyed et al. [16] used Trace Analyzer [15] for automating requirements tracing. Trace Analyzer [15] used trace reasoning and shared use of "common grounds" such as source code to define trace dependencies. Their technique took known dependencies between software artifacts and common grounds to build a graph where nodes represented those common grounds and their overlaps. The graphs were manipulated by moving known artifacts between the nodes. Trace Analyzer used various rules to move nodes around in an iterative fashion. If there was at least one common node between two artifacts, the documents were termed as related. The more nodes the artifacts have in common, the stronger the dependency. Egyed et al. presented a case study using a video-on-demand system. The authors mentioned the following as the benefits of their approach:

- Tracing non-functional requirements,
- Aiding identification of conflicting requirements,

- Verifying requirements,
- Identifyig missing requirements,
- Determining change impact,
- Determining impact of new requirements,
- Understanding level of strength of dependencies,
- Distinguishing domain specific code vs. generic code,
- Determining artifacts needing attention, and
- Balancing granularity of requirements.

The main objective of the work of Egyed et al. [16] was to find out the dependencies between requirements and code. Their work also focused on the similarity between model elements and code. Our work focuses on the similarities between unstructured textual artifacts.

Spanoudakis [49] used heuristic traceability rules to automatically trace requirements artifacts to object models. This work was an extension of the work mentioned in [57]. The heuristic rules syntactically matched textual requirements to related elements in object models such as classes, attributes, and operations. Spanoudakis measured three types of beliefs in this paper, namely belief in rule satisfiability, belief in correctness, and belief in traceability relation. In [49], the author mentioned that these beliefs could assist the users of the traceability system in deciding whether to de-activate or modify specific rules.

Spanoudakis et al. [50] described their approach that automatically traced textual requirements artifacts to use cases and analysis object models. Their approach used two types of rules to automate the generation of traceability relations. It used requirements-to-object-model (ROTM) rules to trace textual requirements and use cases to an analysis object model. Also, it used inter-requirements traceability rules (IREQ) to trace requirements and use cases. Spanoudakis et al. came up with a prototype system that incorporated the rules mentioned earlier to generate traceability relations. The system used XML to represent requirements, use case specification documents, and the analysis object model. They reported all the experiments that they conducted to evaluate the prototype.

Hoffman et al. [31] presented a requirement catalog for requirement management tools based on their experience in the automotive industry, the aircraft industry, and with defense systems. The catalog was helpful in selecting requirements management tools for one's specific needs. Hoffman et al. defined and described requirements management tools from the point of views of developers, project administrators, and tool administrators. They discussed the tools that addressed various requirements management functionalities such as information model, views of the data, format, change management, documentation of history, baselines, tool integration, document generation, workflow management, installation and administration of projects, database, encryption, etc. Our research focuses mainly on candidate link list generation and it does not focus on any requirements management activities. Hence, the requirements presented by Hoffman et al. do not directly apply to our research work.

3.2 Information Retrieval in Requirements Tracing

Some of the research discussed not only focused on requirements tracing, but also on the overall problem of requirements management. Also, manual techniques were often used to improve requirements tracing. The research discussed earlier that did look into automating requirements tracing process did not use information retrieval. This sub-section discusses the research that uses information retrieval to improve the requirements tracing process. This sub-section is also organized in chronological order.

Antoniol et al. [5] apply two IR methods, namely, probabilistic model and vector space model (tf-idf), to recover traceability links between code and documentation. Antoniol et al. believe that the documentation and the free text in the code might help to capture the association between high-level concepts and program concepts.

Their approach works as follows. All documents undergo text normalization that include transforming all upper case letters into lower case letters, removing stopwords, converting plurals into singulars, etc. Stopwords are those words that do not add any significance to the characteristics of the elements. A vocabulary is built using the keywords in the processed document collection and the documents are indexed. The indexed documents are input to the IR methods as the document collection and each source code component is passed as the query. The query is constructed by first extracting all the identifiers, splitting identifiers that are composed of more than one word, and applying text normalization as

described for the document collection. The similarity between each document in the document collection and the given query is computed using a classifier and a list of matching documents for each query, ordered by decreasing similarity, is generated.

Two case studies are reported in [5] where one of them traces C++ source code onto manual pages and the other case study traces JAVA code to functional requirements. Both case studies support the author's hypothesis that IR methods can be used in retrieving traceability links between the source code and documentation. The authors also investigate the use of IR methods to support change impact analysis.

Marcus and Maletic [34] use the Latent Semantic Indexing (LSI) technique to study requirements-to-code traceability. They compare the results obtained from their experiments with the results obtained by Antoniol et al. [5]. Marcus et al. use the same dataset as Antoniol et al. in [5]. However, Marcus et al. [34] trace the links from the manual to the source code where as Antoniol et al. trace the links from the source code to the manual. Marcus et al. report that though LSI requires less computation as compared to Antoniol's methods, it performs as well as the methods used by Antoniol et al.

Marcus et al. mention that LSI can be applied without significant preprocessing of the input as it does not rely on a predefined vocabulary or grammar for the input. They also use the internal documentation in the source code. They state that it helps them achieve better results with one of their datasets. LSI performs as well as the methods used by Antoniol et al. on the dataset that had almost no internal documentation. They believe that the results can be further improved by using structural information in the source code.

Though our research is closely related to that of Marcus et al. [34] and Antoniol et al. [5], there are significant differences too. While Marcus et al. [34] and Antoniol et al. [5] focus on tracing requirements to code, our work focuses on tracing non-structured textual requirements between different documents in the project document hierarchy. It should also be noted that one of the key aspects of our research is the feedback process. Antoniol et al. do not use feedback in their work.

All researchers applying IR methods to tracing use precision and recall measures to assess the quality of the candidate link lists generated. However, Marcus et al. and Antoniol et al. calculate precision and recall differently. They do not have an answerset manually verified by an analyst against which to measure, but we do. Also, we use different datasets

for our work. Hence, comparing the precision and the recall numbers from our experiments to theirs may not be very meaningful. At the same time, we also adopt the tf-idf method and LSI to generate candidate link lists do as Marcus et al. In fact, they show that LSI outperforms the tf-idf method. It will be interesting to observe if the same holds for our experiments. In addition to recall and precision, we also introduce a few more measures to analyze the quality of the candidate link lists generated. In [26], we compare the results of the tf-idf method to that of a manual tracing process.

Huang et al. [13] propose an approach to maximize the return on investment of the traceability effort by applying a heterogeneous set of traceability techniques based on the requirements of the system. The authors discuss different traceability techniques and the advantages and disadvantages of those techniques.

Huang et al. discuss their prototype called TraCS (traceability for complex systems) that allows the users to define their strategy. Based on the strategy assigned, TraCS traces project artifacts using traceability techniques that satisfy those strategies. TraCS incorporates user-defined links that are traced manually, dynamic links that are generated using IR techniques developed by the author, and links generated by Event Based Traceability [11, 12]. Huang et al. also explain their model for comparing the traceability costs of TraCS.

Settimi et al. [48] focus on dynamically generating traces using information retrieval techniques. Their main focus is to trace requirements to UML models. They also discuss using different information retrieval techniques to trace requirements to code and test cases. They analyze the vector space model and pivot normalization techniques. They also analyze if the use of a thesaurus will improve the results. Their experimental results suggest that their methods are more effective in establishing traceability links to UML models than to code. They suggest that it is a good idea to establish traceability links to code through UML models. In a way, their work is similar to ours. They use the vector space model to generate traces like we do. They also analyze the use of a thesaurus, which is similar to our work. However, they trace requirements to UML models whereas our work does not support tracing non-textual/graphical/structured artifacts. Hence, it is not possible to directly compare our work with theirs.

Hayes et al. [28] define and discuss the importance of secondary measures to analyze the quality of a given trace and how secondary measures affect the analyst's perception

of the quality of the given trace. Hayes et al. [24, 29] discuss about the importance of the analyst's decisions on the final outcome of a tracing task. They state that though researchers have been working on automating the tracing process, the analysts still need to make critical decisions such as if all the links have been identified, etc.

Hayes et al. [25] present an experimental framework for evaluating requirements tracing and traceability studies. They use their framework to describe and compare experimental studies reported in [5, 6, 26, 34]. They also use their framework to identify areas for future research and future experimentation.

Chapter 4

Research Approach

This section explains the methods that we adopted, with significant modification, from the IR world. These methods were never used by any other researchers to solve the requirements tracing problem. The feedback algorithm, which is one of the unique features of this work as far as the requirements tracing problem is concerned, is also explained in detail. The later part of the section explains how we use information retrieval methods to solve requirements tracing problems.

4.1 Keyword Extraction by χ^2

Matsuo and Ishizuka [56] propose a method to extract keywords from a single document. Their approach uses word co-occurrence statistical information to identify the importance of the keywords. At first, they stem keywords using the Porter algorithm [36] and extract phrases. They remove all the stop words included in the stop list used in the SMART system [22]. They identify frequent terms by selecting the top frequent terms up to f % of all the keywords. Then, they cluster the similar frequent terms using similarity-based clustering and pairwise clustering. Clustering is an unsupervised learning method to group similar objects or keywords in a collection. Finally, they calculate expected probability and χ^2 and output the keywords in descending order of their χ^2 value.

Matsuo and Ishizuka [56] assume that important keywords show a more irregular pattern of co-occurrence with different terms than generic words. They use the χ^2 value to measure the irregularity of such patterns. They state that "if the probability distribution of co-occurrence between term a and the frequent terms is biased to a particular subset of frequent terms, then term a is likely to be a keyword. The degree of bias of a distribution is measured by the χ^2-measure [56]." The main advantage of the keyword extraction method is that it identifies important keywords without using a corpus.

Here are the steps involved in calculating the χ^2 value to identify the important keywords:

1. Stop words are removed and the keywords are stemmed to their root.

2. The top frequent terms up to 30% of the total terms are selected.

3. The frequent terms are clustered. Two types of clustering techniques are used. A pair of terms are clustered, if their Jensen-Shannon divergence is above a certain threshold value. Also, a part of the terms are clustered if their mutual threshold is above a particular threshold value. Jensen-Shannon divergence is calculated as follows [56]:

$$J(w_1, w_2) = \log 2 + \frac{1}{2} \cdot \sum_{w' \in C} h(P(w'|w_1) + P(w'|w_2)) - h(P(w'|w_1)) - h(P(w'|w_2)). \tag{4.1}$$

   Mutual information is calculated as follows [56]:

$$M(w_1, w_2) = \log \frac{P(w_1, w_2)}{P(w_1) \cdot P(w_2)} \tag{4.2}$$

$$M(w_1, w_2) = \log \frac{N_{total} \cdot freq(w_1, w_2)}{freq(w_1) \cdot freq(w_2)}. \tag{4.3}$$

   In the above equations, N_{total} is the total number of running terms in the document and C is the cluster being identified.

4. The number of terms (n_c) co-occuring with frequent terms c $\in$ C is counted. Using n_c, expected probability is calculated as follows:

$$p_c = \frac{n_c}{N_{total}}. \tag{4.4}$$

5. Finally, a χ^2 value for each term is calculated as follows [56]:

$$\chi^2(w) = \sum_{c \in G} \frac{(freq(w, c) - n_w \cdot p_c)^2}{n_w \cdot p_c} - \max_{c \in G} \frac{(freq(w, c) - n_w \cdot p_c)^2}{n_w \cdot p_c} \tag{4.5}$$

   where freq(w,c) is the co-occurrence frequency of w with c $\in$ C and n_w is the total number of terms in the sentences.

This method can be extended in a straightforward manner to identify the important keywords in the collection of documents that are being traced. False positives in the traces are generated by coincidental matches. Identifying important keywords and using only those keywords while matching documents may help reduce the number of false positives generated.

In the classic vector space model, only the document collection is used to build the vocabulary base as the queries are not known in advance. We later explain, in the case of requirements tracing, how we can use both the documents and the queries to build vocabularies. Similarly, if we are tracing high level documents to low level documents, we can identify the important keywords in the high level document collection or both the high level and the low level document collection. In order to identify the important keywords in the high level document collection only, we can combine all the high level documents and consider the entire collection as a single high level document. In order to identify the important keywords in both the high level and the low level document collections, we can consider both the document collections as a single document. In both these cases, we will use the above mentioned keyword extraction process to identify the important keywords in the single document generated from the document collections.

Hence, we use the approach explained above to generate the list of keywords in descending order of their χ^2 value. The higher the χ^2 value, the more important the keyword. The top $x\%$ of the keywords in the list will be selected. We do not know what the ideal value for x will be. The query and document vectors will be built only with the selected keywords. Then, cosine similarity shall be computed as mentioned in the vector space model section. For each high level document, the list of low level documents in the order of their similarity value will be generated.

Let us look at some critical steps taken to adopt the approach proposed by Matsuo and Ishizuka [56]. They consider two words to be co-occurring if the words appear in the same sentence. In our modified method, we consider two words to be co-occurring if they occur in the same element. Also, our modified method does not cluster the frequent terms. Since the keyword extraction method considers only important keywords above a certain χ^2 value and ignores other keywords, it may not produce better recall (the percentage of correct links found) than the vector space retrieval method. However, it is possible that this method might produce better precision (the percentage of relevant links retrieved) as it avoids keywords that are not important.

4.2 Keyword Extraction by idf

This method is extended from the tf-idf method. In the tf-idf method, term frequency and inverse document frequency are used to compute the weight of a keyword. Inverse document frequency measures how commonly a keyword occurs in the document collection.

If a keyword occurs very often in the document collection, it may not convey any specific information about a particular document. Hence high inverse document frequency means that the keyword occurs only in very few documents and that it might be an important keyword.

In order to consider only the important keywords, the keywords with low idf value can be used while building document and query vectors. In our experiments, all terms with df = 1 are excluded and keywords with top x% of idf are extracted. Then, document and query vectors are built based only on the keywords that have been extracted. The cosine similarity between the vectors of high-level and low-level elements is computed. Either both high-level and low-level elements or just the high-level elements can be used in constructing the initial vocabulary.

4.3 Initial estimates for Probabilistic IR

Let us say that each document **d** in the document collection **D** is represented as a binary vector d = {d_1,d_2,...,d_N}, where N is the total number of terms in the vocabulary of the document collection, d_i = 1 if term i is present in the vector, and d_i = 0 if term i is not present in the vector. Also, consider that the user query is converted into a binary vector $q = (q_1, \ldots, q_N)$. As mentioned earlier, the similarity between q and d is calculated as follows:

$$sim(q,d) \sim \sum_{i:d_i=1,q_i=1}^{N} \log \frac{p_i}{1-p_i} + \log \frac{1-u_i}{u_i}, \tag{4.6}$$

where p_i = P(d_i = 1|R) (i.e,) the probability that the keyword d_i is present in a document randomly selected from the set R of relevant documents and u_i = P(d_i = 1|NR) (i.e,) the probability that the keyword d_i is present in a document randomly selected from the set NR of irrelevant documents.

We do not know the value of p_i and u_i at the start. Hence, we need to derive an initial estimate for p_i and u_i. We estimate u_i for two cases, when the number of documents is more than the number of queries and vice-versa. When the number of documents is higher, u_i is calculated as the ratio of the number of documents in which the keyword d_i occurs to the total number of documents. When the number of queries is higher, u_i is calculated as the ratio of the number of documents in which the keyword d_i occurs to the total number of documents.

Let us say that D_i is the number of documents in which the keyword k_i occurs, n is the number of keywords in the query being considered, ADL is the average length of the documents in the document collection, TotalWords is the total number of words in the vocabulary, N is the number of queries, and M is the number of documents. Hence, when $M > N$

$$u_i = \frac{D_i \cdot N}{(N-1) \cdot M}, \tag{4.7}$$

and when $M \leqslant N$

$$u_i = \frac{D_i}{(M-1) \cdot M}. \tag{4.8}$$

Let L be the product of the average length of the documents in the document collection and the ratio of the number of documents to the number of queries. If there are more queries than documents, L is equal to the average length of the documents in the document collection:

$$L = Max(1, \frac{M}{N}) \cdot ADL. \tag{4.9}$$

In this work, we come up with three different estimates for p_i. The first estimate calculates p_i as the ratio of the average length of the documents in the document collection to the total number of words in the vocabulary. When the number of queries is greater than the number of documents, the estimate is multiplied by the ratio of the number of queries to the number of documents:

$$p_i = \frac{L \cdot N}{M \cdot TotalWords}. \tag{4.10}$$

The second estimate calculates p_i as the ratio of the average length of the documents in the document collection to the total number of words in the vocabulary and the number of keywords in the query under consideration. Just as in the previous case, when the number of queries is greater than the number of documents, the estimate is multiplied by the ratio of the number of queries to the number of documents:

$$p_i = \frac{2 \cdot L \cdot N}{(TotalWords + n) \cdot M}. \tag{4.11}$$

The third estimate calculates p_i as follows:

$$p_i = \frac{N}{M} \cdot (1 - (\frac{n-1}{n})^{\alpha \cdot Max(1, \frac{M}{N}) \cdot AvgDocLength}). \tag{4.12}$$

The list of documents matching the given query q is generated using the p_i and u_i values. The list is improved by adjusting the initial estimates of p_i and u_i, as explained in Chapter 2.

4.4 Vocabulary Base

Standard IR models use only the document collection to create the vocabulary base (the collection of keywords that will be used to match documents and queries). This is mainly due to the fact that the query is not known until it is provided by the user. For example, internet search engines use the WWW to create and maintain vocabulary bases so that the user search can be handled quickly.

In the case of requirements tracing, all the queries are known at the start. Hence, both documents and queries can be used to create vocabulary bases. In this work, we create vocabulary bases using both the approaches mentioned above to see if there is any significant difference in the final result.

4.5 Relevance Feedback

Tracing is a significant part of the V&V and IV&V process. The analyst must go through the generated links to verify the output of the IR methods. The output may have two types of errors:

- errors of commission - a false link is included in the output, and
- errors of omission - a true link is not included in the output.

We observed that the analyst can verify the candidate list and provide minimal information regarding the accuracy of the links which can be used to further improve the quality of the candidate list. The information provided by the analyst can be used to fix some errors of commission, errors of omission, and to restructure the candidate list such that true links have higher relevance weight than false positives. This procedure is called relevance feedback analysis.

Relevance feedback is commonly used in the IR world to utilize user information to improve the quality of the information retrieval algorithm. Relevance feedback algorithms adjust the keyword weight of query vectors based on the feedback information provided. For example, if a link is termed as irrelevant ("Not A Link"), the weight of all the keywords appearing in that query will be reduced. Whereas if a link is termed as relevant ("Link"), the weight of all the keywords related to that particular query will be increased. In order to identify new links similar to the links termed as relevant, the feedback process adds keywords in the relevant documents to the query vector matching those documents.

The feedback process can be defined as follows: Let q be a query vector and D_q be a list of document vectors returned by an IR method given q. Now, consider that the analyst has indicated that D has two subsets: D_{rel} of size R is the subset of documents relevant to q and D_{irr} of size S is the subset of irrelevant documents to q. A document from D cannot be in both D_{irr} and D_{rel} and the analyst does not have to provide feedback information for all the documents in D. The Standard Rochio [7] feedback method used in our research modified the query vector as follows:

$$q_{new} = \alpha \cdot q + (\frac{\beta}{R} \sum_{d_j \in D_{rel}} d_j) - (\frac{\gamma}{S} \sum_{d_k \in D_{irr}} d_k). \tag{4.13}$$

As mentioned earlier, it can be seen in the above formula that the query vector is adjusted by adding to it the sum of all document vectors that are relevant to it and subtracting the sum of all irrelevant document vectors from it.

Once the query vectors are fixed, the similarity between query vectors and document vectors are computed again to generate the new candidate link list that reflects the feedback information. Adding new terms to the query vector will result in new potential relevant links. This will fix errors of omission. Whereas deleting terms from the query vector will drop the potential irrelevant links. This will fix errors of commission. The feedback process can be repeated until the analyst believes that all the potential links have been found.

4.6 Voting Tool

In this work, we adapted more than one infromation retrieval method to generate traceability links. For a particular dataset, certain methods may perform better than others. For a given dataset, we do not know which information retrieval methods will perform well.

We developed a voting tool to take advantage of the availability of multiple information retrieval methods in our RETRO tool.

Consider the scenario where high level elements are traced to low level elements. As stated in [14], each of the methods that we adapted to generate traceability links can be considered a filter. For each high level element and low level element pair, the methods decide if the pair of elements under consideration is a link or not. Hence, each method F can be viewed as a function [14]:

$$F : H \times L \rightarrow \{True, False\} \tag{4.14}$$

where $F(i, j) = True$ means that the method identifies the high level element h_i and the low-level element l_j as links.

In this work, we have implemented five information retrieval methods to generate traceability links. In order to build a voting tool that uses all these methods, we first need to idenfity a voting scheme. The voting scheme is a decision rule that will determine if a given pair of a high level element and low level element are linked or not.

There are different decision rules that can be expressed. We can treat each method equally and simply count for each link (i,j) the total number of $F(i, j)$ which evaluate to True. An alternative is to assign a weight to each method while calculating the total number of $F(i, j)$ which evaluate to True. In this case, the decision rule will consider a high level element and low level element pair as a link iff this number reaches a certain threshold. For example, a majority rule will consider a pair of elements as a link if it is recovered by more than one half of all available methods, three methods in our case.

The voting tool is written in JAVA. It takes the results generated by the information retrieval methods in the RETRO toolbox. It also takes a parameter for the decision rule and the list of methods to be included in the decision process. The output of the voting tool is the candidate link list that corresponds to the decision rule selected.

4.7 Automating Requirements Tracing

Let us revisit the requirements tracing process explained in Table 2.2 and see how it can be automated. A word processing tool can easily assist the analyst in identifying the requirements elements and assigning unique identifiers for the extracted requirements.

Once an RTM is generated, there are tools such as SuperTracePlus [23, 37] etc., that can be used to create summary reports. There are some tools that assist analysts in identifying the candidate links and generating the RTM, but most of them require the analyst to go through each high level and low level element to assign keywords or require some additional task prior to the tracing process. It will be ideal to automate this process without requiring the analyst to do any manual preprocessing. Hence, this becomes an interesting problem for researchers.

Most of the times, requirements tracing is performed for mission- and safety-critical projects. In those cases, it becomes crucial to identify if all the requirements are satisfied. Even though the generation of the RTM can be automated, the analyst should still evaluate the candidate list generated to see if the list is complete, to see if all the requirements are satisfied, or to see if it is necessary to look for more links. Hence, the automated tool should contribute in reducing the analysts' involvement in the tedious and mundane parts of the process and assist the analyst in the effective decision making parts of the process.

The requirements tracing process is highly unreliable as the quality of the resultant RTM depends on the analyst's effort which cannot be controlled. Even a very experienced analyst can have a bad day or make mistakes on a good day. Instead, if we can generate candidate links using automated methods, there will always be consistent effort applied.

Requirements Tracing as an Information Retrieval Problem

In order to understand if IR methods can assist the requirements tracing process, the concept of requirements "similarity" needs to be examined. While an analyst uses this notion of requirements similarity to determine the trace, IR methods rely on the notion of document relevance to match queries and documents. IR methods identify matches purely by arithmetical means whereas human analysts can use any tool available to determine the trace. Requirements tracing can be considered as a repetition of the IR process where high level requirements are considered as queries and all the low level requirements are considered as a document collection. During each iteration, a high level requirement is compared against all the low level requirements in the document collection to generate the candidate list for the high level requirement being considered. At the end of this repetitive process, a candidate list is generated for all the high level requirements.

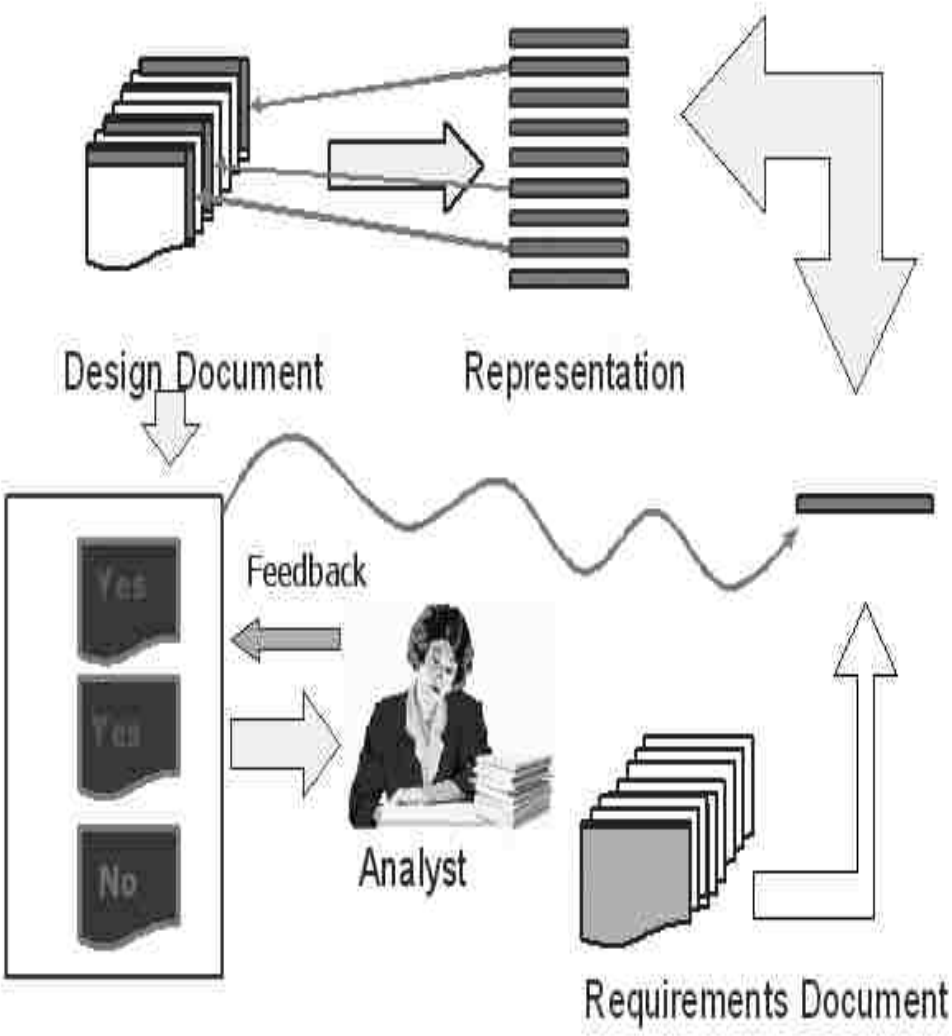


Figure 4.1: Automated Tracing using IR

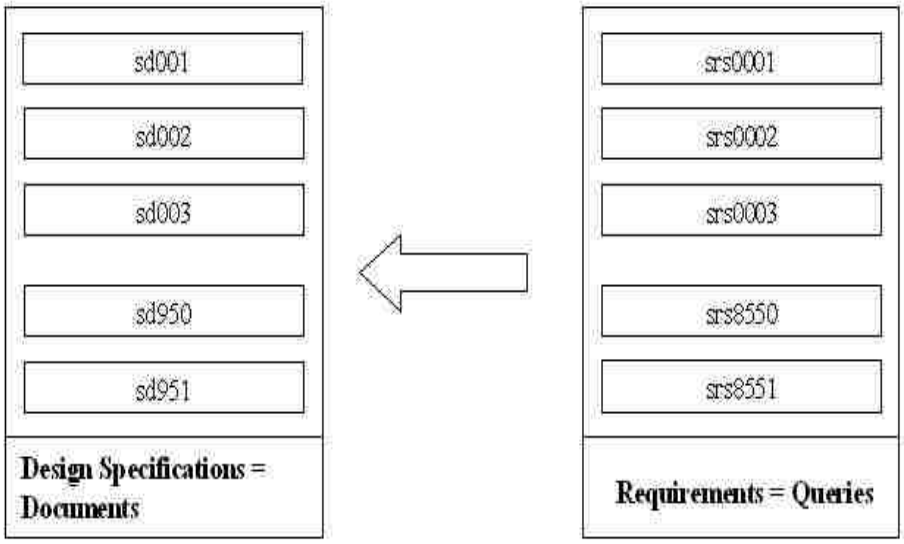


Figure 4.2: Forward Tracing as an IR Problem

Let us revisit the notion of forward tracing and backward tracing. Let us consider the situations stated in Figure 2.4 and Figure 2.5. In the case of forward tracing, each requirement will be considered as a query and design specifications will become the document collection as shown in Figure 4.2. Similarly, in the case of backward tracing, design specifications will become queries and requirements will form the document collection, as shown in Figure 4.3.

While there are many similarities between the information retrieval problem and the tracing problem, there are many unique features of tracing that cannot be found in information retrieval problems. We discuss those next.

Size of the domain

Most of the Information Retrieval algorithms are designed to work on large document collections. The number of requirements or design elements in a large scale software project may be in the order of thousands. But a typical IR problem contains documents in the order of hundreds of thousands and millions. In most cases, the requirement and design elements are only a few sentences long. In IR, the documents are way larger than two sentences long. There is also a huge difference in the size of the vocabulary.

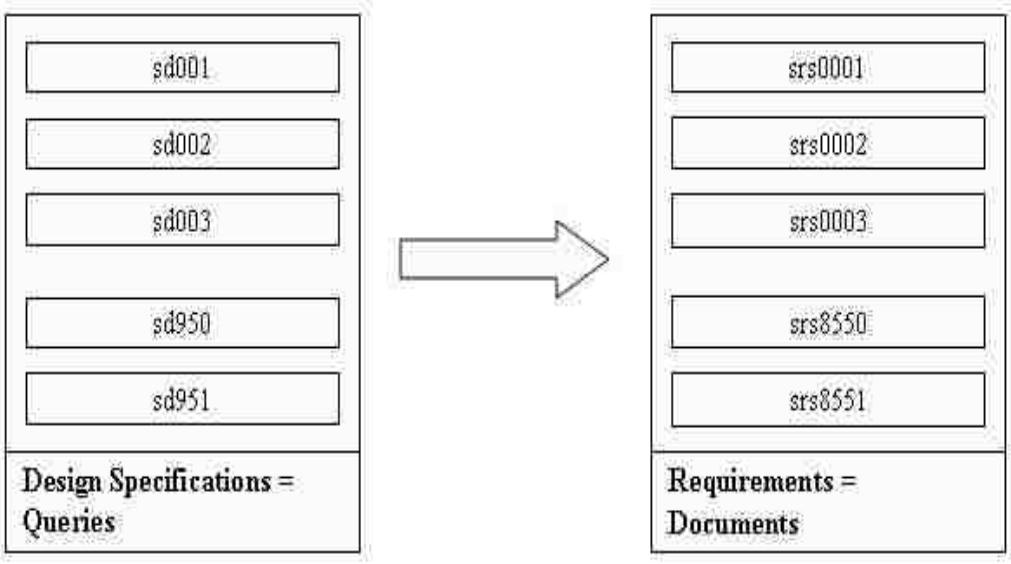


Figure 4.3: Backward Tracing as an IR Problem

Information Retrieval algorithms are robust on large data collections. Their performance on smaller datasets might not be as good. It is possible that coincidental matches might outscore the true matches.

Query independence

In a traditional IR system, the queries are considered as being independent. Consider an internet search engine where the user query cannot be predicted earlier. However, in requirements tracing, the queries are known before-hand. This information can be used to improve the IR methods.

4.8 RETRO

RETRO (REquirements TRacing On-target) is a special purpose tool that we designed exclusively for requirements tracing. Given a list of high level and low level documents, RETRO will produce traceability matrices. RETRO does not focus on requirements management. However, it generates traceability matrices in an easy-to-use XML format which can be used by other requirement management tools, such as DOORS [52], RequisitePro [43], QSSrequireitIt, etc. Figure 4.4, Figure 4.5, Figure 4.6, Figure 4.7, and Figure 4.8 show screenshots of the RETRO Graphical User Interface (GUI).

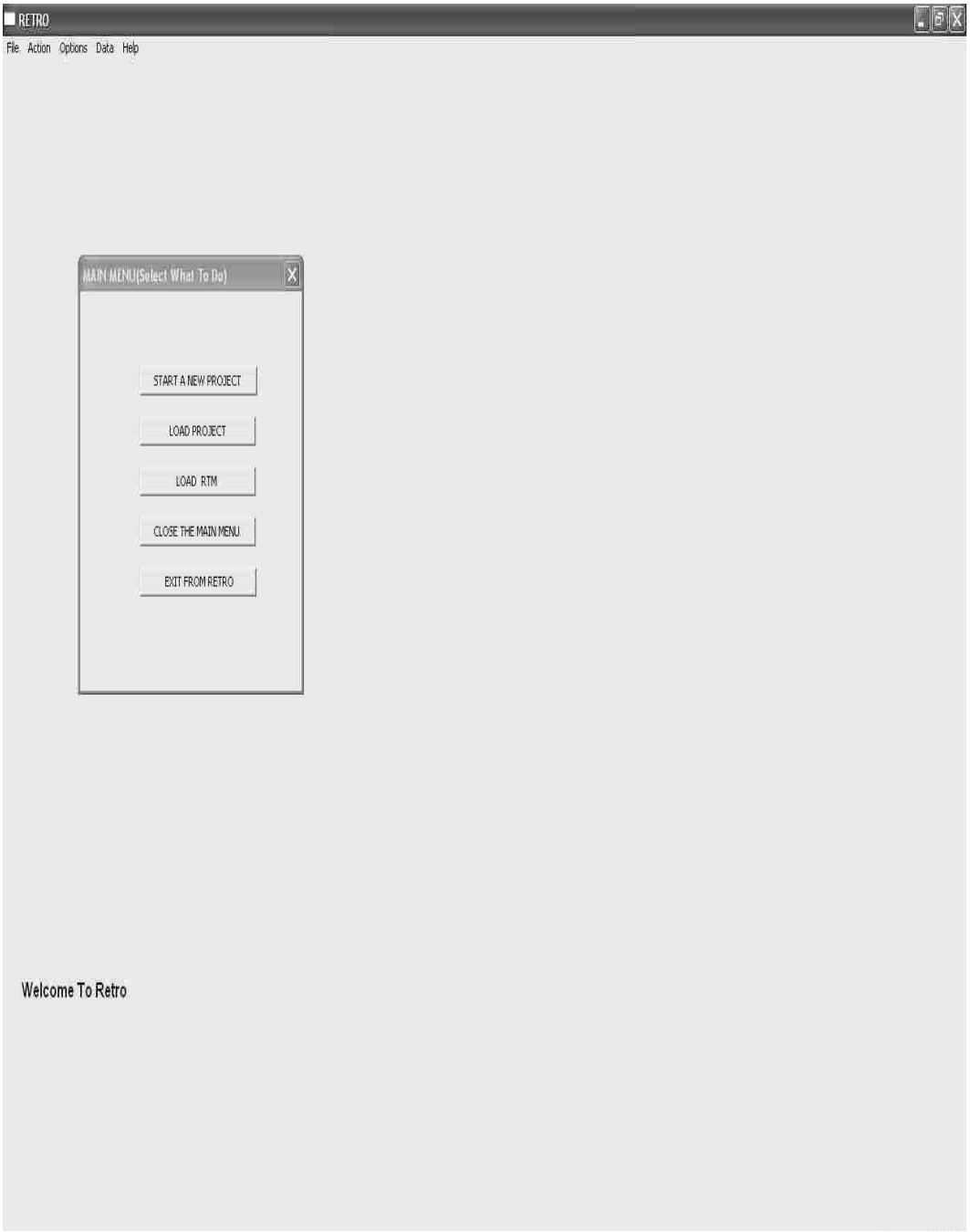


Figure 4.4: Retro

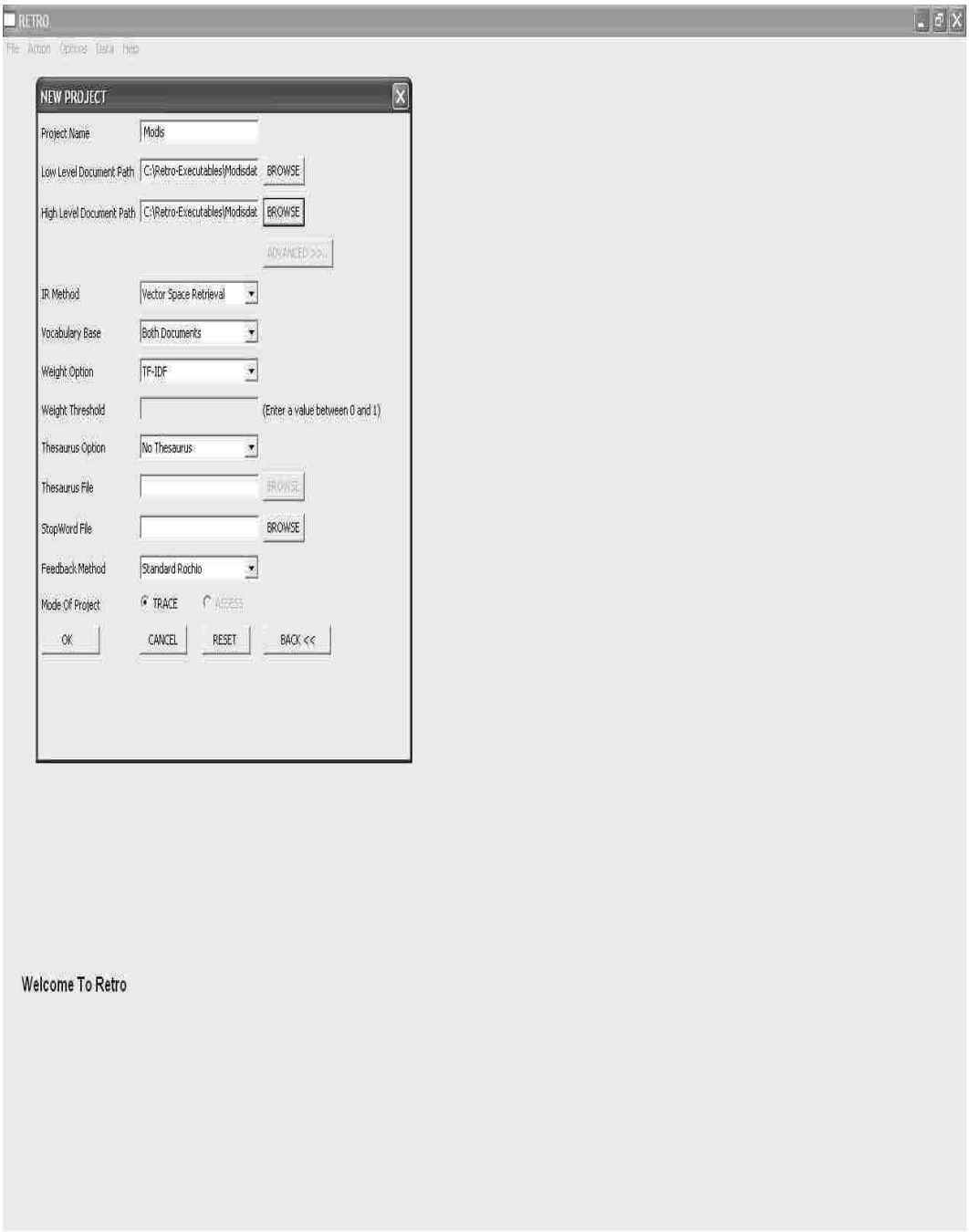


Figure 4.5: Retro - Starting a new Project

RETRO contains an IR toolbox that comprises of a collection of implementations of IR methods adapted for the purposes of the requirements tracing process. Currently, the IR toolbox contains the following methods:

- Vector space model,
- Vector space model with thesaurus,
- Keyword extraction using χ^2, and
- Keyword extraction using IDF.

The IR methods are implemented in VC++ on the Windows platform. The GUI module of RETRO is implemented in JAVA. The GUI can be used to select the IR method and also to pass the information about the documents to be traced. On appropriate Application Programming Interface (API) calls, IR methods in the toolbox generate the candidate list in Extensible Markup Language (XML) format. The candidate list generated is parsed by the GUI and the list is shown to the analyst.

RETRO can work in one of the following three modes:

- Trace mode,
- View mode, and
- Browse mode.

Trace mode is the most commonly used mode as it contains the main functionalities such as starting, running, and terminating a trace. RETRO offers flexibility in displaying the candidate lists generated. The list can be filtered based on the weight factor of the link or the rank of the link. The analyst can also choose to view the content of all the matching low level documents for the high level document selected either based on the order in which they occur in the document collection or based on the weight factor.

As soon as a trace is started, the trace mode shows the list of high level as well as low level documents. Selecting a high or low level document will display the content of the corresponding document. There are buttons at the bottom of the screen to control the tracing activity. Once the analyst presses the "Trace All" button, the candidate lists for all the high level requirements are generated and displayed on the screen.

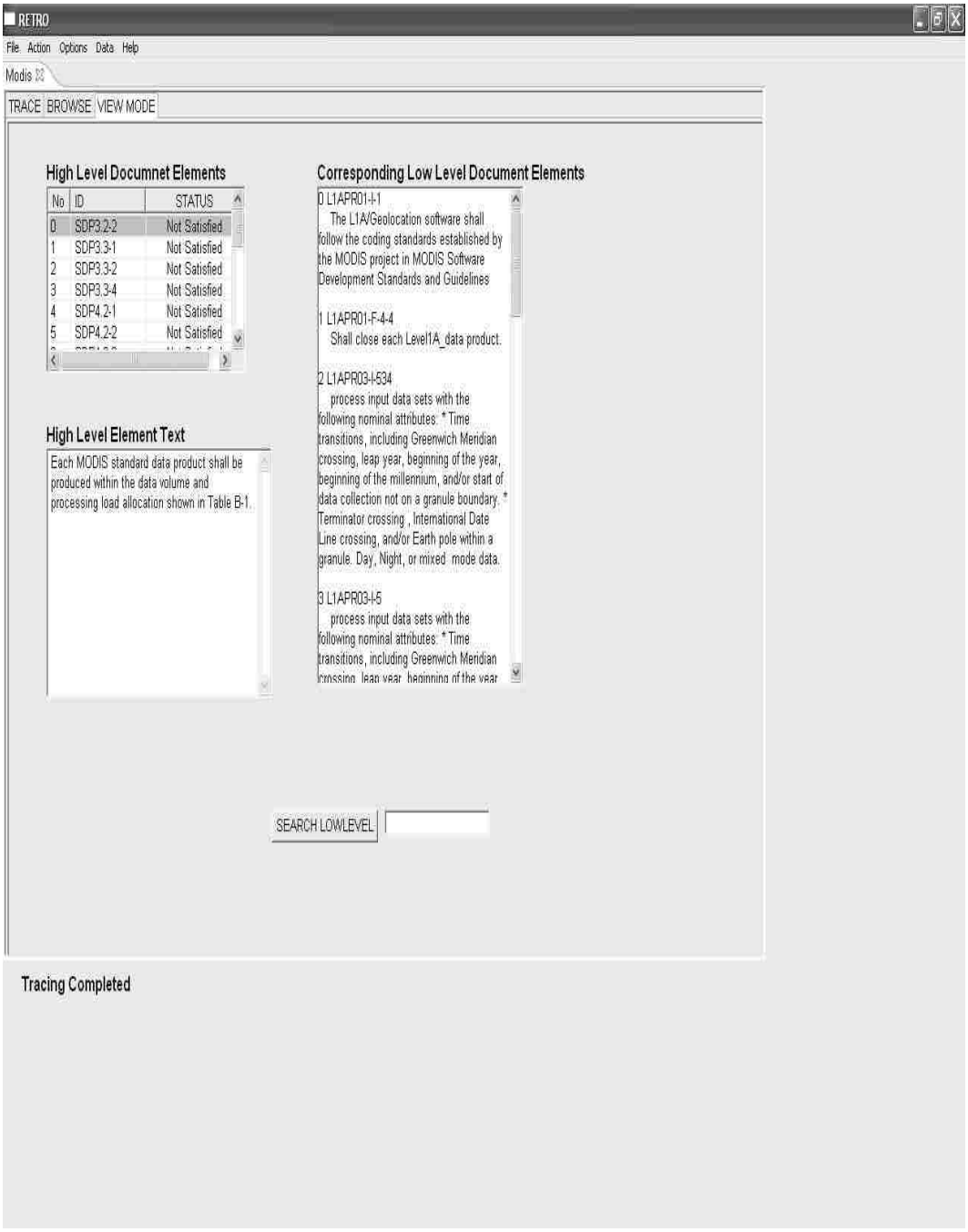


Figure 4.6: Retro View Mode

Figure 4.7: Retro Trace Mode

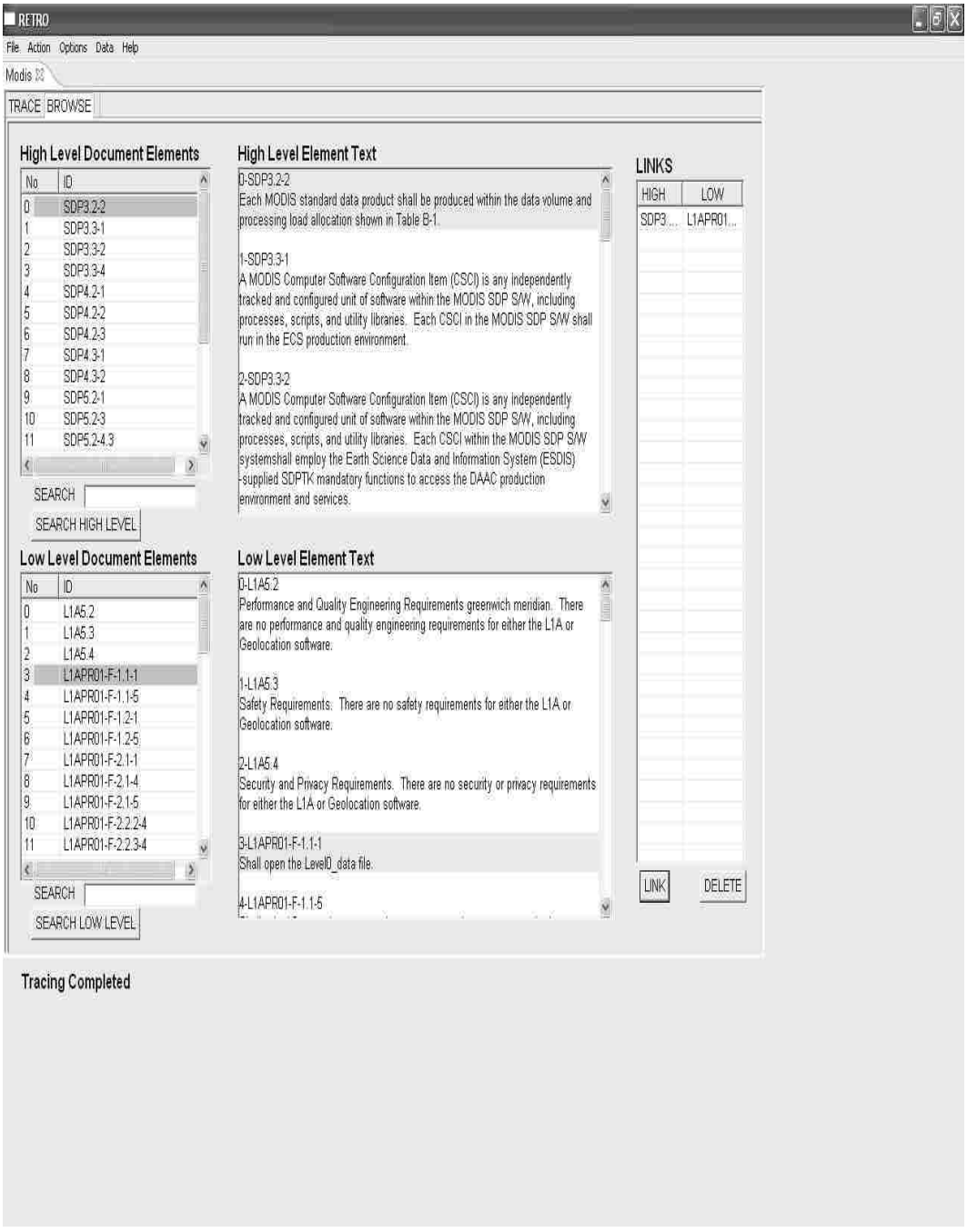


Figure 4.8: Retro Browse Mode

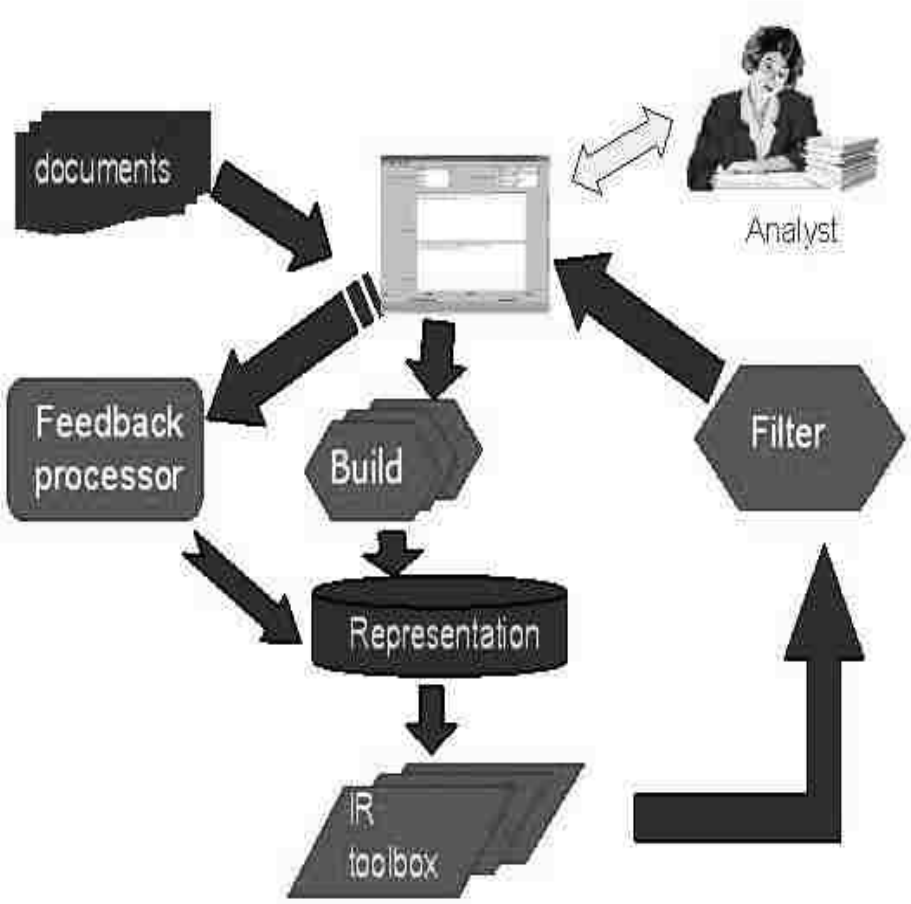


Figure 4.9: Retro Architecture

To begin with, the status of the candidate link is set to "Default." The analyst can make a decision on whether the candidate link under consideration is indeed a link or not by right clicking on the low level document element and changing the status of the candidate link to either "Link" or "Not A Link." This information is accumulated and, upon analyst request, is fed into the feedback processing module in the IR toolbox.

When the analyst presses "Trace All" again, the accumulated information is sent to the feedback process and the candidate list is recalculated for all the candidate links. If the "Trace Current" button is pressed, only the information related to the candidate link under consideration is sent to the feedback process and the candidate list is recalculated only for that particular candidate link.

The "Save Trace" option allows the analyst to save the trace in its current state and reload it later. If "Complete Trace" is selected, the RTM will be saved after throwing out all the candidate links that have been marked as "Not A Link." The XML generated by the IR toolbox can be displayed by selecting "Show XML" under the "Action" menu. RETRO can also generate reports on completed traces.

The browse mode allows the analyst to go through each high level and low level element separately. It also has a search feature that can be used to look for a specific keyword in the collection of documents. The view mode shows the candidate list generated along with the content for each low level and high level document. It also provides functionality to search low level documents for the high level document selected. The view mode can be selected only after the trace has been performed, whereas the browse mode can be selected at any time.

Chapter 5

Experimental Setup and Validation

In order to establish that our automated process indeed helps the analyst in the requirements tracing process, we need to validate the candidate links generated by our methods. As mentioned previously, precision and recall are the most commonly used measures to assess the quality of a search result in the IR world. Though these two measures indicate the quality of the result to some extent, they do not provide sufficient information to totally understand the quality of the candidate links generated. Hence, we came up with a few measures to get an insight into the results. In this section, we discuss all the measures and the datasets that we use to validate our methods. The next subsection explains the format of the datasets used, the format of the candidate list generated (XML), and the DTD (Document Type Definition) that defines the candidate list (XML).

5.1 Datasets

The experiments will be conducted using the following datasets: Modis dataset, CM1 dataset, and four subsets of the CM1 dataset. The Modis and CM1 datasets are available on the PROMISE website [51, 32].

The Modis dataset has been constructed by selecting 19 high level requirements and 49 low level requirements from two publicly available high level requirements [3] and low level requirements [2] documents for NASA's Moderate Resolution Imaging Spectrometer (Modis). A typical requirement in this dataset contains one or two sentences. The Flesch Reading Ease of a typical Modis requirement is 32.1 and the Flesch-Kincaid Grade Level is 12 [17, 18]. A theoretical true trace for the Modis dataset has been constructed and verified manually. The theoretical true trace contains 41 links. Figure 5.1 shows a requirement from the Modis dataset.

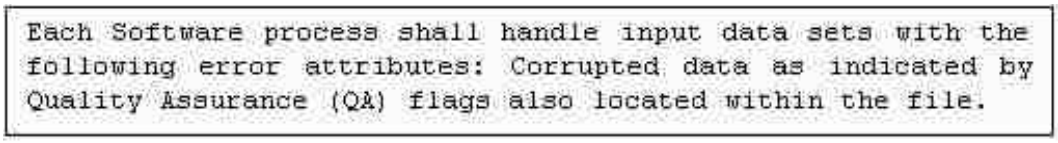


Figure 5.1: Modis

The CM1 dataset was made available by the Metrics Data Program (MDP) [55] and it contains a complete requirements (high level) document and a complete design (low level) document for a NASA scientific instrument. The documents have been sanitized by NASA in order to hide the identity of the instrument. A typical high level element of the CM1 dataset is one or two sentences in length whereas a typical design element is several paragraphs in length. The Flesch Reading Ease of a typical CM1 requirement is 40.5 and the Flesh-Kincaid Grade Level is 12 [17, 18]. The CM1 dataset contains 235 high level requirements and 220 design elements. Again, a theoretical true trace containing 361 links has been constructed and verified manually for the CM1 dataset. Figure 5.2 shows a requirement from the CM1 dataset.

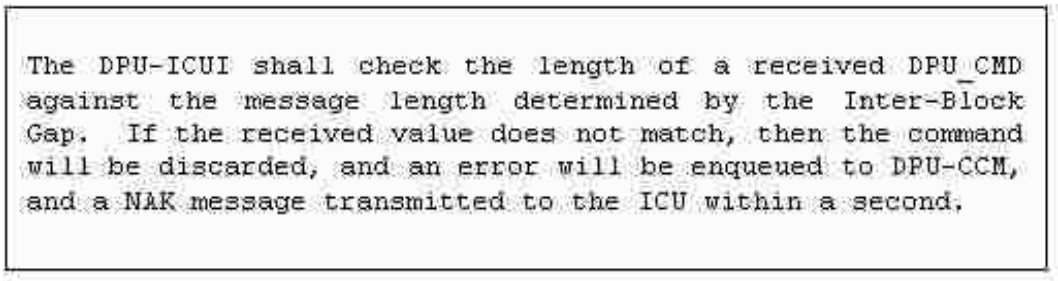


Figure 5.2: CM1

Four datasets have been constructed from the above-mentioned CM1 dataset. All the high level requirements and the low level elements in the subsets occur continuously in requirements and design documents, respectively. The theoretical true traces for these datasets have been generated from the theoretical true trace for the CM1 dataset. Table 5.1 shows the number of high level elements, low level elements, and true links for each CM1 subset generated. The smallest of the subsets is approximately equal to the Modis dataset in size.

Dataset	High level elements	Low level elements	Trace links
Modis	19	49	41
CM1	235	220	361
CM1-subset1	22	52	35
CM1-subset2	61	91	92
CM1-subset3	120	120	117
CM1-subset4	181	165	239

Table 5.1: Datasets

5.2 Input Format and Candidate List Format

RETRO takes the name of the folder containing high level elements and the name of the folder containing low level elements as input. Each element is stored in a seperate file. RETRO currently handles flat text files only. It does not support any graphic input. Figure 5.3 shows the input folders for one of the experiments.

One of the final products of the requirement tracing process is the requirements traceability matrix (RTM) or candidate link list. Our tool generates RTM in XML format. XML offers a universal standard and it can be easily read by using freely available XML parsers. If we have to load the RTM generated by our tool in some other tool, XML and the corresponding DTD can be passed on to the receiving tool. It makes the integration very easy. We already demonstrated that our tool can be easily integrated with SuperTrace Plus [23, 37].

Figure 5.4 shows the DTD describing the format of the XML file generated by RETRO and Figure 5.5 shows a sample XML file generated by the RETRO tool. Each high and low tag contains the name of the high level and low level element, respectively. The high tag also contains an attribute named "freeze." The analyst can set the value to one of the following: "Satisfied," "Not Satisfied," "Postponed," or "Partially Satisfied." "Satisfied" means that the high level requirement has been satisfied by the children elements. In other words, the analyst feels that all the low level requirement elements found for this high level requirement element cover the essence of the high level requirement element. "Not Satisfied" means that the list of low level requirement elements satisfying the high level requirement element under consideration is not complete. Even if the list contains all the low level requirement elements satisfying the high level requirement element under consideration, those low level requirement elements may not adequately address the high level requirement element. "Partially Satisfied" means that some of the matching low level

```
[senthil@bluehat CM1subset]$ dir -R
.:
high  low

./high:
SRS5.12.3.2  SRS5.12.3.6  SRS5.13.1.1  SRS5.13.2.1  SRS5.13.3.2
SRS5.12.2.1  SRS5.12.3.3  SRS5.12.3.7  SRS5.13.1.2  SRS5.13.2.2  SRS5.13.3.3
SRS5.12.2.2  SRS5.12.3.4  SRS5.12.4.1  SRS5.13.1.3  SRS5.13.2.3  SRS5.13.4.1
SRS5.12.3.1  SRS5.12.3.5  SRS5.12.4.2  SRS5.13.1.4  SRS5.13.3.1

./low:
DPUSDS5.12.0.1   DPUSDS5.12.1.5.2  DPUSDS5.13.1.2.1  DPUSDS5.13.2.1
DPUSDS5.12.1.2.1  DPUSDS5.12.1.5.3  DPUSDS5.13.1.3.1  DPUSDS5.13.2.10
DPUSDS5.12.1.2.2  DPUSDS5.12.1.5.4  DPUSDS5.13.1.3.2  DPUSDS5.13.2.2
DPUSDS5.12.1.2.3  DPUSDS5.12.1.5.5  DPUSDS5.13.1.3.3  DPUSDS5.13.2.3
DPUSDS5.12.1.2.4  DPUSDS5.12.1.5.6  DPUSDS5.13.1.4.1  DPUSDS5.13.2.4
DPUSDS5.12.1.3.1  DPUSDS5.12.2.1    DPUSDS5.13.1.5.1  DPUSDS5.13.2.5
DPUSDS5.12.1.3.2  DPUSDS5.12.2.2    DPUSDS5.13.1.5.2  DPUSDS5.13.2.6
DPUSDS5.12.1.3.3  DPUSDS5.12.2.3    DPUSDS5.13.1.5.3  DPUSDS5.13.2.7
DPUSDS5.12.1.4.1  DPUSDS5.12.2.4    DPUSDS5.13.1.5.4  DPUSDS5.13.2.8
DPUSDS5.12.1.4.2  DPUSDS5.12.3.1    DPUSDS5.13.1.6.1  DPUSDS5.13.2.9
DPUSDS5.12.1.4.3  DPUSDS5.13.0.1    DPUSDS5.13.1.6.2
DPUSDS5.12.1.4.4  DPUSDS5.13.0.2    DPUSDS5.13.1.6.3
DPUSDS5.12.1.4.5  DPUSDS5.13.1.1.1  DPUSDS5.13.1.6.4
DPUSDS5.12.1.5.1  DPUSDS5.13.1.1.2  DPUSDS5.13.1.7.1
[senthil@bluehat CM1subset]$
```

Figure 5.3: Input Format

```
<!ELEMENT req (high*)>

<!ELEMENT high (low*)>
<!ATTLIST high id
          ID #REQUIRED freeze ID #IMPLIED>
<!ELEMENT low (weight*)>
<!ATTLIST low id ID #REQUIRED>
<!ELEMENT weight (#PCDATA)>
<!ATTLIST weight change CDATA #IMPLIED>
```

Figure 5.4: DTD

requirement elements have been found. It may also mean that all the matching low level requirement elements have been found, but they do not address all aspects of the high level requirement element. "Postponed" means that the analyst does not currently want the tool to consider this particular high level requirement element while running the feedback process.

Each low tag contains a tag named weight that represents the relevance factor of the link under consideration. The weight tag contains an attribute named "change." This attribute can take the following values: "Default," "Link," "Not A Link." When the link is originally generated by the tool, "change" attribute has the value "Default." The analyst can modify the value to "Link," if he considers the link to be true. Otherwise, the analyst can mark it as "Not A Link." The feedback process will use this information in improving the candidate link list. Figure 5.6 shows a candidate link list with feedback information.

5.3 Measures

In this work, we focus on the following measures to track the quality of the candidate list generated by various methods.

5.3.1 Overall Precision and Overall Recall

These measures calculate how many true links have been captured and how many of the captured links are false positives. Overall recall is a ratio of the total number of true

```
<?xml version="1.0"?>
<!DOCTYPE req SYSTEM "masterDTD.dtd">
<req>
<high id = "SDP3.2-2" freeze = "Not Satisfied">
<low id = "L1APR01-I-2"><weight change = "Default">0.73523</weight></low>
<low id = "L1APR03-I-2"><weight change = "Default">0.54532</weight></low>
</high>
<high id = "SDP3.3-1" freeze = "Not Satisfied">
<low id = "L1APR01-I-1"><weight change = "Default">0.53278</weight></low>
<low id = "L1APR01-F-2.3-1"><weight change = "Default">0.04934</weight></low>
<low id = "L1APR01-F-2.2.4-1"><weight change ="Default">0.01959</weight></low>
</high>
<high id = "SDP3.3-2" freeze = "Not Satisfied">
<low id = "L1APR03-I-2"><weight change = "Default">0.95981</weight></low>
<low id = "L1APR01-I-2"><weight change = "Default">0.75901</weight></low>
<low id = "L1APR01-I-3"><weight change = "Default">0.45863</weight></low>
<low id = "L1APR01-F-2.4-1"><weight change = "Default">0.02765</weight></low>
<low id = "L1APR01-F-5.1-1"><weight change = "Default">0.02327</weight></low>
<low id = "L1APR01-F-2.4-2"><weight change = "Default">0.01988</weight></low>
<low id = "L1APR01-F-4-3"><weight change = "Default">0.01392</weight></low>
<low id = "L1APR01-F-2.2.4-1"><weight change ="Default">0.01169</weight></low>
</high>
<high id = "SDP3.3-4" freeze = "Not Satisfied">
<low id = "L1APR01-I-1"><weight change = "Default">0.75647</weight></low>
<low id = "L1APR01-I-2"><weight change = "Default">0.66885</weight></low>
<low id = "L1APR03-I-2"><weight change = "Default">0.56819</weight></low>
<low id = "L1APR01-I-3"><weight change = "Default">0.29109</weight></low>
<low id = "L1APR03-F-6.1-1"><weight change = "Default">0.19074</weight></low>
</high>
<high id = "SDP4.2-1" freeze = "Not Satisfied">
<low id = "L1APR01-F-4-3"><weight change = "Default">0.75421</weight></low>
<low id = "L1APR01-F-4-4"><weight change = "Default">0.55891</weight></low>
<low id = "L1APR01-F-1.2-1"><weight change = "Default">0.1975</weight></low>
<low id = "L1APR01-I-2"><weight change = "Default">0.1523</weight></low>
<low id = "L1APR03-I-2"><weight change = "Default">0.04532</weight></low>
</high>
</req>
```

Figure 5.5: Candidate Link List

```
<?xml version="1.0"?>
<!DOCTYPE req SYSTEM "masterDTD.dtd">
<req>
<high id = "SDP3.2-2" freeze = "Not Satisfied">
<low id = "L1APR01-I-2"><weight change = "Link">0.73523</weight></low>
<low id = "L1APR03-I-2"><weight change = "Default">0.54532</weight></low>
</high>
<high id = "SDP3.3-1" freeze = "Not Satisfied">
<low id = "L1APR01-I-1"><weight change = "Link">0.53278</weight></low>
<low id = "L1APR01-F-2.3-1"><weight change = "Not A Link">0.04934</weight></low>
<low id = "L1APR01-F-2.2.4-1"><weight change ="Link">0.01959</weight></low>
</high>
<high id = "SDP3.3-2" freeze = "Not Satisfied">
<low id = "L1APR03-I-2"><weight change = "Link">0.95981</weight></low>
<low id = "L1APR01-I-2"><weight change = "Link">0.75901</weight></low>
<low id = "L1APR01-I-3"><weight change = "Default">0.45863</weight></low>
<low id = "L1APR01-F-2.4-1"><weight change = "Not A Link">0.02765</weight></low>
<low id = "L1APR01-F-5.1-1"><weight change = "Default">0.02327</weight></low>
<low id = "L1APR01-F-2.4-2"><weight change = "Not A Link">0.01988</weight></low>
<low id = "L1APR01-F-4-3"><weight change = "Default">0.01392</weight></low>
<low id = "L1APR01-F-2.2.4-1"><weight change ="Default">0.01169</weight></low>
</high>
<high id = "SDP3.3-4" freeze = "Not Satisfied">
<low id = "L1APR01-I-1"><weight change = "Link">0.75647</weight></low>
<low id = "L1APR01-I-2"><weight change = "Link">0.66885</weight></low>
<low id = "L1APR03-I-2"><weight change = "Not A Link">0.56819</weight></low>
<low id = "L1APR01-I-3"><weight change = "Not A Link">0.29109</weight></low>
<low id = "L1APR03-F-6.1-1"><weight change = "Default">0.19074</weight></low>
</high>
<high id = "SDP4.2-1" freeze = "Not Satisfied">
<low id = "L1APR01-F-4-3"><weight change = "Link">0.75421</weight></low>
<low id = "L1APR01-F-4-4"><weight change = "Link">0.55891</weight></low>
<low id = "L1APR01-F-1.2-1"><weight change = "Not A Link">0.1975</weight></low>
<low id = "L1APR01-I-2"><weight change = "Link">0.1523</weight></low>
<low id = "L1APR03-I-2"><weight change = "Default">0.04532</weight></low>
</high>
</req>
```

Figure 5.6: Candidate Link List with Feedback Information

links found for all queries to the total number of possible true links for all queries. Overall precision is computed as the fraction of the true links in the list of all links returned for all queries.

5.3.2 Selectivity

While performing a requirements tracing task manually, an analyst should compare each high and low level pair. Selectivity shows the improvement of an IR method over this number. Selectivity can be defined as the ratio of the total number of candidate links returned for each high level requirement to the total number of high and low level pairs. Let H be the number of high level requirements and L be the number of low level requirements. For a given high level element h, n_q will be the number of candidate links generated. The selectivity can be calculated as follows:

$$selectivity = \frac{\sum_{q=1}^{M} n_q}{H \cdot L}. \tag{5.1}$$

5.3.3 Average Precision and Average Recall

Precision and recall values are calculated for each high level requirement (query) separately and the average of precision and recall for all the queries gives average precision and average recall, respectively. These measures give an indication of the average number of links captured for each query.

5.3.4 Average Expected Precision

Hayes et al. [27] mention that filtering the lists of candidate links using a set threshold α will show how many true links are at the top of the list. If there were two candidate lists with the same recall and precision, the one with more true links at the top of the list will be preferred to the other. Average expected precision allows for the effects of filtering to be incorporated into a single measurement of a candidate link list. It is a secondary measure to evaluate the quality of a given candidate list.

Let Q be a list of high level requirements, D be a list of low level requirements, and $L = \{\langle q, d\rangle, sim(q, d)\}$ be a list of candidate links from D for Q, then average expected precision of L is computed as follows. For each $q \in Q$ we consider the ordered list $D_q = (d_1, \ldots, d_s)$ of candidate links, such that $\langle q, d_i, sim(q, d_i)\rangle \in L$ and $sim(q, d_1) \geqslant sim(q, d_2) \geqslant \ldots \geqslant sim(q, d_s)$. Some of $d_1, \ldots, d_s$ are *relevant* to q while the remaining elements of D_q

are *false positives*. A *recall level* is any position $1 \leqslant i \leqslant s$ such that d_i is *relevant* (i.e., $\langle q, d_i \rangle$ is a true link).

An *expected precision* $Ep(q)$ of D_q is the mean of precisions computed at each recall level of D_q. The mean of expected precisions $Ep(q)$ for all $q \in Q$ gives the *average expected precision*.

For example, let us assume that there were candidate link lists for two high-level requirements (query) q_1 and q_2 with five elements each. Also, consider that in the list D_{q_1} recall levels are $1, 3$, and 5. In the list D_{q_2}, recall levels are $2, 4$, and 5. Then, for q_1, precisions at recall levels are $\frac{1}{1} = 1, \frac{2}{3} = 0.67$, and $\frac{3}{5} = 0.6$, and therefore $E_{q_1} = (1 + 0.67 + 0.6)/3 = 0.757$. For q_2, precisions at recall levels are $\frac{1}{2}, \frac{2}{4} = \frac{1}{2}$, and $\frac{3}{5}$. Hence $E_{q_2} = \frac{\frac{1}{2}+\frac{1}{2}+\frac{3}{5}}{3} = \frac{8}{15} = 0.533$. Therefore, the average expected precision for q_1 and q_2 is $\frac{0.757+0.5333}{2} = 0.645$. Figure 5.7 shows how to calculate average expected precision.

5.3.5 F-measure

An ideal candidate list should have 100% recall and 100% precision. Unfortunately, it is highly unlikely that a method will always produce an ideal candidate list. Often, recall of a candidate list can be improved by sacrificing some precision and vice-versa. Low recall means that many true links are missing in the candidate list while low precision means that the candidate list contains too many false positives. In order to fix a candidate list with low precision, an analyst has to go through the candidate list to identify the false positives and drop them from the list. To fix a candidate list with low recall, the analyst must notice that there are few links and manually search for them by going through low level and high level artifacts.

Consider a dataset with 100 low-level and 50 high-level artifacts with a candidate list containing about 500 links. If the candidate list has low precision, the analyst must go through 500 links. At the same time, if the candidate list has low recall, the analyst must go through approximately 4500 (100×50 less 500) links to identify the missing links. Clearly, identifying false positives is easier than finding missing links. Sundaram et al. [51] argue that achieving high recall is more important in tracing tasks than achieving the highest possible precision.

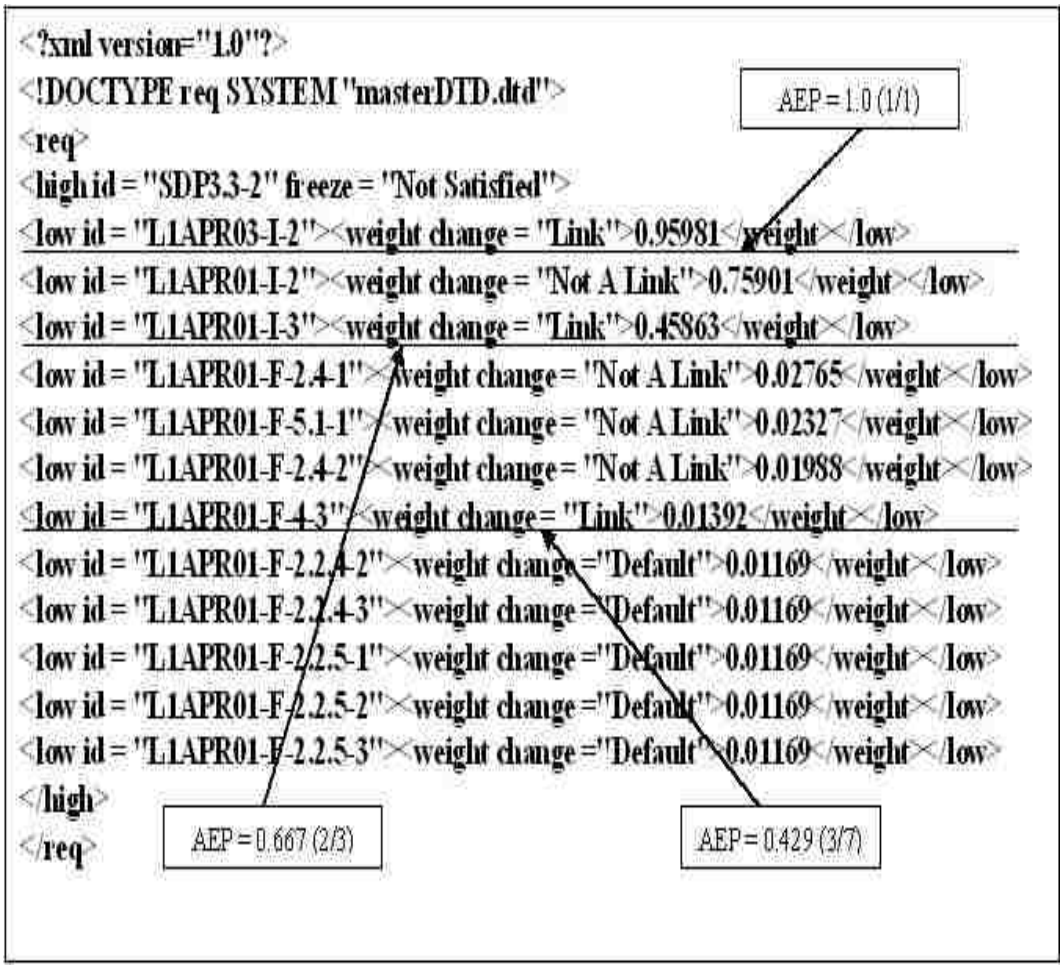


Figure 5.7: Average Expected Precision Calculation

F-measure is defined as a harmonic mean of precision and recall:

$$F = \frac{2 \cdot precision \cdot recall}{precision + recall}. \tag{5.2}$$

It can be seen that achieving high precision and high recall is a balancing act. The above mentioned formula puts equal preference to both recall and precision. The β parameter is introduced in the above formula to tilt the balance one way or the other. The parameter β can be altered to set desirable significance for either recall or precision:

$$F_\beta = \frac{(\beta + 1) \cdot precision \cdot recall}{recall + \beta \cdot precision}. \tag{5.3}$$

If $\beta > 1$, the recall will be valued more than precision, and if $\beta < 1$, the precision will be valued more than recall.

5.3.6 Analysis Tool

We implemented an analysis tool in JAVA to automatically generate the above mentioned measures. The tool takes the XML file generated by RETRO and the answerset as inputs and prints the measures to a text file. We use a script to run the analysis tool on all the results generated by RETRO at once. The analysis tool can also generate results in different formats convenient for analysis and plotting graphs.

5.4 Experimental Setup

The objectives of our experiments were to determine whether our methods are capable of producing accurate tracing results, to determine whether our methods are capable of separating true links from false positives as a result of a feedback process, to identify if one method performs better than other methods for datasets of a particular size and characteristics, and to observe if the methods scale well.

Table 5.2 shows the list of IR methods that have been implemented. In our experiments, each high level element and low level element was stored in a separate file. The names of the files are the unique identifiers of the corresponding low level element and high level element. First, for each dataset, all the high level elements and low level elements were extracted from the original documents and saved in a format readable by RETRO.

The words that do not add any significance to the characteristics of the elements, called stopwords, were removed. If the stopwords had not been removed, this might contribute to the generation of coincidental links. For example, two elements might be termed

Method
Vector space retrieval (tf-idf)
Vector space retrieval plus thesaurus
Keyword extraction
IDF
LSI
Probabilistic IR

Table 5.2: Information Retrieval Methods

as related just because they both had the words "and," "the," and "a." In order to avoid this situation, common stopwords such as "and," "a," "the," etc. were removed. Finally, the elements were stemmed using Porter's algorithm [36]. The purpose of stemming keywords to their root is to convert all forms of the same word to a single form. For example, "computed," "computable," and "compute" were all stemmed to "comput." After the preprocessing stage, each high level element and low level element was converted to a vector of term weights.

Once the vectors were created, the selected IR method went through each high level element vector and low level element vector to produce the list of candidate links for each high level element. When an analyst uses RETRO to trace a pair of software artifacts, the candidate list generated is displayed in the RETRO GUI. The analyst goes through the candidate list and provides feedback on each link. The feedback information is then fed to the feedback process and the candidate list is corrected accordingly. In other words, the feedback method looks for more of the links similar to the ones that are termed as "Link" by the analyst. Similarly, the links similar to the ones termed as "Not A Link" by the analyst are discarded.

All the datasets used in our experiments have a true answerset verified by an analyst. For experimental purposes, we replaced the human analyst with an automated feedback simulator. The feedback simulator was provided with a copy of the answerset for the dataset under consideration. The feedback simulator acted as an ideal analyst. In other words, the feedback provided by the simulator was always correct.

5.4.1 Feedback

We studied four different feedback strategies: Top 1, Top 2, Top 3, and Top 4. Using strategy Top i, the feedback process examined, for each high level element, the top i unexamined candidate links in the list, and specified whether each examined link is a true link or a false positive. This information was used by the feedback process to update the query vectors appropriately. Once query (high level element) vectors were updated, the selected IR method went through each high level element vector and low level element vector and regenerated the candidate list. Each experiment ran eight iterations of the above mentioned process.

5.4.2 Filter

We also produced candidate link lists with relevance higher than one of the predefined levels: 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, and 0.5. We call this process "filtering." Filtering offers an insight into the quality of the candidate link list. Let us say that two candidate link lists, list A and list B, have the same recall and precision. If the true links appear at the top of list A compared to list B, then clearly list A will be preferred to list B by an analyst. Figure 5.8 shows how filtering works. For the filter value 0.8, only the links above the first line will be included in the candidate list. Similarly, for the filter value 0.4, the links above the second line will be included in the candidate list and for the filter value 0.1, the links above the third line will be included.

As mentioned earlier, the ideal candidate link list needs to achieve 100% recall and 100% precision. However, it is very hard to achieve the ideal result. Instead, we prefer to have as high precision and recall as possible. We also calculated the F-measure for all the candidate link lists generated. We calculated the F-measure with β value 1, 2, and 3. This allows us to identify which IR method produced results with better harmonic mean of precision and recall.

5.4.3 Thesaurus

We analyzed the tf-idf method and tf-idf+thesaurus method to identify if a thesaurus impacts the result significantly. We also included the feedback process in these experiments to see if there is any difference in the results generated by the tf-idf method and tf-idf+thesaurus method after feedback processing. This shows us if it is worth spending

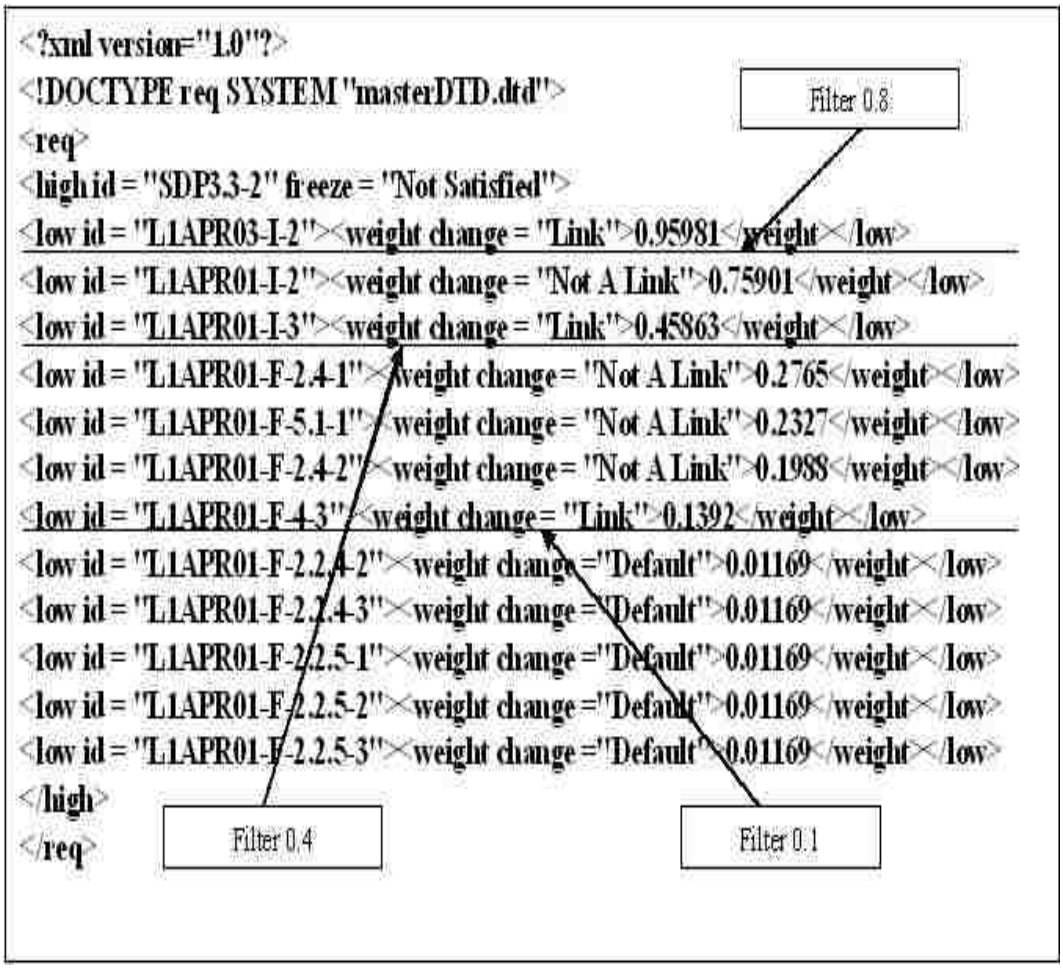


Figure 5.8: Filter Example

the time and effort to build a thesaurus. If the same results can be achieved without using a thesaurus but with minimal feedback, it might be better to opt for the latter option.

5.4.4 Vocabulary Base

In the IR world, only the document collection is used to build the vocabulary base because the query is not known beforehand. However, in the case of the requirements tracing process, we know both the document and the query collection in advance. We can use this additional information to build the vocabulary base. In other words, both the documents and the queries can be used to build the vocabulary base. We ran experiments with both types of vocabulary bases to identify if they produce significantly different results.

5.4.5 Weight Method

We extended the vector space model (tf-idf) by modifying the computation of term weights in the document and query and the computation of similarity between the document and query vectors. Totally, we used three weighting schemes in our experiments. In addition to the regular tf-idf weighting scheme, we also used the *Okapi* [46] and *LTU* [33] weighting schemes.

5.4.6 Voting Tool

We also ran experiments with the voting tool. We formed a committee of methods with all the methods implemented in the RETRO tool box. We came up with different rules to see which one works better. We ran experiments with majority, parity, and consensus rules. The results from these experiemnts have been published in [14]. The rules we used can be defined as follows:

- majority - at least more than half the number of methods need to identify the link,
- parity - exactly half the number of methods or more need to idenfity the link, and
- consensus - all the methods need to identify the link.

The voting tool allows new rules to be framed. The consensus rule may help reduce the false positives significantly, but at the same time it is quite possible that it may also throw away true links. At the same time, including a link as a candidate link if it is found by at least one of the methods may reduce the precision significantly. We believe that picking

Study	Dataset	Measure
Feedback for small datasets	Modis	F-measure
Feedback for large datasets	CM1	F-measure
Feedback with filter for small datasets	Modis	F-measure
Feedback with filter for large datasets	CM1	F-measure
Comparison of weight methods	CM1	Recall
Comparison of weight methods	CM1	Precision

Table 5.3: Empirical Study List

the correct rule depends on the needs of the type of trace being performed. It should be noted that the rules discussed above give equal weight to all the methods.

5.4.7 Empirical Assessment

We conducted an empirical study to assess the effect of using feedback and the effect using feedback with filter. We also conducted an empirical study to assess the effectiveness of the tf-idf, Okapi, LTU weight methods. Totally we conducted six studies as shown in Table 5.3.

In the first study, we compared the F-measure obtained for the MODIS dataset by the tf-idf method, tf-idf + thesaurus, KE, IDF, LSI, and probabilistic IR at iteration 0 (when no feedback was used) and the F-measure obtained by these methods at iteration 8 (when the feedback was used). In the second study, we compared the F-measure obtained for the CM1 dataset by the tf-idf method, KE, IDF, LSI, and probabilistic IR at iteration 0 and the F-measure obtained by these methods at iteration 8. The third and the fourth studies were similar to the first and the second studies, respectively, except that we also combined the feedback with filter. We used the F-measure value obtained at iteration 8 with filter value 0.15. In fifth and the sixth studies, we compared the recall and precision obtained by the tf-idf, Okapi, and LTU weighting schemes, respectively.

Dependent and Independent Variables

The independent variable for the first two studies was the use of feedback. The independent variable for the third and the fourth study was the use of feedback and filter. The independent variable for the last two studies was the weighting schemes used. The dependent variable in the first four studies was the F-measure. The dependent variables in the fifth and sixth studies were the recall and the precision, respectively.

Hypotheses

The null and alternative hypotheses for these studies are:

- H_{01} - There will be no difference in the F-measure obtained for the MODIS dataset by using feedback and by not using feedback.
- H_{A1} - There will be a difference in the F-measure obtained for the MODIS dataset by using feedback and by not using feedback.
- H_{02} - There will be no difference in the F-measure obtained for the CM1 dataset by using feedback and by not using feedback.
- H_{A2} - There will be a difference in the F-measure obtained for the CM1 dataset by using feedback and by not using feedback.
- H_{03} - There will be no difference in the F-measure obtained for the MODIS dataset by using feedback and filter and by not using feedback and filter.
- H_{A3} - There will be a difference in the F-measure obtained for the MODIS dataset by using feedback and filter and by not using feedback and filter.
- H_{04} - There will be no difference in the F-measure obtained for the CM1 dataset by using feedback and by not using feedback.
- H_{A4} - There will be a difference in the F-measure obtained for the CM1 dataset by using feedback and by not using feedback.
- H_{05} - There will be no difference in the recall obtained by tf-idf, Okapi, and LTU weighting schemes.
- H_{A5} - There will be a difference in the recall obtained by tf-idf, Okapi, and LTU weighting schemes.
- H_{06} - There will be no difference in the precision obtained by tf-idf, Okapi, and LTU weighting schemes.
- H_{A6} - There will be a difference in the precision obtained by tf-idf, Okapi, and LTU weighting schemes.

Statistical Analysis

We used paired t-test to analyze the effect of using feedback and the effect of using feedback and filter together. Hence, we used paired t-test to validate the null hypotheses H_{01}, H_{02}, H_{03}, and H_{04}. For the paired t-test, we assumed that our data was obtained from a Gaussian distribution. We used Kruskal-Wallis test to analyze the effect of using different weighting methods. As the recall and the precision values generated by the different weighting methods did not follow a normal distribution, we could not use Analysis of Variance (ANOVA). We performed all pairwise comparisons using Dwass-Steel-Chritchlow-Fligner and Conover-Inman. Hence, we used Kruskal-Wallis to validate the null hypotheses H_{05} and H_{06}.

The factor for the first two studies was the use of feedback and the factor for the third and the fourth studies was the use of feedback with filter. Whereas, the factor for the fifth and the sixth studies was the weight method used. The treatments in all the studies were the IR methods used. In the case of paired t-test, if the t value was less than $t_{critical}$ for a given α value, the null hypothesis was accepted. If the t value was greater than $t_{critical}$ for a given α value, the null hypothesis was rejected and the alternative hypothesis was accepted. In our case studies, we used 0.05 as α value.

Threats to Validity

Let us look at various threats to the validity of our experiments.

External Validity The IR methods used in this work do no depend on any domain specific information. However, the thesaurus used in the tf-idf method is domain specific, but the tf-idf method can be run without a thesaurus. The stopword file we used in our experiments included the stopwords for the English language only. In order to use our approach to trace requirements that are not written in English, a new stopword file may have to be built.

Internal Validity For our feedback experiments, we simulated the analyst behavior using the answerset. Hence we avoided validity threats due to human factors. The answersets used to calculate the F-measure, recall and precision were built by human analysts. Hence, there was a potential for bias. Also, the answersets may not be accurate. However, the answersets were verified by multiple analysts to address these concerns. LSI typically works well for large datasets with thousands and millions of documents. The largest dataset we

used had only 235 high-level elements and 220 low-level elements. Hence, it is possible that the results obtained for LSI may not show the actual capability of LSI.

Construct Validity Okapi and LTU weighting schemes use document length as a factor in calculating the weight for each keyword. All the datasets used in our work had documents that were only two or three sentences long. For these weighting schemes to work well, it may be necessary to have documents that are longer than two or three sentences.

Conclusion Validity The paired t-test requires that the data points used be drawn from a Gaussian distribution. We assume that our data points follow a Gaussian distribution. This is a threat to the validity of our empirical analysis.

5.4.8 Objectives

All the results were archived. The analysis tool was used to collect metrics from the results. We conducted various experiments on our datasets using all the IR methods that we adopted to identify the quality of the candidate lists generated. As explained earlier in this section, the objectives of these experiments were quite different. We can summarize our experiments based on their objectives as follows:

- Compare the quality of the results generated by all the IR methods on each of the datasets,
- Compare the effect of the feedback process using all the IR methods,
- Analyze the effect of the thesaurus,
- Use filters to better quantify the results,
- Compare the effect of different weighting schemes,
- Compare different vocabulary bases, and
- Conduct empirical study and analyze the results.

In the following chapter, we report the most interesting results obtained in our experiments. The results of the empirical studies are also discussed in the following chapter. More results and graphs can be found in the appendix section.

Chapter 6

Results

Hayes et al. [30] discuss what constitute good values for recall and precision. Obtaining high recall is more significant than obtaining high precision, as it is harder for an analyst to search for the links that were not found than to identify the false positives. Hayes et al. [30] consider 60% - 69% recall as acceptable, 70% - 79% recall as good, and 80% - 100% as excellent. Similarly, they consider 20% - 29% precision as acceptable, 30% - 49% precision as good, and 50% - 100% precision as excellent. We will use the classication proposed by Hayes et al. [30] to assess our results.

Before we start analyzing the results, let us look at the abbreviations used in the tables. Following is the list of abbreviations used in the tables:

- Rec - Recall,
- Pr - Precision,
- Sel - Selectivity,
- AEPr - Average Expected Precision,
- AvPr - Average Precision,
- AvRec - Average Recall,
- f1 - F-measure with $\beta = 1$,
- f2 - F-measure with $\beta = 2$, and
- f3 - F-measure with $\beta = 3$.

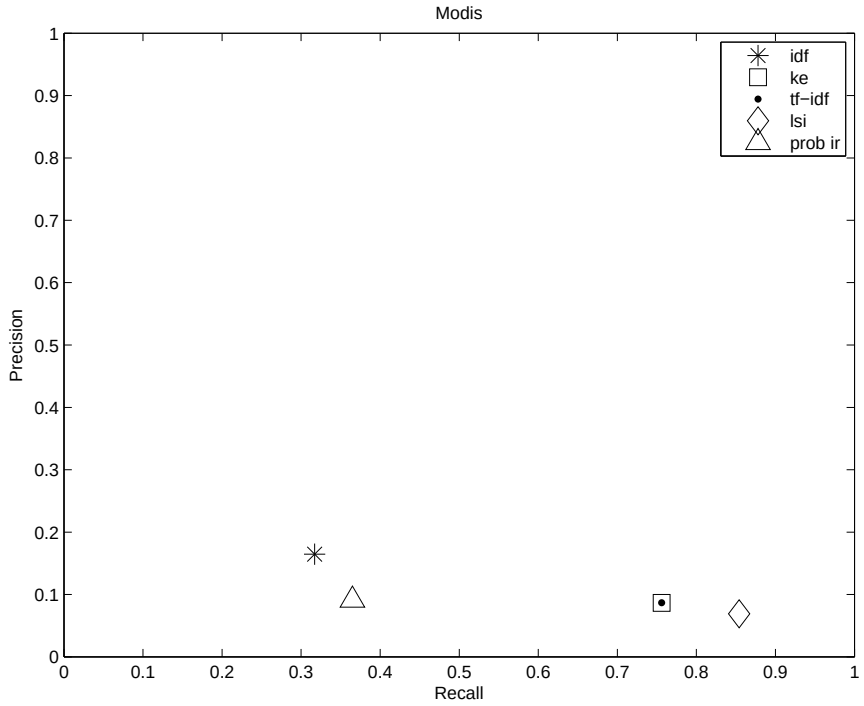


Figure 6.1: Recall vs Precision - Modis

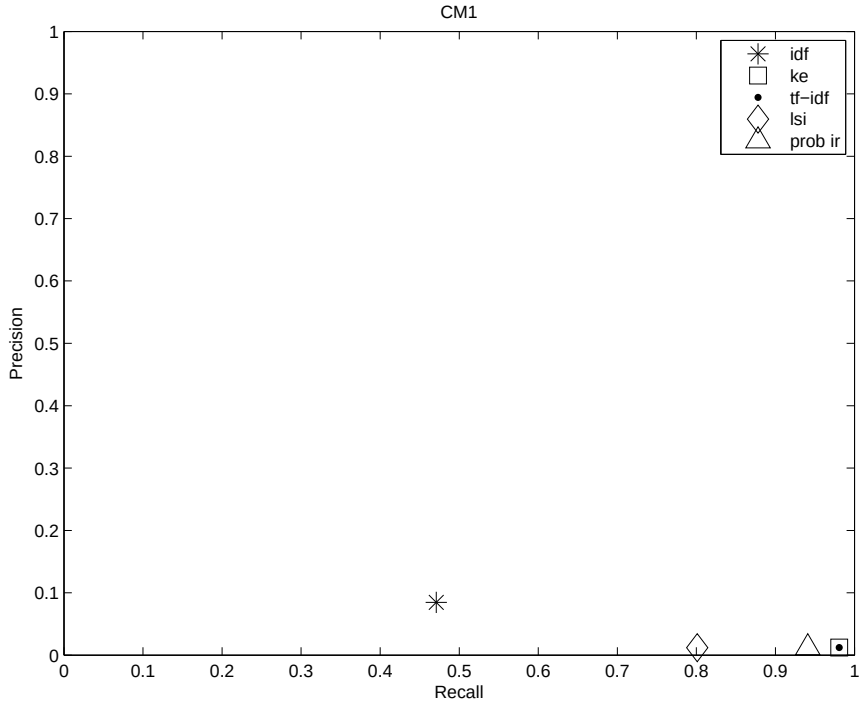


Figure 6.2: Recall vs Precision - CM1

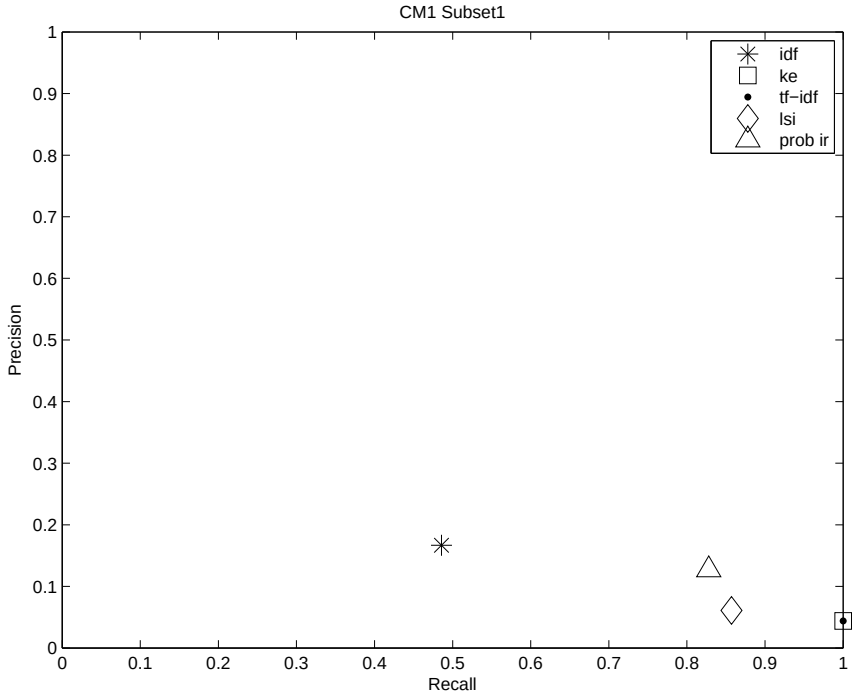


Figure 6.3: Recall vs Precision - CM1 Subset1

6.1 Analysis of IR methods

Table 6.1 and Table 6.2 show the performance of the tf-idf method, KE, LSI, probabilistic IR, and IDF on all the datasets. Evidently, IDF produced very low recall with fair precision. It can be noticed that there was no difference in the performance of the tf-idf method and KE. They both produced the same recall and precision values except for the CM1 Subset 2 dataset. For the Modis dataset, the tf-idf method and KE were able to achieve good recall with poor precision. For the CM1 dataset and its subsets, the tf-idf method and KE produced excellent recall and poor precision.

Figure 6.1, Figure 6.2, Figure 6.3, Figure 6.4, Figure 6.5, and Figure 6.6 show the recall vs. precision graphs for the tf-idf method, KE, LSI, probabilistic IR, and IDF running on all the datasets. Also, Figure 6.7, Figure 6.8, Figure 6.9, Figure 6.10, Figure 6.11, and Figure 6.12 show the average recall vs. average precision graphs for the tf-idf method, KE, LSI, probabilistic IR, and IDF running on all the datasets. Average recall and average precision for the tf-idf method and KE behaved the same way as recall and precision. The tf-idf method and KE produced the same average recall and average precision for the Modis, CM1, CM1 Subset1, CM1 Subset3, and CM1 Subset4. However, for the CM1 Subset2, the

tf-idf method produced slightly better average recall and average precision.

6.1.1 Observations

- It can be seen clearly that keyword extraction using IDF performs poorly compared to other methods, especially keyword extraction using χ^2. Hence, it can be concluded that using IDF to identify important keywords in the document collection is not a useful idea.
- All the methods are able to produce good or excellent recall and average recall values.
- None of the methods is able to produce good precision and average precision values.

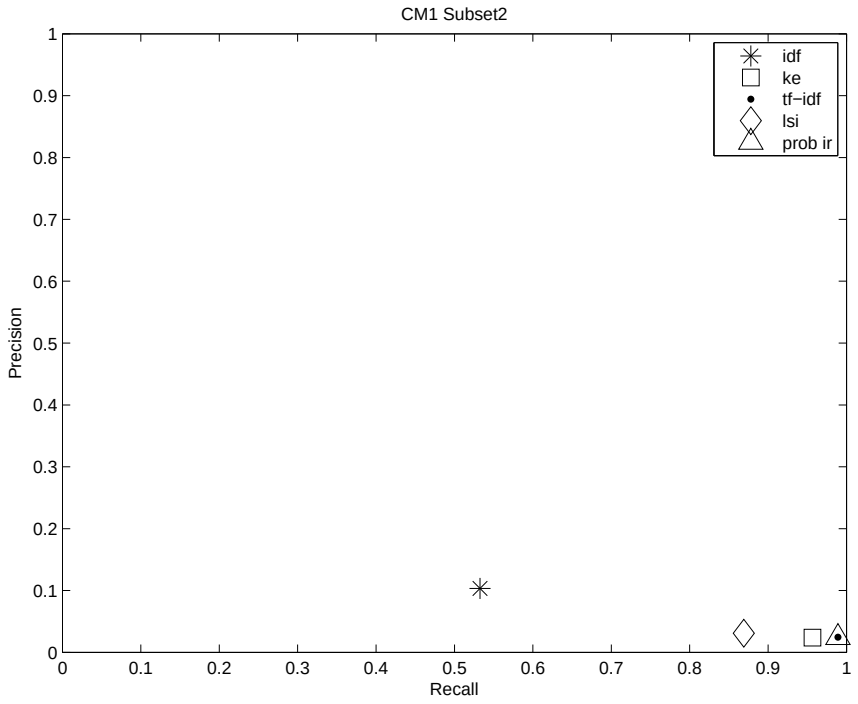


Figure 6.4: Recall vs Precision - CM1 Subset2

6.2 Feedback Analysis

We can see that all the methods were able to produce candidate link lists with good recall, but with poor precision. We believe that we can use the feedback process to address this issue. In our experiments, we found the Top 2 strategy to be a good balance of quality of results and amount of feedback per iteration. Results for the Top 1 strategy

Dataset:	**Modis**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.756	0.087	0.383	0.775	0.113	0.768	0.155	0.297	0.426
tf-idf+thesaurus	1.0	0.11	0.401	0.791	0.132	1.0	0.197	0.381	0.552
KE	0.756	0.087	0.384	0.787	0.114	0.769	0.155	0.297	0.426
IDF	0.317	0.164	0.084	0.833	0.225	0.611	0.216	0.267	0.290
LSI	0.854	0.069	0.544	0.815	0.093	0.977	0.242	0.261	0.400
Probabistic IR	0.365	0.091	0.176	0.739	0.231	0.660	0.194	0.228	0.281

Dataset:	**CM1**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.980	0.012	0.565	0.639	0.022	0.990	0.023	0.057	0.109
KE	0.980	0.012	0.565	0.624	0.022	0.990	0.023	0.057	0.109
IDF	0.470	0.084	0.038	0.775	0.177	0.539	0.143	0.246	0.323
LSI	0.801	0.012	0.464	0.591	0.018	0.818	0.027	0.057	0.106
Probabilistic IR	0.941	0.012	0.542	0.534	0.020	0.952	0.029	0.058	0.109

Dataset:	**CM1 Subset1**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	1	0.043	0.697	0.706	0.062	1	0.084	0.186	0.314
KE	1	0.043	0.697	0.711	0.062	1	0.084	0.186	0.314
IDF	0.485	0.166	0.089	0.886	0.304	0.505	0.248	0.351	0.407
LSI	0.857	0.061	0.430	0.668	0.087	0.838	0.140	0.237	0.371
Probabilistic IR	0.828	0.127	0.199	0.707	0.174	0.822	0.219	0.394	0.534

Table 6.1: Results Obtained by the Tf-idf, Tf-idf with Thesaurus, Keyword Extraction, IDF, LSI and Probabilistic Methods on Modis, CM1, and CM1 Subset1

Dataset:	**CM1 Subset2**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.989	0.024	0.667	0.648	0.066	1	0.047	0.111	0.200
KE	0.956	0.023	0.663	0.611	0.042	0.954	0.046	0.108	0.195
IDF	0.532	0.103	0.085	0.814	0.26	0.600	0.173	0.291	0.376
LSI	0.869	0.031	0.458	0.550	0.050	0.889	0.703	0.137	0.237
Probabilistic IR	0.989	0.024	0.660	0.526	0.067	1.0	0.598	0.112	0.202

Dataset:	**CM1 Subset3**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.991	0.014	0.548	0.742	0.038	0.994	0.029	0.069	0.129
KE	0.991	0.014	0.548	0.725	0.038	0.994	0.029	0.069	0.129
IDF	0.589	0.094	0.050	0.847	0.280	0.605	0.163	0.288	0.387
LSI	0.846	0.015	0.459	0.674	0.029	0.873	0.036	0.069	0.129
Probabilistic IR	0.854	0.040	0.123	0.636	0.105	0.879	0.073	0.171	0.285

Dataset:	**CM1 Subset4**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.983	0.013	0.578	0.676	0.028	0.991	0.026	0.064	0.120
KE	0.983	0.013	0.578	0.664	0.028	0.991	0.026	0.064	0.120
IDF	0.485	0.081	0.047	0.769	0.185	0.550	0.139	0.243	0.324
LSI	0.799	0.013	0.462	0.606	0.022	0.824	0.029	0.064	0.119
Probabilistic IR	0.815	0.040	0.159	0.571	0.082	0.858	0.075	0.170	0.281

Table 6.2: Results Obtained by the Tf-idf, Tf-idf with Thesaurus, Keyword Extraction, IDF, LSI, and Probabilistic IR Methods on CM1 Subset2, CM1 Subset3, and CM1 Subset4

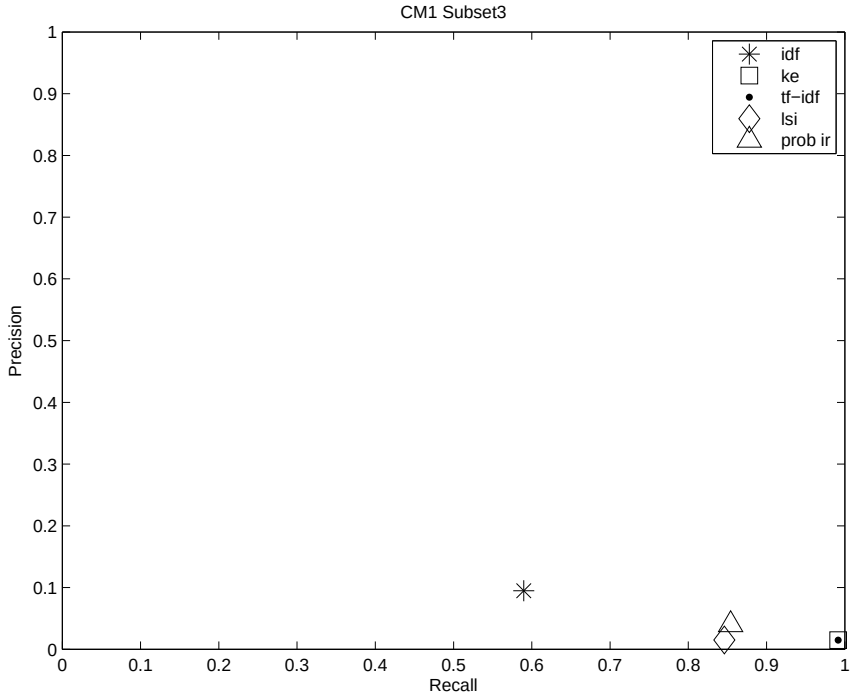


Figure 6.5: Recall vs Precision - CM1 Subset 3

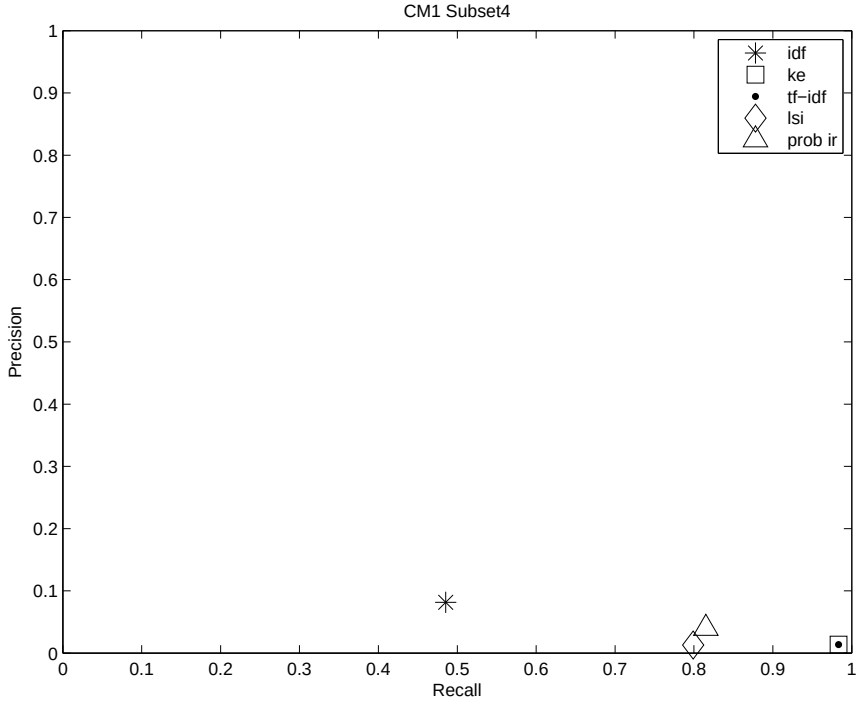


Figure 6.6: Recall vs Precision - CM1 Subset 4

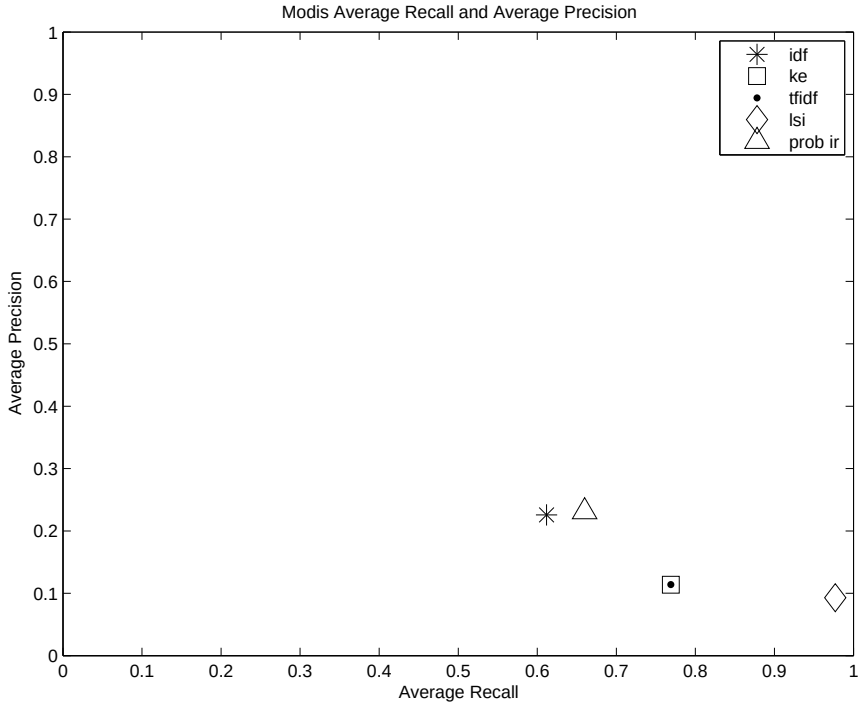


Figure 6.7: Average Recall vs Average Precision - Modis

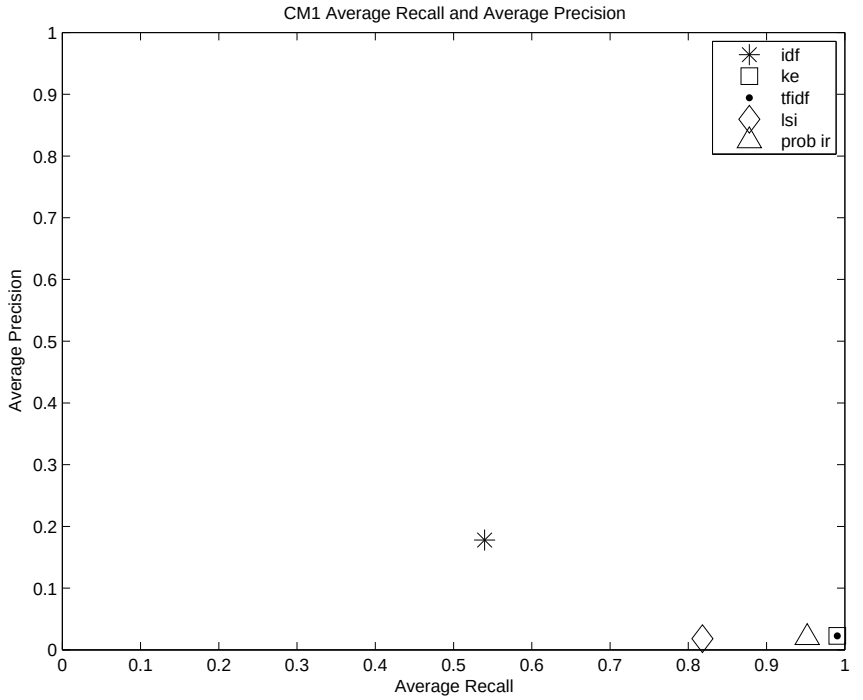


Figure 6.8: Average Recall vs Average Precision - CM1

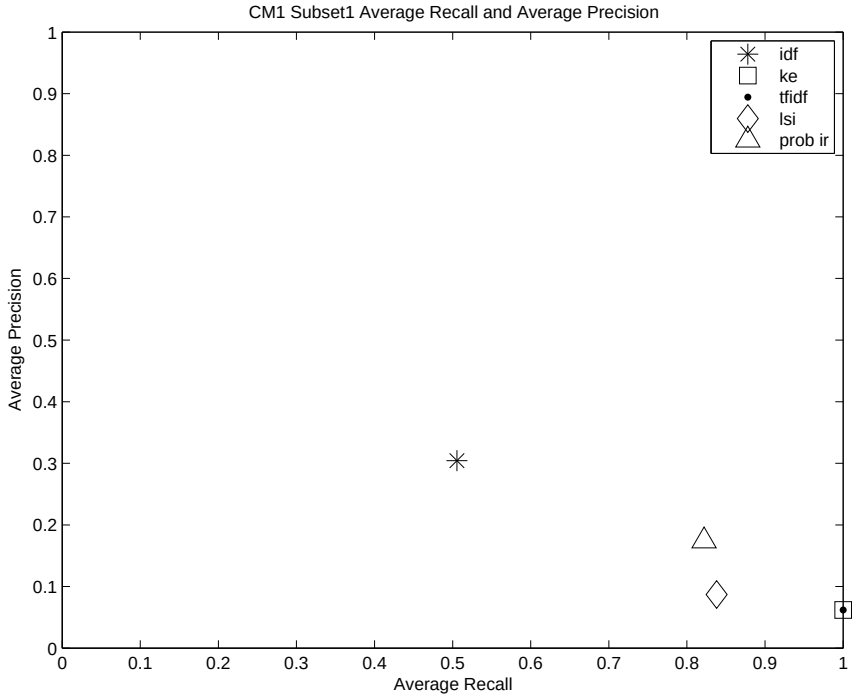


Figure 6.9: Average Recall vs Average Precision - CM 1 Subset1

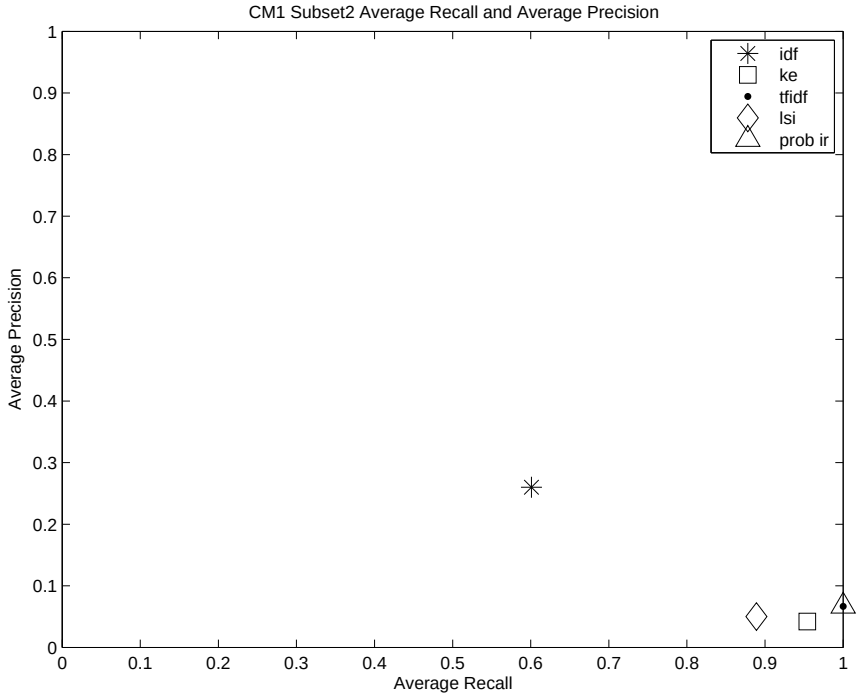


Figure 6.10: Average Recall vs Average Precision - CM 1 Subset2

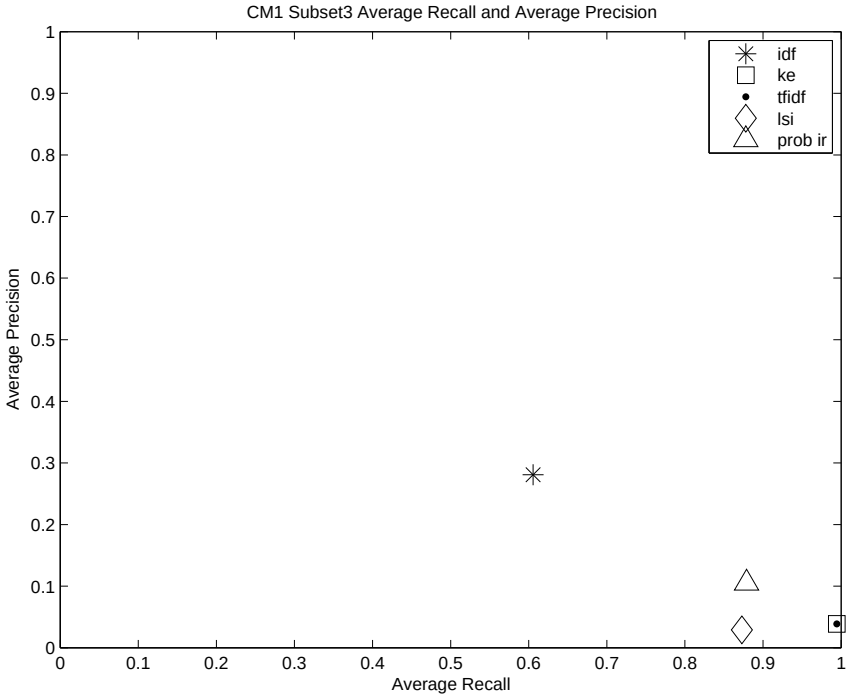


Figure 6.11: Average Recall vs Average Precision - CM 1 Subset3

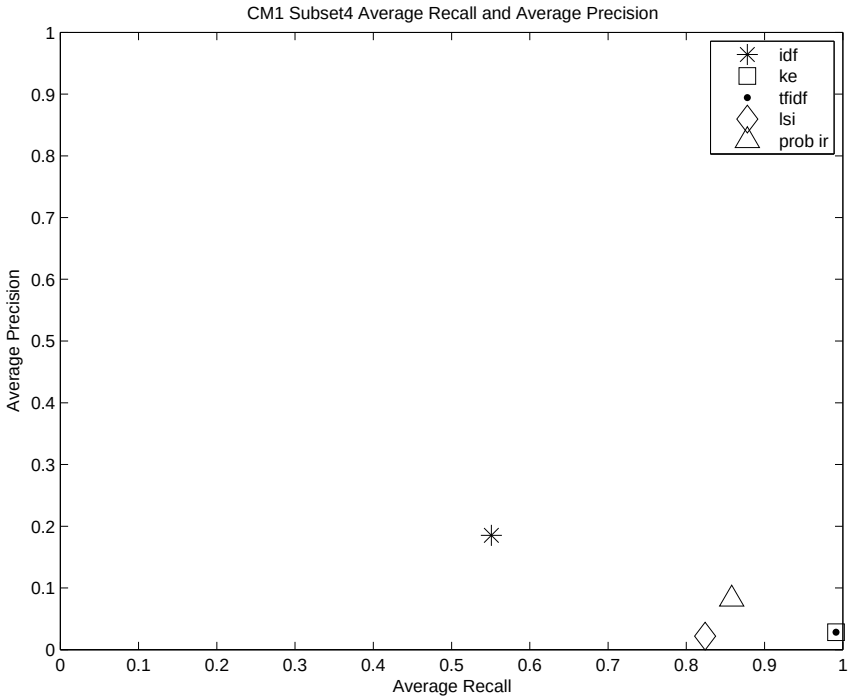


Figure 6.12: Average Recall vs Average Precision - CM 1 Subset4

were significantly worse, while results for Top 3 tended to be very similar to those of Top 2, occasionally reaching the same precision/recall numbers one or two iterations earlier. Hence, we only include the Top 2 strategy results in this report.

Table 6.5 shows all the results achieved by the tf-idf method on the Modis, CM1, and CM1 Subset1, when feedback was used. It can be seen that recall and precision improved at each iteration. Table 6.6 and Table 6.7 shows the results obtained at the eighth iteration of the feedback process. For the Modis and CM1 Subset1, the precision improved significantly, but for the other datasets, the increase in precision was insignificant.

Figure 6.13 and Figure 6.14 show the recall and precision values obtained over iterations for all the datasets for tf-idf with feedback. For the MODIS dataset, the recall value reduces slightly, but the precision goes from poor to acceptable. For CM1 and CM1 Subset1, recall improved significantly. For CM1, the precision did not improve significantly. For CM1 Subset1, the precision value doubled from 5% to 10%. There was no significant improvement in recall and precision for CM1 Subset2, CM1 Subset3, and CM1 Subset4.

Figure 6.15 and Figure 6.16 show the f-measure obtained for all the datasets at each iteration of the feedback. The f-measure increased from 0.3 to 0.62 for the MODIS dataset. The f-measure increased significantly for the CM1 Subset1 also. However, there was no significant increase in the f-measure for CM1 Subset2, CM1 Subset3, CM1 Subset4, and CM1 datasets.

Figure 6.17 and Figure 6.18 show the effect of feedback on average recall and average precision for all the datasets. For the MODIS dataset, we were able to achieve 40% average precision while maintaining the average recall at 75%. Similarly, we were able to achieve 22% average precision while maintaining the average recall at 95%. There was no significant increase in the average recall and the average precision values for CM1 Subset2, CM1 Subset3, CM1 Subset4, and CM1 datasets.

Figure 6.19 and Figure 6.20 show the average expected precision values obtained at each iteration of the feedback process. For the MODIS and CM1 datasets, the average expected precision improved from around 78% to 97%-99%. Whereas, for the CM1 Subset1, the average expected precision improved from 70% to 97%. Similarly, for the CM1 Subset2, CM1 Subset3, and CM1 Subset4, we were able to achieve around 95% average expected precision.

Figure 6.20 compares the selectivity and f-measure (f2) obtained at each iteration of the feedback for the Modis, CM1, and CM1 Subset1 datasets. Remember that we want to obtain as low selectivity as possible with as high f-measure as possible. Figure 6.20 As desired, the selectivity decreases significantly over the iterations, while the f-measure increases. Figure 6.20 compares the selectivity and f-measure (f2) obtained at each iteration of the feedback for the CM1 Subset2, CM1 Subset3, and CM1 Subset4 datasets. Notice that, for the CM1 subset2, CM1 subset3, and CM1 subset4, the selectivity and f-measure follow the same pattern as with the other three datasets.

6.2.1 Statistical Analysis

Table 6.3 and Table 6.4 show the data used for the statistical analysis and the result of the studies using the MODIS and CM1 datasets, respectively. Now let us analyze the result to see if the feedback increased the F-measure.

We used the F-measure obtained by five IR methods listed in Table 6.3 at iteration 0 and iteration 8. The t-statistic obtained for the first study using MODIS dataset was -5.4006. We carried out paired t test for α value 0.05. The degree of freedom for this test was 5. The $t_{critical}$ value obtained for one-tail test was 0.00147 and the $t_{critical}$ value obtained for two-tail test was 0.00294. We can see that, in the case of one-tail and two-tail tests, $|t| > t_{critical}$. Hence, we reject the null hypothesis H_{01} and accept the alternative hypothesis H_{A1}. Hence, In the case of the MODIS dataset, the difference between the F-measure of the candidate link lists obtained by using feedback and the F-measure of the candidate link lists obtained by not using feedback was statistically significant. The Pearson correlation shows a strong positive correlation between the F-measure obtained at iteration 0 and the F-measure obtained at iteration 8.

We used the F-measure obtained by tf-idf, KE, IDF, LSI, and probabilistic IR at iteration 0 and iteration 8. Table 6.4 shows that the t-statistic obtained for the second study using CM1 dataset was 0.51197. In this study, we used 0.05 as our α value. The degree of freedom for this test was 4. The $t_{critical}$ value obtained for one-tail test was 2.1318 and the $t_{critical}$ value obtained for two-tail test was 2.7764. In the case of one-tail as well as two-tail tests, $|t| < t_critical$. Hence, we cannot reject the null hypothesis H_{02}. Hence, In the case of the CM1 dataset, the difference between the F-measure of the candidate link lists obtained by using feedback and the F-measure of the candidate link lists obtained by not using feedback was not statistically significant. We can conclude that, in the case of

Method	Iteration 0	Iteration 8
Tf-idf	0.297	0.616
Tf-idf + Thesaurus	0.381	0.634
KE	0.297	0.614
IDF	0.267	0.363
LSI	0.261	0.392
Probabilistic IR	0.228	0.39

Groups	Observations	Sum	Mean	Variance
Iteration 0	6	1.731	0.2885	0.002716
Iteration 8	6	3.009	0.5015	0.017386

Metric	Value
Pearson Correlation	0.783543
df	5
t Stat	-5.400574263
P($T <= t$) one-tail	0.00147045
$t_{critical}$ one-tail	2.015048372
P($T <= t$) two-tail	0.0029409
$t_{critical}$ two-tail	2.570581835

Table 6.3: Modis - No Feedback vs. Feedback - Paired T-test

CM1 dataset, using feedback did not increase the F-measure. Notice that, similar to the first study, the Pearson correlation shows a very strong positive correlation between the F-measure obtained at iteration 0 and the F-measure obtained at iteration 8.

6.2.2 Observations

- All the measures improved significantly for the MODIS and CM1 Subset1 datasets. It should be noticed that these are the smallest of the six datasets. It shows that feedback works really well for small datasets.

- Recall, precision, f-measure, average recall, and average precision did not improve significantly for CM1 Subset2, CM1 Subset3, CM1 Subset4, and CM1 datasets. Hence, feedback did not really help in improving these measures for the large datasets.

- Average expected precision improved significantly for all the datasets. Recollect that

Method	Iteration 0	Iteration 8
Tf-idf	0.057	0.057
KE	0.057	0.057
IDF	0.246	0.208
LSI	0.057	0.066
Probabilistic IR	0.058	0.065

Groups	Observations	Sum	Mean	Variance
Iteration 0	5	0.475	0.095	0.0071
Iteration 8	5	0.453	0.0906	0.0043

Metric	Value
Pearson Correlation	0.998051072
df	4
t Stat	0.51197414
P($T <= t$) one-tail	0.317819826
$t_{critical}$ one-tail	2.131846782
P($T <= t$) two-tail	0.635639652
$t_{critical}$ two-tail	2.776445105

Table 6.4: CM1 - No Feedback vs. Feedback - Paired T-test

the high average expected precision value means that the true links occur at the top of the candidate list. Hence, it can be concluded that feedback helps in improving the similarity measures of the true links.

- Feedback decreases selectivity while increasing the f-measure. Hence, feedback improves the qualify of the trace results.
- The statistical analysis shows that, in the case of the MODIS dataset, there is a significant increase in the F-measure when the feedback was used. However, in the case of the CM1 dataset, there is no significant increase in the F-measure. Hence, we can see that the feedback increases the quality of the traces in the case of small datasets, but not in the case of large datasets.

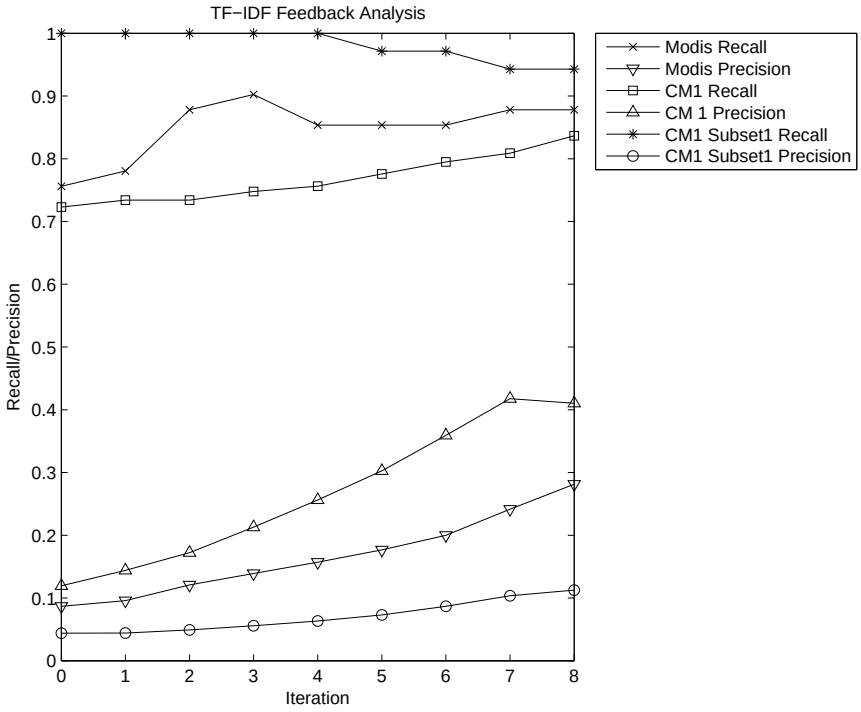


Figure 6.13: Feedback Analysis - Recall/Precision - Modis, CM1 and CM1 Subset1

6.3 Filter Analysis

Now, let us analyze the filtering results. Table 6.10 and Table 6.11 display the results obtained by the tf-idf method on all the datasets using the predefined filter values as mentioned in the experimental setup section. Note that these results were obtained without using the

Dataset:	**Modis**								
Iteration	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0	0.756	0.087	0.383	0.775	0.114	0.769	0.156	0.298	0.427
1	0.780	0.096	0.359	0.833	0.113	0.773	0.171	0.321	0.455
2	0.878	0.121	0.320	0.888	0.131	0.788	0.212	0.390	0.540
3	0.902	0.139	0.286	0.917	0.149	0.792	0.241	0.430	0.583
4	0.854	0.157	0.240	0.986	0.202	0.736	0.265	0.452	0.591
5	0.854	0.177	0.213	0.990	0.323	0.736	0.293	0.483	0.617
6	0.854	0.200	0.188	0.992	0.351	0.736	0.324	0.516	0.643
7	0.878	0.242	0.160	0.971	0.419	0.757	0.379	0.575	0.695
8	0.878	0.281	0.137	0.976	0.351	0.757	0.426	0.616	0.724

Dataset:	**CM1**								
Iteration	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0	0.981	0.012	0.565	0.639	0.023	0.990	0.024	0.058	0.109
1	0.981	0.010	0.662	0.709	0.015	0.990	0.020	0.050	0.094
2	0.981	0.010	0.653	0.767	0.015	0.990	0.021	0.050	0.096
3	0.981	0.011	0.636	0.813	0.015	0.990	0.021	0.052	0.098
4	0.986	0.011	0.622	0.856	0.015	0.993	0.022	0.053	0.101
5	0.989	0.011	0.607	0.886	0.016	0.995	0.022	0.054	0.103
6	0.989	0.012	0.592	0.904	0.016	0.994	0.023	0.056	0.105
7	0.986	0.012	0.577	0.927	0.016	0.993	0.024	0.057	0.108
8	0.986	0.012	0.571	0.926	0.016	0.993	0.024	0.058	0.109

Dataset:	**CM1 Subset1**								
Iteration	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0	1.000	0.044	0.698	0.706	0.062	1.000	0.084	0.187	0.314
1	1.000	0.044	0.691	0.785	0.055	1.000	0.085	0.188	0.316
2	1.000	0.049	0.623	0.900	0.059	1.000	0.094	0.205	0.340
3	1.000	0.056	0.548	0.906	0.071	1.000	0.106	0.228	0.372
4	1.000	0.063	0.483	0.929	0.078	1.000	0.119	0.253	0.403
5	0.971	0.073	0.407	0.958	0.093	0.978	0.136	0.281	0.435
6	0.971	0.087	0.343	0.956	0.138	0.978	0.159	0.320	0.481
7	0.943	0.103	0.279	0.977	0.234	0.956	0.186	0.359	0.521
8	0.943	0.113	0.256	0.977	0.194	0.956	0.201	0.381	0.543

Table 6.5: Feedback Analysis, Tf-idf - Modis, CM1, and CM1 Subset1

Dataset:	**Modis**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.878	0.281	0.137	0.975	0.350	0.756	0.426	0.616	0.724
tf-idf+thesaurus	0.975	0.264	0.162	0.914	0.443	0.916	0.416	0.634	0.769
KE	0.878	0.279	0.138	0.995	0.494	0.756	0.423	0.614	0.722
IDF	0.317	0.866	0.016	1.0	0.888	0.611	0.464	0.363	0.338
LSI	0.780	0.131	0.262	0.959	0.212	0.965	0.216	0.392	0.522
Probabilistic IR	0.390	0.390	0.044	0.922	0.661	0.747	0.385	0.390	0.390

Dataset:	**CM1**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.986	0.012	0.570	0.926	0.016	0.992	0.023	0.057	0.108
KE	0.986	0.012	0.573	0.916	0.016	0.992	0.023	0.057	0.108
IDF	0.554	0.059	0.064	0.953	0.155	0.582	0.107	0.208	0.303
LSI	0.797	0.014	0.392	0.809	0.021	0.811	0.0395	0.066	0.122
Probabilistic IR	0.925	0.014	0.467	0.587	0.023	0.940	0.039	0.065	0.121

Dataset:	**CM1 Subset1**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.942	0.112	0.256	0.976	0.194	0.955	0.201	0.381	0.542
KE	0.971	0.096	0.306	0.971	0.176	0.977	0.176	0.346	0.510
IDF	0.657	0.239	0.083	0.952	0.484	0.611	0.351	0.487	0.559
LSI	0.885	0.159	0.169	0.990	0.208	0.861	0.294	0.464	0.609
Probabilistic IR	0.7428	0.097	0.234	0.771	0.146	0.711	0.198	0.319	0.446

Table 6.6: Results Obtained by the Tf-idf, Tf-idf with thesaurus, Keyword Extraction, IDF, LSI and Probabilistic IR Methods on MODIS, CM1, and CM1 Subset1 at Iteration 8

Dataset:	**CM1 Subset2**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.978	0.035	0.458	0.934	0.043	0.975	0.068	0.154	0.266
KE	0.945	0.035	0.438	0.922	0.044	0.929	0.068	0.155	0.266
IDF	0.663	0.083	0.131	0.973	0.134	0.676	0.148	0.278	0.392
LSI	0.880	0.047	0.307	0.873	0.073	0.901	0.929	0.195	0.319
Probabilistic IR	0.891	0.043	0.341	0.610	0.117	0.868	0.108	0.181	0.300

Dataset:	**CM1 Subset3**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.974	0.016	0.468	0.969	0.031	0.969	0.033	0.078	0.146
KE	0.974	0.016	0.475	0.970	0.034	0.969	0.032	0.077	0.144
IDF	0.649	0.085	0.062	0.971	0.179	0.631	0.150	0.279	0.390
LSI	0.846	0.020	0.332	0.879	0.040	0.873	0.048	0.094	0.169
Probabilistic IR	0.863	0.020	0.253	0.672	0.077	0.888	0.052	0.092	0.166

Dataset:	**CM1 Subset4**								
Method	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
tf-idf	0.983	0.014	0.551	0.949	0.020	0.990	0.028	0.067	0.126
KE	0.983	0.014	0.554	0.945	0.020	0.990	0.027	0.067	0.125
IDF	0.581	0.066	0.070	0.960	0.123	0.603	0.118	0.227	0.326
LSI	0.778	0.017	0.369	0.860	0.027	0.804	0.039	0.077	0.141
Probabilistic IR	0.958	0.017	0.436	0.644	0.050	0.962	0.053	0.081	0.150

Table 6.7: Results Obtained by the Tf-idf, Tf-idf with thesaurus, Keyword Extraction, IDF, LSI, and Probabilistic IR Methods on CM1 Subset2, CM1 Subset3, and CM1 Subset4 at Iteration 8

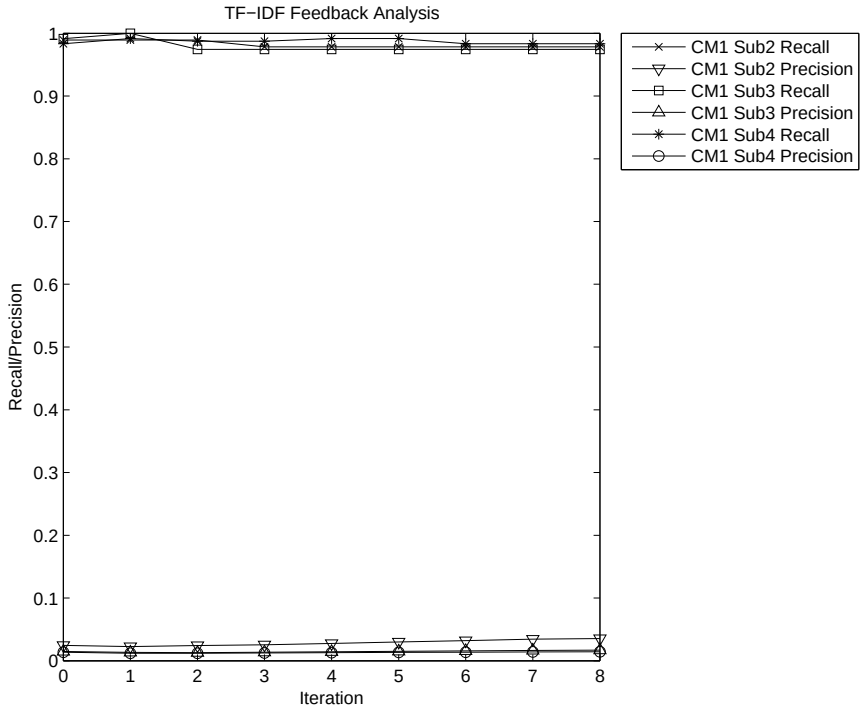


Figure 6.14: Feedback Analysis - Recall/Precision - CM1 Subset2, CM1 Subset3 and CM1 Subset4

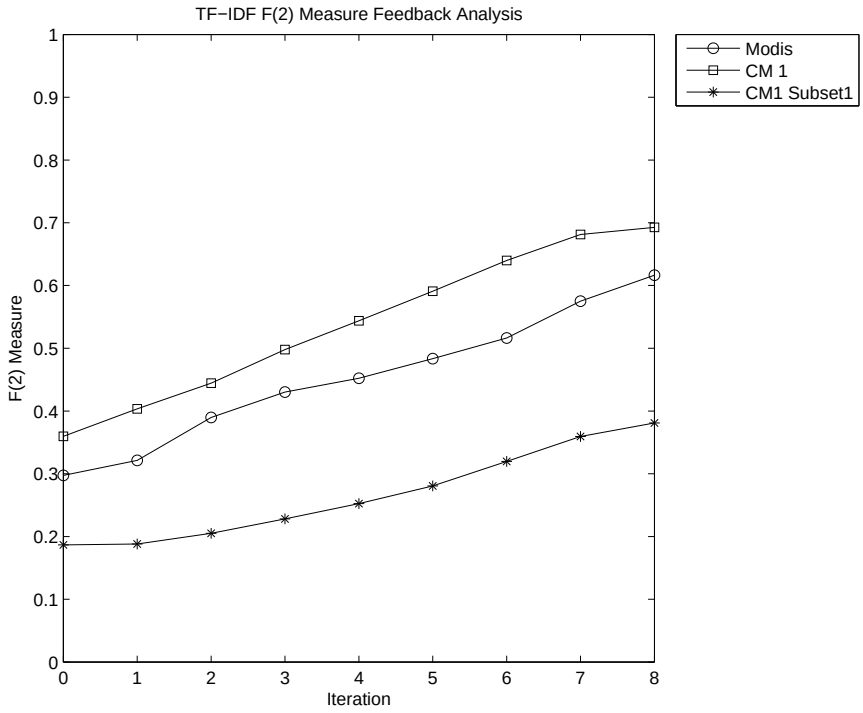


Figure 6.15: Feedback Analysis - F-measure - Modis, CM1 and CM1 Subset1

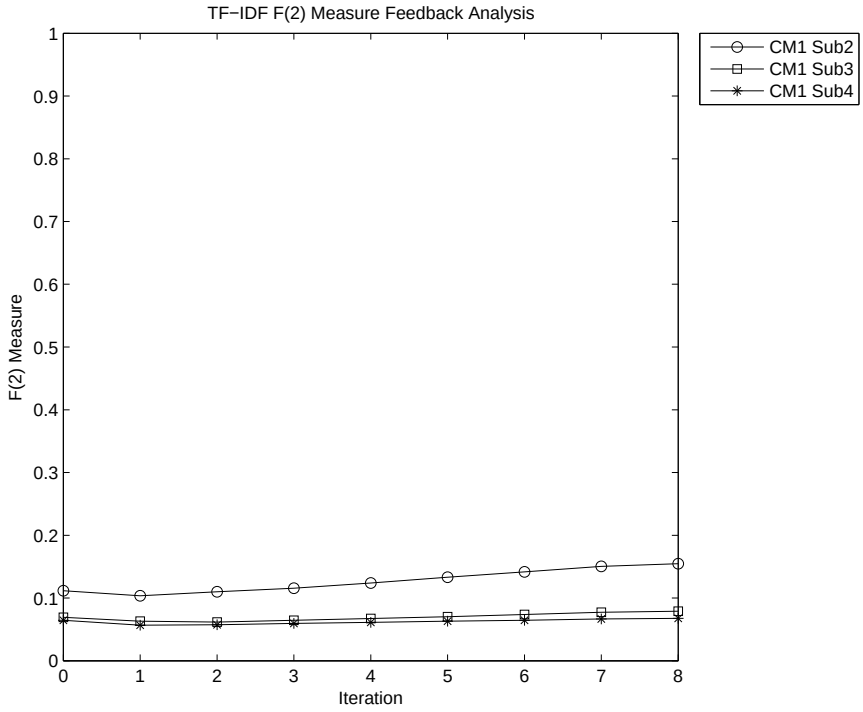


Figure 6.16: Feedback Analysis - F-measure - CM1 Subset2, CM1 Subset3 and CM1 Subset4

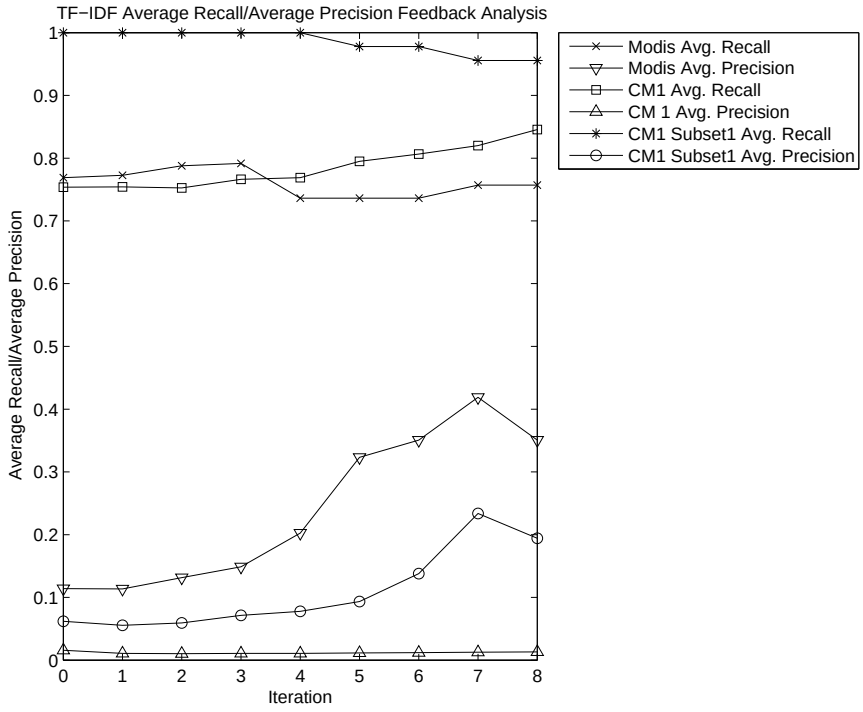


Figure 6.17: Feedback Analysis - Average Recall/Average Precision - Modis, CM1 and CM1 Subset1

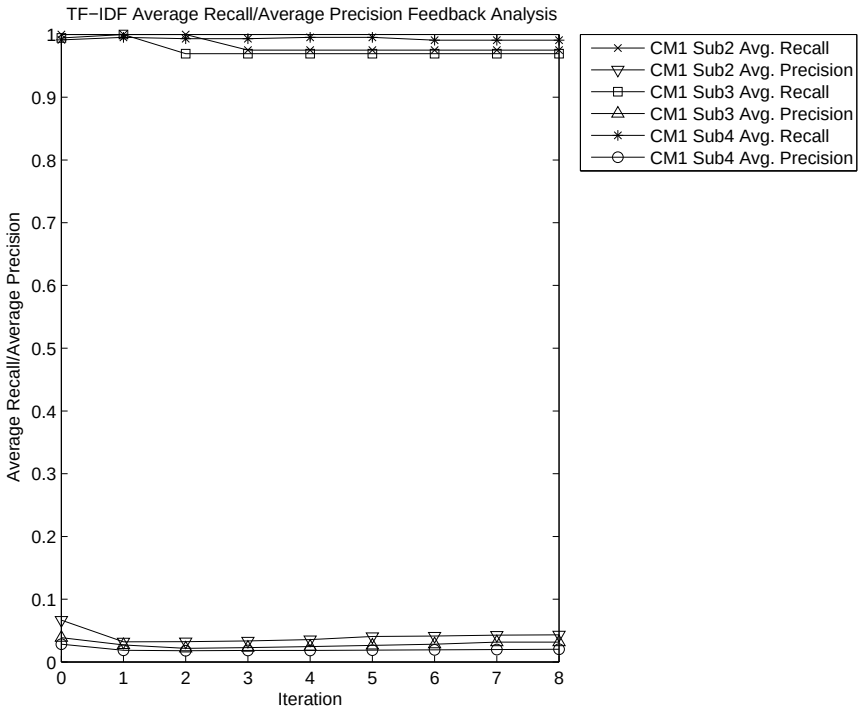


Figure 6.18: Feedback Analysis - Average Recall/Average Precision - CM1 Subset2, CM1 Subset3 and CM1 Subset4

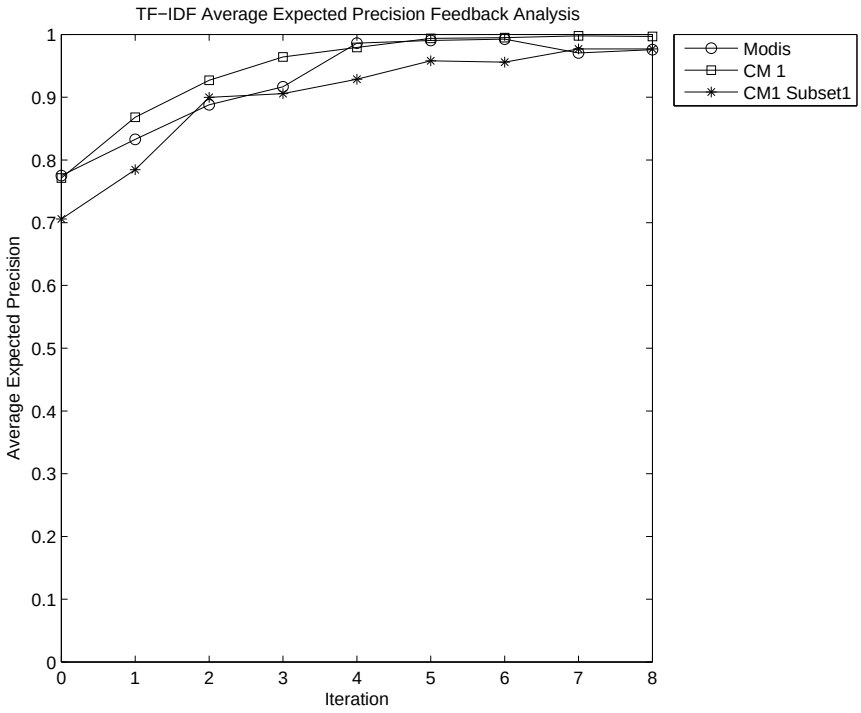


Figure 6.19: Feedback Analysis - Average Expected Precision - Modis, CM1 and CM1 Subset1

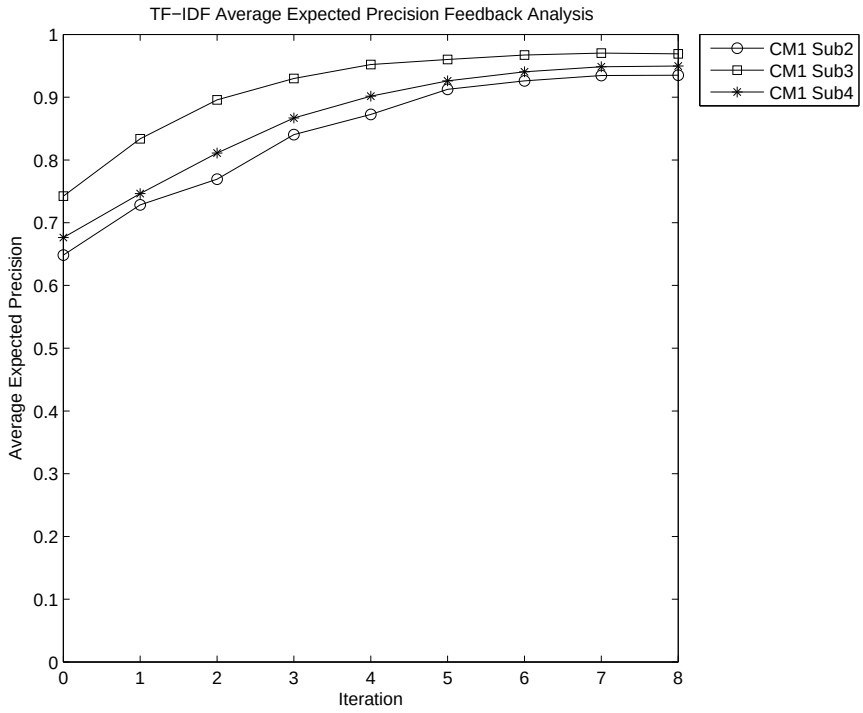


Figure 6.20: Feedback Analysis - Average Expected Precision - CM1 Subset2, CM1 Subset3 and CM1 Subset4

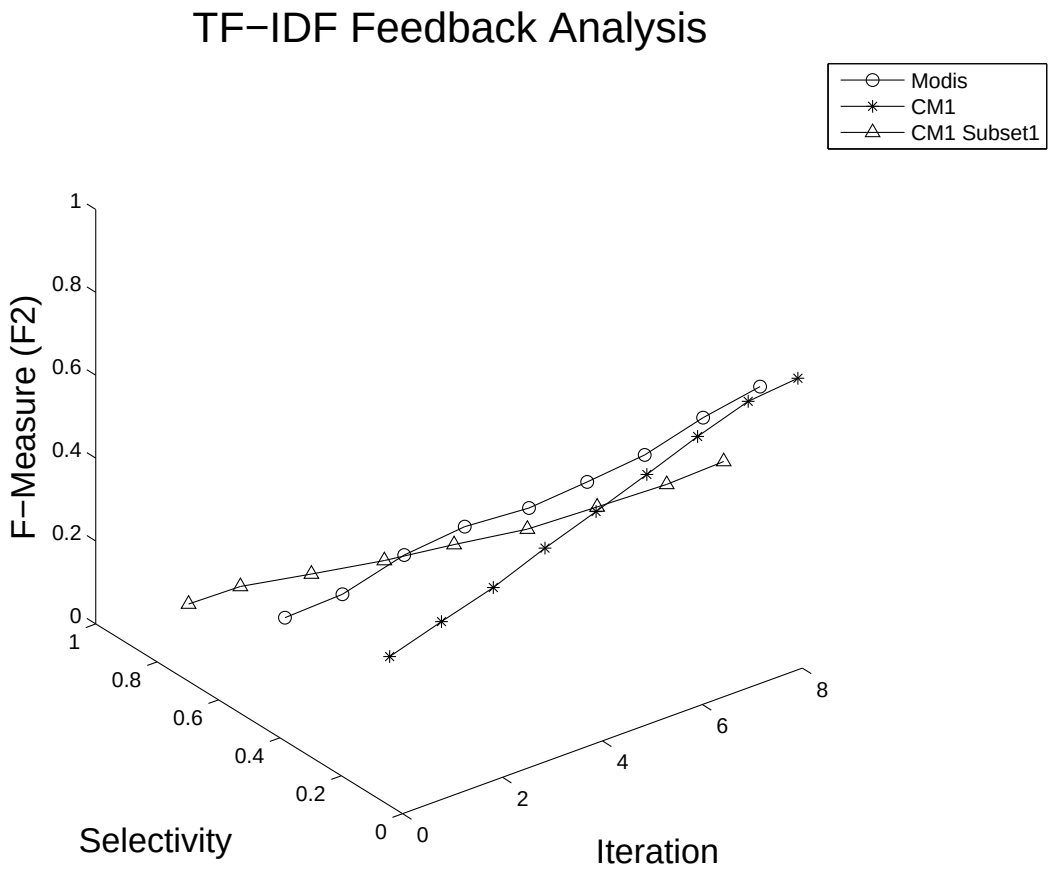


Figure 6.21: Feedback Analysis - F-Measure vs. Selectivity - Modis, CM1, and CM1 Subset1

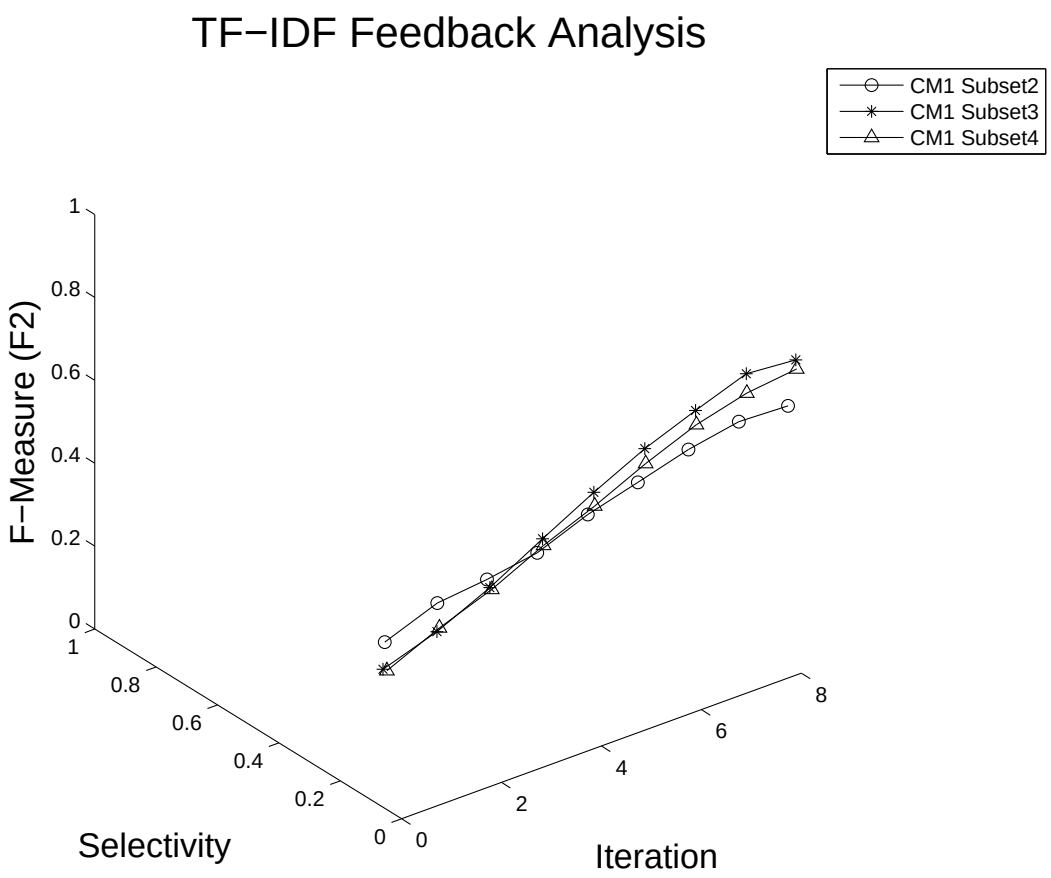


Figure 6.22: Feedback Analysis - F-Measure vs. Selectivity - CM1 Subset2, CM1 Subset3 and CM1 Subset4

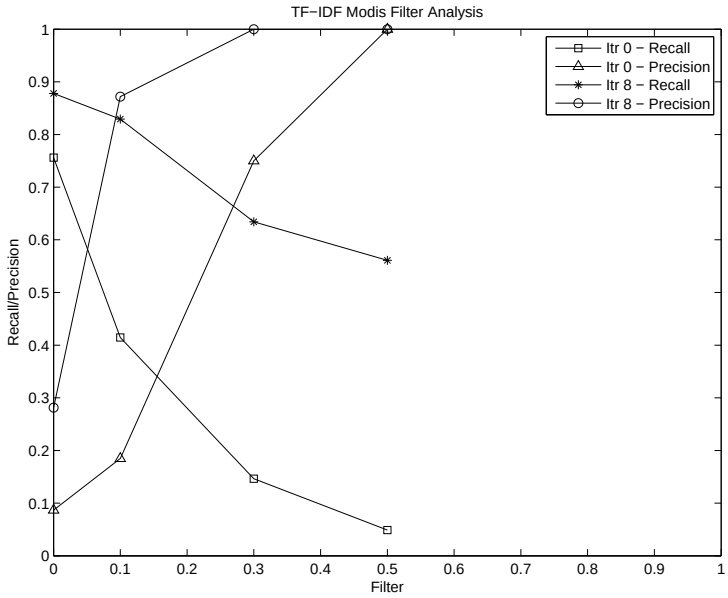
feedback process. It can be seen that the recall value decreased and the precision value increased as the filter value increased. Observe that for filter 0.5 for the Modis dataset, the recall and precision values achieved were 4.9% and 100%, respectively. This means that only 4.9% of the true links had relevance values higher than 0.5, but all the links that had relevance values of more than 0.5 were true links. This was true for all the datasets.

In an attempt to improve both precision and recall, we combined the feedback process with the filter. The feedback process and the filter significantly improved the recall and the precision, respectively. Table 6.12 and Table 6.13 show the results obtained by running the feedback process using filter 0.1 on all the datasets. The Modis dataset and CM1 Subset1 achieved excellent recall (80%-100%) and precision (50%-100%). The rest of the datasets achieved excellent recall (80%-100%) and good precision (30%-50%).

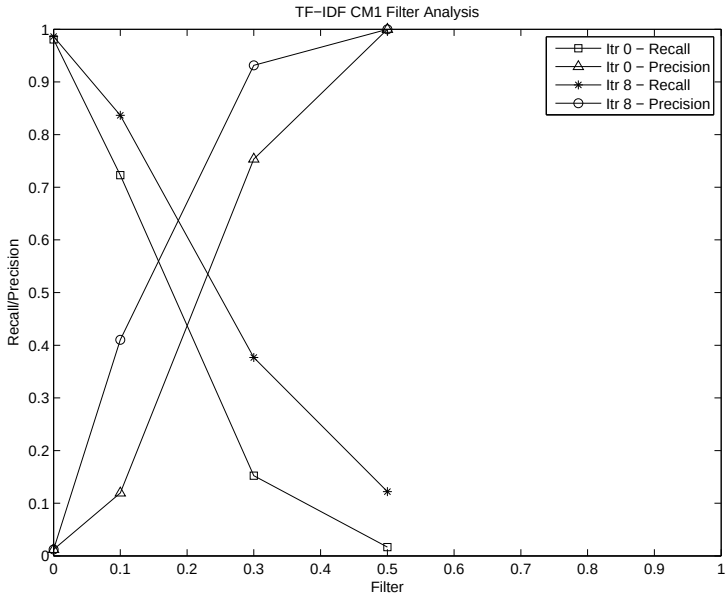
Figure 6.23, Figure 6.24, and Figure 6.25 display the effect of filtering and feedback on recall and precision. It can be seen that the filtering decreases recall and increases precision. However, when it is combined with feedback, it produces high recall-precision value pairs. For example, for the MODIS dataset, we are able to achieve about 82% recall and 82% precision at iteration 8. Similarly, for CM1 dataset, we are able to achieve about 85% recall and 50% precision. The same effect can be seen for other datasets too.

Figure 6.26, Figure 6.27, and Figure 6.28 show how f-measure behaves when both filtering and feedback are used. Both with and without feedback, the f-measure increases for filter values going from 0.0 to 0.1, but it decreases for filter values going from 0.1 to 0.5. The highest f-measure value achieved for the MODIS dataset at filter value 0.1 is 0.85. Similarly, the highest f-measure achieved for CM1 dataset at filter value 0.1 is 0.7. The highest f-measure value obtained for other datasets range from 0.6 to 0.8.

Figure 6.29, Figure 6.30, and Figure 6.31 show the effect of filtering and feedback on average precision and average recall. Strangely, for all the datasets, average precision and average recall decrease with filtering. The average recall and average precision values for iteration 8 are very high compared to those of iteration 0. However, even at iteration 8, the filtering still has the same effect on all the datasets. Figure 6.32 shows that, in the case of iteration 0, the average expected precision reaches 100% at filter 0.3. Whereas, in the case of iteration 8, the average expected precision reaches 100% at filter 0.1.

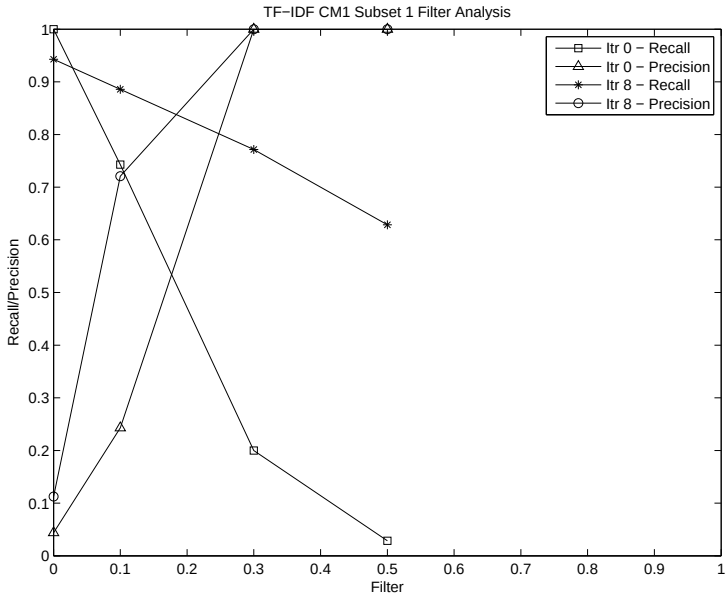


(a)

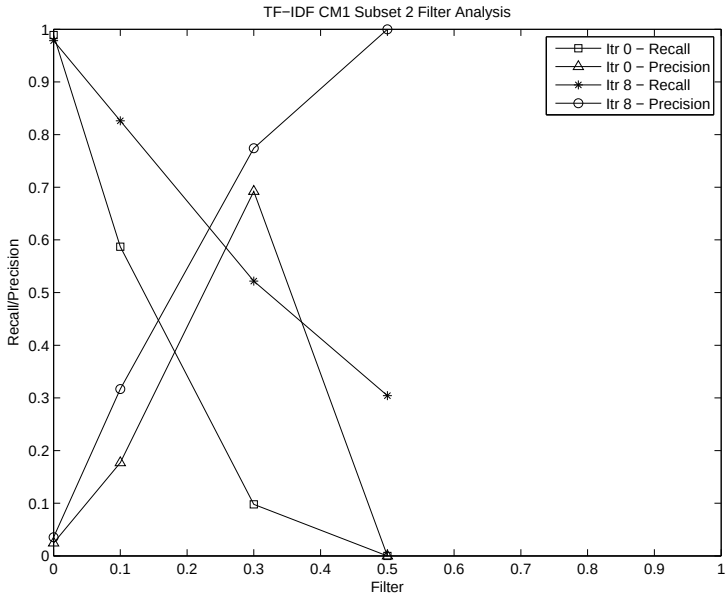


(b)

Figure 6.23: Filter Analysis - Recall/Precision (a) Modis and (b) CM1

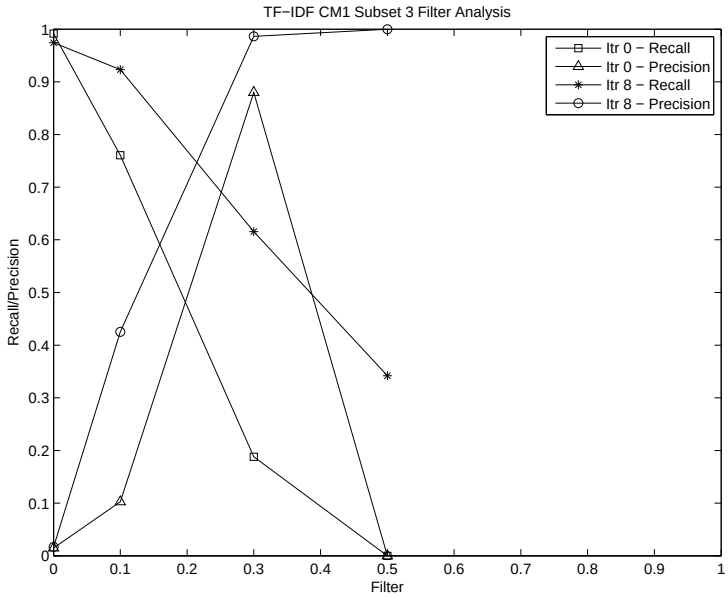


(a)

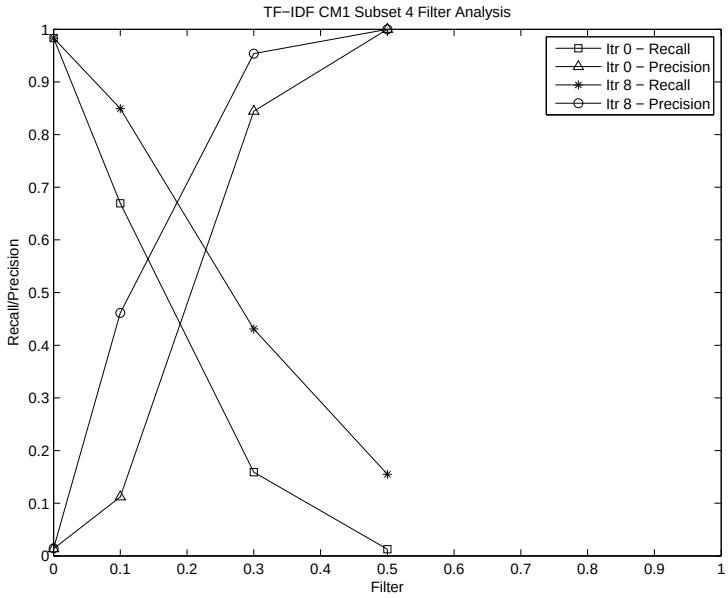


(b)

Figure 6.24: Filter Analysis - Recall/Precision (a) CM1 Subset1 and (b) CM1 Subset2

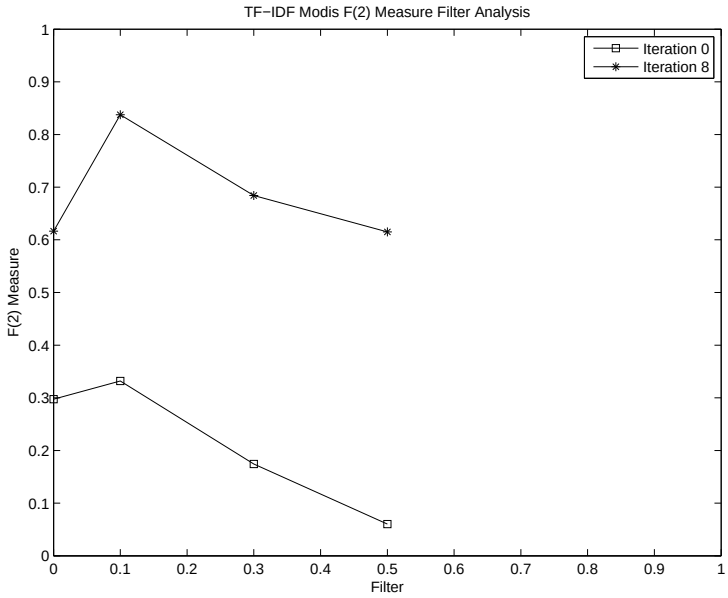


(a)

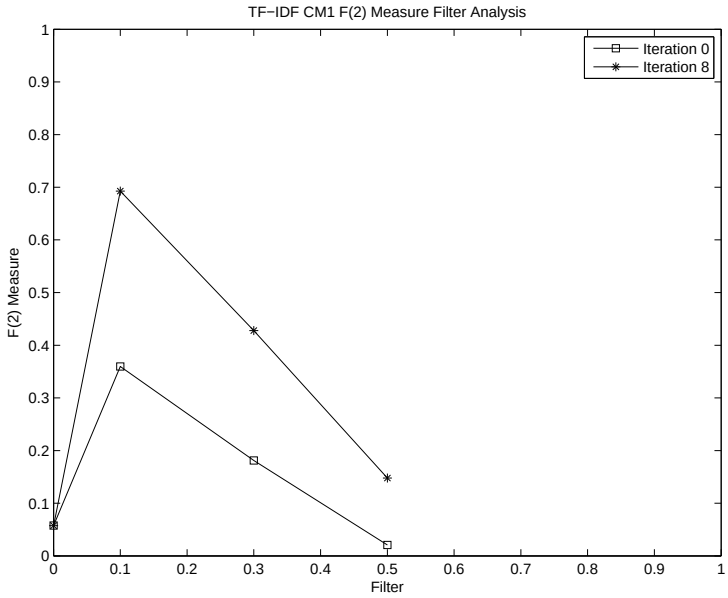


(b)

Figure 6.25: Filter Analysis - Recall/Precision (a) CM1 Subset3 and (b) CM1 Subset4



(a)



(b)

Figure 6.26: Filter Analysis - F-measure (a) Modis and (b) CM1

6.3.1 Statistical Analysis

We already saw in the previous section that, in the case of CM1 dataset, the feedback did not really increase the F-measure. Now, let us analyze if using feedback with filter helped to overcome this issue. Table 6.8 and Table 6.9 show the data used for the statistical analysis and the results of the studies using the MODIS and CM1 datasets, respectively.

We used the F-measure obtained by five IR methods listed in Table 6.8 at iteration 0 and iteration 8. The t-statistic obtained for the study using MODIS dataset was -4.3264. We carried out paired t test for α value 0.05. The degree of freedom for this test was 5. The $t_{critical}$ value obtained for one-tail test was 0.00376 and the $t_{critical}$ value obtained for two-tail test was 0.00752. We can see that, in the case of one-tail and two-tail tests, $|t| > t_critical$. Hence, we reject the null hypothesis H_{03} and accept the alternative hypothesis H_{A3}. Hence, In the case of the MODIS dataset, the difference between the F-measure of the candidate link lists obtained by using feedback with filter and the F-measure of the candidate link lists obtained by not using feedback and filter was statistically significant. The Pearson correlation shows a strong positive correlation between the F-measure obtained at iteration 0 and the F-measure obtained at iteration 8.

We used the F-measure obtained by tf-idf, KE, IDF, LSI, and probabilistic IR at iteration 0 and iteration 8. Table 6.9 shows that the t-statistic obtained for the study using CM1 dataset was -2.8907. In this study, we used 0.05 as our α value. The degree of freedom for this test was 4. The $t_{critical}$ value obtained for one-tail test was 0.02226 and the $t_{critical}$ value obtained for two-tail test was 0.04453. In the case of one-tail as well as two-tail tests, $|t| > t_critical$. Hence, we can reject the null hypothesis H_{04} and accept the alternative hypothesis H_{A4}. Hence, In the case of the CM1 dataset, the difference between the F-measure of the candidate link lists obtained by using feedback and the F-measure of the candidate link lists obtained by not using feedback and filter was statistically significant. Notice that the Pearson correlation obtained for this study was negative and close to zero. Hence, in the case of the CM1 dataset, there is a weak negative correlation between the F-measure obtained at iteration 0 and the F-measure obtained at iteration 8. In both the cases, using feedback with filter increased the F-measure significantly.

Method	Iteration 0	Iteration 8
Tf-idf	0.297	0.812
Tf-idf + Thesaurus	0.381	0.891
KE	0.297	0.792
IDF	0.267	0.367
LSI	0.261	0.482
Probabilistic IR	0.228	0.406

Groups	Observations	Sum	Mean	Variance
Iteration 0	6	1.731	0.2885	0.002716
Iteration 8	6	3.75	0.625	0.053718

Metric	Value
Pearson Correlation	0.833592694
df	5
t Stat	-4.326428351
P($T <= t$) one-tail	0.003762361
$t_{critical}$ one-tail	2.015048372
P($T <= t$) two-tail	0.007524721
$t_{critical}$ two-tail	2.570581835

Table 6.8: Modis - No Feedback vs. Feedback + Filter - Paired T-test

Method	Iteration 0	Iteration 8
Tf-idf	0.057	0.721
KE	0.057	0.71
IDF	0.246	0.458
LSI	0.057	0.366
Probabilistic IR	0.058	0.066

Groups	Observations	Sum	Mean	Variance
Iteration 0	5	0.475	0.095	0.0071
Iteration 8	5	2.321	0.4642	0.0737

Metric	Value
Pearson Correlation	-0.017133225
df	4
t Stat	-2.890685856
P($T <= t$) one-tail	0.022266113
$t_{critical}$ one-tail	2.131846782
P($T <= t$) two-tail	0.044532227
$t_{critical}$ two-tail	2.776445105

Table 6.9: CM1 - No Feedback vs. Feedback + Filter - Paired T-test

6.3.2 Observations

- Filtering increases precision, average precision, and average expected precision. It decreases recall and average recall.

- Filtering increases the f-measure for smaller values of the filter, but decreases f-measure for higher values of filter.

- When filtering is combined with feedback, the recall and the precision values improved significantly for all the datasets.

- The statistical analysis shows that, in the case of the MODIS and CM1 datasets, the F-measure increased significantly when the filtering was combined with feedback. Hence, we can conclude that combining filtering and feedback improves the quality of the traceability links obtained for both, the large and the small datasets.

6.4 Selectivity

One of the main objectives of this work is to reduce the workload of the analyst. Let us analyze the selectivity values to find out how much of the analyst's work was decreased by our methods. Figure 6.35 shows the selectivity values achieved by the tf-idf method for all the datasets. Note that the lower the selectivity, the fewer the number of links the analyst needs to analyze. When the filter was not used, the tf-idf method achieved selectivity values of less than 0.7 for all the datasets. The selectivity decreased when the filter value was increased. In the case of filter 0.1, the selectivity for all the datasets ranged from 4.8% to 9.9%. This indicates that our methods are capable of reducing the analyst's workload significantly.

6.5 Weight Analysis

The results obtained by the tf-idf method using the Okapi and LTU weighting schemes are listed in Table 6.16. Figure 6.36 compares the results obtained by the tf-idf method while using different weighting schemes. It is very clear that there is no significant difference between the performances of the weighting schemes. In our experiments, we observed that the behavior of the weighting schemes were the same for the other datasets too.

Dataset:	**Modis**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.756	0.087	0.383	0.775	0.114	0.769	0.156	0.298	0.427
0.05	0.488	0.100	0.216	0.790	0.084	0.679	0.165	0.274	0.351
0.1	0.415	0.185	0.099	0.859	0.072	0.596	0.256	0.332	0.369
0.15	0.244	0.345	0.031	0.958	0.050	0.535	0.286	0.259	0.251
0.2	0.195	0.615	0.014	1.000	0.043	0.490	0.296	0.226	0.209
0.25	0.171	0.700	0.011	1.000	0.040	0.448	0.275	0.201	0.185
0.3	0.146	0.750	0.009	1.000	0.030	0.365	0.245	0.174	0.159
0.35	0.098	1.000	0.004	1.000	0.020	0.198	0.178	0.119	0.107
0.4	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.45	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.5	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054

Dataset:	**CM1**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.981	0.012	0.565	0.639	0.023	0.990	0.024	0.058	0.109
0.05	0.889	0.046	0.134	0.678	0.019	0.915	0.088	0.191	0.315
0.1	0.723	0.120	0.042	0.771	0.016	0.754	0.205	0.360	0.480
0.15	0.512	0.236	0.015	0.925	0.011	0.546	0.323	0.415	0.459
0.2	0.330	0.415	0.006	0.956	0.007	0.363	0.367	0.344	0.337
0.25	0.247	0.593	0.003	0.993	0.006	0.265	0.348	0.279	0.262
0.3	0.152	0.753	0.001	0.996	0.003	0.165	0.253	0.181	0.166
0.35	0.089	0.800	0.001	0.994	0.002	0.093	0.160	0.108	0.097
0.4	0.047	0.895	0.000	1.000	0.001	0.051	0.089	0.058	0.052
0.45	0.028	1.000	0.000	1.000	0.001	0.033	0.054	0.034	0.031
0.5	0.017	1.000	0.000	1.000	0.000	0.024	0.033	0.021	0.018

Dataset:	**CM1 Subset1**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	1.000	0.044	0.698	0.706	0.062	1.000	0.084	0.187	0.314
0.05	0.914	0.122	0.229	0.758	0.056	0.922	0.215	0.398	0.555
0.1	0.743	0.243	0.094	0.852	0.045	0.756	0.366	0.526	0.616
0.15	0.571	0.435	0.040	0.979	0.034	0.567	0.494	0.538	0.554
0.2	0.286	0.526	0.017	0.976	0.015	0.306	0.370	0.314	0.299
0.25	0.257	0.818	0.010	1.000	0.014	0.289	0.391	0.298	0.276
0.3	0.200	1.000	0.006	1.000	0.011	0.233	0.333	0.238	0.217
0.35	0.114	1.000	0.003	1.000	0.006	0.172	0.205	0.139	0.125
0.4	0.086	1.000	0.003	1.000	0.004	0.150	0.158	0.105	0.094
0.45	0.029	1.000	0.001	1.000	0.002	0.067	0.056	0.035	0.032
0.5	0.029	1.000	0.001	1.000	0.002	0.067	0.056	0.035	0.032

Table 6.10: Filter Analysis - Modis, CM1, and CM1 Subset1

Dataset:	**CM1 Subset2**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.989	0.025	0.668	0.648	0.067	1.000	0.048	0.112	0.201
0.05	0.826	0.087	0.158	0.708	0.058	0.880	0.157	0.305	0.445
0.1	0.587	0.177	0.055	0.803	0.023	0.658	0.272	0.401	0.477
0.15	0.380	0.302	0.021	0.960	0.014	0.439	0.337	0.362	0.371
0.2	0.207	0.463	0.007	0.990	0.008	0.247	0.286	0.232	0.219
0.25	0.152	0.737	0.003	0.986	0.006	0.154	0.252	0.181	0.165
0.3	0.098	0.692	0.002	0.976	0.004	0.104	0.171	0.118	0.107
0.35	0.087	0.889	0.002	0.972	0.004	0.098	0.158	0.106	0.096
0.4	0.033	0.750	0.001	1.000	0.001	0.040	0.063	0.040	0.036
0.45	0.022	1.000	0.000	1.000	0.001	0.031	0.043	0.027	0.024
0.5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Dataset:	**CM1 Subset3**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.991	0.015	0.548	0.742	0.039	0.994	0.029	0.069	0.130
0.05	0.940	0.045	0.169	0.763	0.034	0.950	0.086	0.189	0.315
0.1	0.761	0.103	0.060	0.840	0.027	0.800	0.181	0.333	0.463
0.15	0.530	0.211	0.020	0.963	0.019	0.563	0.302	0.407	0.460
0.2	0.410	0.495	0.007	0.975	0.016	0.439	0.449	0.425	0.417
0.25	0.325	0.864	0.003	0.994	0.013	0.368	0.472	0.371	0.346
0.3	0.188	0.880	0.002	0.991	0.006	0.238	0.310	0.223	0.204
0.35	0.128	0.938	0.001	0.987	0.005	0.165	0.226	0.155	0.140
0.4	0.009	1.000	0.000	1.000	0.000	0.006	0.017	0.011	0.009
0.45	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Dataset:	**CM1 Subset4**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.983	0.014	0.579	0.676	0.028	0.991	0.027	0.064	0.121
0.05	0.887	0.047	0.151	0.709	0.024	0.932	0.089	0.194	0.318
0.1	0.669	0.112	0.048	0.814	0.017	0.744	0.192	0.336	0.447
0.15	0.473	0.217	0.017	0.950	0.012	0.547	0.298	0.383	0.423
0.2	0.322	0.450	0.006	0.967	0.009	0.377	0.376	0.342	0.332
0.25	0.238	0.704	0.003	0.978	0.007	0.281	0.356	0.275	0.255
0.3	0.159	0.844	0.002	0.995	0.004	0.203	0.268	0.190	0.173
0.35	0.092	0.917	0.001	0.991	0.002	0.122	0.167	0.112	0.101
0.4	0.038	0.900	0.000	1.000	0.001	0.056	0.072	0.047	0.042
0.45	0.021	1.000	0.000	1.000	0.000	0.030	0.041	0.026	0.023
0.5	0.013	1.000	0.000	1.000	0.000	0.023	0.025	0.016	0.014

Table 6.11: Filter Analysis - CM1 Subset2, CM1 Subset3, and CM1 Subset4

Dataset:	**Modis**								
Iteration	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0	0.415	0.185	0.099	0.859	0.072	0.596	0.256	0.332	0.369
1	0.463	0.292	0.070	0.921	0.080	0.652	0.358	0.415	0.438
2	0.439	0.500	0.039	1.000	0.086	0.631	0.468	0.450	0.444
3	0.439	0.563	0.034	1.000	0.102	0.631	0.493	0.459	0.449
4	0.488	0.714	0.030	1.000	0.163	0.662	0.580	0.521	0.504
5	0.634	0.897	0.031	0.999	0.299	0.685	0.743	0.674	0.653
6	0.707	0.906	0.034	0.999	0.338	0.713	0.795	0.740	0.723
7	0.780	0.889	0.039	0.998	0.406	0.725	0.831	0.800	0.790
8	0.829	0.872	0.042	0.998	0.342	0.732	0.850	0.837	0.833

Dataset:	**CM1**								
Iteration	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0	0.723	0.120	0.042	0.771	0.016	0.754	0.205	0.360	0.480
1	0.734	0.144	0.036	0.868	0.011	0.754	0.241	0.403	0.521
2	0.734	0.172	0.030	0.927	0.010	0.752	0.279	0.444	0.554
3	0.748	0.213	0.025	0.964	0.011	0.766	0.331	0.498	0.598
4	0.756	0.256	0.021	0.979	0.011	0.769	0.383	0.544	0.633
5	0.776	0.303	0.018	0.994	0.011	0.795	0.435	0.591	0.671
6	0.795	0.359	0.015	0.995	0.012	0.806	0.495	0.640	0.709
7	0.809	0.418	0.014	0.998	0.013	0.820	0.551	0.681	0.740
8	0.837	0.410	0.014	0.997	0.013	0.845	0.551	0.693	0.758

Dataset:	**CM1 Subset1**								
Iteration	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0	0.743	0.243	0.094	0.852	0.045	0.756	0.366	0.526	0.616
1	0.771	0.307	0.077	0.932	0.042	0.722	0.439	0.592	0.670
2	0.771	0.355	0.066	0.949	0.042	0.722	0.486	0.625	0.691
3	0.800	0.431	0.057	0.976	0.050	0.800	0.560	0.683	0.737
4	0.829	0.527	0.048	0.984	0.066	0.867	0.644	0.744	0.784
5	0.857	0.600	0.044	0.987	0.082	0.883	0.706	0.789	0.822
6	0.886	0.689	0.039	1.000	0.130	0.917	0.775	0.838	0.861
7	0.886	0.795	0.034	1.000	0.228	0.917	0.838	0.866	0.876
8	0.886	0.721	0.038	1.000	0.187	0.917	0.795	0.847	0.866

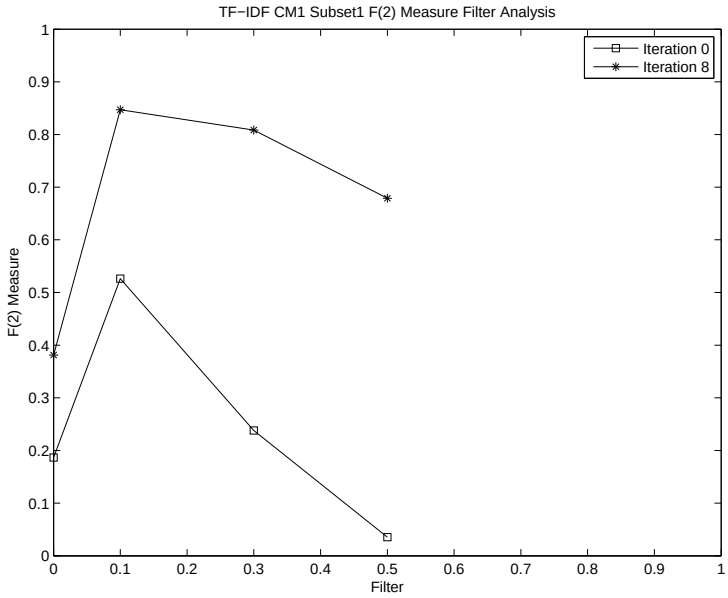
Table 6.12: Filter Analysis, Filter 0.1 - Modis, CM1, and CM1 Subset1

Dataset:	**CM1 Subset2**								
Iteration	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0	0.587	0.177	0.055	0.803	0.023	0.658	0.272	0.401	0.477
1	0.620	0.221	0.046	0.931	0.018	0.673	0.326	0.455	0.525
2	0.641	0.225	0.047	0.950	0.019	0.685	0.333	0.468	0.541
3	0.663	0.238	0.046	0.963	0.020	0.714	0.351	0.489	0.563
4	0.717	0.269	0.044	0.977	0.024	0.755	0.392	0.538	0.615
5	0.761	0.287	0.044	0.992	0.028	0.811	0.417	0.572	0.653
6	0.793	0.315	0.042	0.994	0.032	0.836	0.451	0.608	0.689
7	0.815	0.333	0.041	0.995	0.034	0.857	0.473	0.632	0.712
8	0.826	0.317	0.043	0.996	0.035	0.870	0.458	0.625	0.712

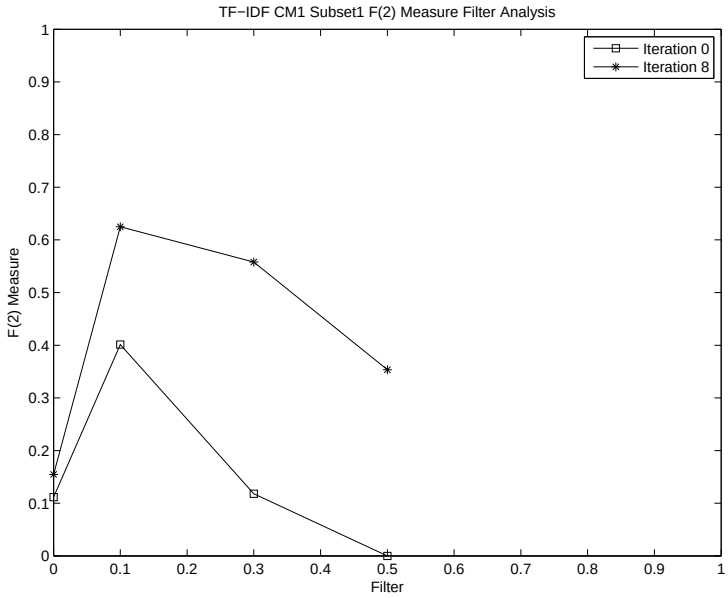
Dataset:	**CM1 Subset3**								
Iteration	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0	0.761	0.103	0.060	0.840	0.027	0.800	0.181	0.333	0.463
1	0.761	0.130	0.048	0.948	0.017	0.783	0.222	0.386	0.512
2	0.786	0.168	0.038	0.971	0.017	0.818	0.276	0.452	0.574
3	0.803	0.225	0.029	0.979	0.018	0.831	0.351	0.530	0.639
4	0.846	0.278	0.025	0.992	0.020	0.864	0.419	0.601	0.703
5	0.880	0.334	0.021	0.996	0.023	0.899	0.485	0.664	0.757
6	0.897	0.392	0.019	0.998	0.025	0.911	0.545	0.713	0.795
7	0.915	0.451	0.016	0.999	0.028	0.924	0.605	0.759	0.829
8	0.923	0.425	0.018	1.000	0.030	0.929	0.582	0.748	0.826

Dataset:	**CM1 Subset4**								
Iteration	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0	0.669	0.112	0.048	0.814	0.017	0.744	0.192	0.336	0.447
1	0.711	0.145	0.039	0.881	0.012	0.768	0.240	0.399	0.511
2	0.715	0.182	0.031	0.932	0.012	0.769	0.290	0.451	0.553
3	0.741	0.234	0.025	0.969	0.013	0.791	0.355	0.516	0.608
4	0.762	0.283	0.021	0.989	0.013	0.813	0.413	0.569	0.652
5	0.791	0.346	0.018	0.996	0.014	0.831	0.482	0.629	0.701
6	0.812	0.413	0.016	0.998	0.016	0.863	0.547	0.680	0.740
7	0.824	0.464	0.014	1.000	0.016	0.870	0.593	0.713	0.765
8	0.849	0.461	0.015	1.000	0.017	0.898	0.598	0.727	0.783

Table 6.13: Filter Analysis, Filter 0.1 - CM1 Subset2, CM1 Subset3, and CM1 Subset4

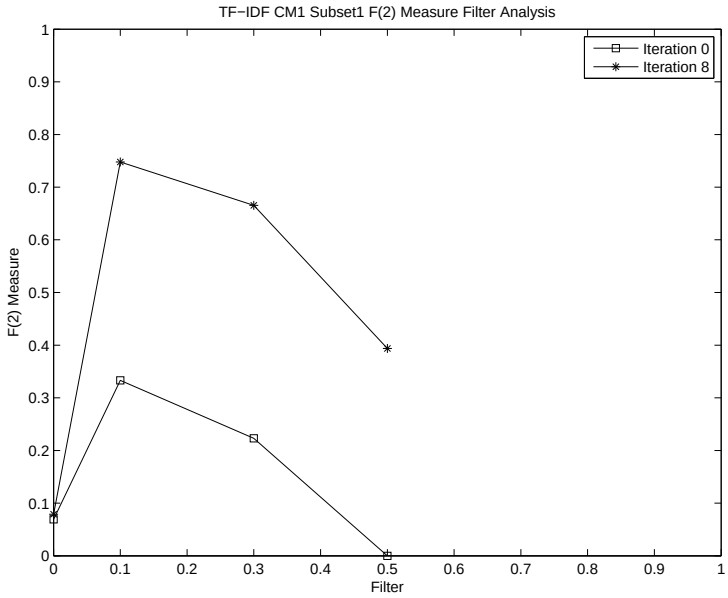


(a)

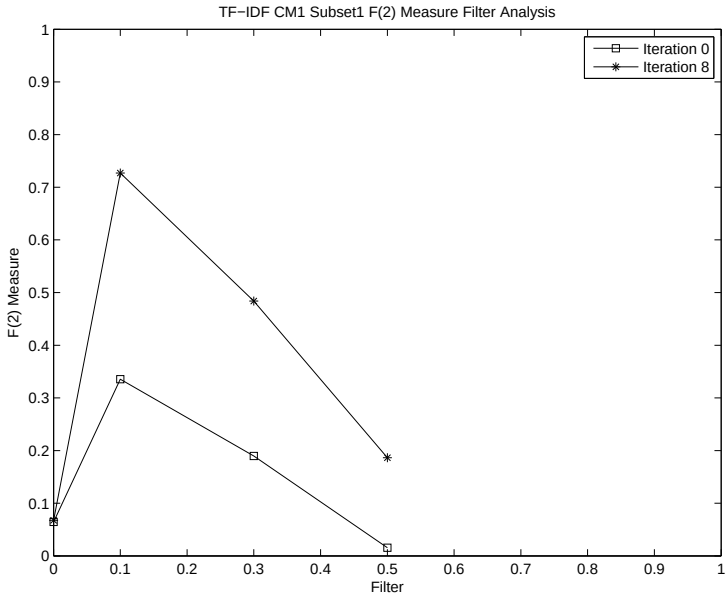


(b)

Figure 6.27: Filter Analysis - F-measure (a) CM1 Subset1 and (b) CM1 Subset2

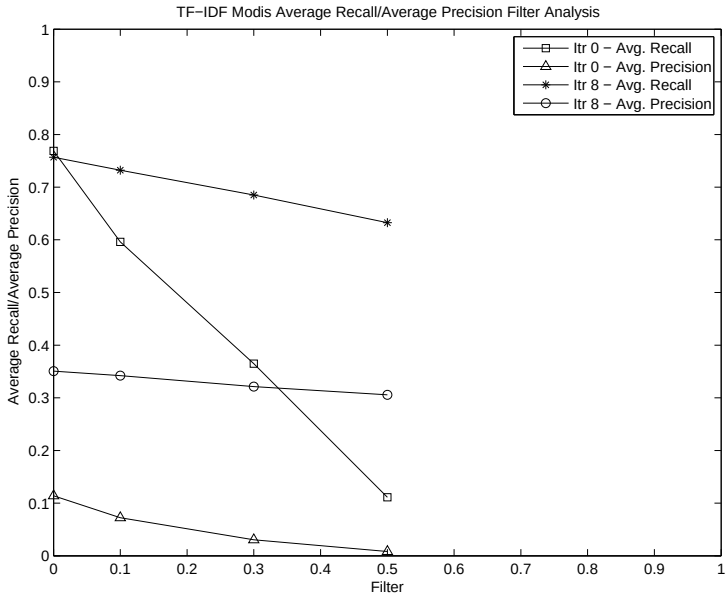


(a)

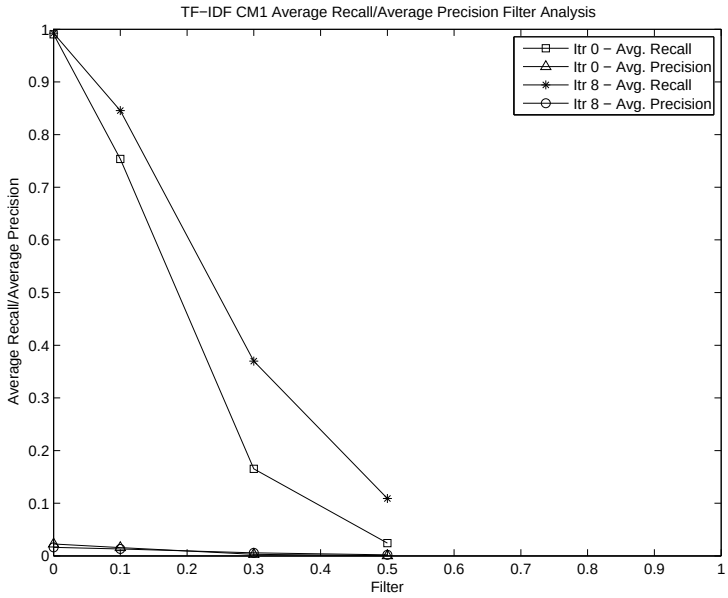


(b)

Figure 6.28: Filter Analysis - F-measure (a) CM1 Subset3 and (b) CM1 Subset4

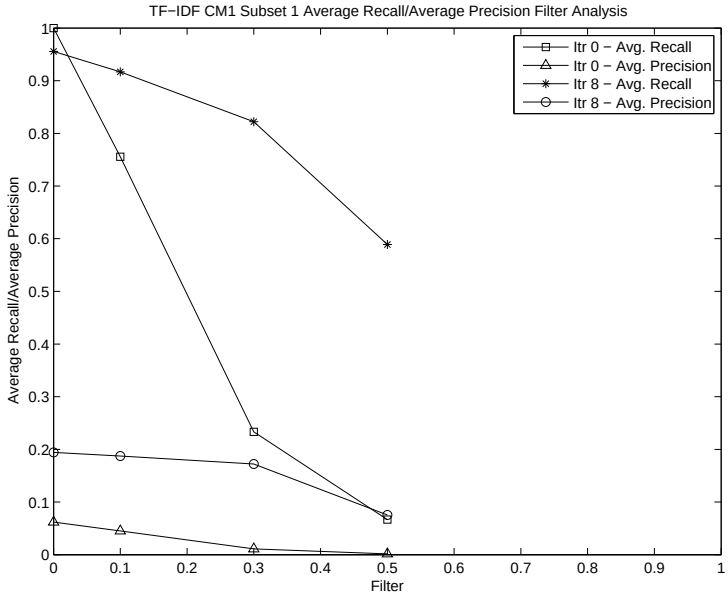


(a)

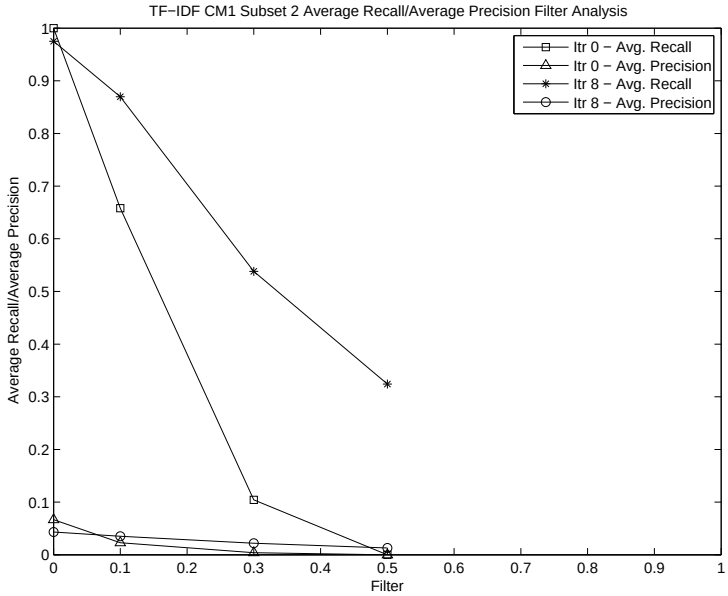


(b)

Figure 6.29: Filter Analysis - Average Recall/Average Precision (a) Modis and (d) CM1

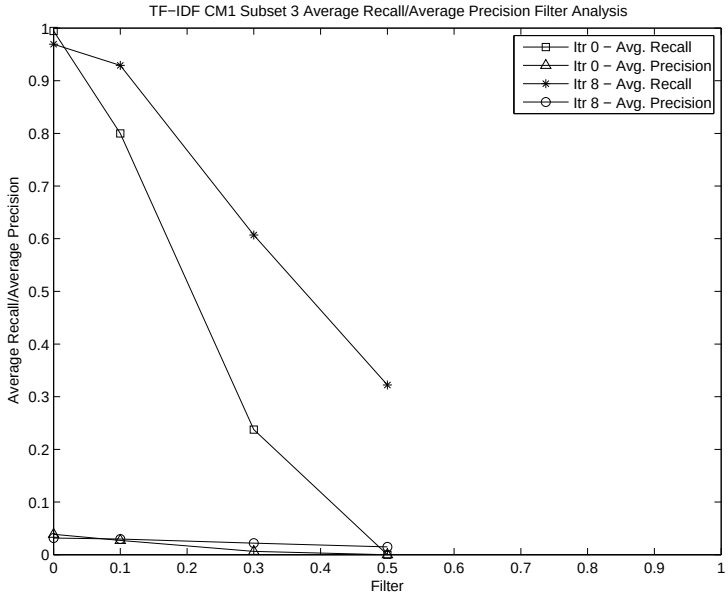


(a)

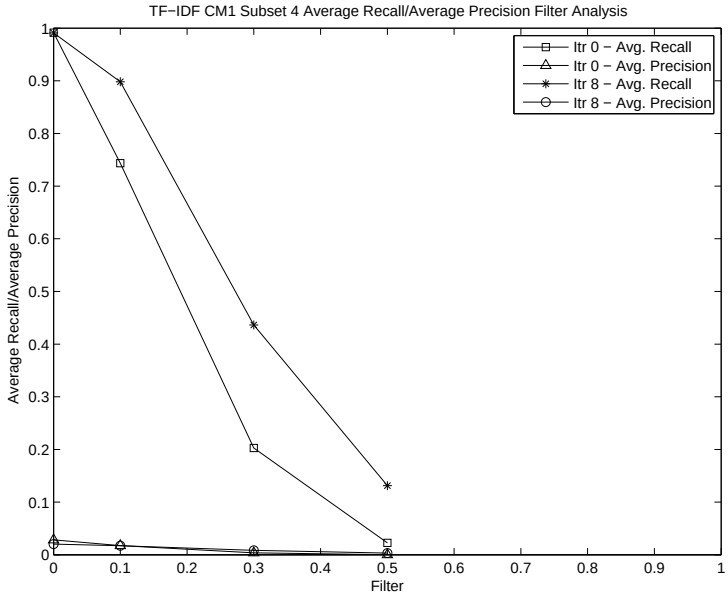


(b)

Figure 6.30: Filter Analysis - Average Recall/Average Precision (a) CM1 Subset1 and (d) CM1 Subset2

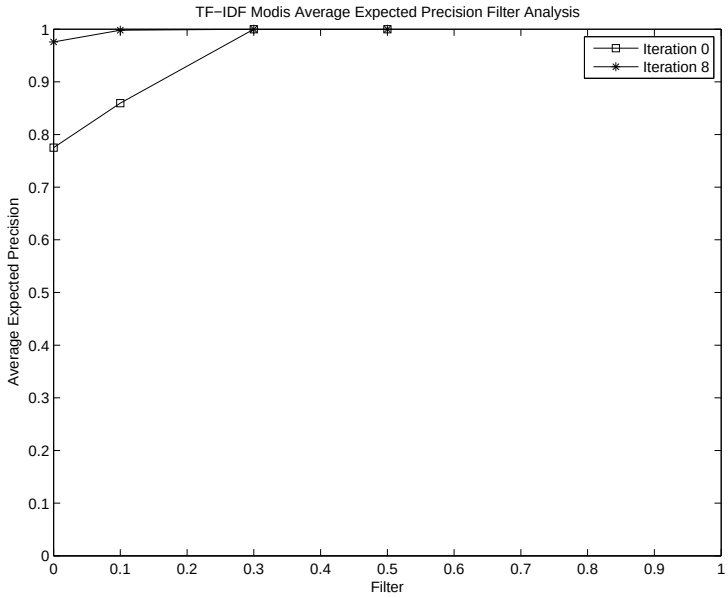


(a)

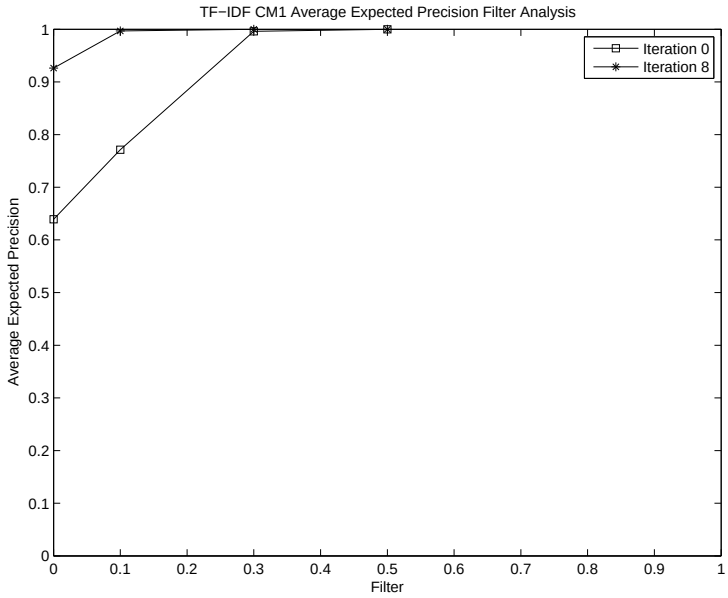


(b)

Figure 6.31: Filter Analysis - Average Recall/Average Precision (a) CM1 Subset3 and (d) CM1 Subset4



(a)



(b)

Figure 6.32: Filter Analysis - Average Expected Precision (a) Modis and (b) CM1

6.5.1 Statistical Analysis

Let us analyze if the difference among the recall and precision values of the results obtained by tf-idf, Okapi, and LTU weight methods were statistically significant. Table 6.14 and Table 6.15 show the data used for the statistical analysis and the results of the studies. For the weight analysis using recall and precision, we had totally 33 observations in three groups, namely, tf-idf, Okapi, and LTU. Also the degree of freedom for these studies was 2.

The columns in Table 6.14 show the recall values obtained by each weighting method. The T and P values obtained for the study comparing the recall values were 0.249392 and 0.8828, respectively. When adjusted for ties, the T and P values obtained were 0.24981 and 0.8826, respectively. The all pairwise comparisons using Dwass-Steel-Chritchlow-Fligner and Conover-Inman tests showed no significant difference among the recall values obtained by tf-idf, Okapi, and LTU weighting methods. Hence, we cannot reject the null hypothesis H_{05}.

The columns in Table 6.15 show the precision values obtained by each weighting method. The T and P values obtained for the study comparing the recall values were 0.910063 and 0.6344, respectively. When adjusted for ties, the T and P values obtained were 0.945455 and 0.6233, respectively. The all pairwise comparisons using Dwass-Steel-Chritchlow-Fligner and Conover-Inman tests showed no significant difference among the precision values obtained by tf-idf, Okapi, and LTU weighting methods. Hence, we cannot reject the null hypothesis H_{06}.

Since, the difference among the recall and the precision values obtained by tf-idf, Okapi, and LTU weighting methods is not statistically significant, we can conclude that there is no advantage in using one weighting method over the other.

6.6 Thesaurus Analysis

Figure 6.37 compares the recall and precision obtained by the tf-idf method and the tf-idf+thesaurus method. The usage of the thesaurus improved the recall and precision. Also, few links would not have been found without the thesaurus. Unlike the tf-idf method, the tf-idf+thesaurus method was able to achieve 100% recall.

Tf-idf	Okapi	LTU
0.9806	0.9806	0.9806
0.8892	0.8947	0.8864
0.7230	0.6620	0.6814
0.5125	0.4681	0.4792
0.3296	0.2909	0.3019
0.2465	0.1911	0.2022
0.1524	0.1080	0.1274
0.0886	0.0471	0.0582
0.0471	0.0194	0.0332
0.0277	0.0111	0.0166
0.0166	0.0111	0.0111

Groups	Count	Sum	Average	Variance
Tf-idf	11	4.0138	0.3649	0.1277
Okapi	11	3.6842	0.3349	0.1321
LTU	11	3.7784	0.3435	0.1297

Groups	df	Observations	T	P	T (ties)	P (ties)
3	2	33	0.24932	0.8828	0.24981	0.8826

All Pairwise Comparisons (Dwass-Steel-Chritchlow-Flinger)	
Critical q (range) = 3.314493	
Tf-idf vs. Okapi ($\|-0.60413\| > 3.314493$)	Not significant P = 0.9043
Tf-idf vs. LTU ($\|-0.511043\| > 3.314493$)	Not significant P = 0.9306
Okapi vs. LTU ($\|0.371983\| > 3.314493$)	Not significant P = 0.9626

All Pairwise Comparisons (Conover-Inman)	
Critical t (30 df) = 2.042272	
Tf-idf and Okapi ($2.045455 > 8.662705$)	Not significant P = 0.6331
Tf-idf and LTU ($1.227273 > 8.662705$)	Not significant P = 0.7743
Okapi and LTU ($0.818182 > 8.662705$)	Not significant P = 0.8483

Table 6.14: CM1 - Recall - Weight Methods Comparison - Kruskal-Wallis

Tf-idf	Okapi	LTU
0.0121	0.0121	0.0121
0.0462	0.0475	0.0436
0.1195	0.1245	0.1218
0.2357	0.3288	0.2993
0.4146	0.6402	0.5989
0.5933	0.8519	0.8295
0.7534	1.0000	0.9020
0.8000	1.0000	1.0000
0.8947	1.0000	1.0000
1.0000	1.0000	1.0000
1.0000	1.0000	1.0000

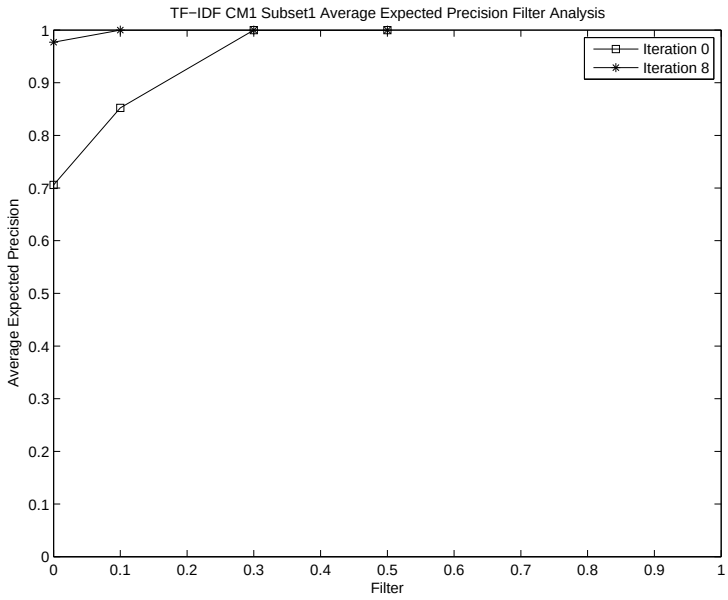
Groups	Count	Sum	Average	Variance
Tf-idf	11	5.8696	0.5336	0.1472
Okapi	11	7.0050	0.6368	0.1800
LTU	11	6.8072	0.6188	0.1754

Groups	df	Observations	T	P	T (ties)	P (ties)
3	2	33	0.910063	0.6344	0.945455	0.6233

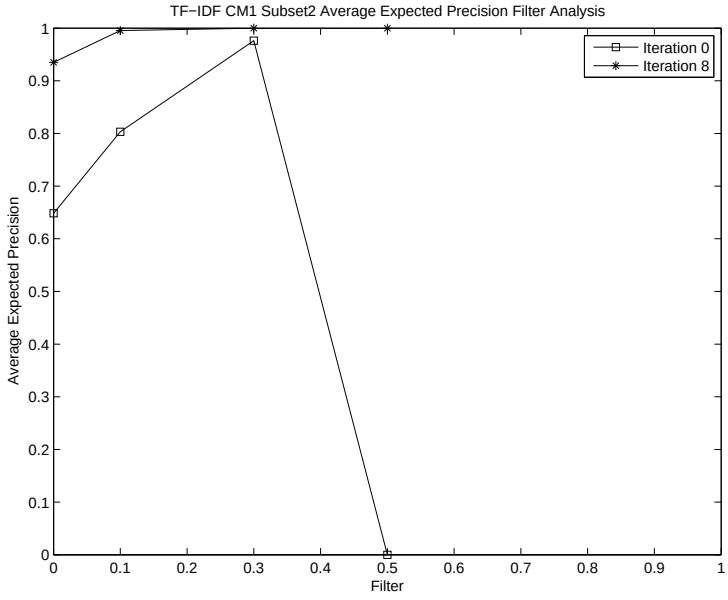
All Pairwise Comparisons (Dwass-Steel-Chritchlow-Flinger)	
Critical q (range) = 3.314493	
Tf-idf vs. Okapi ($\|1.227146\| > 3.314493$)	Not significant P = 0.6607
Tf-idf vs. LTU ($\|1.032051\| > 3.314493$)	Not significant P = 0.7458
Okapi vs. LTU ($\|-0.481046\| > 3.314493$)	Not significant P = 0.9382

All Pairwise Comparisons (Conover-Inman)	
Critical t (30 df) = 2.042272	
Tf-idf and Okapi ($3.818182 > 8.572104$)	Not significant P = 0.3702
Tf-idf and LTU ($2.727273 > 8.572104$)	Not significant P = 0.5208
Okapi and LTU ($1.090909 > 8.572104$)	Not significant P = 0.7967

Table 6.15: CM1 - Precision - Weight Methods Comparison - Kruskal Wallis

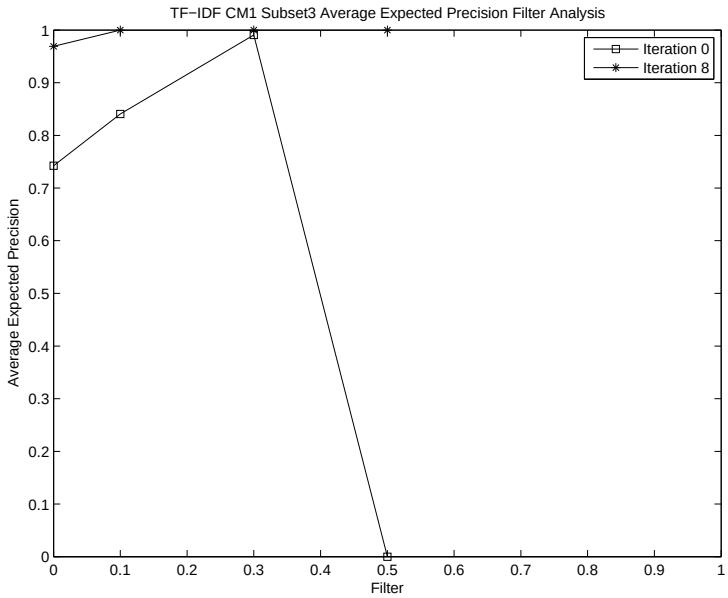


(a)

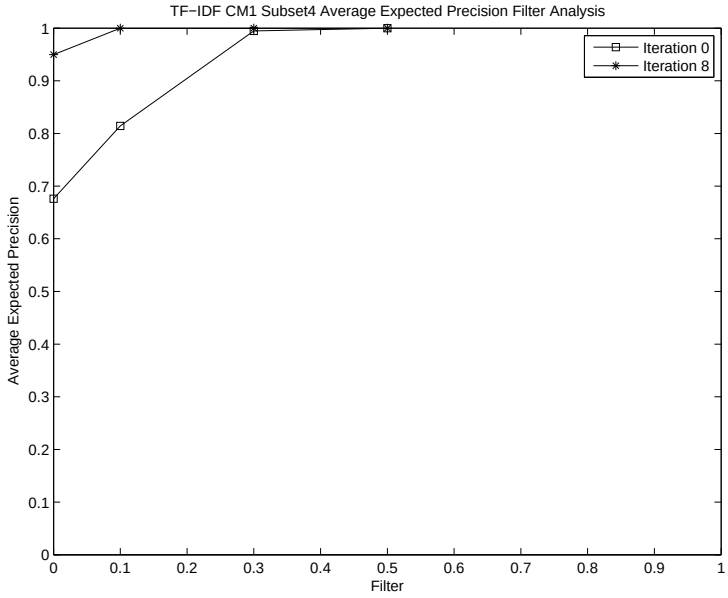


(b)

Figure 6.33: Filter Analysis - Average Expected Precision (a) CM1 Subset1 and (b) CM1 Subset2



(a)



(b)

Figure 6.34: Filter Analysis - Average Expected Precision (a) CM1 Subset3 and (b) CM1 Subset4

6.7 Vocabulary Base Analysis

The results obtained by the tf-idf method when both the documents and the queries were used to build the vocabulary base is shown in Table 6.17 and Table 6.18. Figure 6.38, Figure 6.39, and Figure 6.40 compare the precision and recall values obtained by both types of vocabulary bases for the tf-idf method. In the case of the tf-idf method with no filter, the difference in the vocabulary base did not seem to impact the results. However, the recall and precision values are slightly better for some of the filter values when both the documents and the queries were used to build the vocabulary base. This means that the new vocabulary base did not identify any new true links. However, it increased the relevance value of the true links.

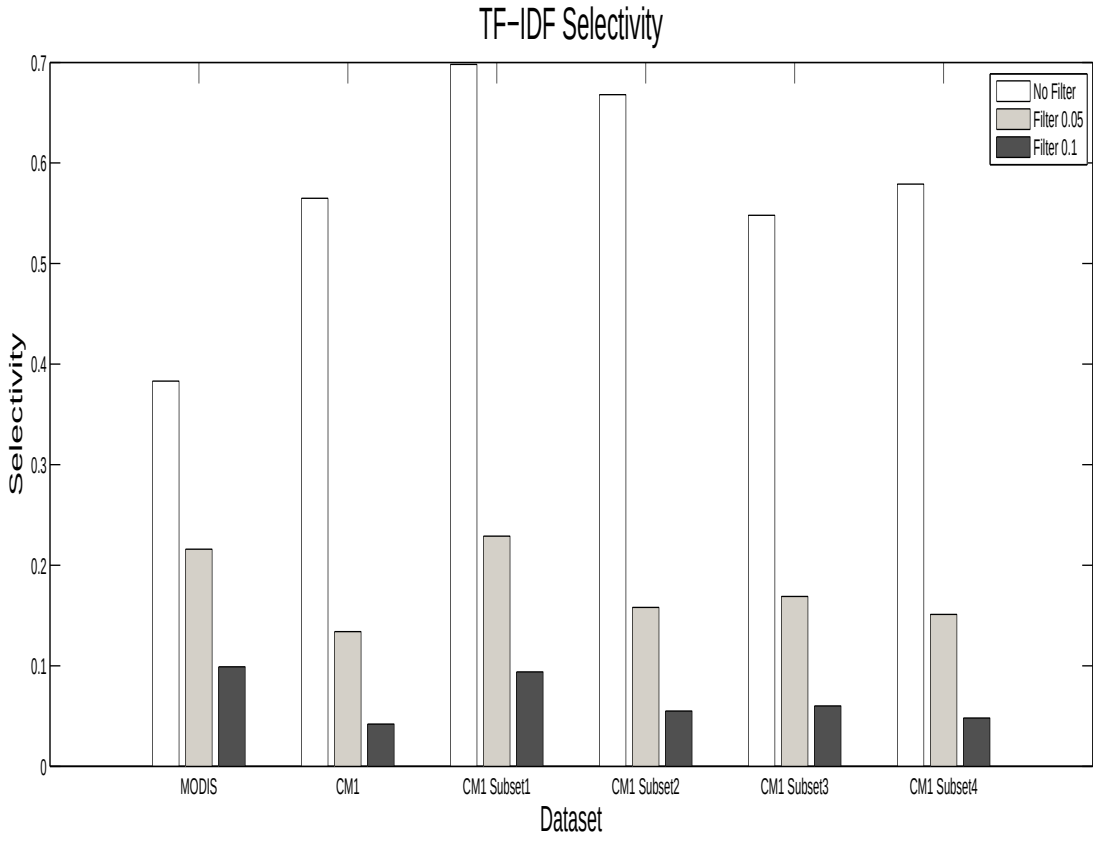


Figure 6.35: Selectivity

Dataset:	**Modis Weight 2**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.756	0.086	0.386	0.784	0.113	0.769	0.155	0.296	0.426
0.05	0.488	0.091	0.236	0.805	0.083	0.679	0.153	0.260	0.340
0.1	0.317	0.135	0.103	0.820	0.060	0.612	0.190	0.250	0.280
0.15	0.244	0.294	0.037	0.979	0.049	0.535	0.267	0.253	0.248
0.2	0.220	0.643	0.015	1.000	0.046	0.532	0.327	0.253	0.235
0.25	0.146	0.667	0.010	1.000	0.035	0.365	0.240	0.173	0.159
0.3	0.122	0.714	0.008	1.000	0.026	0.282	0.208	0.146	0.133
0.35	0.098	1.000	0.004	1.000	0.020	0.198	0.178	0.119	0.107
0.4	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.45	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.5	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054

Dataset:	**Modis Weight 3**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.756	0.086	0.386	0.794	0.113	0.769	0.155	0.296	0.426
0.05	0.488	0.092	0.234	0.812	0.083	0.679	0.154	0.262	0.341
0.1	0.415	0.173	0.105	0.881	0.072	0.603	0.245	0.324	0.364
0.15	0.244	0.313	0.034	0.979	0.049	0.535	0.274	0.255	0.249
0.2	0.220	0.600	0.016	1.000	0.046	0.532	0.321	0.251	0.234
0.25	0.171	0.700	0.011	1.000	0.040	0.448	0.275	0.201	0.185
0.3	0.122	0.714	0.008	1.000	0.026	0.282	0.208	0.146	0.133
0.35	0.098	1.000	0.004	1.000	0.020	0.198	0.178	0.119	0.107
0.4	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.45	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.5	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054

Table 6.16: Weight Analysis - CM1 Subset2 Filter 0.1, CM1 Subset3 Filter 0.1, and CM1 Subset4 Filter 0.1

Dataset:	**Modis**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.756	0.087	0.383	0.794	0.114	0.769	0.156	0.298	0.427
0.05	0.488	0.153	0.141	0.811	0.084	0.679	0.233	0.339	0.400
0.1	0.220	0.321	0.030	1.000	0.047	0.532	0.261	0.234	0.227
0.15	0.146	0.667	0.010	1.000	0.031	0.365	0.240	0.173	0.159
0.2	0.146	0.857	0.008	1.000	0.031	0.365	0.250	0.175	0.160
0.25	0.073	1.000	0.003	1.000	0.014	0.194	0.136	0.090	0.081
0.3	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.35	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.4	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.45	0.049	1.000	0.002	1.000	0.008	0.111	0.093	0.060	0.054
0.5	0.024	1.000	0.001	1.000	0.003	0.083	0.048	0.030	0.027

Dataset:	**CM1**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.981	0.012	0.565	0.630	0.023	0.990	0.024	0.058	0.109
0.05	0.895	0.048	0.131	0.665	0.020	0.918	0.090	0.196	0.322
0.1	0.662	0.124	0.037	0.814	0.015	0.698	0.210	0.355	0.462
0.15	0.468	0.329	0.010	0.934	0.010	0.515	0.386	0.432	0.449
0.2	0.291	0.640	0.003	0.984	0.007	0.325	0.400	0.326	0.308
0.25	0.191	0.852	0.002	1.000	0.004	0.208	0.312	0.226	0.207
0.3	0.108	1.000	0.001	1.000	0.002	0.134	0.195	0.131	0.119
0.35	0.047	1.000	0.000	1.000	0.001	0.053	0.090	0.058	0.052
0.4	0.019	1.000	0.000	1.000	0.001	0.027	0.038	0.024	0.021
0.45	0.011	1.000	0.000	1.000	0.000	0.019	0.022	0.014	0.012
0.5	0.011	1.000	0.000	1.000	0.000	0.019	0.022	0.014	0.012

Dataset:	**CM1 Subset1**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	1.000	0.044	0.698	0.711	0.062	1.000	0.084	0.187	0.314
0.05	0.886	0.166	0.163	0.775	0.055	0.906	0.279	0.474	0.618
0.1	0.514	0.367	0.043	0.981	0.031	0.511	0.429	0.476	0.495
0.15	0.257	0.529	0.015	1.000	0.014	0.289	0.346	0.287	0.271
0.2	0.171	0.857	0.006	1.000	0.009	0.217	0.286	0.204	0.186
0.25	0.114	1.000	0.003	1.000	0.006	0.178	0.205	0.139	0.125
0.3	0.114	1.000	0.003	1.000	0.006	0.178	0.205	0.139	0.125
0.35	0.057	1.000	0.002	1.000	0.003	0.089	0.108	0.070	0.063
0.4	0.029	1.000	0.001	1.000	0.002	0.067	0.056	0.035	0.032
0.45	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 6.17: Vocabulary Base Analysis - Modis, CM1, and CM1 Subset1

Dataset:	**CM1 Subset2**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.989	0.025	0.668	0.635	0.067	1.000	0.048	0.112	0.201
0.05	0.804	0.098	0.136	0.689	0.032	0.842	0.174	0.329	0.467
0.1	0.489	0.232	0.035	0.879	0.019	0.554	0.315	0.400	0.440
0.15	0.228	0.438	0.009	0.971	0.009	0.266	0.300	0.252	0.240
0.2	0.141	0.650	0.004	0.955	0.005	0.167	0.232	0.168	0.153
0.25	0.065	0.857	0.001	1.000	0.003	0.081	0.121	0.080	0.072
0.3	0.043	1.000	0.001	1.000	0.001	0.067	0.083	0.054	0.048
0.35	0.033	1.000	0.001	1.000	0.001	0.042	0.063	0.040	0.036
0.4	0.011	1.000	0.000	1.000	0.000	0.008	0.022	0.014	0.012
0.45	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Dataset:	**CM1 Subset3**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.991	0.015	0.549	0.725	0.039	0.994	0.029	0.069	0.130
0.05	0.923	0.052	0.146	0.742	0.033	0.929	0.098	0.211	0.343
0.1	0.641	0.129	0.040	0.882	0.023	0.681	0.215	0.358	0.459
0.15	0.427	0.331	0.010	0.962	0.013	0.453	0.373	0.404	0.415
0.2	0.316	0.597	0.004	0.951	0.010	0.361	0.413	0.349	0.332
0.25	0.222	0.867	0.002	0.992	0.007	0.265	0.354	0.261	0.240
0.3	0.103	0.857	0.001	0.981	0.003	0.119	0.183	0.124	0.112
0.35	0.017	0.667	0.000	0.833	0.000	0.011	0.033	0.021	0.019
0.4	0.009	0.500	0.000	1.000	0.000	0.006	0.017	0.011	0.009
0.45	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Dataset:	**CM1 Subset4**								
Filter	Rec	Pr	Sel	AEPr	AvPr	AvRec	f_1	f_2	f_3
0.0	0.983	0.014	0.579	0.665	0.028	0.991	0.027	0.064	0.121
0.05	0.870	0.052	0.133	0.701	0.023	0.916	0.099	0.211	0.340
0.1	0.582	0.133	0.035	0.867	0.015	0.658	0.217	0.348	0.435
0.15	0.368	0.310	0.010	0.947	0.008	0.428	0.337	0.355	0.361
0.2	0.247	0.488	0.004	0.958	0.005	0.306	0.328	0.274	0.260
0.25	0.163	0.780	0.002	0.995	0.004	0.201	0.270	0.194	0.177
0.3	0.109	0.839	0.001	0.992	0.002	0.146	0.193	0.132	0.119
0.35	0.038	0.818	0.000	0.979	0.001	0.054	0.072	0.047	0.042
0.4	0.025	0.857	0.000	1.000	0.001	0.038	0.049	0.031	0.028
0.45	0.013	1.000	0.000	1.000	0.000	0.023	0.025	0.016	0.014
0.5	0.013	1.000	0.000	1.000	0.000	0.023	0.025	0.016	0.014

Table 6.18: Vocabulary Base Analysis - CM1 Subset2, CM1 Subset3, and CM1 Subset4

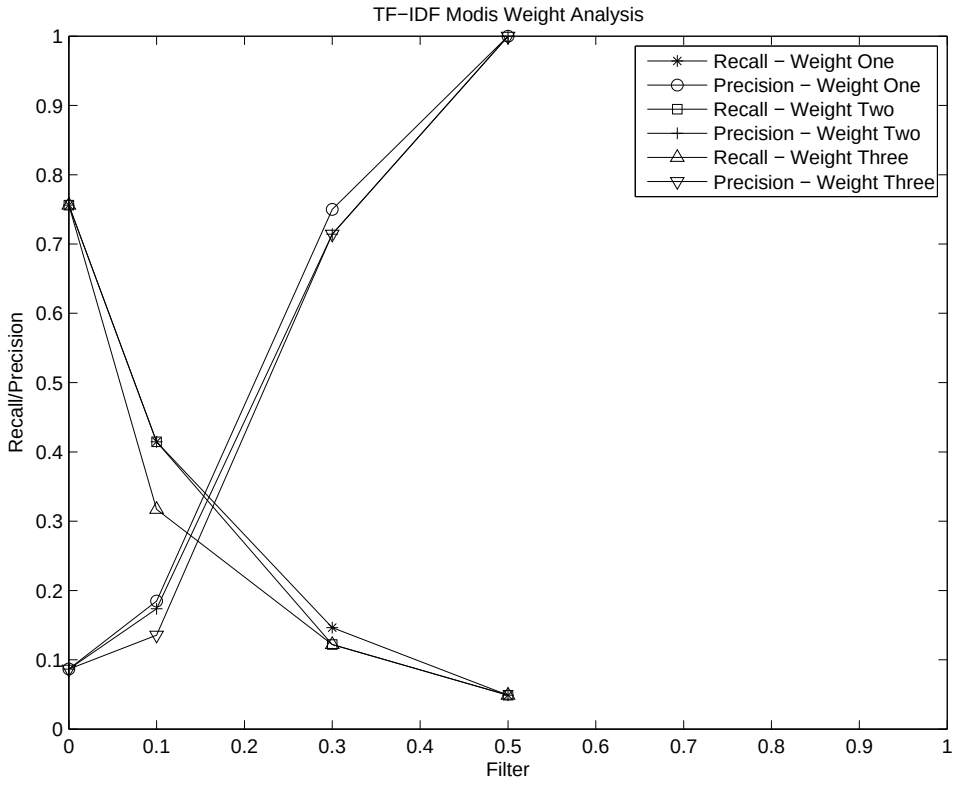


Figure 6.36: Weight Analysis

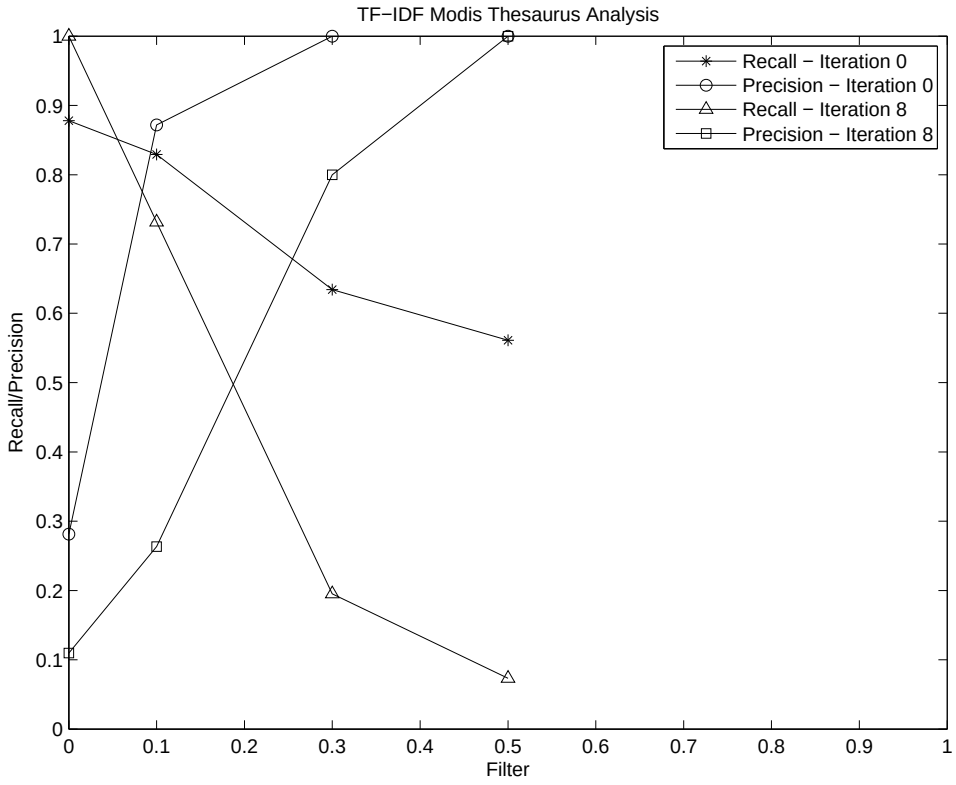
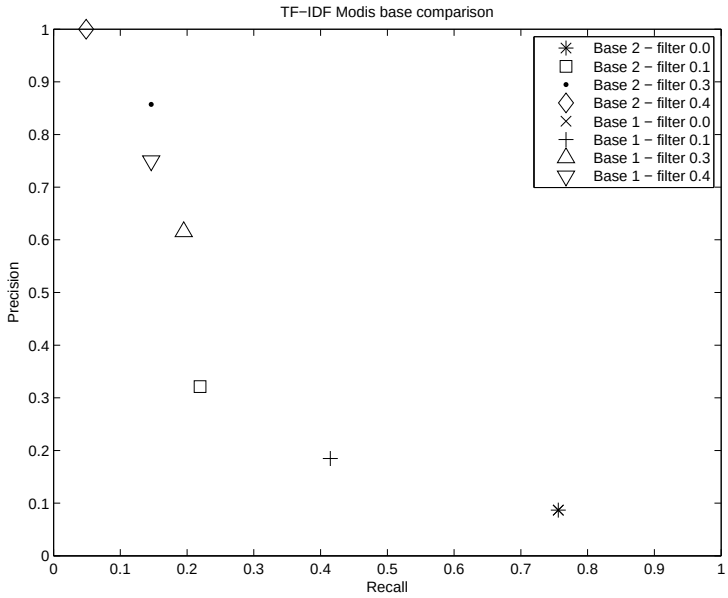
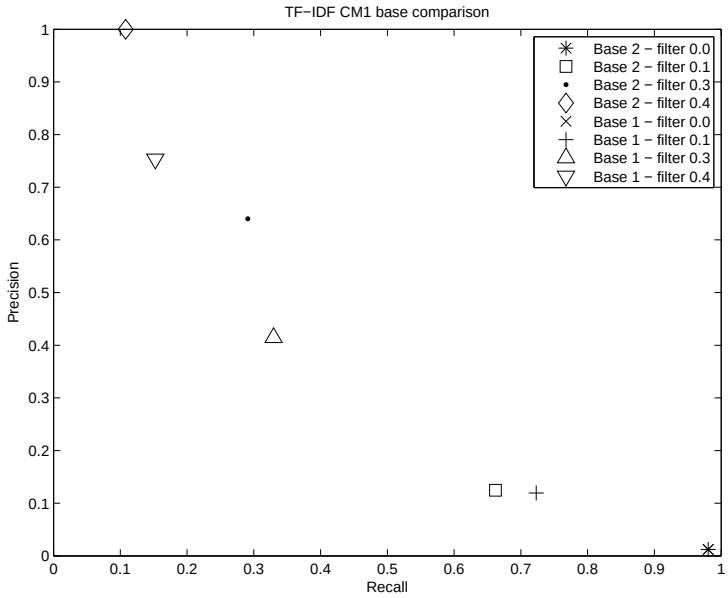


Figure 6.37: Thesaurus Analysis

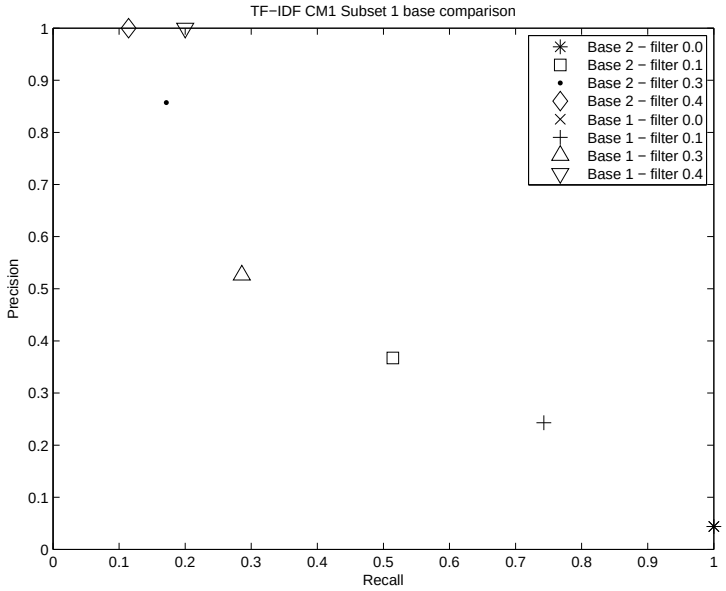


(a)

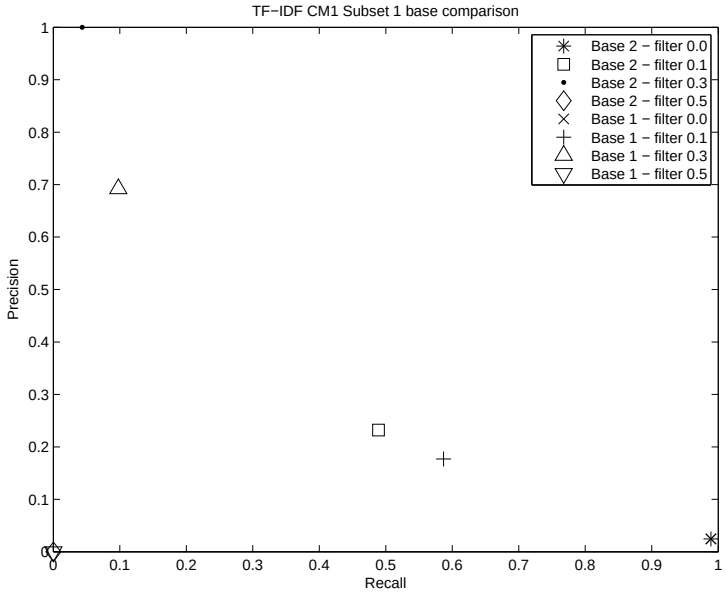


(b)

Figure 6.38: Vocabulary Base (a) Modis and (b) CM1

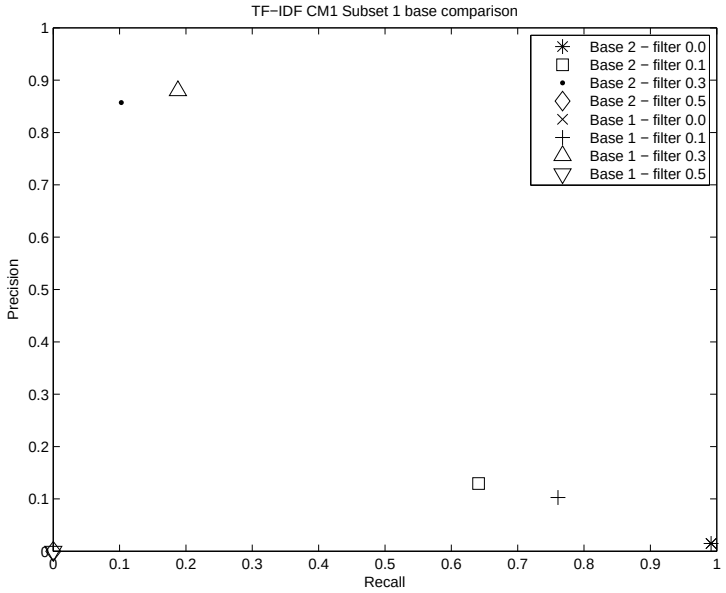


(a)

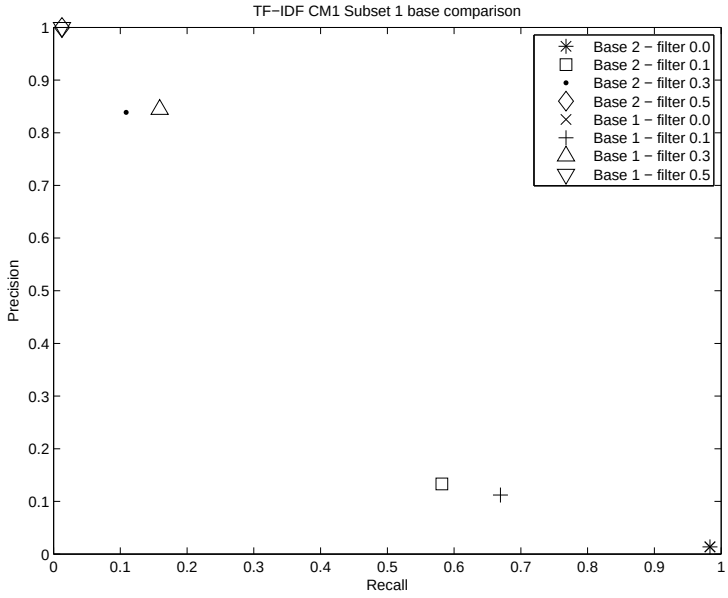


(b)

Figure 6.39: Vocabulary Base (a) CM1 Subset1 and (b) CM1 Subset2

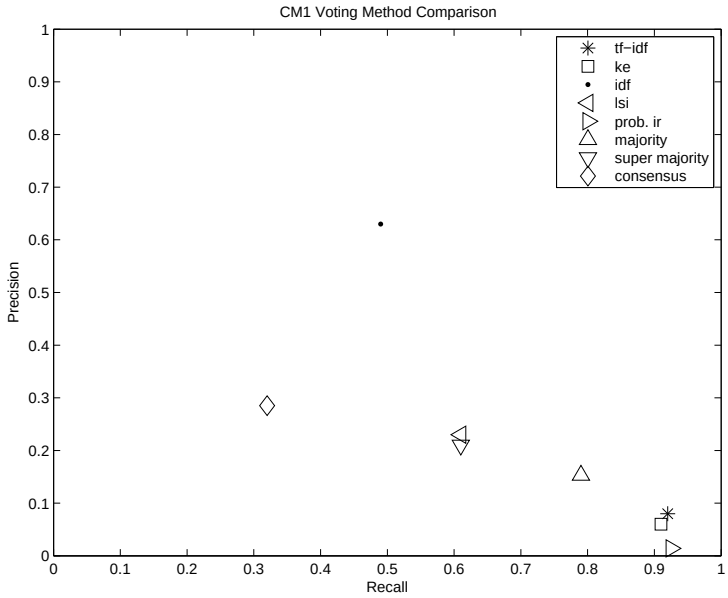


(a)

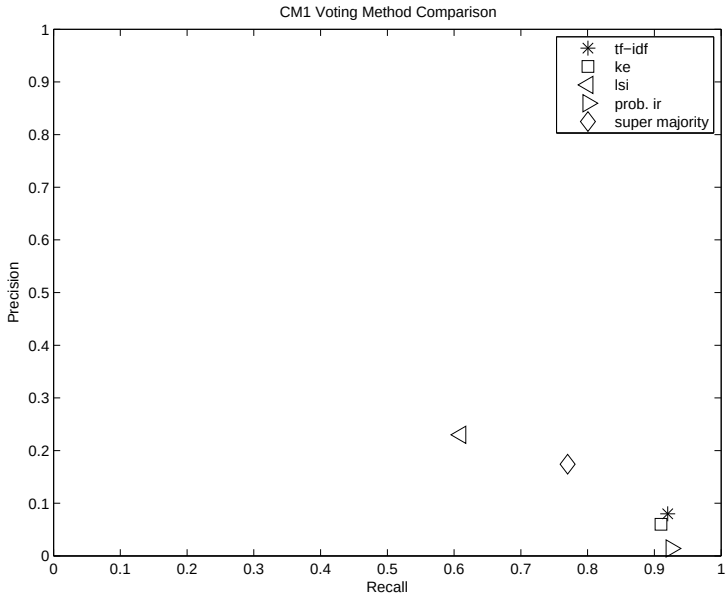


(b)

Figure 6.40: Vocabulary Base (a) CM1 Subset3 and (b) CM1 Subset4

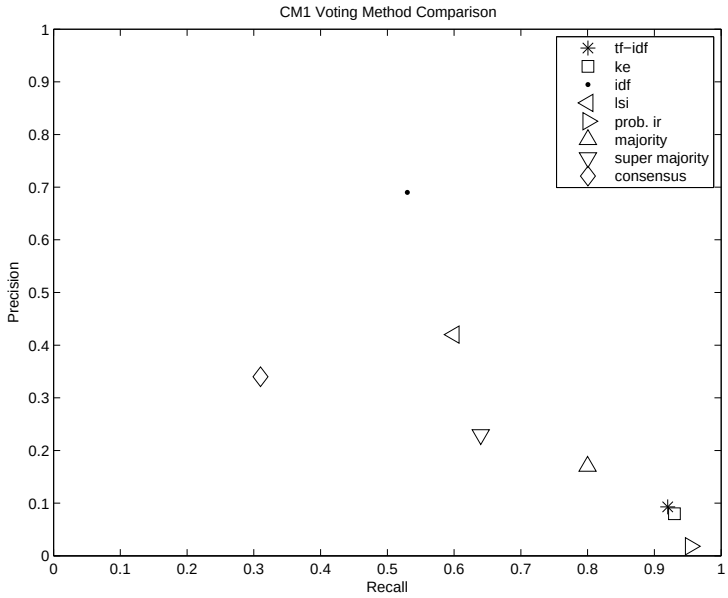


(a)

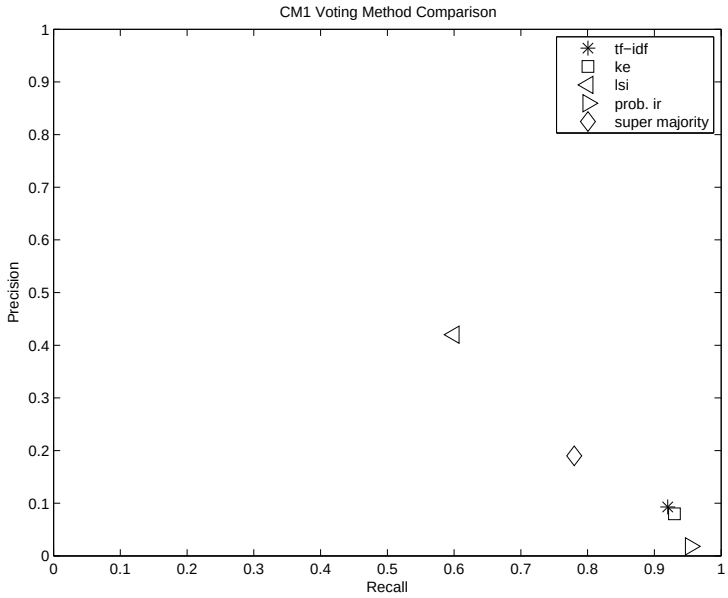


(b)

Figure 6.41: Voting Method - CM1 (a) With IDF and (b) Without IDF

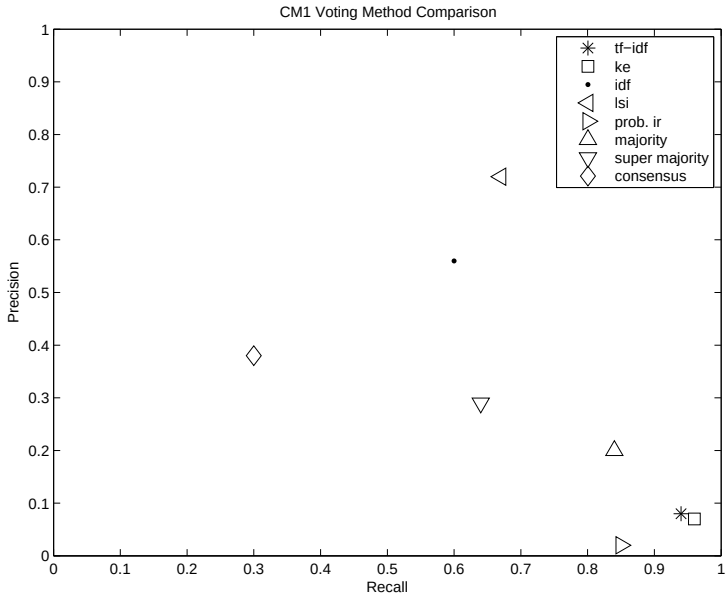


(a)

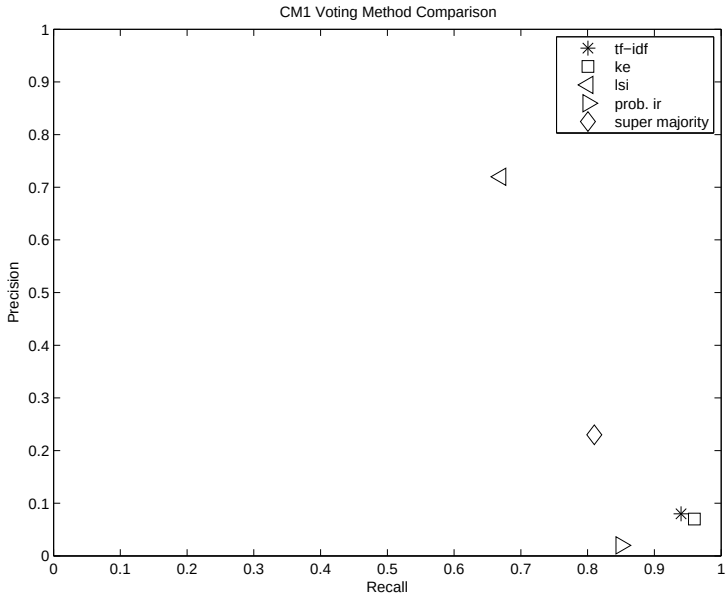


(b)

Figure 6.42: Voting Method - CM1 Subset4 (a) With IDF and (b) Without IDF



(a)



(b)

Figure 6.43: Voting Method - CM1 Subset3 (a) With IDF and (b) Without IDF

6.8 Voting Tool Analysis

Table 6.19 shows the recall and precision values obtained for the CM1 dataset by tf-idf, KE, IDF, LSI, and probabilistic IR at filter 0.05. It also shows the recall and precision values obtained by the voting rules majority, super majority, and consensus. Table 6.19 also shows the recall and precision values obtained by super majority when tf-idf, KY, IDF, and LSI were used. Notice that LSI and IDF produced acceptable and excellent precision, respectively. However, LSI and IDF produced acceptable and poor recall, respectively. Conversely, for tf-idf, KE, and probabilistic IR, had excellent recall, but they had very poor precision. Notice that, we were able to achieve 79% recall and 17% precision with majority rule.

Figure 6.41 compares the recall and precision obtained for the CM1 dataset by tf-idf, KE, IDF, LSI, and probabilistic IR with that of majority, super majority, and consensus rules. Since, the consensus rule is very strict, it produced poor recall and good precision. The super majority rule produced acceptable recall and precision. The majority rule produced good recall and poor precision. When tf-idf, KE, IDF, and LSI methods were used, the super majority rule produced good recall and poor precision. Though the precision produced by most of the voting rules were poor, the precision still increased significantly with an insignificant decrease in recall.

Table 6.20 shows the recall and precision values obtained for the CM1 subset4 dataset by tf-idf, KE, IDF, LSI, and probabilistic IR at filter 0.05. It also shows the recall and precision values obtained by the voting rules majority, super majority, consensus, and super majority when IDF was not used. Again, IDF had excellent precision and LSI had acceptable precision. Also, LSI had acceptable recall and IDF had poor recall. The other three methods had very poor precision with excellent recall. The majority rule produced 80% recall and 17% precision.

Figure 6.42 compares the recall and precision obtained for the CM1 subset4 dataset by tf-idf, KE, IDF, LSI, and probabilistic IR with that of majority, super majority, and consensus rules. It can be seen from the graph that majority rule outperformed other rules when all the five methods were used. When IDF was not used, the super majority rule produced good recall with excellent precision.

Method	Recall	Precision
Tf-idf	0.92	0.08
KE	0.91	0.06
IDF	0.49	0.63
LSI	0.61	0.23
Probabilistic IR	0.93	0.014

Rule	Recall	Precision
Majority	0.79	0.15
Super majority	0.61	0.21
Consnsus	0.32	0.29
Super majority without IDF	0.77	0.17

Table 6.19: Voting Tool Analysis - CM1 - Comparison of Recall/Precision values obtained by different methods and rules

Method	Recall	Precision
Tf-idf	0.92	0.09
KE	0.93	0.08
IDF	0.53	0.69
LSI	0.60	0.42
Probabilistic IR	0.95	0.018

Rule	Recall	Precision
Majority	0.80	0.17
Super majority	0.64	0.23
Consnsus	0.31	0.34
Super majority without IDF	0.78	0.19

Table 6.20: Voting Tool Analysis - CM1 Subset4 - Comparison of Recall/Precision values obtained by different methods and rules

Method	Recall	Precision
Tf-idf	0.94	0.08
KE	0.96	0.07
IDF	0.60	0.56
LSI	0.67	0.72
Probabilistic IR	0.85	0.02

Rule	Recall	Precision
Majority	0.84	0.20
Super majority	0.64	0.29
Consnsus	0.30	0.38
Super majority without IDF	0.81	0.23

Table 6.21: Voting Tool Analysis - CM1 Subset3 - Comparison of Recall/Precision values obtained by different methods and rules

Table 6.21 shows the recall and precision values obtained for the CM1 subset3 dataset by tf-idf, KE, IDF, LSI, and probabilistic IR at filter 0.05. It also shows the recall and precision values obtained by the voting rules majority, super majority, and consensus. Table 6.21 also shows the recall and precision values obtained by super majority when tf-idf, KY, IDF, and LSI were used. Both, LSI and IDF had acceptable recall and excellent precision. Similar to the previous cases, the other three methods had excellent recall and very poor precision. The best recall-precision pair, 84% recall and 20% precision, was obtained by the majority rule. When IDF was not used, the super majority rule produced 81% recall and 23% precision.

Figure 6.42 compares the recall and precision obtained for the CM1 subset3 dataset by tf-idf, KE, IDF, LSI, and probabilistic IR with that of majority, super majority, and consensus rules. Notice that the majority rule obtained almost the same recall as the probabilistic IR, but with 10 times better precision. The super majority rule without IDF obtained slightly better precision compared to the majority rule using all the five methods.

Observations

- Voting mechanism improves the precision significantly without reducing the recall

significantly.

- It is possible to obtain excellent recall and acceptable precision using methods that have either, excellent recall and poor precision, or, poor recall and good precision.
- The consensus rule is very strict and it may not be effective when one of the methods in the committee produces poor recall.
- The majority rule seems to strike a good balance balance between recall and precision.
- In our experiments, the super majority rule works better when IDF was not used.

Chapter 7

Conclusion and Future Work

This chapter summarizes this work with some concluding remarks, points out the contributions of this work, and discusses future work.

7.1 Conclusion

The primary objective of this work was to assist analysts in the traceability links generation process using information retrieval techniques to improve the quality of the traceability links generated and to save analysts' time. We detailed the significance of the traceability links generation process in the software development lifecycle. We illustrated the mundaneness and lengthiness of the tracing process. We also explained the similarities of the tracing and information retrieval processes and showed that the tracing process can be reduced to an information retrieval process.

In this work, we adapted the vector space model (tf-idf), keyword extraction using χ^2, LSI, and probabilistic information retrieval methods. We also analyzed the use of a thesaurus to improve the vector space model. We implemented keyword extraction using IDF to cross check the validity of the keyword extraction using χ^2. We also used filtering and analyst feedback to improve the results generated by the above mentioned methods. We looked at using different weighting schemes and vocabulary bases.

We implemented a requirements tracing tool, RETRO, containing all the methods and features explained in the previous paragraph. Also, we developed a voting tool that filters traceability links based on whether the links were generated by certain methods in the IR toolbox. We validated our methods using six datasets. We also wrote a tool to automatically collect metrics from the traceability links generated.

From the selectivity values calculated for different methods, we see that, when using RETRO, the analyst needs to verify fewer links. In [26], we compared the time taken by an analyst to perform tracing using RETRO versus tracing manually. It is clear from the results that the analysts' workload has been greatly reduced. At the same time, we made sure that the time saved was not at the cost of the quality of the traceability links generated. Hence, we used recall, precision, and other secondary metrics to measure the quality of the traceability links generated.

The vector space model produced results with good recall for the MODIS dataset. However, the precision was poor. Using filters, we improved precision at the cost of recall. At the same time, analyst feedback improved the recall significantly, but sometimes at the cost of precision. When filters and analyst feedback were combined, we were able to obtain good recall and precision values. The keyword extraction method using χ^2 was able to produce the same recall and precision values as the vector space model, but with better similarity measures for the individual links. Also, the keyword extraction using IDF did not perform well, as expected. This proved that using the keyword extraction method to idenfity important keywords certainly provided good payoff.

7.2 Contributions

As discussed in the related works chapter, there are other researchers who use information retrieval techniques to generate traceability links for different software artifacts. It should be noted that most of this work was performed in parallell with our work, and we did not refer to these works when we started our research. However, this dissertation has the following unique contributions to the traceability research community:

- using analyst feedback to improve the quality of the treaceability links generated,
- using the keyword extraction technique to identify the important keywords and thus obtain results with better secondary measures,
- using both the document and the query collection to build the vocabulary base, and
- using a committe of methods to decide on the traceability links to be included in the candidate list.

7.3 Future Work

This work focuses on the dynamic generation of traceability links for unstructured textual artifacts. All the methods discussed in this work can be extended to generate traceability links for other structured software artifacts such as design documents, source code, etc. For example, in order to use our methods to trace design documents with design diagrams, the infromation represented in the design diagrams should be interpreted and converted to a format that our methods can read.

The structure of the textual software artifacts may contain useful information. This information can be used to improve the trace results. For example, if a document is organized into sections and subsections, the hierarchical information can be used while generating traceability links. It is quite possible that two matching sections may have matching sub-sections too.

Currently, our approach does not distinguish the links into different types. Classifying the links into different types may make the traces more understandable. Also, our approach does not identify if a particular element is completely satisfied by all the links found. Identifying the traceability link satisfaction will help the analyst decide when to stop tracing.

Appendix A

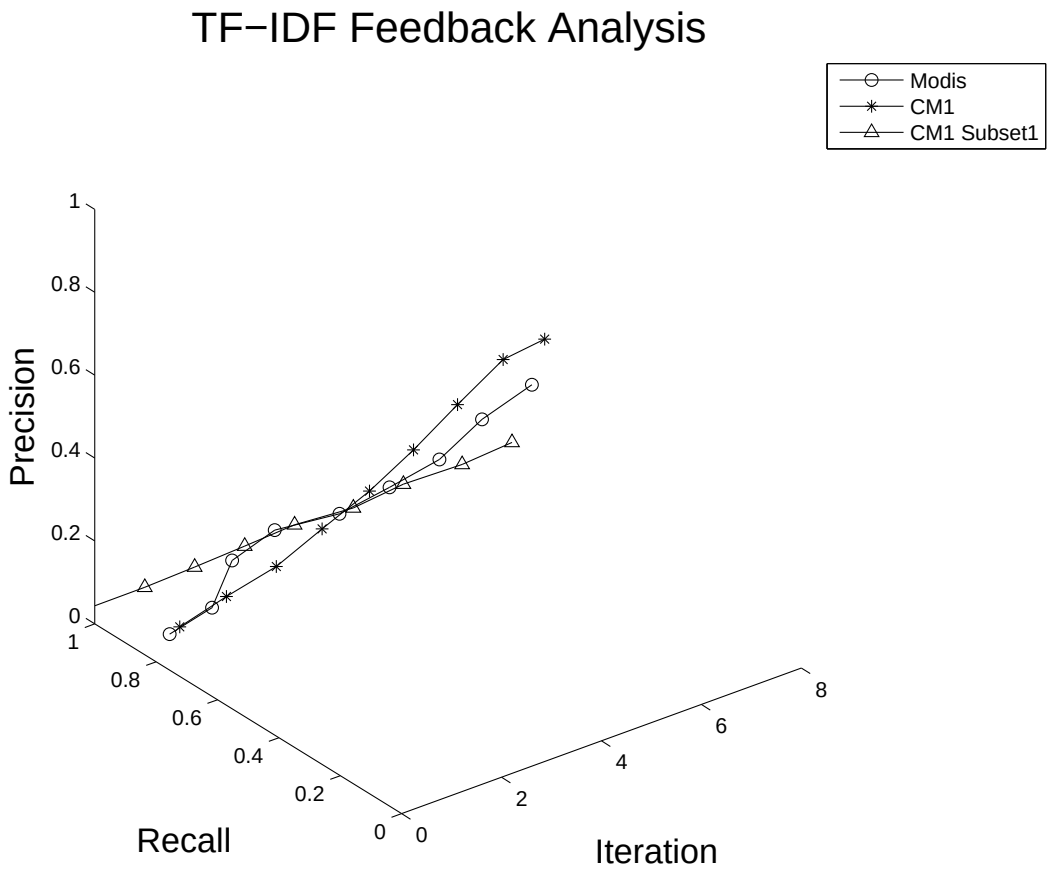


Figure A.1: Recall vs. Precision - Tf-idf - Modis, CM1, and CM1 Subset1.

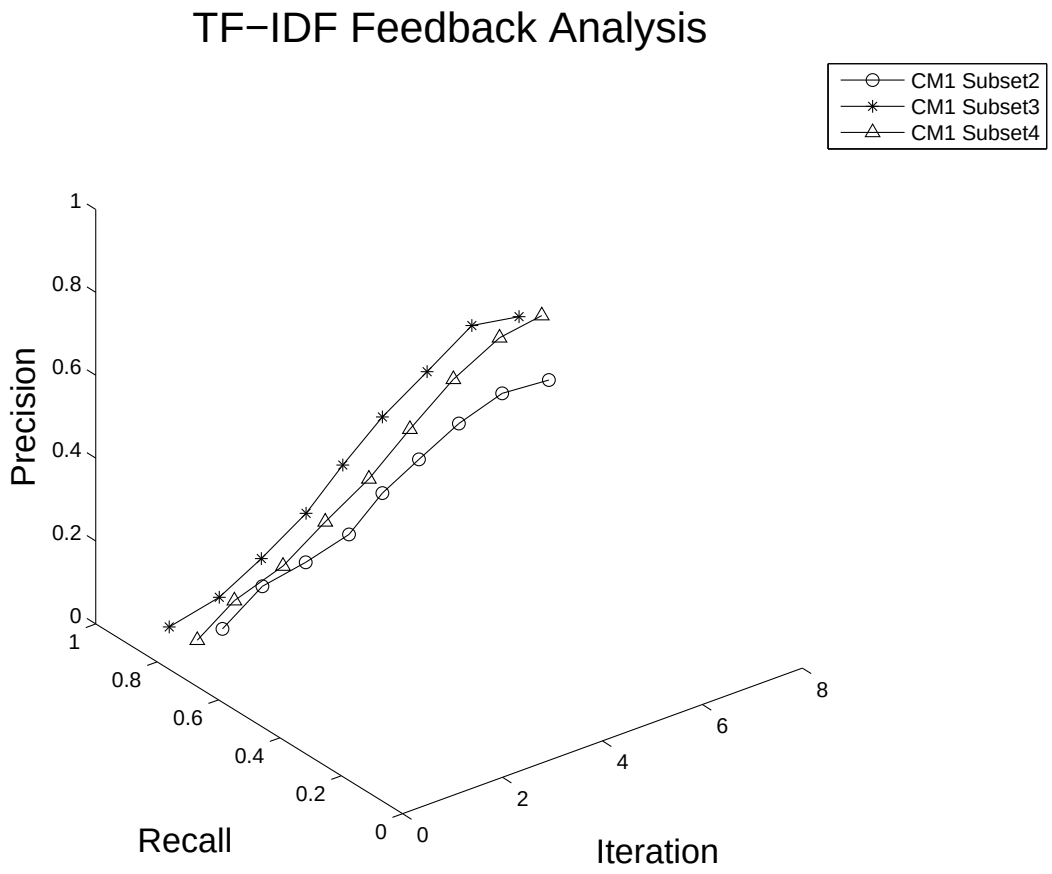


Figure A.2: Recall vs. Precision - Tf-idf - CM1 Subset2, CM1 Subset3 and CM1 Subset4.

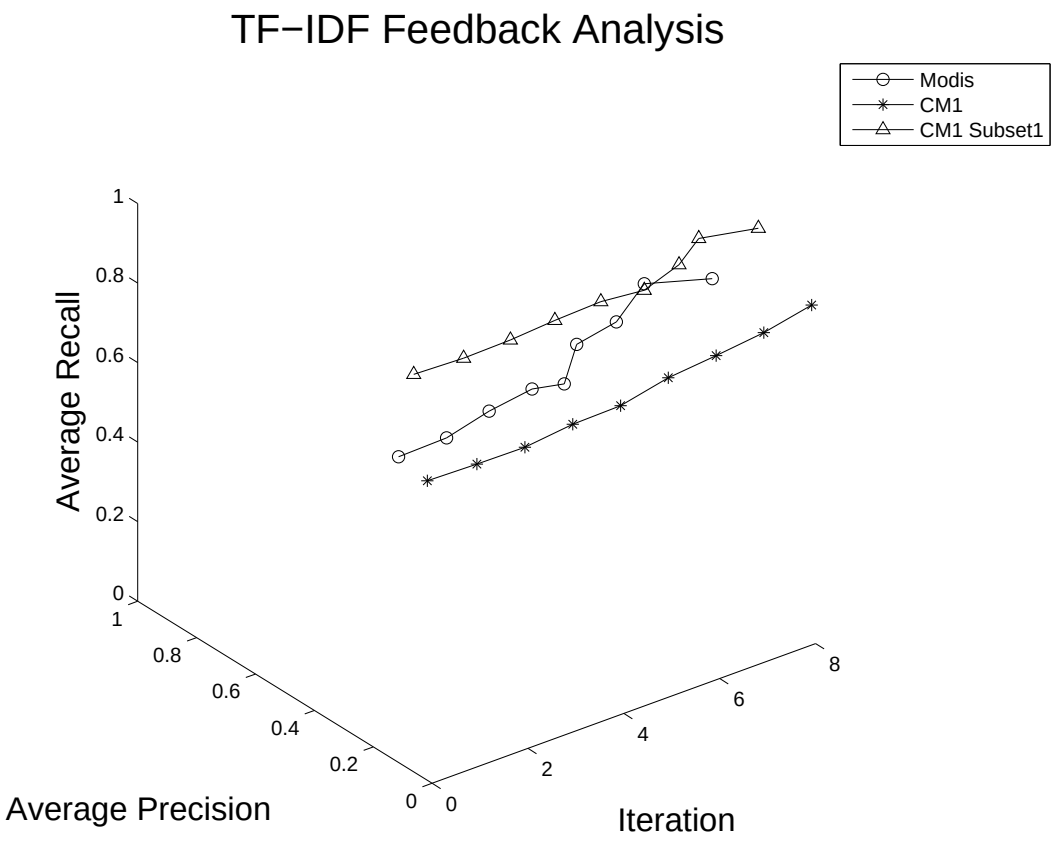


Figure A.3: Average Recall vs. Average Precision - Tf-idf - Modis, CM1, and CM1 Subset1.

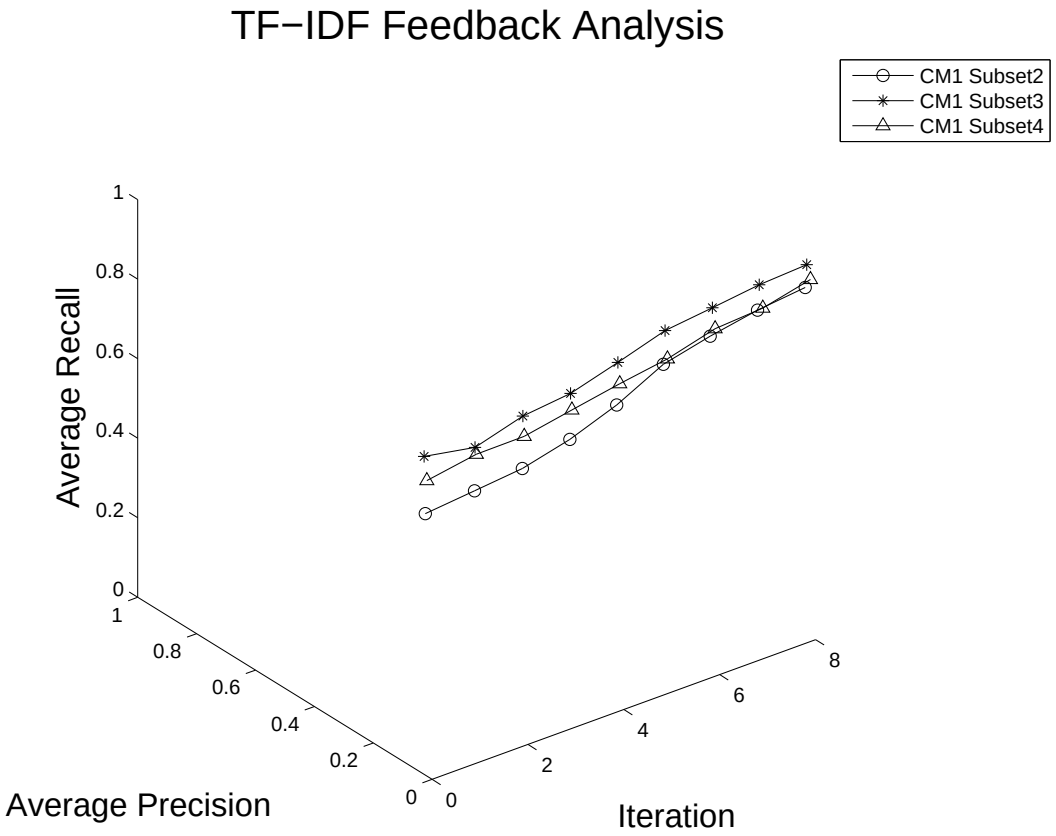


Figure A.4: Average Recall vs. Average Precision - Tf-idf - CM1 Subset2, CM1 Subset3 and CM1 Subset4.

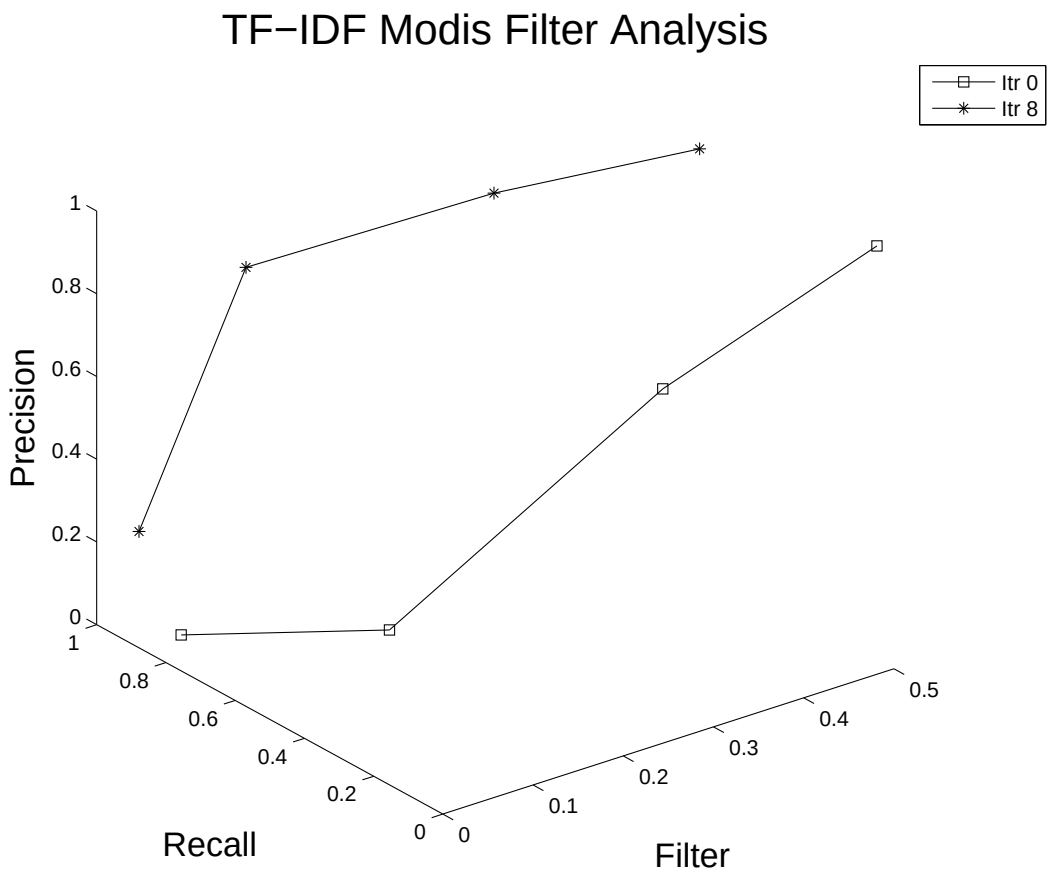


Figure A.5: Filter Analysis - Modis - Recall vs. Precision (Tf-idf)

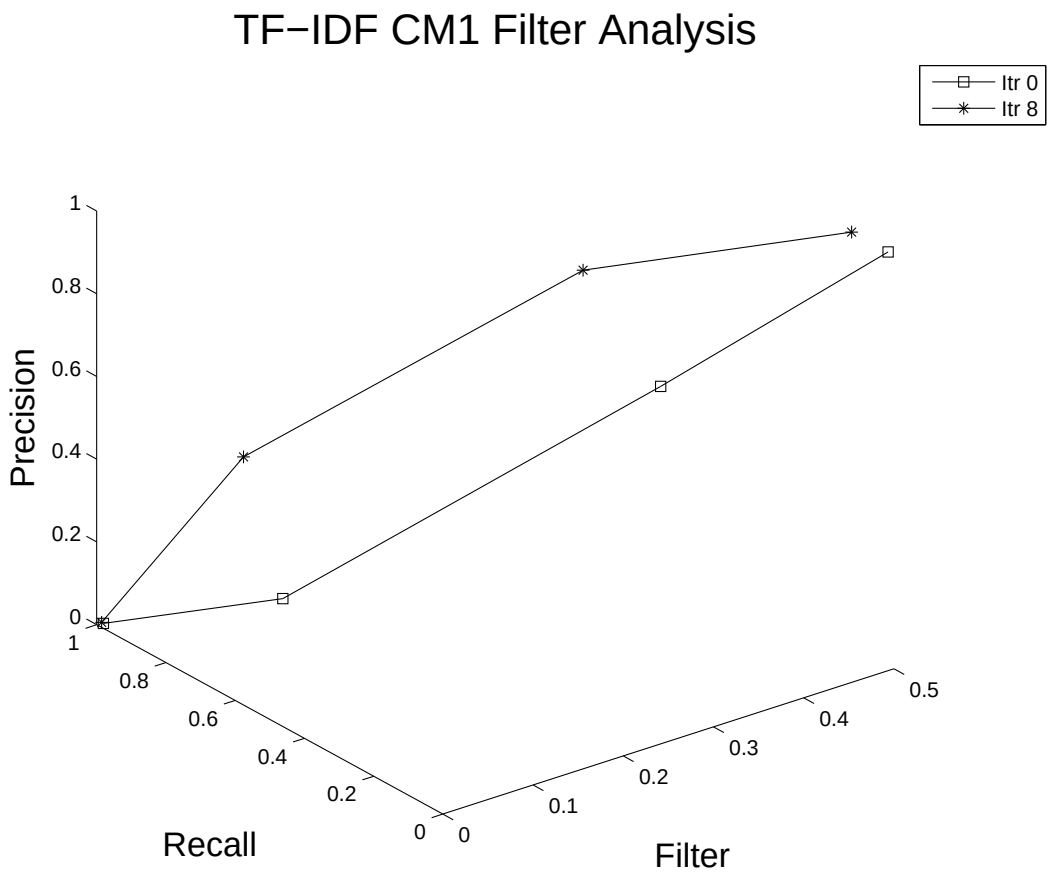


Figure A.6: Filter Analysis - CM1 - Recall vs. Precision (Tf-idf)

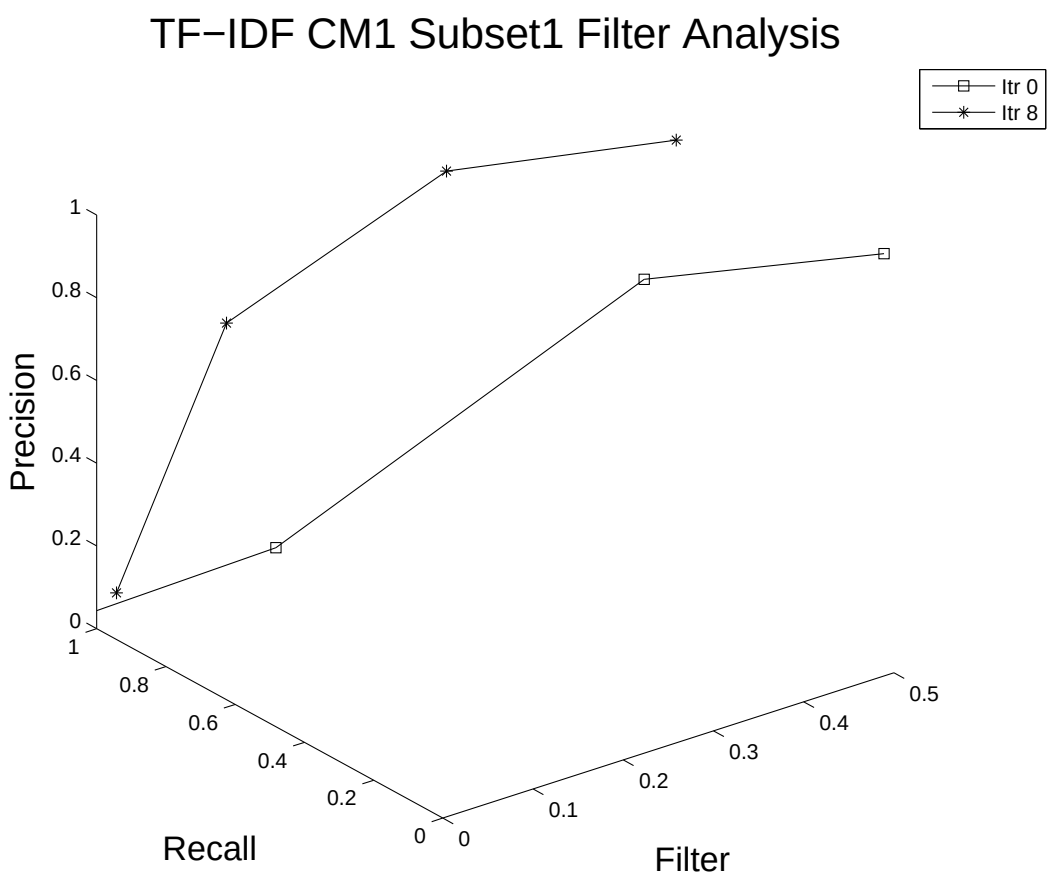


Figure A.7: Filter Analysis - CM1 Subset1 - Recall vs. Precision (Tf-idf)

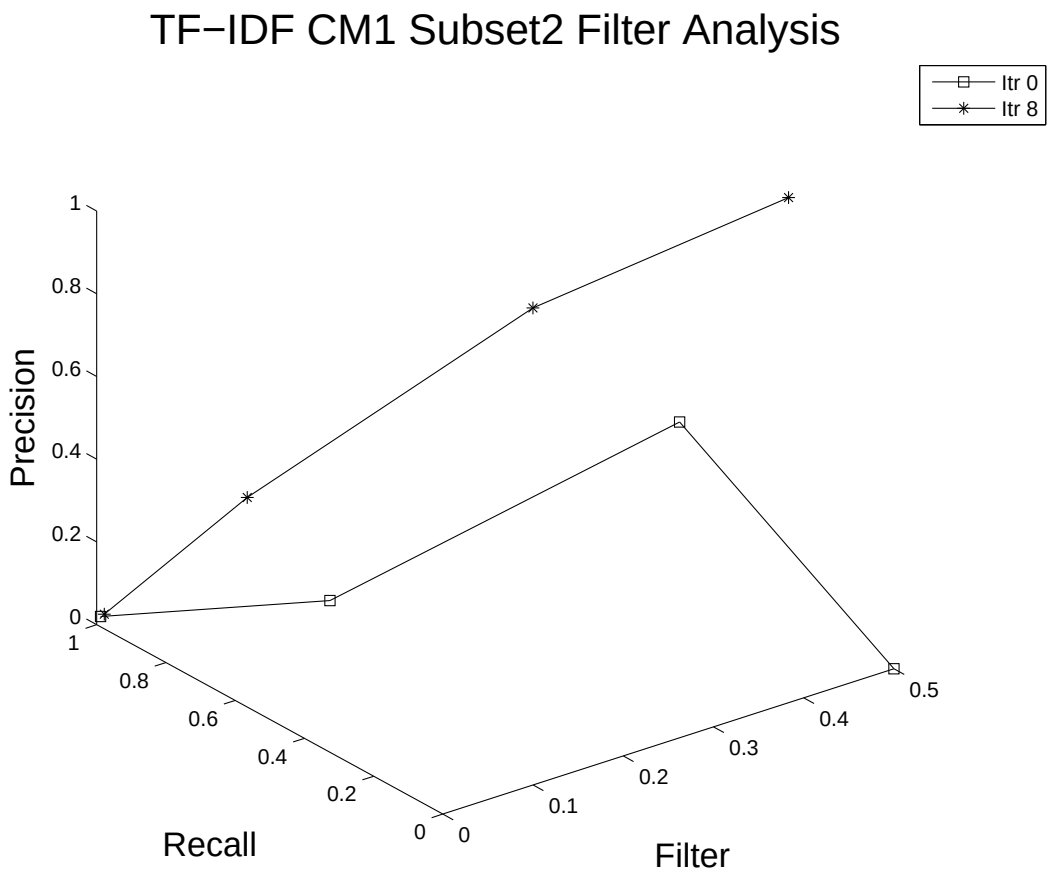


Figure A.8: Filter Analysis - CM1 Subset2 - Recall vs. Precision (Tf-idf)

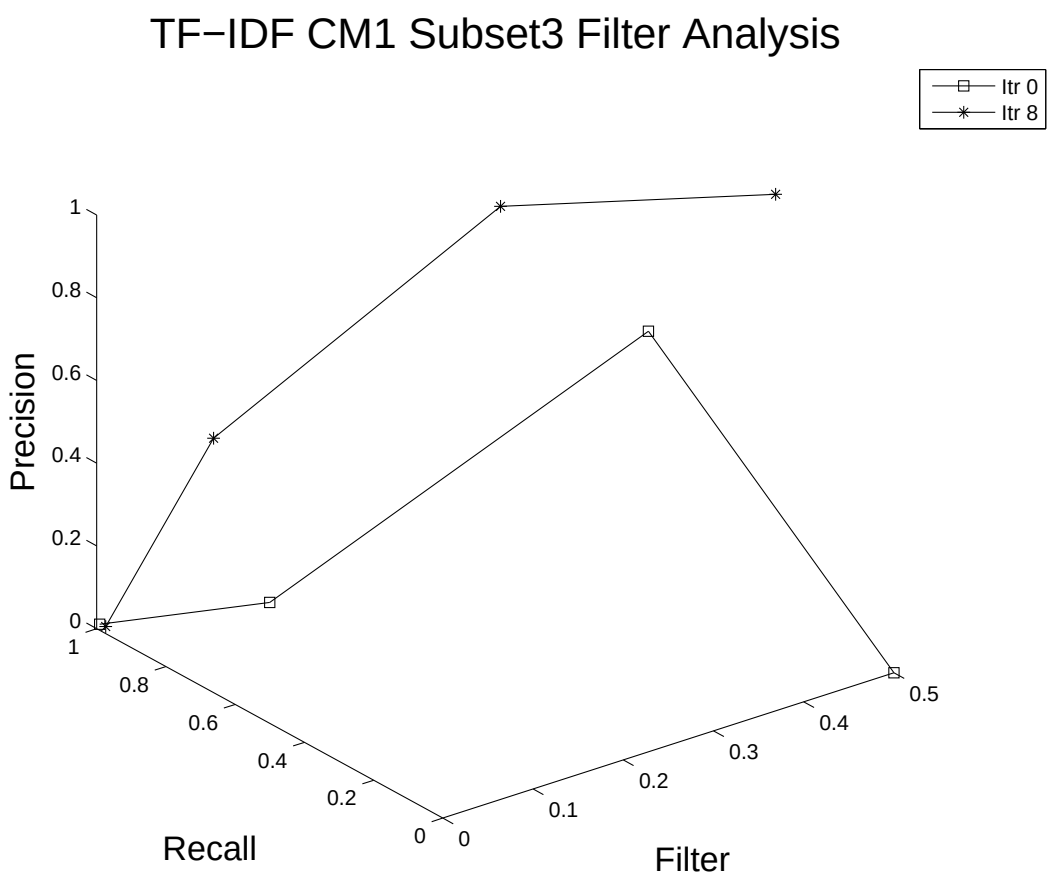


Figure A.9: Filter Analysis - CM1 Subset3 - Recall vs. Precision (Tf-idf)

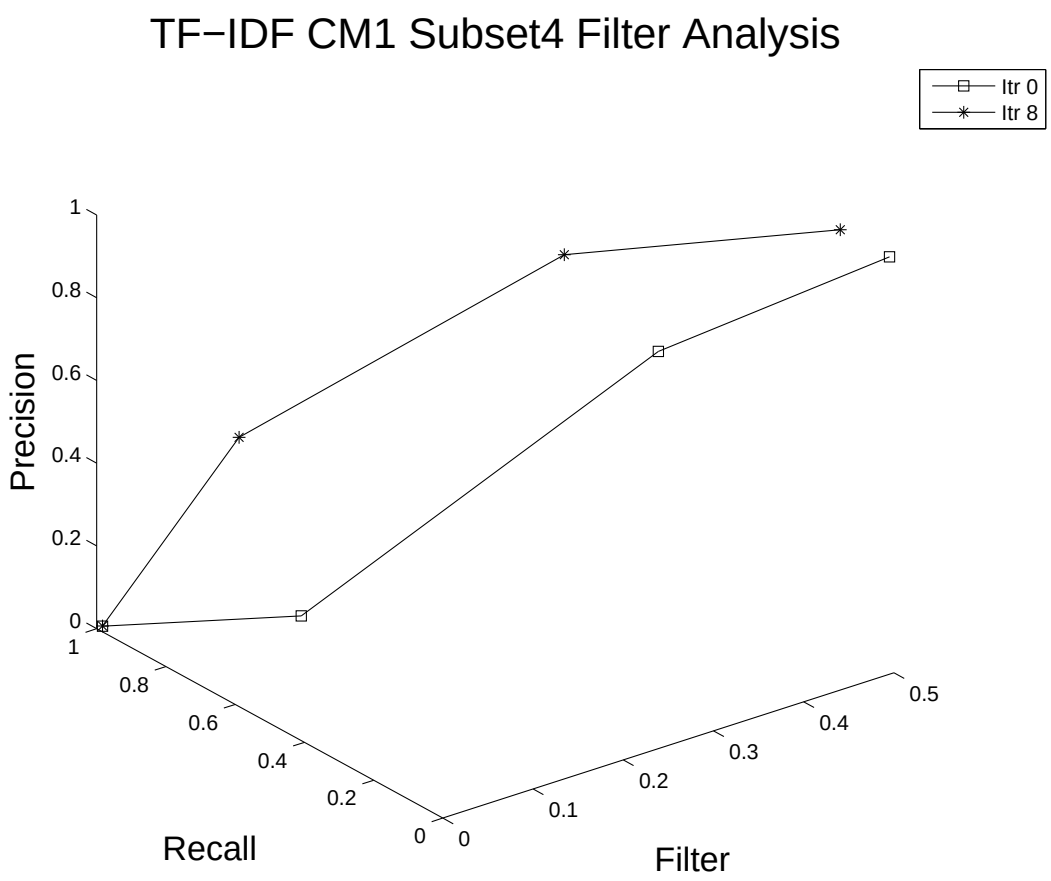


Figure A.10: Filter Analysis - CM1 Subset4 - Recall vs. Precision (Tf-idf)

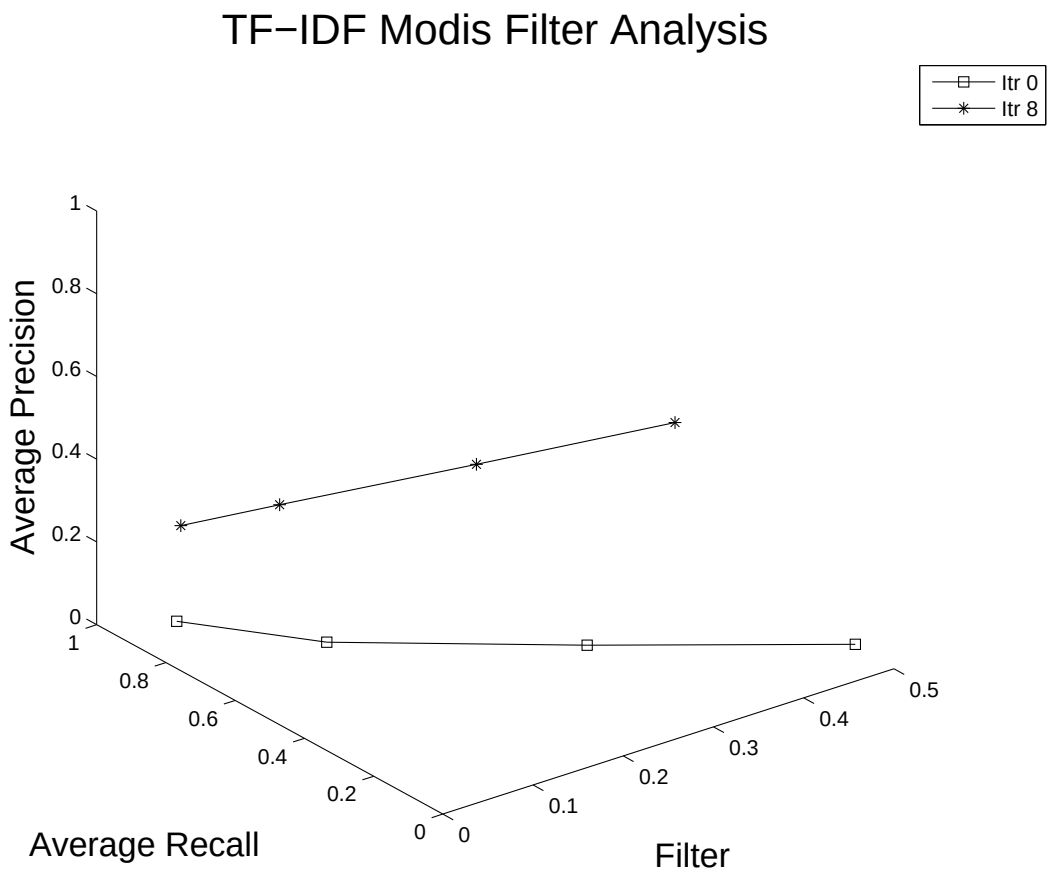


Figure A.11: Filter Analysis - Modis - Average Recall vs. Average Precision (Tf-idf)

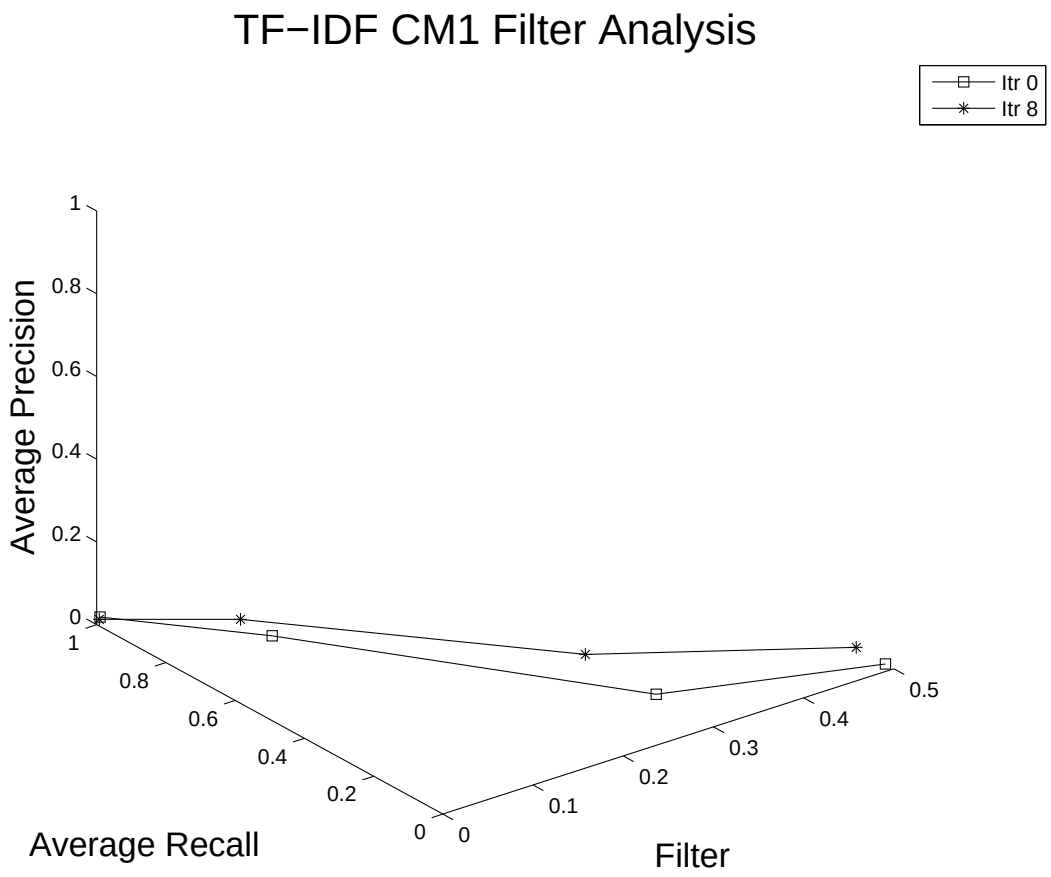


Figure A.12: Filter Analysis - CM1 - Average Recall vs. Average Precision (Tf-idf)

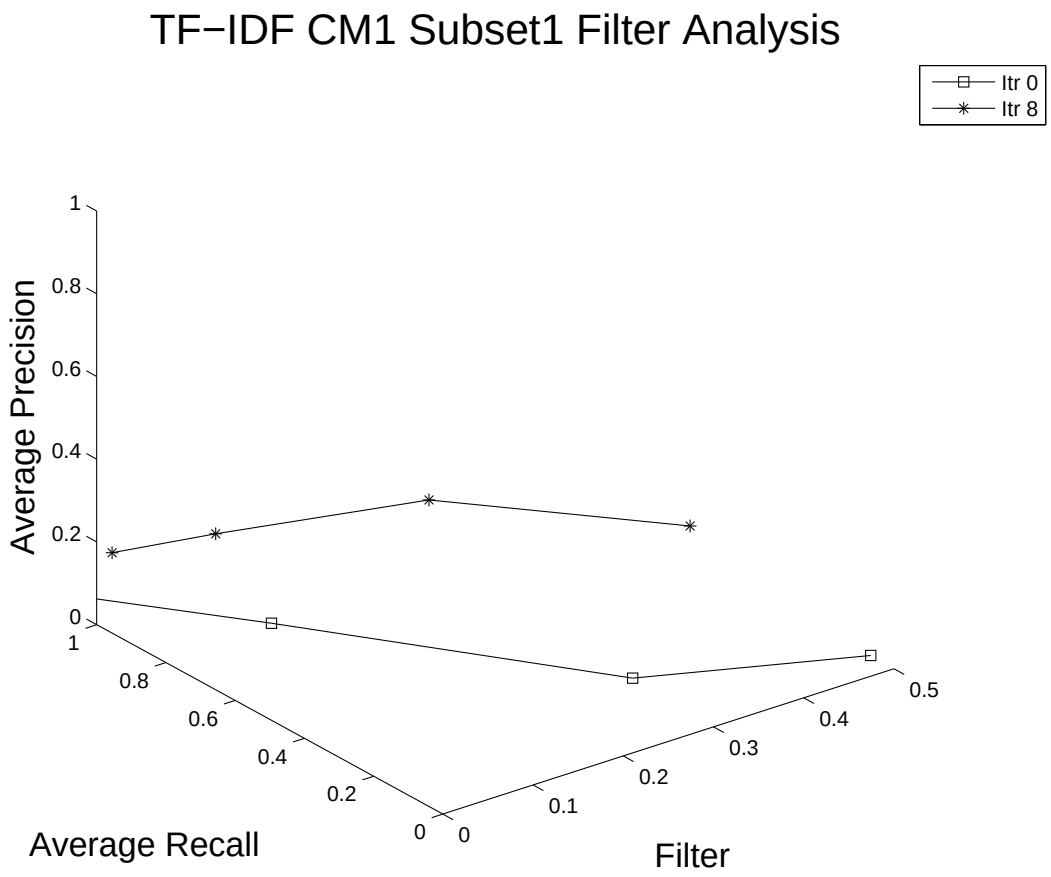


Figure A.13: Filter Analysis - CM1 Subset1 - Average Recall vs. Average Precision (Tf-idf)

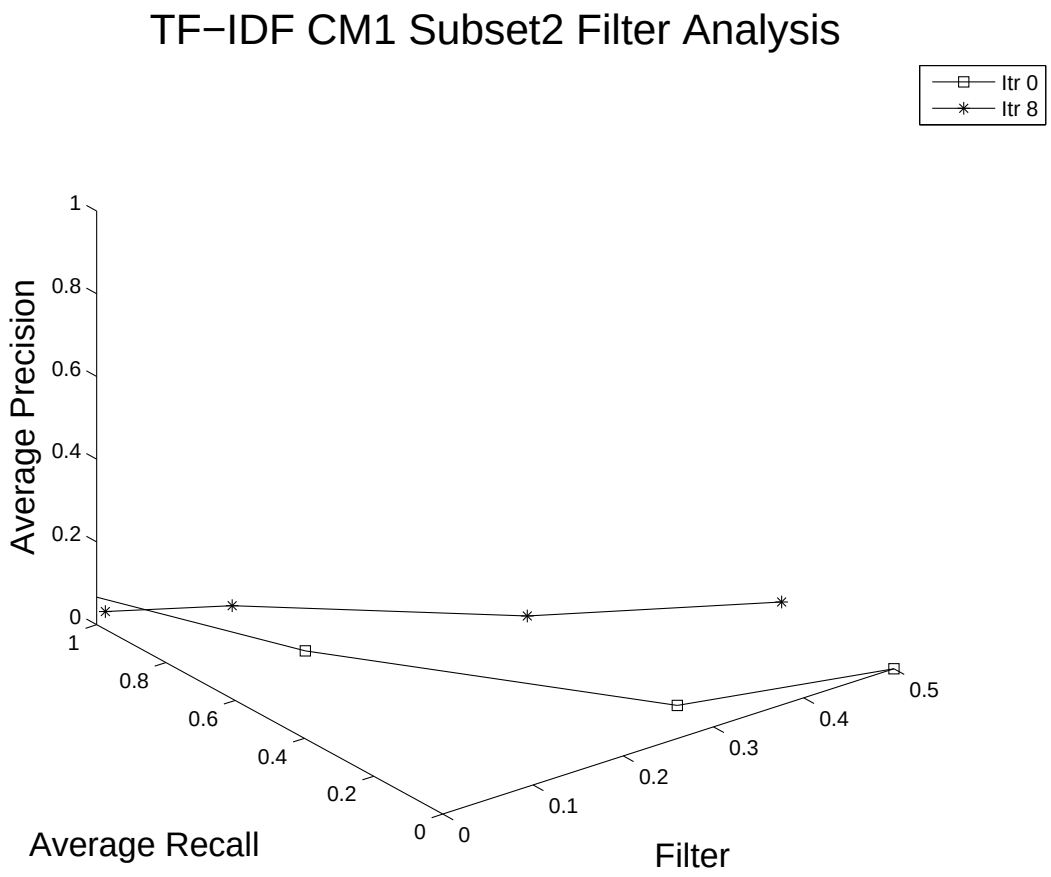


Figure A.14: Filter Analysis - CM1 Subset2 - Average Recall vs. Average Precision (Tf-idf)

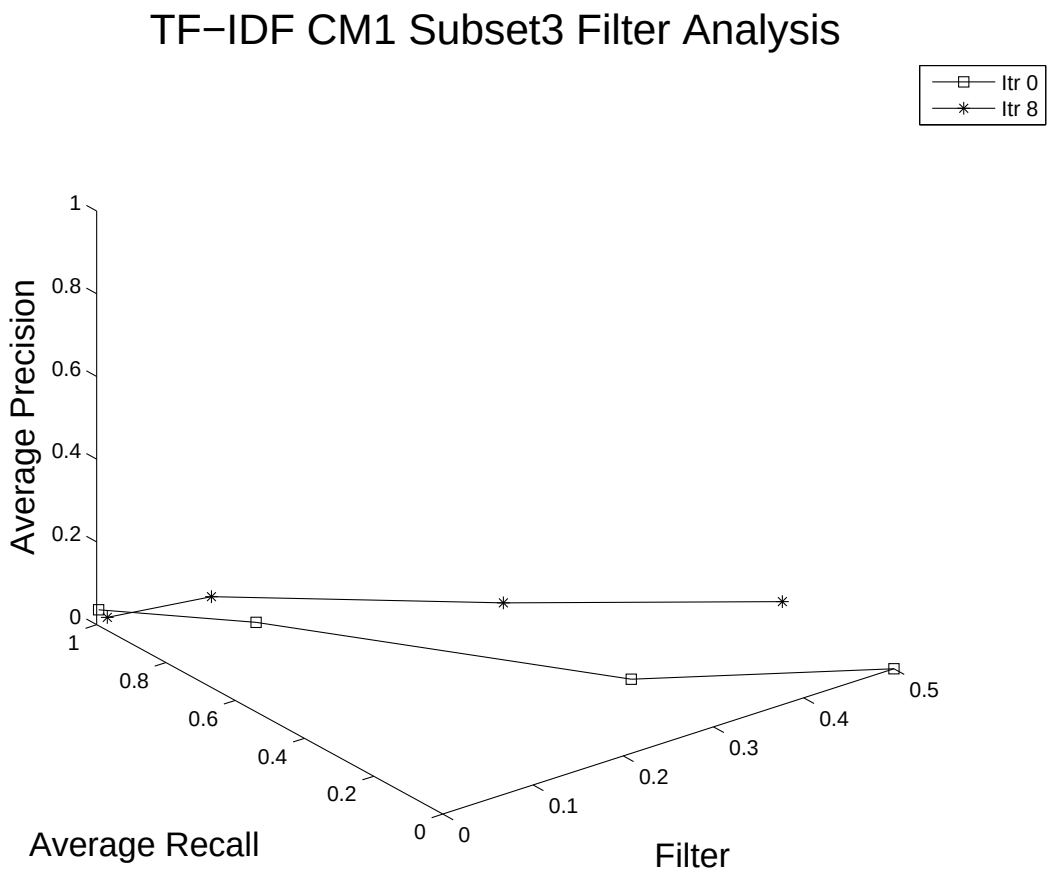


Figure A.15: Filter Analysis - CM1 Subset3 - Average Recall vs. Average Precision (Tf-idf)

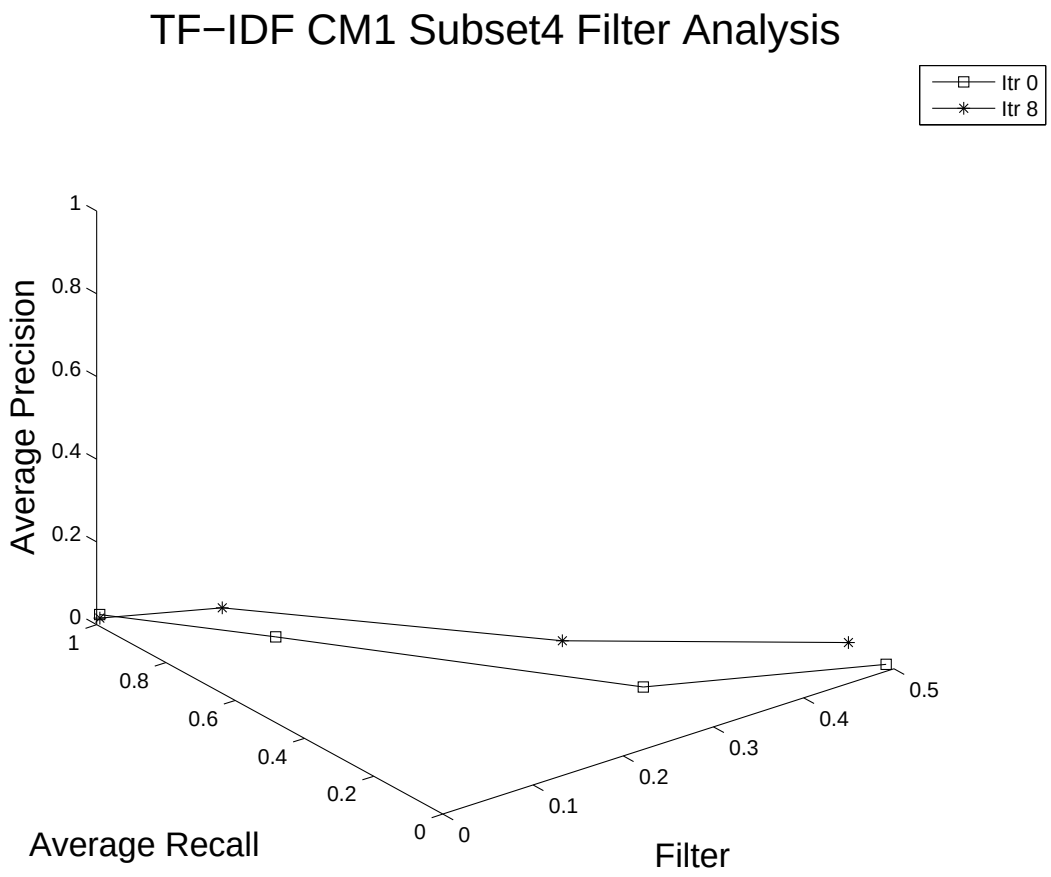


Figure A.16: Filter Analysis - CM1 Subset4 - Average Recall vs. Average Precision (Tf-idf)

References

[1] IEEE Std 1012-1986. IEEE standard for software verification and validation plans. *IEEE*, Piscataway, N.J, 1986.

[2] Level 1A(L1A) and Geolocation Processing Software Requirements Specification. SDST-059A. *GSFC SBRS*, September 11, 1997.

[3] MODIS Science Data Processing Software Requirements Specification Version 2. SDST-089. *GSFC SBRS*, November 10, 1997.

[4] D. Anezin. *Process and Methods for Requirements Tracing (Software Development Life Cycle).* Dissertation, George Mason University, 1994.

[5] G. Antoniol, G Canfora, G Casazza, A. De Lucia, and E Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, Volume 28, No 10, October 2002, 970-983.

[6] G. Antoniol, B. Caprile, A. Potrich, and P. Tonella. Design-code traceability of object oriented systems. *Annals of Software Engineering*, volume 9, 35-38, 1999.

[7] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[8] S. Bohner. *A Graph Traceability Approach for Software Change Impact Analysis.* Dissertation, George Mason University, 1995.

[9] P. Brouse. *A Process for Use of Multimedia Information in Requirements Identification and Traceability.* Dissertation, George Mason University, 1992.

[10] A. Casotto. Run-time requirement tracing. *Proceeding of the IEEE/ACM International Conference on Computer-aided Design*, pp. 350-355, Santa Clara, CA, 1993.

[11] J. Cleland-Huang, C.K. Chang, G. Sethi, K. Javvaji, H. Hu, and J Xia. Automating speculative queries through event-based requirements traceability. *Proceedings of the IEEE Joint INternational Requirements Engineering Conference (RE'02)*, Essex, Germany, Sept. 2002.

[12] J. Cleland-Huang, C.K. Chang, and J. Wise. Automating performance related impact analysis through event based traceability. *Requirements Engineering Journal, Springer-Verlag, Vol. 8, No.3*, Aug.2003, pp. 171-182.

[13] Jane Cleland-Huang, Grant Zemont, and Wiktor Lukasik. A heterogeneous solution for improving the return on investment of requirements traceability. *Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'04)*, Kyoto, Japan, Sept. 2004.

[14] A. Dekhtyar, J. Huffman Hayes, K.S. Sundaram, A. Holbrook, and O. Dekhtyar. Two heads are better than one, or too many cooks in the kitchen? technique integration for requirements assessment. *Accepted to the International Conference on Requirements Engineering (RE'2007)*, Delhi, India, October 2007.

[15] A. Egyed. A scenario-driven approach to traceability. *In Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, Toronto, Canada, May 2001, pp. 123-132.

[16] A. Egyed and P. Gruenbacher. Automating requirements traceability: Beyond the record and replay paradigm. *In Proceedings of the 17th IEEE International Conference on Automated Software Engineering*, pp. 163-171, Edinburgh, UK, September, 2002.

[17] R.F. Flesch. Estimating the comprehension difficulty of magazine articles. *Journal of General Psychology, 28: 63-80*, (1943).

[18] R.F Flesch. A new readability yardstick. *Journal of Applied Psychology, 32: 221-233*, (1948).

[19] W. Frakes and R.Baeza-Yates (Eds.). *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, 1992.

[20] G. W. Furnas, S. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. *In Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 465-480, 1988.

[21] O. C. Z. Gotel and A. C. W. Finkelstein. An analysis of requirements traceability problem. *In Proceedings of the IEEE International Conference on Requirements Engineering (ICRE'94)*, Colorado Springs, Colorado & Taipei, Taiwan, 18-22 April, 1994.

[22] G.Salton. *Automatic Text Processing*. Addison-Wesley, 1998.

[23] J. Huffman Hayes. Risk reduction through requirements tracing. *In The Conference Proceedings of Software Quality Week 1990*, San Francisco, California, May 1990.

[24] J. Huffman Hayes and A. Dekhtyar. Humans in the traceability loop: Can't live with'em, can't live without'em. *in Proceeding of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'05)*, pp. 20-23, Long Beach, CA, 2005.

[25] J. Huffman Hayes and A. Dekhtyar. A framework for comparing requirements tracing experiments. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Vol. 15, No. 5 (October 2005), pp. 185-215, 2005.

[26] J. Huffman Hayes, A. Dekhtyar, and J. Osbourne. Improving requirements tracing via information retrieval. *In Proceedings of the International Conference on Requirements Engineering (RE'2003)*, Monterey, California, September 2003, pp. 151-161.

[27] J. Huffman Hayes, A. Dekhtyar, K.S. Sundaram, and S. Howard. Helping analysts trace requirements: An objective look. *In Proceedings of the International Conference on Requirements Engineering (RE'2004)*, pp. 249-261, Kyoto, Japan, September 2004.

[28] J. Huffman Hayes, A. Dekhtyar, and S. Sundaram. Measuing the effectiveness of retrieval techniques in software engineering. *University of Kentucky Technial Report TR 422-04*, October 2004.

[29] J. Huffman Hayes, A. Dekhtyar, and S. Sundaram. Text mining for software engineering: How analyst feedback impacts final results. *In Proceedings of the 2nd International Workshop on Mining Software Repositories (MSR'2005)*, pp. 58-42, 2005.

[30] J. Huffman Hayes, A. Dekhtyar, and S. Sundaram. Advancing candidate link generation for requirements tracing: the study of methods. *IEEE Transactions on Software Engineering*, Vol. 32, No. 1,pp. 4-19, January 2006.

[31] M. Hoffman, N. Kuhn, M. Weber, and M. Bittner. Requirements for requirements management tools. *In International Conference on Requirements Engineering*, Los Alamitos, California:IEEE Computer Society Press, 2004, pp. 301-308.

[32] Predictor Models in Software Engineering (PROMISE). Software engineering repository. *http://promise.site.uottawa.ca/SERepository*.

[33] K.S. Jones and P.Willet. *Reading in Information Retrieval*. Morgan Kaufmann Publishers, 1997.

[34] A. Marcus and J. Maletic. Recovering Documentation-to-Source Code Traceability Links using Latent Semantic Indexing. *Proceedings of the Twenty-Fifth International Conference on Software Engineering 2003*, Portland, Oregon, 2-10 May 2003, pp. 125-135.

[35] J. Matthias. Requirements tracing. *Communications of the ACM*, 41(12), 1998.

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

[37] T. Mundie and F. Hallsworth. Requirements analysis using SuperTrace PC. *In Proceedings of the American Society of Mechanical Engineers (ASME) for Computers in Engineering Symposium at the Energy & Environmental Expo*, 1995, Houston, Texas.

[38] R Pierce. A requirements tracing tool. *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues*, 1978.

[39] F. Pinheiro and J. Goguen. An object-oriented tool for tracing requirements. *IEEE Software, 52-64*, March 1996.

[40] K. Pohl. PRO-ART: Enabling requirements pre-traceability. *In Proceedings of the IEEE International Conference on Requirements Engineering (RE)*, pp. 76-85, 1996.

[41] B Ramesh. Factors influencing requirements traceability practice. *Communications of the ACM*, Volume 41, No. 12 pp. 37-44, December 1998.

[42] B. Ramesh and V. Dhar. Supporting systems development by capturing deliberations during requirements engineering. *IEEE Transactions on Software Engineering*, 498-510, June 1992.

[43] Rational RequisitePro. *http://www.rational.com/products/reqpro/index.jsp.*

[44] C. J. Van Rijsbergen. A theoretical basis for the use of co-occurence data in information retrieval. *Journal of Documentation, 33(2):106-119*, 1977.

[45] S. Robertson. The probability ranking principle in ir. *Journal of Documentation, 33(4):294-304*, 1977.

[46] S. Robertson and S. Walker. Okapi/Keenbow at TREC-8. *In Proceedings Eighth Text Retrieval Conference (TREC-8)*, 2000.

[47] S. E Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of American Society for Information Science, 27(3):129-146*, 1976.

[48] R. Settimi, J. Cleland-Huang, O. Ben Khadra, J. Mody, W. Lukasok, and C. DePalma. Supporting software evolution through dynamically retrieving traces to UML artifacts. *In Proceedings of 7th International Workshop on Principles of Software Evolution*, 2004, 49-54.

[49] G. Spanoudakis. Plausible and Adaptive Requirement Traceability Structures. *In 14th International Conference on Software Engineering and Knowledge Engineering*, pp. 135-142, Ischia, Italy, July 2002.

[50] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause. Rule-Based Generation of Requirements Traceability Relations. *Journal of Systems and Software*, pp. 105-127, 2004.

[51] S.K. Sundaram, J.H. Hayes, and A. Dekhtyar. Baselines in requirements tracing. *In IEEE International Workshop on Predictive Models for Software Engineering (PROMISE2005)*, 2005.

[52] Telelogic product DOORS. *http://www.telelogic.com/products/doors/index.cfm.*

[53] T. Tsumaki and Y.A. Morisawa. Framework of Requirements Tracing using UML. *In Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, 2000, 5-8 December 2000, pp. 206-213.

[54] R Watkins and M. Neal. Why and how of requirements tracing. *IEEE Software*, Volume 11, Issue 4, pp. 104-106, July 1994.

[55] MDP website. CM-1 Project. *http://mdp.ivv.nasa.gov/mdp_glossary.html#CM1.*

[56] Y.Matsuo and M.Ishizuka. Keyword extraction from a single document using word co-occurence statistical information. *International Journal on Artificial Intelligence Tools, 13(1):157-169*, 2004.

[57] A. Zisman, G. Spanoudakis, E. Perez-Minana, and P. Krause. Tracing Product Families Requirements: An XML-based Approach. *Technical Report, Technical Report Series, Department of Computing, City University*, 2001.

Vita

Date of Birth: July 24, 1981

Place of Birth: Chennai, India

Education:

Bachelors of Engineering,
Computer Science and Engineering,
University of Madras, 2002

Professional Positions:

Lead Software Engineer,
Arts & Sciences Computing Services,
University of Kentucky, Lexington, Kentucky,
Dec 2005 - Present

Software Engineer,
Hensley Elam and Associates, Lexington, Kentucky,
Jun 2005 - Dec 2005

Graduate Research Assistant,
Department of Computer Science,
University of Kentucky, Lexington, Kentucky,
Jun 2004 - May 2005

Graduate Teaching Assistant,
Department of Computer Science,
University of Kentucky, Lexington, Kentucky,
Jan 2004 - May 2004

Graduate Research Assistant,
Department of Computer Science,
University of Kentucky, Lexington, Kentucky,
Feb 2003 - Dec 2003

Web Developer,
Appalachian Rural Systemic Initiative,
University of Kentucky, Lexington, Kentucky,
Nov 2002 - Jan 2003

Honors and Awards:

Best Research Paper Award for "Advancing Candidate Link Generation For Requirements Tracing: the Study of Methods,"
IEEE Transactions on Software Engineering, January 2006.,
Software Assurance Symposium, 2006,
NASA Office of Safety & Mission Assurance (OSMA)

Viji Jeganathan Scholarship for Cross-Cultural Understanding,
University of Kentucky, 2004

Professional Publications:

"Advancing Candidate Link Generation For Requirements Tracing: the Study of Methods," (2006), Jane Hayes, Alex Dekhtyar, Senthil Sundaram, IEEE Transactions on Software Engineering, Vol. 32, No. 1., pp. 4-19, January 2006.

"Improving After-the-Fact Tracing and Mapping: Supporting Software Quality Predictions," (2005), Jane Hayes, Alex Dekhtyar, Senthil Sundaram, IEEE Software, Vol. 22, No. 6 (November/December), pp. 30-37.

"Two Heads are Better than One, or Too Many Cooks in the Kitchen? Technique Integration for Requirements Assessment" (2007), Alexander Dekhtyar, Jane Hayes, Senthil Sundaram, Ashlee Holdbrook, and Olga Dekhtyar, accepted to, 15th International Requirements Engineering Conference (RE 2007), October 2007, Delhi, India.

"Helping Analysts Trace Requirements: An Objective Look," (2004), Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, and Sarah Howard, in Proceedings, 12th International Requirements Engineering Conference (RE 2004), pp. 249-261, September 2004, Kyoto, Japan.

"Baselines in Requirements Tracing," Senthil Karthikeyan Sundaram, Jane Huffman Hayes, and Alex Dekhtyar, in Proceedings, PROMISE' 2005: International Workshop on Predictive Models in Software Engineering , St. Louis, MO, May 2005.

"Text Mining for Software Engineering: How Analyst Feedback Impacts Final Results, Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram, in Proceedings, MSR'2005: Second International Workshop on Mining Software Repositories, pp. 58-62, St. Louis, MO, May 2005.

"Will Johnny/Joanie Make a Good Software Engineer? Are Course Grades Showing the Whole Picture?," (2006), Jane Huffman Hayes, Alex Dekhtyar, Ashlee Holbrook, Olga

Dekhtyar, Senthil Sundaram, in Proceedings, 19th Conference on Software Engineering Education and Training (CSEET), Hawaii, April 2006.

"Are Strong Research Skills in Traceability Necessary and Sufficient to Complete Good Traces?," Holbrook, E. Ashlee, Senthil Karthikeyan Sundaram, Presented at the 21st Century Workshop on Independent Validation and Verification, co-located with the 19th Conference on Software Engineering Education and Training (CSEET), Hawaii, April 2006.

Senthil Karthikeyan Sundaram