# Contents

Dedication v

List of Figures xi

Preface xv

Acknowledgments xix

1. INTRODUCTION 1

2. RETRIEVAL STRATEGIES 9
   - 2.1 Vector Space Model 11
   - 2.2 Probabilistic Retrieval Strategies 21
   - 2.3 Language Models 45
   - 2.4 Inference Networks 57
   - 2.5 Extended Boolean Retrieval 67
   - 2.6 Latent Semantic Indexing 70
   - 2.7 Neural Networks 74
   - 2.8 Genetic Algorithms 80
   - 2.9 Fuzzy Set Retrieval 84
   - 2.10 Summary 90
   - 2.11 Exercises 91

3. RETRIEVAL UTILITIES 93
   - 3.1 Relevance Feedback 94
   - 3.2 Clustering 105
   - 3.3 Passage-based Retrieval 113
   - 3.4 N-grams 115
   - 3.5 Regression Analysis 119

3.6 Thesauri 122
3.7 Semantic Networks 132
3.8 Parsing 139
3.9 Summary 146
3.10 Exercises 146

4. CROSS-LANGUAGE INFORMATION RETRIEVAL 149
4.1 Introduction 149
4.2 Crossing the Language Barrier 151
4.3 Cross-Language Retrieval Strategies 157
4.4 Cross Language Utilities 170
4.5 Summary 178
4.6 Exercises 179

5. EFFICIENCY 181
5.1 Inverted Index 182
5.2 Query Processing 195
5.3 Signature Files 199
5.4 Duplicate Document Detection 203
5.5 Summary 208
5.6 Exercises 209

6. INTEGRATING STRUCTURED DATA AND TEXT 211
6.1 Review of the Relational Model 215
6.2 A Historical Progression 222
6.3 Information Retrieval as a Relational Application 228
6.4 Semi-Structured Search using a Relational Schema 245
6.5 Multi-dimensional Data Model 250
6.6 Mediators 250
6.7 Summary 253
6.8 Exercises 254

7. PARALLEL INFORMATION RETRIEVAL 257
7.1 Parallel Text Scanning 258
7.2 Parallel Indexing 263
7.3 Clustering and Classification 270
7.4 Large Parallel Systems 271

7.5 Summary 272
7.6 Exercises 274

8. DISTRIBUTED INFORMATION RETRIEVAL 275
8.1 A Theoretical Model of Distributed Retrieval 276
8.2 Web Search 281
8.3 Result Fusion 284
8.4 Peer-to-Peer Information Systems 286
8.5 Other Architectures 289
8.6 Summary 290
8.7 Exercises 290

9. SUMMARY AND FUTURE DIRECTIONS 291

References 299

Index 331

# List of Figures

| | | |
|---|---|---|
| 1.1 | Document Retrieval | 2 |
| 1.2 | Document Routing | 3 |
| 1.3 | Result Set: Relevant Retrieved, Relevant, and Retrieved | 4 |
| 1.4 | Precision and Two Points of Recall | 5 |
| 1.5 | Typical and Optimal Precision/Recall Graph | 5 |
| 2.1 | Vector Space Model | 13 |
| 2.2 | Vector Space Model with a Two Term Vocabulary | 14 |
| 2.3 | Inverted Index | 17 |
| 2.4 | Training Data for Probabilistic Retrieval | 25 |
| 2.5 | Language Model | 46 |
| 2.6 | Simple Inference Network | 58 |
| 2.7 | Document-Term-Query Inference Network | 63 |
| 2.8 | Inference Network | 64 |
| 2.9 | Neural Network with Feedback | 76 |
| 3.1 | Relevance Feedback Process | 95 |
| 3.2 | Document Clustering | 106 |
| 3.3 | Overlapping vs Non-Overlapping Passages | 115 |
| 3.4 | Using a Thesaurus to Expand a Query | 122 |
| 4.1 | Translate the Query | 152 |
| 4.2 | Translate the Documents | 153 |
| 5.1 | Inverted Index | 183 |
| 6.1 | IR as an Application of a RDBMS | 213 |
| 6.2 | Intranet Mediator Architecture | 254 |
| 7.1 | Partitioning an Inverted Index | 264 |
| 8.1 | Distributed Document Retrieval | 276 |

8.2 Simple PageRank Calculation 283
9.1 Sample TREC document 294
9.2 Sample TREC query 294

# Foreword

The past five years, as Grossman and Frieder acknowledge in their preface, have been a period of considerable progress for the field of information retrieval (IR). To the general public, this is reflected in the maturing of commercial Web search engines. To the IR practitioner, research has led to an improved understanding of the scope and limitations of the Web search problem, new insights into the retrieval process through the development of the formal underpinnings and models for IR, and a variety of exciting new applications such as cross-language retrieval, peer-to-peer search, and music retrieval, which have expanded the horizons of the research landscape. In addition, there has been an increasing realization on the part of the database and IR communities that solving the information problems of the future will involve the integration of techniques for unstructured and structured data. The revised edition of this book addresses many of these important new developments, and is currently the only textbook that does so.

Two particular examples that stood out for me are the descriptions of language models for IR and cross-language retrieval. Language models have become an important topic at the major IR conferences and many researchers are adapting this framework due to its power and simplicity, as well as the availability of tools for experimentation and application building. Grossman and Frieder provide an excellent overview of the topic in the retrieval strategies chapter, together with examples of different smoothing techniques. Cross-language retrieval, which involves the retrieval of text in different languages than the query source language, has been driven by government interest in Europe and the U.S. A number of approaches have been developed that can exploit available resources such as parallel and comparable corpora, and the effectiveness of these systems now approaches (or even surpasses in some cases) monolingual retrieval. The revised version of this book contains a chapter on cross-language retrieval that clearly describes the major approaches and gives examples of how the algorithms involved work with real data. The combination of up-to-date coverage, straightforward treatment, and the frequent use of examples makes this book an excellent choice for undergraduate or graduate IR courses.

W. Bruce Croft
August 2004

# Preface

When we wrote the first edition of this book in 1998, the Web was relatively new, and information retrieval was an old field but it lacked popular appeal. Today the word *Google* has joined the popular lexicon, and *Google* indexes more than four billion Web pages. In 1998, only a few schools taught graduate courses in information retrieval; today, the subject is commonly offered at the undergraduate level. Our experience with teaching information retrieval at the undergraduate level, as well as a detailed analysis of the topics covered and the effectiveness of the class, are given in [Goharian et al., 2004].

The term *Information Retrieval* refers to a search that may cover any form of information: structured data, text, video, image, sound, musical scores, DNA sequences, etc. The reality is that for many years, database systems existed to search structured data, and information retrieval meant the search of documents. The authors come originally from the world of structured search, but for much of the last ten years, we have worked in the area of document retrieval. To us, the world should be data type agnostic. There is no need for a special delineation between structured and unstructured data. In 1998, we included a chapter on data integration, and reviews suggested the only reason it was there was because it covered some of our recent research. Today, such an allegation makes no sense, since information *mediators* have been developed which operate with both structured and unstructured data. Furthermore, the eXtensible Markup Language (XML) has become prolific in both the database and information retrieval domains.

We focus on the ad hoc information retrieval problem. Simply put, ad hoc information retrieval allows users to search for documents that are relevant to user-provided queries. It may appear that systems such as *Google* have solved this problem, but effectiveness measures for *Google* have not been published. Typical systems still have an effectiveness (accuracy) of, at best, forty percent [TREC, 2003]. This leaves ample room for improvement, with the prerequisite of a firm understanding of existing approaches.

Information retrieval textbooks on the market are relatively unfocused, and we were uncomfortable using them in our classes. They tend to leave out details of a variety of key retrieval models. Few books detail inference networks, yet an inference network is a core model used by a variety of systems. Additionally, many books lack much detail on efficiency, namely, the execution speed of a query. Efficiency is potentially of limited interest to those who focus only on effectiveness, but for the practitioner, efficiency concerns can override all others.

Additionally, for each strategy, we provide a detailed running example. When presenting strategies, it is easy to gloss over the details, but examples keep us honest. We find that students benefit from a single example that runs through the whole book. Furthermore, every section of this book that describes a core retrieval strategy was reviewed by either the inventor of the strategy (and we thank them profusely; more thanks are in the acknowledgments!) or someone intimately familiar with it. Hence, to our knowledge, this book contains some of the gory details of some strategies that cannot be found anywhere else in print.

Our goal is to provide a book that is sharply focused on ad hoc information retrieval. To do this, we developed a taxonomy of the field based on a model that a *strategy* compares a document to a query and a utility can be plugged into any strategy to improve the performance of the given strategy. We cover all of the basic strategies, not just a couple of them, and a variety of utilities. We provide sufficient detail so that a student or practitioner who reads our book can implement any particular strategy or utility. The book, *Managing Gigabytes* [Witten et al., 1999], does an excellent job of describing a variety of detailed inverted index compression strategies. We include the most recently developed and the most efficient of these, but we certainly recommend Managing Gigabytes as an excellent side reference.

So what is new in this second edition? Much of the core retrieval strategies remain unchanged. Since 1998, numerous papers were written about the use of language models for information retrieval. We have added a new section on language models. Furthermore, cross-lingual information retrieval, that is, the posting of a query in one language and finding documents in another language, was just in its infancy at the time of the first version. We have added an entire chapter on the topic that incorporates information from over 100 recent references.

Naturally, we have included some discussion on current topics such as XML, peer-to-peer information retrieval, duplicate document detection, parallel document clustering, fusion of disparate retrieval strategies, and information mediators.

Finally, we fixed a number of bugs found by our alert undergraduate and graduate students. We thank them all for their efforts.

This book is intended primarily as a textbook for an undergraduate or graduate level course in Information Retrieval. It has been used in a graduate course, and we incorporated student feedback when we developed a set of overhead transparencies that can be used when teaching with our text. The presentation is available at *www.ir.iit.edu*.

Additionally, practitioners who build information retrieval systems or applications that use information retrieval systems will find this book useful when selecting retrieval strategies and utilities to deploy for production use. We have heard from several practitioners that the first edition was helpful, and we incorporated their comments and suggested additions into this edition.

We emphasize that the focus of the book is on algorithms, not on commercial products, but, to our knowledge, the basic strategies used by the majority of commercial products are described in the book. We believe practitioners may find that a commercial product is using a given strategy and can then use this book as a reference to learn what is known about the techniques used by the product.

Finally, we note that the information retrieval field changes daily. For the most up to date coverage of the field, the best sources include journals like the *ACM Transactions on Information Systems*, the *Journal of the American Society for Information Science and Technology*, *Information Processing and Management*, and *Information Retrieval*. Other relevant papers are found in the various information retrieval conferences such as ACM SIGIR *www.sigir.org*, NIST TREC *trec.nist.gov*, and the ACM CIKM *www.cikm.org*.

# Acknowledgments

The first edition of this book was published in 1998. Since then, numerous advancements in Information Retrieval have occurred.

For help with most of the material for the first edition, we are still deeply indebted to Paul Kantor and Don Kraft for their insightful comments during some of the early stages of the development of the book. We also greatly thank Steve Robertson, K.L. Kwok, and Jamie Callan for critically reviewing the sections describing their efforts, and Warren Greiff whose extreme patience taught and retaught us the details of inference networks.

With the addition of language models and cross-language information retrieval we are extremely thankful for the insightful comments and suggestions from ChengXiang Zhai, Doug Oard, and James Mayfield.

General critical feedback came from multiple individuals. Readability for classroom use was enhanced by feedback given from the students in our undergraduate Information Retrieval course and our graduate Advanced Information Retrieval. John Juilfs spent countless hours checking all of the new examples for this edition and checking our math (we take full responsibility for any remaining errors, but he saved us from lots of them). Abdur Chowdhury carefully reviewed and provided input for our new section on duplicate document detection. Wei Gen Yee provided significant input to our modifications in the distributed information retrieval chapter. Steve Beitzel and Eric Jensen provided significant input to updates in our efficiency and parallel processing sections. Rebecca Cathey provided examples for our new language modeling and cross-language sections. Michael Saelee produced and refined our figures. Finally, Wei Guan carefully and repeatedly checked all of our examples and notation.

We also thank Wendy Grossman, Ben Goldfarb, David Roberts, and Jordan Wilberding for detailed comments. We also thank all others who read the book and provided excellent feedback, and our friends and colleagues who provided moral support.

Finally, we would not have ever finished the book without constant moral support from our families and close friends. Special thanks go to Mary Catherine McCabe and Nazli Goharian who made countless sacrifices of their time and gave us constant encouragement and support.

From both of us to all of you, THANKS.

Chapter 1

# INTRODUCTION

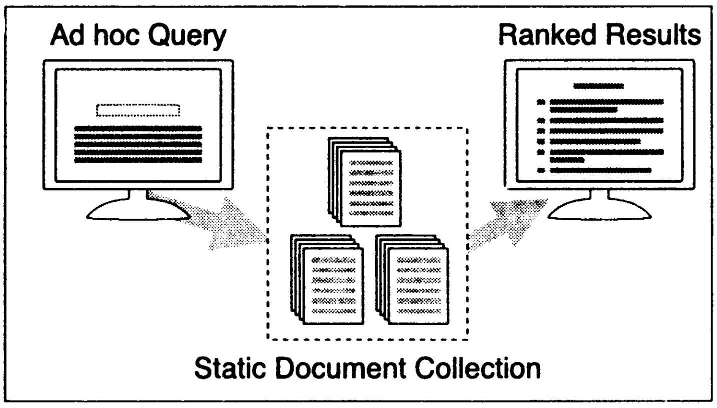
Since the near beginnings of civilization, human beings have focused on written communication. From cave drawings to scroll writings, from printing presses to electronic libraries, communicating was of primary concern to man's existence. Today, with the emergence of digital libraries and electronic information exchange there is clear need for improved techniques to organize large quantities of information. Applied and theoretical research and development in the areas of information authorship, processing, storage, and retrieval is of interest to all sectors of the community. In this book, we survey recent research efforts that focus on the electronic searching and retrieving of documents.

Our focus is strictly on the retrieval of information in response to user queries. That is, we discuss algorithms and approaches for ad hoc information retrieval, or simply, information retrieval. Figure 1.1 illustrates the basic process of ad hoc information retrieval. A static, or relatively static, document collection is indexed prior to any user query. A query is issued and a set of documents that are deemed relevant to the query are ranked based on their computed similarity to the query and presented to the user. Numerous techniques exist to identify how these documents are ranked, and that is a key focus of this book (effectiveness). Other techniques also exist to rank documents quickly, and these are also discussed (efficiency).

Information Retrieval (IR) is devoted to finding *relevant* documents, not finding simple matches to patterns. Yet, often when information retrieval systems are evaluated, they are found to miss numerous relevant documents [Blair and Maron, 1985]. Moreover, users have become complacent in their expectation of accuracy of information retrieval systems [Gordon, 1997].

A related problem is that of document routing or filtering. Here, the queries are static and the document collection constantly changes. An environment where corporate e-mail is routed based on predefined queries to different parts

*Figure 1.1.* Document Retrieval

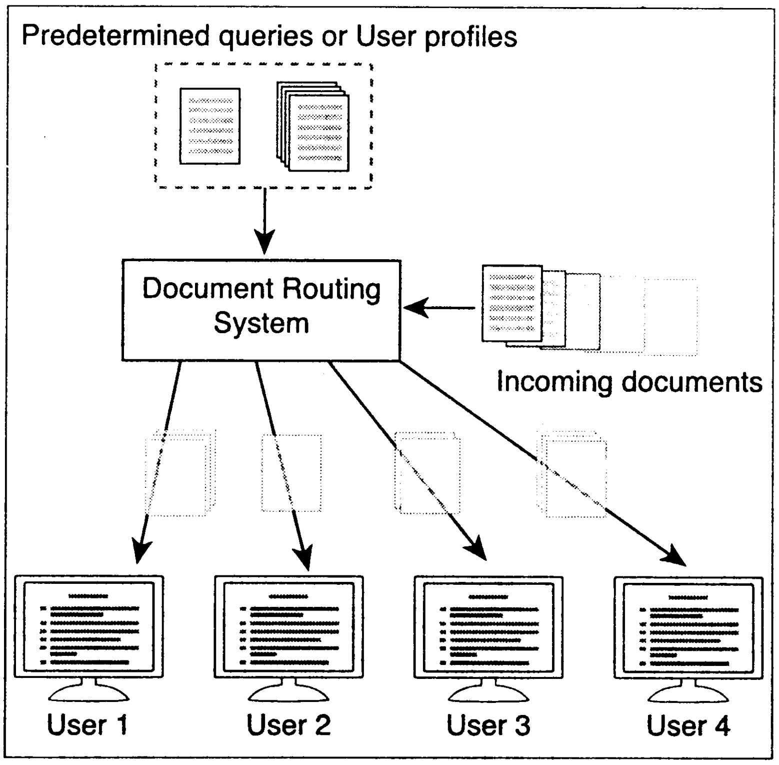


of the organization (i.e., e-mail about sales is routed to the sales department, marketing e-mail goes to marketing, etc.) is an example of an application of document routing. Figure 1.2 illustrates document routing. Document routing algorithms and approaches also widely appear in the literature, but are not addressed in this book.

In Figure 1.3, we illustrate the critical document categories that correspond to any issued query. Namely, in the collection there are documents which are retrieved, and there are those documents that are relevant. In a perfect system, these two sets would be equivalent; we would only retrieve relevant documents. In reality, systems retrieve many non-relevant documents. To measure effectiveness, two ratios are used: *precision* and *recall*. Precision is the ratio of the number of relevant documents retrieved to the total number retrieved. Precision provides an indication of the quality of the answer set. However, this does not consider the total number of relevant documents. A system might have good precision by retrieving ten documents and finding that nine are relevant (a 0.9 precision), but the total number of relevant documents also matters. If there were only nine relevant documents, the system would be a huge success — however if millions of documents were relevant and desired, this would not be a good result set.

Recall considers the total number of relevant documents; it is the ratio of the number of relevant documents retrieved to the total number of documents in the collection that are believed to be relevant. Computing the total number of relevant documents is non-trivial. The only sure means of doing this is to read the entire document collection. Since this is clearly not feasible, an approximation of the number is obtained (see Chapter 9). A good survey of

*Figure 1.2.* Document Routing

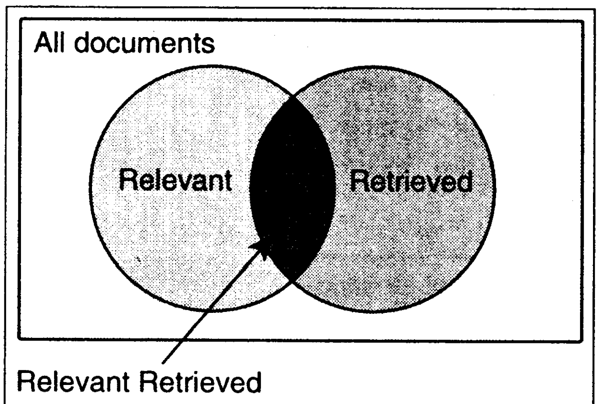


effectiveness measures, as well as a brief overview of information retrieval, is found in [Kantor, 1994].

Precision can be computed at various points of recall. Consider an example query $q$. For this query, we have estimated that there are two relevant documents. Now assume that when the user submits query $q$ that ten documents are retrieved, including the two relevant documents. In our example, documents two and five are relevant. The sloped line in Figure 1.4 shows that after retrieving two documents, we have found one relevant document, and hence have achieved fifty percent *recall.* At this point, *precision* is fifty percent as we have retrieved two documents and one of them is relevant.

To reach one hundred percent recall, we must continue to retrieve documents until both relevant documents are retrieved. For our example, it is necessary to retrieve five documents to find both relevant documents. At this point, precision is forty percent because two out of five retrieved documents are relevant. Hence, for any desired level of recall, it is possible to compute precision.

*Figure 1.3.* Result Set: Relevant Retrieved, Relevant, and Retrieved



$$\text{Precision} = \frac{\text{Relevant Retrieved}}{\text{Retrieved}}$$

$$\text{Recall} = \frac{\text{Relevant Retrieved}}{\text{Relevant}}$$

Graphing precision at various points of recall is referred to as a *precision/recall curve.*

A typical precision/recall curve is shown in Figure 1.5. Typically, as higher recall is desired, more documents must be retrieved to obtain the desired level of recall. In a perfect system, only relevant documents are retrieved. This means that at any level of recall, precision would be 1.0. The optimal precision/recall line is shown in Figure 1.5.

Average precision refers to an average of precision at various points of recall. Many systems today, when run on a standard document collection, report an average precision of between 0.2 and 0.3. Certainly, there is some element of fuzziness here because relevance is not a clearly defined concept, but it is clear that there is significant room for improvement in the area of effectiveness.

Finding relevant documents is not enough. The goal is to identify relevant documents within an acceptable response time. This book describes the current strategies to find relevant documents *quickly.* The quest to find efficient and effective information retrieval algorithms continues.

We explain each algorithm in detail, and for each topic, include examples for the most crucial algorithms. We then switch gears into survey mode and provide references to related and follow-on work. We explain the key aspects of the algorithms and then provide references for those interested in further

*Figure 1.4.* Precision and Two Points of Recall

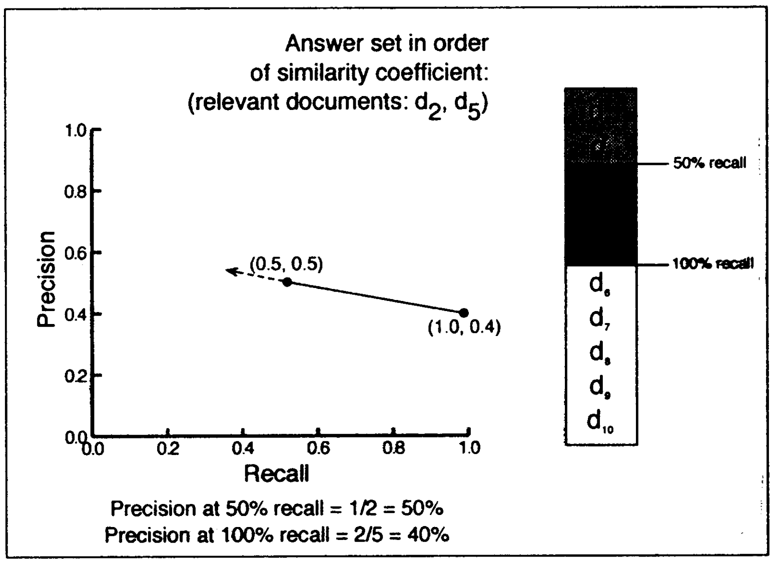
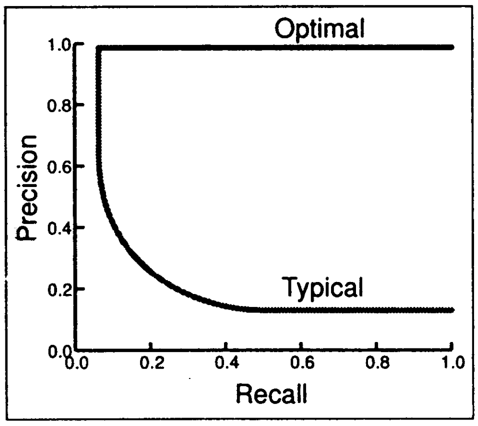


*Figure 1.5.* Typical and Optimal Precision/Recall Graph



details. A collection of key information retrieval research papers is found in [Sparck Jones and Willett, 1997].

Recent algorithms designed to search large bodies of information are discussed throughout this book. Many research publications describe these algorithms in detail, but they are spread across numerous journals and written in a variety of different styles. Also, they have differing expectations of their reader's background. We provide a relatively brief, but sufficiently detailed overview of the field.

A sophisticated mathematical background is not required. Whenever detailed mathematical constructs are used, we provide a quick refresher of the key points needed to understand the algorithms and detailed examples.

We believe this book is valuable to a variety of readers. Readers familiar with the core of computer science and interested in learning more about information retrieval algorithms should benefit from this text. We provide explanations of the fundamental problems that exist and how people have addressed them in the past.

This book also has value for anyone who currently uses and supports a Relational Database Management System (RDBMS). Chapter 6 gives detailed algorithms that treat text retrieval as an application of a RDBMS. This makes it possible to integrate both structured data and text. We also include a section describing relational database approaches to process semi-structured documents such as those tagged with XML.

To guide the reader through the key issues in ad hoc information retrieval, we partitioned this book into separate but inter-linked processing avenues. In the first section, covered in Chapters 2 and 3, we overview retrieval processing strategies and utilities. All of these strategies and utilities focus on one and only one critical issue, namely, the improvement of retrieval accuracy. In Chapter 2, we describe nine models that were either developed for or adapted to information retrieval specifically for the purpose of enhancing the evaluation or ranking of documents retrieved in response to user queries. Chapter 3 describes utilities that could be applied to enhance any strategy described in Chapter 2.

In Chapter 3, we focus on techniques that are applicable to either all or most of the models. Several of those utilities described are language dependent, e.g., parsing and thesauri, others focus specifically on being language independent, namely, N-gram processing. We note in Chapter 3, that some of the described utilities were proposed as individual processing strategies. In reality, however, it is the combination of these techniques that yields the best improvements. An approach to precisely determine the optimal mix of techniques, the order to execute them, and the underlying models to operate them so as to yield the optimal processing strategy is still unknown.

After describing models and utilities that address accuracy demands, we turn our attention towards processing efficiency. In Chapter 4, we describe various document access schemes. That is, we describe both the constructs and usage of inverted indices as well as other representation schemes such as signature files. Each of these access schemes has advantages and disadvantages. The tradeoffs lie in terms of storage overhead and maintainability versus search and retrieval processing times. After describing the various access methods, we overview several compression schemes.

Chapters 2 and 3 cover the basics of traditional information retrieval models, utilities, and processing strategies. Chapter 4 provides a brief overview of cross-language information retrieval. Chapter 5 describes efficiency issues in Information Retrieval. In Chapters 6, 7, and 8, we focus on special topics in information retrieval. The three topics addressed, namely data integration, parallel, and distributed information retrieval systems, were selected based on where the commercial sector is focusing.

Traditionally, there was a clear separation between structured data, typically stored and accessed via relational database management systems, and semi-structured data such as text, typically stored and accessed via information retrieval systems. Each processing system supported its own data storage files and access methods. Today, the distinction between structured and semi-structured data is quickly vanishing. In fact, we no longer are concerned with just structured and semi-structured data, but also text and often include unstructured data, such as images, in the same storage repository.

To address the integration of structured and unstructured data, commercial vendors such as Oracle, IBM, and Microsoft have integrated information retrieval functionality with their traditional relational database engines. Furthermore, text retrieval vendors such as Convera and Verity have added relational processing components. In all of these cases, however, additional functionality came at the expense of requiring additional, separate, processing units. In Chapter 6, we discuss the issues related to adding processing units and suggest an alternative method that involves implementing information retrieval processing capability as an application of relational databases. Using such an approach, the traditional benefits of relational database processing (i.e., portability, concurrency, recovery, etc.) are made available without requiring additional software development. Since all traditional relational database vendors provide parallel implementations of their database software, implementing an information retrieval system as a relational database application further provides for a parallel instantiation of an information retrieval system.

Having recognized the need for a parallel information retrieval capability, we also describe recent developments in this area. In Chapter 7, we initially describe the earlier parallel processing efforts in information retrieval. These approaches predominantly focus on the use of Single Instruction Multiple Data (SIMD) multiprocessors to efficiently scan the text. However, as the understanding of parallel processing techniques in information retrieval grew, inverted index-based approaches were developed to reduce the unnecessarily high I/O demands commonly associated with text scanning schemes. We discuss several of these approaches and conclude Chapter 7 with recent work in parallel information retrieval focusing on the parallelization of document clustering algorithms.

In the information processing world of today, no treatment of the field is complete without addressing the most frequently used retrieval paradigm, the World Wide Web. Thus, in Chapter 8, we describe the encompassing topic of the Web, namely, distributed information retrieval systems. We overview some of the early theoretical foundations and culminate with a discussion of peer-to-peer information retrieval.

The problem of searching document collections to find relevant documents has been addressed for more than forty years. However, until the advent of the Text REtrieval Conference (TREC) in 1990 (which is hosted by the National Institute of Standards and Technology), there was no standard test bed to judge information retrieval algorithms. Without the existence of a standard test data collection and a standard set of queries, there was no effective mechanism by which to objectively compare the algorithms. Many of these algorithms were run against only a few megabytes of text. It was hoped that the performance of these would scale to larger document collections. A seminal paper showed that some approaches that perform well on small document collections did not perform as well on large collections [Blair and Maron, 1985].

We include a brief description of TREC in Chapter 9 — our final chapter. Given all of the models, utilities, and performance enhancements proposed over the years, clearly measures and procedures to evaluate their effectiveness in terms of accuracy and processing times are needed. Indeed, that was part of the motivation behind the creation of the benchmark data and query sets and evaluation forum called TREC. Today, TREC serves as the de facto forum for comparison across systems and approaches. Unfortunately, only accuracy evaluations are currently supported. Hopefully, in the future, processing efficiency will also be evaluated.

We conclude this book with a discussion of the current limitations of information retrieval systems. We review our successes and project future needs. It is our hope that after reading this text, you the reader, will be interested in furthering the field of information retrieval. In our future editions, we hope to incorporate your contributions.

Chapter 2

# RETRIEVAL STRATEGIES

Retrieval strategies assign a measure of similarity between a query and a document. These strategies are based on the common notion that the more often terms are found in both the document and the query, the more "relevant" the document is deemed to be to the query. Some of these strategies employ counter measures to alleviate problems that occur due to the ambiguities inherent in language—the reality that the same concept can often be described with many different terms (e.g., *new york* and *the big apple* can refer to the same concept). Additionally, the same term can have numerous semantic definitions (terms like *bark* and *duck* have very different meanings in their noun and verb forms).

A retrieval strategy is an algorithm that takes a query Q and a set of documents $D_1, D_2, \ldots, D_n$ and identifies the Similarity Coefficient SC(Q,$D_i$) for each of the documents $1 \leq i \leq n$. (Note: SC is short for Similarity Coefficient, sometimes it is written RSV for Retrieval Status Value).

The retrieval strategies identified are:

- **Vector Space Model**—Both the query and each document are represented as vectors in the term space. A measure of the similarity between the two vectors is computed.

- **Probabilistic Retrieval**—A probability based on the likelihood that a term will appear in a relevant document is computed for each term in the collection. For terms that match between a query and a document, the similarity measure is computed as the combination of the probabilities of each of the matching terms.

- **Language Models**—A language model is built for each document, and the likelihood that the document will *generate* the query is computed.

- **Inference Networks**—A Bayesian network is used to infer the relevance of a document to a query. This is based on the "evidence" in a document that allows an inference to be made about the relevance of the document. The strength of this inference is used as the similarity coefficient.

- **Boolean Indexing**—A score is assigned such that an initial Boolean query results in a ranking. This is done by associating a weight with each query term so that this weight is used to compute the similarity coefficient.

- **Latent Semantic Indexing**—The occurrence of terms in documents is represented with a term-document matrix. The matrix is reduced via Singular Value Decomposition (SVD) to filter out the noise found in a document so that two documents which have the same semantics are located close to one another in a multi-dimensional space.

- **Neural Networks**—A sequence of "neurons," or nodes in a network, that fire when activated by a query triggering links to documents. The strength of each link in the network is transmitted to the document and collected to form a similarity coefficient between the query and the document. Networks are "trained" by adjusting the weights on links in response to predetermined relevant and irrelevant documents.

- **Genetic Algorithms**—An optimal query to find relevant documents can be generated by evolution. An initial query is used with either random or estimated term weights. New queries are generated by modifying these weights. A new query survives by being close to known relevant documents and queries with less "fitness" are removed from subsequent generations.

- **Fuzzy Set Retrieval**—A document is mapped to a fuzzy set (a set that contains not only the elements but a number associated with each element that indicates the strength of membership). Boolean queries are mapped into fuzzy set intersection, union, and complement operations that result in a strength of membership associated with each document that is relevant to the query. This strength is used as a similarity coefficient.

For a given retrieval strategy, many different utilities are employed to improve the results of the retrieval strategy. These are described in Chapter 3. Note that some strategies and utilities are based on very different mathematical constructs. For example, a probabilistic retrieval strategy should theoretically not be used in conjunction with a thesaurus based on the vector space model. However, it might be the case that such a combination could improve effectiveness. We merely note that care should be taken when mixing and matching strategies and utilities that are based on very different mathematical models.

Attempting to refine the query, most of these utilities add or remove terms from the initial query. Others simply refine the focus of the query (using subdocuments or passages instead of whole documents). The key is that each of these utilities (although rarely presented as such) are plug-and-play utilities that should work with an arbitrary retrieval strategy.

Before delving into the details of each strategy, we wish to somewhat caution the reader. In our attempt to present the algorithms in their original form, we intentionally left inconsistencies present. For example, some inventors used $\ln(x)$ while other use $\log(x)$ to achieve a slow growing function. Clearly, we are aware that these functions are strictly a constant multiple of each other, but we felt that presenting the original description was still advantageous although it does introduce some minor confusion. Towards clarity, we tried to use common notation across strategies and provided a running example that uses the same query and documents regardless of each strategy.

## 2.1 Vector Space Model

The vector space model computes a measure of similarity by defining a vector that represents each document, and a vector that represents the query [Salton et al., 1975]. The model is based on the idea that, in some rough sense, the meaning of a document is conveyed by the words used. If one can represent the words in the document by a vector, it is possible to compare documents with queries to determine how similar their content is. If a query is considered to be like a document, a similarity coefficient (SC) that measures the similarity between a document and a query can be computed. Documents whose content, as measured by the terms in the document, correspond most closely to the content of the query are judged to be the most relevant. Figure 2.1 illustrates the basic notion of the vector space model in which vectors that represent a query and three documents are illustrated.

This model involves constructing a vector that represents the terms in the document and another vector that represents the terms in the query. Next, a method must be chosen to measure the closeness of any document vector to the query vector. One could look at the magnitude of the difference vector between two vectors, but this would tend to make any large document appear to be not relevant to most queries, which typically are short. The traditional method of determining closeness of two vectors is to use the size of the angle between them. This angle is computed by using the inner product (or dot product); however, it is not necessary to use the actual angle. Any monotonic function of the angle suffices. Often the expression "similarity coefficient" is used instead of an angle. Computing this number is done in a variety of ways, but the inner product generally plays a prominent role. Underlying this whole discussion is the idea that a document and a query are similar to the extent that their associated vectors point in the same general direction.

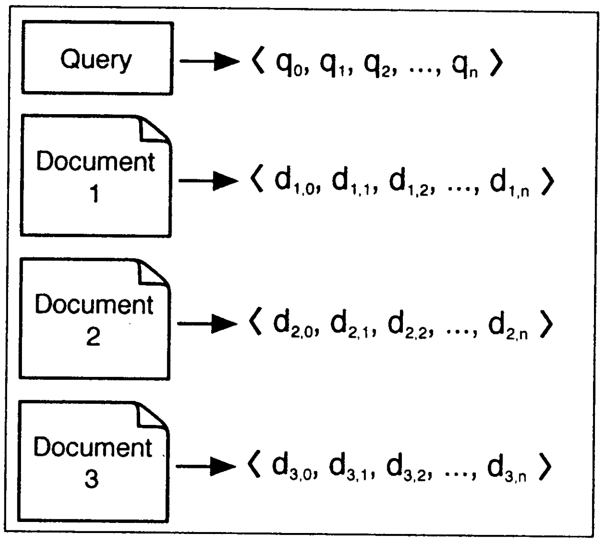
There is one component in these vectors for every distinct term or concept that occurs in the document collection. Consider a document collection with only two distinct terms, $\alpha$ and $\beta$. All vectors contain only two components, the first component represents occurrences of $\alpha$, and the second represents occurrences of $\beta$. The simplest means of constructing a vector is to place a one in the corresponding vector component if the term appears, and a zero if the term does not appear. Consider a document, $D_1$, that contains two occurrences of term $\alpha$ and zero occurrences of term $\beta$. The vector $< 1, 0 >$ represents this document using a binary representation. This binary representation can be used to produce a similarity coefficient, but it does not take into account the frequency of a term within a document. By extending the representation to include a count of the number of occurrences of the terms in each component, the frequency of the terms can be considered. In this example, the vector would now appear as $< 2, 0 >$.

A simple example is given in Figure 2.2. A component of each vector is required for each distinct term in the collection. Using the toy example of a language with a two word vocabulary (only *A* and *I* are valid terms), all queries and documents can be represented in two dimensional space. A query and three documents are given along with their corresponding vectors and a graph of these vectors. The similarity coefficient between the query and the documents can be computed as the distance from the query to the two vectors. In this example, it can be seen that document one is represented by the same vector as the query so it will have the highest rank in the result set.

Instead of simply specifying a list of terms in the query, a user is often given the opportunity to indicate that one term is more important than another. This was done initially with manually assigned term weights selected by users. Another approach uses automatically assigned weights — typically based on the frequency of a term as it occurs across the entire document collection. The idea was that a term that occurs infrequently should be given a higher weight than a term that occurs frequently. Similarity coefficients that employed automatically assigned weights were compared to manually assigned weights [Salton, 1969, Salton, 1970b]. It was shown that automatically assigned weights perform at least as well as manually assigned weights [Salton, 1969, Salton, 1970b]. Unfortunately, these results did not include the relative weight of the term across the entire collection.

The value of a collection weight was studied in the 1970's. The conclusion was that relevance rankings improved if collection-wide weights were included. Although relatively small document collections were used to conduct the experiments, the authors still concluded that, "in so far as anything can be called a solid result in information retrieval research, this is one" [Robertson and Sparck Jones, 1976].

*Figure 2.1.* Vector Space Model


Query
$\langle q_0, q_1, q_2, \ldots, q_n \rangle$
Document 1
$\langle d_{1,0}, d_{1,1}, d_{1,2}, \ldots, d_{1,n} \rangle$
Document 2
$\langle d_{2,0}, d_{2,1}, d_{2,2}, \ldots, d_{2,n} \rangle$
Document 3
$\langle d_{3,0}, d_{3,1}, d_{3,2}, \ldots, d_{3,n} \rangle$

This more formal definition, and slightly larger example, illustrates the use of weights based on the collection frequency. Weight is computed using the *Inverse Document Frequency (IDF)* corresponding to a given term.

To construct a vector that corresponds to each document, consider the following definitions:

$t$ = number of distinct terms in the document collection

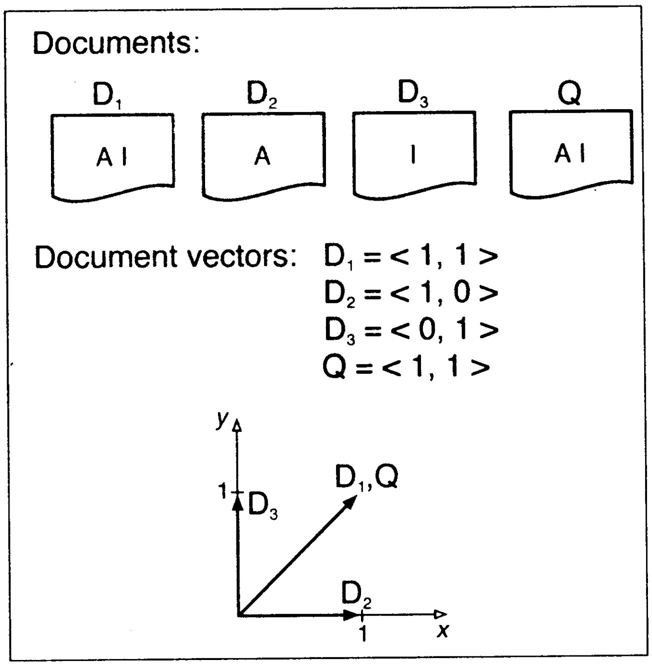
$tf_{ij}$= number of occurrences of term $t_j$ in document $D_i$.
This is referred to as the *term frequency*.

$df_j$= number of documents which contain $t_j$.
This is the *document frequency*.

$idf_j$= $\log(\frac{d}{df_j})$ where $d$ is the total number of documents.
This is the *inverse document frequency*.

The vector for each document has $n$ components and contains an entry for each distinct term in the entire document collection. The components in the vector are filled with weights computed for each term in the document collection. The terms in each document are automatically assigned weights based

*Figure 2.2.* Vector Space Model with a Two Term Vocabulary



on how frequently they occur in the entire document collection and how often a term appears in a particular document. The weight of a term in a document increases the more often the term appears in one document and decreases the more often it appears in all other documents.

A weight computed for a term in a document vector is non-zero only if the term appears in the document. For a large document collection consisting of numerous small documents, the document vectors are likely to contain mostly zeros. For example, a document collection with 10,000 distinct terms results in a 10,000-dimensional vector for each document. A given document that has only 100 distinct terms will have a document vector that contains 9,900 zero-valued components.

The weighting factor for a term in a document is defined as a combination of term frequency, and inverse document frequency. That is, to compute the value of the $j$th entry in the vector corresponding to document $i$, the following equation is used:

$$d_{ij} = tf_{ij} \times idf_j$$

Consider a document collection that contains a document, $D_1$, with ten occurrences of the term *green* and a document, $D_2$, with only five occurrences of the term *green*. If *green* is the only term found in the query, then document $D_1$ is ranked higher than $D_2$.

When a document retrieval system is used to query a collection of documents with $t$ distinct collection-wide terms, the system computes a vector D $(d_{i1}, d_{i2}, \ldots, d_{it})$ of size $t$ for each document. The vectors are filled with term weights as described above. Similarly, a vector Q $(w_{q1}, w_{q2}, \ldots, w_{qt})$ is constructed for the terms found in the query.

A simple similarity coefficient (SC) between a query Q and a document $D_i$ is defined by the dot product of two vectors. Since a query vector is similar in length to a document vector, this same measure is often used to compute the similarity between two documents. We discuss this application of an SC as it applies to document clustering in Section 3.2.

$$SC(Q, D_i) = \sum_{j=1}^{t} w_{qj} \times d_{ij}$$

### 2.1.1 Example of Similarity Coefficient

Consider a case insensitive query and document collection with a query Q and a document collection consisting of the following three documents:

Q: "gold silver truck"
$D_1$: "Shipment of gold damaged in a fire"
$D_2$: "Delivery of silver arrived in a silver truck"
$D_3$: "Shipment of gold arrived in a truck"

In this collection, there are three documents, so $d = 3$. If a term appears in only one of the three documents, its *idf* is $\log \frac{d}{df_j} = \log \frac{3}{1} = 0.477$. Similarly, if a term appears in two of the three documents its *idf* is $\log \frac{3}{2} = 0.176$, and a term which appears in all three documents has an *idf* of $\log \frac{3}{3} = 0$.

The *idf* for the terms in the three documents is given below:

| | |
|---|---|
| $idf_a = 0$ | $idf_{in} = 0$ |
| $idf_{arrived} = 0.176$ | $idf_{of} = 0$ |
| $idf_{damaged} = 0.477$ | $idf_{silver} = 0.477$ |
| $idf_{delivery} = 0.477$ | $idf_{shipment} = 0.176$ |
| $idf_{fire} = 0.477$ | $idf_{truck} = 0.176$ |

$idf_{gold} = 0.176$

Document vectors can now be constructed. Since eleven terms appear in the document collection, an eleven-dimensional document vector is constructed. The alphabetical ordering given above is used to construct the document vector so that $t_1$ corresponds to term number one which is *a* and $t_2$ is *arrived*, etc. The weight for term $i$ in vector $j$ is computed as the $idf_i \times tf_{ij}$. The document vectors are shown in Table 2.1.

*Table 2.1.* Document Vectors

| docid | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_1$ | 0 | 0 | .477 | 0 | .477 | .176 | 0 | 0 | .176 | 0 | 0 |
| $D_2$ | 0 | .176 | 0 | .477 | 0 | 0 | 0 | 0 | 0 | .954 | .176 |
| $D_3$ | 0 | .176 | 0 | 0 | 0 | .176 | 0 | 0 | .176 | 0 | .176 |
| Q | 0 | 0 | 0 | 0 | 0 | .176 | 0 | 0 | 0 | .477 | .176 |

$$\begin{aligned} SC(Q, D_1) = & (0)(0) + (0)(0) + (0)(0.477) + (0)(0) \\ & +(0)(0.477) + (0.176)(0.176) + (0)(0) + (0)(0) \\ & +(0)(0.176) + (0.477)(0) + (0.176)(0) \\ & = (0.176)^2 \approx 0.031 \end{aligned}$$

Similarly,

$$SC(Q, D_2) = (0.954)(0.477) + (0.176)^2 \approx 0.486$$

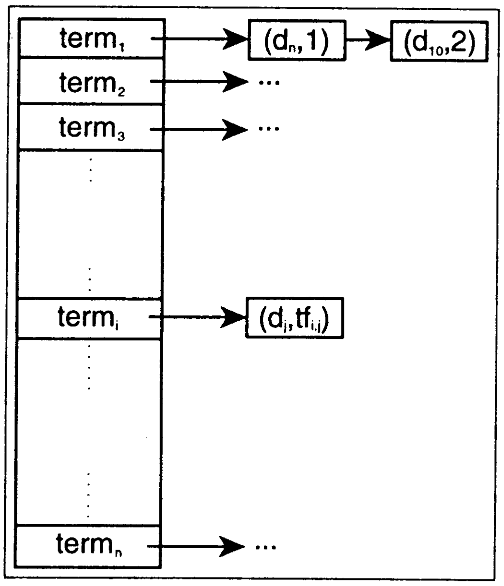
$$SC(Q, D_3) = (0.176)^2 + (0.176)^2 \approx 0.062$$

Hence, the ranking would be $D_2$, $D_3$, $D_1$.

Implementations of the vector space model and other retrieval strategies typically use an inverted index to avoid a lengthy sequential scan through every document to find the terms in the query. Instead, an inverted index is generated prior to the user issuing any queries. Figure 2.3 illustrates the structure of the inverted index. An entry for each of the $n$ terms is stored in a structure called the *index*. For each term, a pointer references a logical linked list called the *posting list*. The posting list contains an entry for each unique document that contains the term. In the figure below, the posting list contains both a document identifier and the term frequency. (In practice, structures more efficient than linked lists are often used, but conceptually they serve the same purpose).

The posting list in the figure indicates that term $t_1$ appears once in document one and twice in document ten. An entry for an arbitrary term $t_i$ indicates that it occurs $tf$ times in document $j$. Details of inverted index construction and use are provided in Chapter 5, but it is useful to know that inverted indexes are commonly used to improve run-time performance of various retrieval strategies.

*Figure 2.3.* Inverted Index



Early work described the vector space model in the late 1960's [Salton and Lesk, 1968]. This model became popular in the mid-1970's [Salton et al., 1975] and is still an extremely popular means of computing a measure of similarity between a query and a document [TREC, 2003]. The measure is important as it is used by a retrieval system to identify which documents are displayed to the user. Typically, the user requests the top $n$ documents, and these are displayed ranked according to the similarity coefficient.

Subsequently, work on term weighting was done to improve on the basic combination of *tf-idf* weights [Salton and Buckley, 1988]. Many variations were studied, and the following weight for term $j$ in document $i$ was identified as a good performer:

$$w_{ij} = \frac{(\log tf_{ij} + 1.0) * idf_j}{\sum_{j=1}^{t}[(\log tf_{ij} + 1.0) * idf_j]^2}$$

The motivation for this weight is that a single matching term with a high term frequency can skew the effect of remaining matches between a query and a given document. To avoid this, the $\log(tf) + 1$ is used reduce the range of term frequencies. A variation on the basic theme is to use weight terms in the query differently than terms in the document.

One term weighting scheme, referred to as *lnc.ltc*, was effective. It uses a document weight of $(1 + \log(tf))(idf)$ and query weight of $(1 + \log(tf))$. The label *lnc.ltc* is of the form: *qqq.ddd* where *qqq* refers to query weights and *ddd* refers to document weights. The three letters: *qqq* or *ddd* are of the form *xyz*.

The first letter, *x*, is either $n$, $l$, or $a$. $n$ indicates the "natural" term frequency or just $tf$ is used. $l$ indicates that the logarithm is used to scale down the weight so $1 + \log(tf)$ is used. $a$ indicates that an augmented weight was used where the weight is $0.5 + 0.5 \times \frac{tf}{tf_{max}}$.

The second letter, *y*, indicates whether or not the $idf$ was used. A value of $n$ indicates that no $idf$ was used while a value of $t$ indicates that the $idf$ was used.

The third letter, *z*, indicates whether or not document length normalization was used. By normalizing for document length, we are trying to reduce the impact document length might have on retrieval (see Equation 2.1). A value of $n$ indicates no normalization was used, a value of $c$ indicates the standard cosine normalization was used, and a value of $u$ indicates pivoted length normalization was used in [Singhal, 1997].

## 2.1.2 Similarity Measures

Several different means of comparing a query vector with a document vector have been implemented. These are well documented and are presented here simply as a quick review. The most common of these is the cosine measure where the cosine of the angle between the query and document vector is given:

$$SC(Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} d_{ij}}{\sqrt{\sum_{j=1}^{t} (d_{ij})^2 \sum_{j=1}^{t} (w_{qj})^2}}$$

Since the $\sqrt{\sum_{j=1}^{t} (w_{qj})^2}$ appears in the computation for every document, the cosine coefficient should give the same relevance results as dividing the inner product by the magnitude of the document vector. Note that the cosine measure "normalizes" the result by considering the length of the document. With the inner product measure, a longer document can result in a higher score simply because it is longer, and thus, has a higher chance of containing terms that match the query—not necessarily because it is relevant.

The Dice coefficient is defined as:

$$SC(Q, D_i) = \frac{2\sum_{j=1}^{t} w_{qj} d_{ij}}{\sum_{j=1}^{t}(d_{ij})^2 \sum_{j=1}^{t}(w_{qj})^2}$$

The Jaccard coefficient is defined as:

$$SC(Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} d_{ij}}{\sum_{j=1}^{t}(d_{ij})^2 + \sum_{j=1}^{t}(w_{qj})^2 - \sum_{j=1}^{t} w_{qj} d_{ij}}$$

The cosine measure levels the playing field by dividing the computation by the length of the document vector. The assumption used in the cosine measure is that document length has no impact on relevance. Without a normalization factor, longer documents are more likely to be found relevant simply because they have more terms which increases the likelihood of a match. Dividing by the document vector removes the size of the document from consideration.

It turns out that (at least for the TREC data), this basic assumption is not correct. Taking all of the relevant documents found for a set of fifty TREC queries, Singhal found that more documents judged to be relevant actually were found in longer documents [Singhal, 1997]. The reason for this might be that a longer document simply has more opportunity to have some components that are indeed relevant to a given query.

To identify a means of adjusting the normalization factor, Singhal compared the likelihood of relevance with the likelihood of retrieval in a collection where the documents relevant to a set of queries was known. Ideally, if the probability of retrieval and the probability of relevance are both plotted against the length of the document, the two curves should be roughly the same. Since this is not the case (the two curves actually cross), there must be a document length in which the probability of relevance equals the probability of retrieval. Before this point (referred to as the *pivot*), a document is more likely to be retrieved than relevant. After this point, the reverse is true. Once the pivot is found, a "correction factor" can be used to adjust the normalization. The "correction factor" is computed from a linear equation whose value at *pivot* is equal to *pivot* and whose slope is selected to increase the normalization for shorter documents so that their probability of selection is equal to their probability of relevance. Thus, the similarity coefficient is:

$$SC(Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} d_{ij}}{(1.0 - s)p + (s)\sqrt{\sum_{j=1}^{t}(d_{ij})^2}}$$

This scheme has two variables: $s$ and $p$ for the slope and pivot, respectively. However, it is possible to express the slope as a function of pivot. Singhal selects as pivot the average normalization factor taken over the entire collection prior to any correction and adjusts the slope accordingly. At the same time the normalization factor is divided by $(1.0 - s)p$. The resulting equation for the similarity coefficient:

$$SC(Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} d_{ij}}{(1.0 - s) + (s) \frac{\sqrt{\sum_{j=1}^{t} (d_{ij})^2}}{avgn}} \tag{2.1}$$

where $avgn$ is the average document normalization factor before any correction is made.

The pivoted scheme works fairly well for short and moderately long documents, but extremely long documents tend to be more favored than those without any normalization. To remedy this, the number of unique terms in a document, $|d_i|$ is proposed as the normalization function prior to any adjustment.

A final adjustment is made to account for extremely high term frequencies that occur in very large documents. First, a weight of $(1 + \log tf)$ is used to scale the frequency. To account for longer documents, an individual term weight is divided by the weight given to the average term frequency.

The new weight, $d_{ij}$, is computed as—

$$d_{ij} = \frac{1 + \log tf}{1 + \log(atf)}$$

Using this new weight, and dividing it by the correction factor gives the following equation:

$$SC(Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} d_{ij}}{((1.0 - s)p + (s)(|d_i|))} \tag{2.2}$$

We then compute the average number of unique terms in a document for a given collection and use this as the pivot, $p$. Once this is done, the collection can be trained for a good slope. Equation 2.2 is referred to as *pivoted unique normalization* and it was shown to provide improved effectiveness over *pivoted cosine normalization* given in Equation 2.1. The modified normalization factor makes it more likely to retrieve longer documents and consistently shows about a ten percent improvement for TREC queries.

It should also be noted that the vector space model assumes terms are independent. One approach to alleviating the question of term independence in the vector space model is to change the basis. Although changing the basis does not totally eliminate the problem, it can reduce it. The idea is to pick a basis vector for each combination of terms that exist in a document (regardless of the number of occurrences of the term). The new basis vectors can be made mutually orthogonal and can be scaled to be unit vectors. The documents and the query can be expressed in terms of the new basis vectors. Using this procedure in conjunction with other (possibly probabilistic) methods avoids independence assumptions, but in practice, it has not been shown to significantly improve effectiveness.

## 2.2 Probabilistic Retrieval Strategies

The probabilistic model computes the similarity coefficient (SC) between a query and a document as the probability that the document will be relevant to the query. This reduces the relevance ranking problem to an application of probability theory. A survey on probabilistic methods is given in [Fuhr, 1992].

Probability theory can be used to compute a measure of relevance between a query and a document. Two fundamentally different approaches were proposed. The first relies on usage patterns to predict relevance [Maron and Kuhns, 1960], the second uses each term in the query as clues as to whether or not a document is relevant [Robertson and Sparck Jones, 1976].

The original work on the use of probability theory to retrieve documents can be traced to Maron and Kuhns. Their work developed an area of research where the probability that a document will be relevant given a particular term is estimated.

All of the work on probabilistic retrieval stems from the concept of estimating a term's weight based on how often the term appears or does not appear in relevant documents and non-relevant documents, respectively. Section 2.2.1 describes the simple term weight model, a non-binary independence model is discussed in Section 2.2.2, and Sections 2.2.3 and 2.2.4 describe the Poisson and component-based models which have both performed well on the TREC collection. Finally, Section 2.2.5 focuses on two large issues with the model—parameter estimation and independence assumptions.

### 2.2.1 Simple Term Weights

The use of term weights is based on the Probability Ranking Principle (PRP), which assumes that optimal effectiveness occurs when documents are ranked based on an estimate of the probability of their relevance to a query [Robertson, 1977].

The key is to assign probabilities to components of the query and then use each of these as evidence in computing the final probability that a document is relevant to the query.

The terms in the query are assigned weights which correspond to the probability that a particular term, in a match with a given query, will retrieve a relevant document. The weights for each term in the query are combined to obtain a final measure of relevance.

Most of the papers in this area incorporate probability theory and describe the validity of independence assumptions, so a brief review of probability theory is in order.

Suppose we are trying to predict whether or not a softball team called the Salamanders will win one of its games. We might observe, based on past experience, that they usually win on sunny days when their best shortstop plays. This means that two pieces of evidence, outdoor-conditions and presence of good-shortstop, might be used. For any given game, there is a seventy five percent chance that the team will win if the weather is sunny and a sixty percent chance that the team will win if the shortstop plays. Therefore, we write:

P(win | sunny) = 0.75
P(win | good-shortstop) = 0.6

The conditional probability that the team will win given both situations is written as p(win | sunny, good-shortstop). This is read "the probability that the team will win given that there is a sunny day and the good-shortstop plays." We have two pieces of evidence indicating that the Salamanders will win. Intuition says that together the two pieces should be stronger than either alone. This method of combining them is to "look at the odds." A seventy-five percent chance of winning is a twenty-five percent chance of losing, and a sixty percent chance of winning is a forty percent chance of losing. Let us assume the independence of the pieces of evidence.

P(win | sunny, good-shortstop) = $\alpha$
P(win | sunny) = $\beta$
P(win | good-shortstop) = $\gamma$

By Bayes' Theorem:

$$\alpha = \frac{P(win, sunny, good\text{–}shortstop)}{P(sunny, good\text{–}shortstop)} = \frac{P(sunny, good\text{–}shortstop|win)P(win)}{P(sunny, good\text{–}shortstop)}$$

Therefore:

$$\frac{\alpha}{1-\alpha} = \frac{P(sunny, good\text{–}shortstop|win)P(win)}{P(sunny, good\text{–}shortstop|lose)P(lose)}$$

Solving for the first term (because of the independence assumptions):

$$\frac{P(sunny, good\text{–}shortstop|win)}{P(sunny, good\text{–}shortstop|lose)} = \frac{P(sunny|win)P(good\text{–}shortstop|win)}{P(sunny|lose)P(good\text{–}shortstop|lose)}$$

Similarly,

$$\frac{\beta}{1-\beta} = \frac{P(sunny|win)P(win)}{P(sunny|lose)P(lose)}$$

$$\frac{\gamma}{1-\gamma} = \frac{P(good\text{–}shortstop|win)P(win)}{P(good\text{–}shortstop|lose)P(lose)}$$

Making all of the appropriate substitutions, we obtain:

$$\frac{\alpha}{1-\alpha} = \left(\frac{\beta}{1-\beta}\right)\left(\frac{P(lose)}{P(win)}\right)\left(\frac{\gamma}{1-\gamma}\right)\left(\frac{P(lose)}{P(win)}\right)\left(\frac{P(win)}{P(lose)}\right)$$

Simplifying:

$$\frac{\alpha}{1-\alpha} = \left(\frac{\beta}{1-\beta}\right)\left(\frac{\gamma}{1-\gamma}\right)\left(\frac{P(lose)}{P(win)}\right)$$

Assume the Salamanders are a 0.500 ball club (that is they win as often as they lose) and assume numeric values for $\beta$ and $\gamma$ of 0.6 and 0.75, respectively. We then obtain:

$$\frac{\alpha}{1-\alpha} = \left(\frac{0.6}{0.4}\right)\left(\frac{0.75}{0.25}\right)\left(\frac{0.500}{0.500}\right) = (1.5)(3.0)(1.0) = 4.5$$

Solving for $\alpha$ gives a value of $\frac{9}{11} = 0.818$.

Note the combined effect of both sunny weather and the good-shortstop results in a higher probability of success than either individual condition.

The key is the independence assumptions. The likelihood of the weather being nice and the good-shortstop showing up are completely independent. The chance the shortstop will show up is not changed by the weather. Similarly, the weather is not affected by the presence or absence of the good-shortstop. If the independence assumptions are violated : suppose the shortstop prefers sunny weather — special consideration for the dependencies is required. The independence assumptions also require that the weather and the appearance of the good-shortstop are independent given either a win or a loss.

For an information retrieval query, the terms in the query can be viewed as indicators that a given document is relevant. The presence or absence of query term A can be used to predict whether or not a document is relevant. Hence, after a period of observation, it is found that when term A is in both the query and the document, there is an $x$ percent chance the document is relevant. We then assign a probability to term A. Assuming independence of terms, this can be done for each of the terms in the query. Ultimately, the product of all the weights can be used to compute the probability of relevance.

We know that independence assumptions are really not a good model of reality. Some research has investigated why systems with these assumptions have performed reasonably well, despite their theoretical problems [Cooper, 1991]. For example, a relevant document that has the term *apple* in response to a query for *apple pie* probably has a better chance of having the term *pie* than some other randomly selected term. Hence, the key independence assumption is violated.

Most work in the probabilistic model assumes independence of terms because handling dependencies involves substantial computation. It is unclear whether or not effectiveness is improved when dependencies are considered. We note that relatively little work has been done implementing these approaches. They are computationally expensive, but more importantly, they are difficult to estimate. It is necessary to obtain sufficient training data about term co-occurrence in both relevant and non-relevant documents. Typically, it is very difficult to obtain sufficient training data to estimate these parameters.

In figure 2.4, we illustrate the need for training data with most probabilistic models. A query with two terms, $q_1$ and $q_2$, is executed. Five documents are returned and an assessment is made that documents two and four are relevant. From this assessment, the probability that a document is relevant (or non-relevant) given that it contains term $q_1$ is computed. Likewise, the same probabilities are computed for term $q_2$. Clearly, these probabilities are estimates based on training data. The idea is that sufficient training data can be obtained so that when a user issues a query, a good estimate of which documents are relevant to the query can be obtained.

Consider a document, $d_i$, consisting of $t$ terms $(w_1, w_2, \ldots, w_t)$, where $w_i$ is the estimate that term $i$ will result in this document being relevant. The weight or "odds" that document $d_i$ is relevant is based on the probability of relevance for each term in the document. For a given term in a document, its contribution to the estimate of relevance for the entire document is computed as:

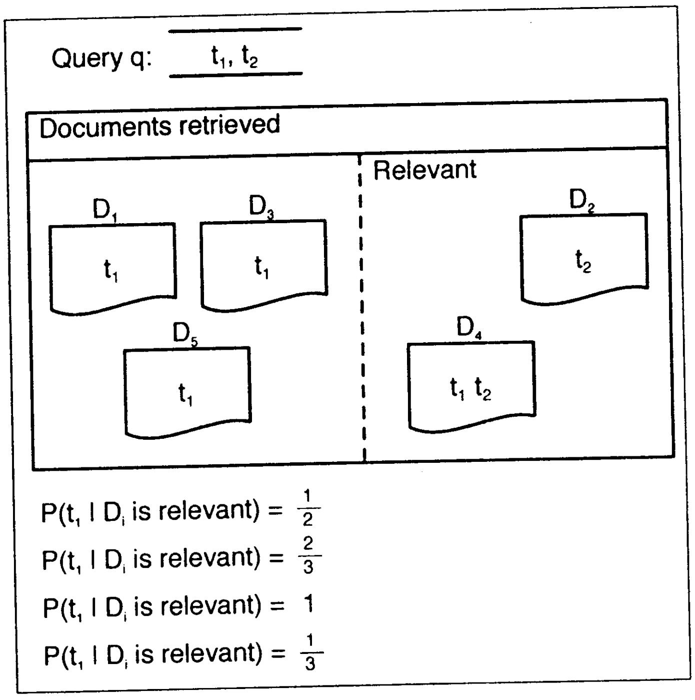
$$\frac{P(w_i|rel)}{P(w_i|nonrel)}$$

The question is then: How do we combine the odds of relevance for each term into an estimate for the entire document? Given our independence assumptions, we can multiply the odds for each term in a document to obtain the odds that the document is relevant. Taking the log of the product yields:

$$\log\left(\prod_{i=1}^{t} \frac{P(w_i|\ rel)}{P(w_i|\ nonrel)}\right) = \sum_{i=1}^{t} \log\left(\frac{P(w_i|rel)}{P(w_i|nonrel)}\right)$$

We note that these values are computed based on the assumption that terms will occur independently in relevant and non-relevant documents. The assumption is also made that if one term appears in a document, then it has no impact on whether or not another term will appear in the same document.

*Figure 2.4.* Training Data for Probabilistic Retrieval



Query q: $t_1, t_2$
Documents retrieved
Relevant
$D_1$
$t_1$
$D_3$
$t_1$
$D_5$
$t_1$
$D_2$
$t_2$
$D_4$
$t_1$ $t_2$
$P(t_1 \mid D_i \text{ is relevant}) = \frac{1}{2}$
$P(t_1 \mid D_i \text{ is relevant}) = \frac{2}{3}$
$P(t_1 \mid D_i \text{ is relevant}) = 1$
$P(t_1 \mid D_i \text{ is relevant}) = \frac{1}{3}$

Now that we have described how the individual term estimates can be combined into a total estimate of relevance for the document, it is necessary to describe a means of estimating the individual term weights. Several different means of computing the probability of relevance and non-relevance for a given term were studied since the introduction of the probabilistic retrieval model. In their 1976 paper, Robertson and Sparck Jones considered several methods [Robertson and Sparck Jones, 1976]. They began by presenting two mutually exclusive independence assumptions:

**I1:** The distribution of terms in relevant documents is independent and their distribution in all documents is independent.

**I2:** The distribution of terms in relevant documents is independent and their distribution in non-relevant documents is independent.

They also presented two methods, referred to as ordering principles, for presenting the result set:

**O1:** Probable relevance is based only on the presence of search terms in the documents.

**O2:** Probable relevance is based on both the presence of search terms in documents and their absence from documents.

I1 indicates that terms occur randomly within a document—that is, the presence of one term in a document in no way impacts the presence of another term in the same document. This is analogous to our example in which the presence of the good-shortstop had no impact on the weather given a win. This also states that the distribution of terms across all documents is independent unconditionally for all documents—that is, the presence of one term in a document in no way impacts the presence of the same term in other documents. This is analogous to saying that the presence of a good-shortstop in one game has no impact on whether or not a good-shortstop will play in any other game. Similarly, the presence of good-shortstop in one game has no impact on the weather for any other game.

I2 indicates that terms in relevant documents are independent—that is, they satisfy I1 and terms in non-relevant documents also satisfy I1. Returning to our example, this is analogous to saying that the independence of a good-shortstop and sunny weather holds regardless of whether the team wins or loses.

O1 indicates that documents should be highly ranked only if they contain matching terms in the query (i.e., the only evidence used is which query terms are actually present in the document). We note that this ordering assumption is not commonly held today because it is also important to consider when query terms are not found in the document. This is inconvenient in practice. Most systems use an inverted index that identifies for each term, all occurrences of that term in a given document. If absence from a document is required, the index would have to identify all terms *not* in a document (for a detailed discussion of inverted indexes see Section 5.1). To avoid the need to track the absence of a term in a document, the estimate makes the zero point correspond to the probability of relevance of a document lacking all the query terms—as opposed to the probability of relevance of a random document. The zero point does not mean that we do not know anything: it simply means that we have some evidence for non-relevance. This has the effect of converting the O2 based weights to presence-only weights.

O2 takes O1 a little further and says that we should consider both the *presence* and the *absence* of search terms in the query. Hence, for a query that asks for term $t_1$ and term $t_2$—a document with just one of these terms should be ranked lower than a document with both terms.

Four weights are then derived based on different combinations of these ordering principles and independence assumptions. Given a term, $t$, consider the following quantities:

$$
\begin{aligned}
N &= \text{number of documents in the collection} \\
R &= \text{number of relevant documents for a given query } q \\
n &= \text{number of documents that contain term } t \\
r &= \text{number of relevant documents that contain term } t
\end{aligned}
$$

Choosing I1 and O1 yields the following weight:

$$w_1 = \log\left(\frac{\frac{r}{R}}{\frac{n}{N}}\right)$$

Choosing I2 and O1 yields the following weight:

$$w_2 = \log\left(\frac{\frac{r}{R}}{\frac{n-r}{N-R}}\right)$$

Choosing I1 and O2 yields the following weight:

$$w_3 = \log\left(\frac{\frac{r}{R-r}}{\frac{n}{N-n}}\right)$$

Choosing I2 and O2 yields the following weight:

$$w_4 = \log\left(\frac{\frac{r}{R-r}}{\frac{n-r}{(N-n)-(R-r)}}\right)$$

Robertson and Sparck Jones argue that O2 is correct and that I2 is more likely than I1 to describe what actually occurs. Hence, $w_4$ is most likely to yield the best results. They then present results that indicate that $w_4$ and $w_3$ performed better than $w_1$ and $w_2$. Most subsequent work starts with $w_4$ and extends it to contain other important components such as the within-document frequency of the term and the relative length of a document. We describe these extensions to $w_4$ in Section 2.2.3.

When incomplete relevance information is available, 0.5 is added to the weights to account for the uncertainty involved in estimating relevance. Robertson and Sparck Jones suggest that, "This procedure may seem somewhat arbitrary, but it does in fact have some statistical justification." The modified weighting function appears as:

$$w = \log\left(\frac{\frac{r+0.5}{(R-r)+0.5}}{\frac{(n-r)+0.5}{(N-n)-(R-r)+0.5}}\right)$$

The claimed advantage to the probabilistic model is that it is entirely based on probability theory. The implication is that other models have a certain arbitrary characteristic. They might perform well experimentally, but they lack a sound theoretical basis because the parameters are not easy to estimate. Either complete training data are required, or an inaccurate estimate must be made.

This debate is similar to one that occurs when comparing a relational to an object-oriented database management system (DBMS). Object-oriented DBMS are sometimes said to model "real world" data, but lack sound theoretical basis. Relational DBMS, on the other hand, have very solid set-theoretic underpinnings, but sometimes have problems modeling real data.

#### 2.2.1.1 Example

Using the same example we used previously with the vector space model, we now show how the four different weights can be used for relevance ranking. Again, the documents and the query are:

| | |
|---|---|
| $Q$ : | "gold silver truck" |
| $D_1$: | "Shipment of gold damaged in a fire." |
| $D_2$: | "Delivery of silver arrived in a silver truck." |
| $D_3$: | "Shipment of gold arrived in a truck." |

Since training data are needed for the probabilistic model, we assume that these three documents are the training data and we deem documents $D_2$ and $D_3$ as relevant to the query.

To compute the similarity coefficient, we assign term weights to each term in the query. We then sum the weights of matching terms. There are four quantities we are interested in:

| | | |
|---|---|---|
| $N$ | $=$ | number of documents in the collection |
| $n$ | $=$ | number of documents indexed by a given term |
| $R$ | $=$ | number of relevant documents for the query |
| $r$ | $=$ | number of relevant documents indexed by the given term |

These values are given in the Table 2.2 for each term in the query. As we stated previously, Robertson and Sparck Jones described the following four different weighting equations to estimate, for a given term, the likelihood that a document which contains the query term is relevant.

*Table 2.2.* Frequencies for Each Query Term

| | *gold* | *silver* | *truck* |
|---|---|---|---|
| $N$ | 3 | 3 | 3 |
| $n$ | 2 | 1 | 2 |
| $R$ | 2 | 2 | 2 |
| $r$ | 1 | 1 | 2 |

$$w_1 = \log\left[\frac{\frac{r}{R}}{\frac{n}{N}}\right]$$

$$w_2 = \log\left[\frac{\frac{r}{R}}{\frac{(n-r)}{(N-R)}}\right]$$

$$w_3 = \log\left[\frac{\frac{r}{(R-r)}}{\frac{n}{(N-n)}}\right]$$

$$w_4 = \log\left[\frac{\frac{r}{(R-r)}}{\frac{(n-r)}{(N-n)-(R-r)}}\right]$$

Note that with our collection, the weight for *silver* is infinite, since $(n-r) = 0$. This is because "silver" only appears in relevant documents. Since we are using this procedure in a predictive manner, Robertson and Sparck Jones recommended adding constants to each quantity [Robertson and Sparck Jones, 1976]. The new weights are:

$$w_1 = \log\left[\frac{\frac{(r+0.5)}{(R+1)}}{\frac{(n+1)}{(N+2)}}\right]$$

$$w_2 = \log\left[\frac{\frac{(r+0.5)}{(R+1)}}{\frac{(n-r+0.5)}{(N-R+1)}}\right]$$

$$w_3 = \log\left[\frac{\frac{(r+0.5)}{(R-r+0.5)}}{\frac{(n+1)}{(N-n+1)}}\right]$$

$$w_4 = \log \left[ \frac{\frac{(r+0.5)}{(R-r+0.5)}}{\frac{(n-r+0.5)}{(N-n-(R-r)+0.5}} \right]$$

Using these equations, we derive the following weights:

**gold**

[illegible] $\frac{.5}{.6} = -0.079$

**silver**

$$w_1 = \log \left[ \frac{\frac{(1+0.5)}{(2+1)}}{\frac{(1+1)}{(3+2)}} \right] = \log \frac{0.5}{0.4} = 0.097$$

**truck**

$$w_1 = \log \left[ \frac{\frac{(2+0.5)}{(2+1)}}{\frac{(2+1)}{(3+2)}} \right] = \log \frac{0.833}{0.6} = 0.143$$

**gold**

$$w_2 = \log \left[ \frac{\frac{(1+0.5)}{(2+1)}}{\frac{(2-1+0.5)}{(3-2+1)}} \right] = \log \frac{0.5}{0.75} = -0.176$$

**silver**

$$w_2 = \log \left[ \frac{\frac{(1+0.5)}{(2+1)}}{\frac{(1-1+0.5)}{3-2+1}} \right] = \log \frac{0.5}{0.25} = 0.301$$

**truck**

$$w_2 = \log \left[ \frac{\frac{(2+0.5)}{(2+1)}}{\frac{(2-2+0.5)}{3-2+1}} \right] = \log \frac{0.833}{0.25} = 0.523$$

**gold**

$$w_3 = \log \left[ \frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(2+1)}{(3-2+1)}} \right] = \log \frac{1.0}{1.5} = -0.176$$

**silver**

$$w_3 = \log \left[ \frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(1+1)}{(3-1+1)}} \right] = \log \frac{1.0}{0.667} = 0.176$$

**truck**

$$w_3 = \log \left[ \frac{\frac{(2+0.5)}{(2-2+0.5)}}{\frac{(2+1)}{(3-2+1)}} \right] = \log \frac{5}{1.5} = 0.523$$

**gold**

$$w_4 = \log \left[ \frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(2-1+0.5)}{(3-2-2+1+0.5)}} \right] = \log \frac{1}{3} = -0.477$$

**silver**

$$w_4 = \log \left[ \frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(1-1+0.5)}{(3-1-2+1+0.5)}} \right] = \log \frac{1}{0.333} = 0.477$$

**truck**

$$w_4 = \log \left[ \frac{\frac{(2+0.5)}{(2-2+0.5)}}{\frac{(2-2+0.5)}{(3-2-2+2+0.5)}} \right] = \log \left( \frac{5}{0.333} \right) = 1.176$$

The results are summarized in Table 2.3.

*Table 2.3.* Term Weights

| | $w_1$ | $w_2$ | $w_3$ | $w_4$ |
|---|---|---|---|---|
| gold | -0.079 | -0.176 | -0.176 | -0.477 |
| silver | 0.097 | 0.301 | 0.176 | 0.477 |
| truck | 0.143 | 0.523 | 0.523 | 1.176 |

*Table 2.4.* Document Weights

| | $w_1$ | $w_2$ | $w_3$ | $w_4$ |
|---|---|---|---|---|
| $D_1$ | -0.079 | -0.176 | -0.176 | -0.477 |
| $D_2$ | 0.240 | 0.824 | 0.699 | 1.653 |
| $D_3$ | 0.064 | 0.347 | 0.347 | 0.699 |

The similarity coefficient for a given document is obtained by summing the weights of the terms present. Table 2.4 gives the similarity coefficients for each of the four different weighting schemes. For $D_1$, *gold* is the only term to appear so the weight for $D_1$ is just the weight for *gold*, which is -0.079. For $D_2$, *silver* and *truck* appear so the weight for $D_2$ is the sum of the weights for *silver* and *truck*, which is $0.097 + 0.143 = 0.240$. For $D_3$, *gold* and *truck* appear so the weight for $D_3$ is the sum for *gold* and *truck*, which is $-0.079 + 0.143 = 0.064$.

### 2.2.1.2 Results

Initial tests of the four weights were done on the 1,400 document Cranfield collection. These showed that the third and fourth weights performed somewhat comparably, but were superior to the first and second weights. An additional study against the 27,361 document UKCIS collection measured the difference in the first weight and the fourth weight [Sparck Jones, 1979a]. Again a significant improvement was found in the use of the fourth weight.

Two other baseline tests were run. The first simply ranked documents based on the number of term matches, the second test used inverse document frequency as an estimated weight. Both of these approaches were inferior to any of the four weights, but the use of the *idf* was better than simply counting term matches. In all cases, the ranking of the documents was D2, D3, D1—the same ranking that was obtained with the vector space model in Section 2.1.

The number of times a term appears in a given document is not used, as the weighting functions are based on whether or not the term appears in lots of relevant documents. Thus, if term $t$ appears 50 times over the span of 10 relevant documents and term $u$ appears only 10 times in the same relevant documents, they are given the same weight.

#### 2.2.1.3 Incorporating Term Frequency

Term frequency was not used in the original probabilistic model. Croft and Harper incorporated term frequency weights in [Croft and Harper, 1979]. Relevance is estimated by including the probability that a term will appear in a given document, rather than the simple presence or absence of a term in a document. The term frequency is used to derive an estimate of how likely it is for the term to appear in a document. This new coefficient is given below.

$$SC(Q, D_j) = C\sum_{i=1}^{t} q_i d_{ij} + \sum_{i=1}^{t} P(d_{ij}) q_i d_{ij} \log\left(\frac{N - n_i}{n_i}\right)$$

The $P(d_{ij})$ indicates the probability that term $i$ appears in document $j$, and can be estimated simply as the term frequency of term $i$ in document $j$. Unfortunately, this frequency is not a realistic probability so another estimate, *normalized term frequency* is used. The normalized term frequency is computed as:

$$ntf_{ij} = \frac{tf_{ij}}{max(tf_{1j}, tf_{2j}, \ldots, tf_{tj})}$$

Normalized term frequency is the ratio of the term frequency of a given term to the maximum term frequency of any term in the document. If term $i$ appears ten times in the document, and the highest term frequency of any other term in the document is 100, the $ntf_{ij}$ is 0.1.

Croft and Harper compared the use of the normalized term frequency, the unnormalized term frequency, and a baseline without any use of term frequency for the Cranfield collection and the 11,429 document NPL collection. The results were statistically significant in that the normalized term frequency outperformed the baseline. In many cases, the unnormalized term frequency performed worse than the baseline.

### 2.2.2 Non-Binary Independence Model

The non-binary independence model developed by Yu, Meng, and Park incorporates term frequency and document length, somewhat naturally, into the calculation of term weights [Yu et al., 1989]. Once the term weights are computed, the vector space model (see Section 2.1) is used to compute an inner product for obtaining a final similarity coefficient.

The simple term weight approach estimates a term's weight based on whether or not the term appears in a relevant document. Instead of estimating the probability that a given term will identify a relevant document, the probability that a *term which appears tf times* will appear in a relevant document is estimated. For example, consider a ten document collection in which document one contains the term *blue* once and document two contains ten occurrences of the term *blue*. Assume both documents one and two are relevant, and the eight other documents are not relevant. With the simple term weight model, we would compute the P(Rel | *blue*) = 0.2 because *blue* occurs in two out of ten relevant documents.

With the non-binary independence model, we calculate a separate probability for each term frequency. Hence, we compute the probability that *blue* will occur one time P(1 | R) = 0.1, because it did occur one time in document one. The probability that *blue* will occur ten times is P(10 | R) = 0.1, because it did occur ten times in one out of ten documents.

To incorporate document length, weights are normalized based on the size of the document. Hence, if document one contains five terms and document two contains ten terms, we recompute the probability that *blue* occurs only once in a relevant document to the probability that *blue* occurs 0.5 times in a relevant document.

The probability that a term will result in a non-relevant document is also used. The final weight is computed as the ratio of the probability that a term will occur *tf* times in relevant documents to the probability that the term will occur *tf* times in non-relevant documents.

More formally:

$$\log \frac{P(d_i|R)}{P(d_i|N)}$$

where $P(d_i|R)$ is the probability that a relevant document will contain $d_i$ occurrences of the $i^{th}$ term, and $P(d_i|N)$ is the probability that a non-relevant document has $d_i$ occurrences of the $i^{th}$ term.

### 2.2.2.1 Example

Returning to our example, the documents and the query are:

$Q$ : "gold silver truck"
$D_1$: "Shipment of gold damaged in a fire."
$D_2$: "Delivery of silver arrived in a silver truck."
$D_3$: "Shipment of gold arrived in a truck."

*Table 2.5.* Term to Document Mapping

| docid | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_1$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| $D_2$ | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 2 | 1 |
| $D_3$ | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| Q | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

Thus, we have three documents with eleven terms and a single query (see Table 2.5). The training data include both relevant and non-relevant documents. We assume that document two and three are relevant and document one is not relevant (we are free to do this as relevance, after all, is in the eyes of the beholder). Normalizing by document length yields as shown in Table 2.6:

*Table 2.6.* Normalized Document Length

| docid | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_1$ | $\frac{1}{7}$ | 0 | $\frac{1}{7}$ | 0 | $\frac{1}{7}$ | $\frac{1}{7}$ | $\frac{1}{7}$ | $\frac{1}{7}$ | $\frac{1}{7}$ | 0 | 0 |
| $D_2$ | $\frac{1}{8}$ | $\frac{1}{8}$ | 0 | $\frac{1}{8}$ | 0 | 0 | $\frac{1}{8}$ | $\frac{1}{8}$ | 0 | $\frac{1}{4}$ | $\frac{1}{8}$ |
| $D_3$ | $\frac{1}{7}$ | $\frac{1}{7}$ | 0 | 0 | 0 | $\frac{1}{7}$ | $\frac{1}{7}$ | $\frac{1}{7}$ | $\frac{1}{7}$ | 0 | $\frac{1}{7}$ |

We do not normalize the query. The terms present in the query are *gold*, *silver*, and *truck*. For $D_1$ the weight of *gold* is

$$\log\left[\frac{P(\frac{1}{7}|R)}{P(\frac{1}{7}|N)}\right] = \log\frac{\frac{1}{2}}{1} = -0.3010.$$

Of the two relevant documents, one has a frequency of $\frac{1}{7}$ and one does not, so $P(\frac{1}{7}|R) = \frac{1}{2}$. However, the only non-relevant document has *gold* with a frequency of $\frac{1}{7}$, so $P(\frac{1}{7}|N) = 1$.

For *silver* in $D_1$ we obtain:

$$\log\left[\frac{P(0|R)}{P(0|N)}\right] = \log\frac{1}{2} = -0.3010.$$

Weights for each term and a given term frequency can be computed in this way for each term in a document. Vectors can then be constructed and a similarity coefficient can be computed between a query and each document.

With our example, there are only a few frequencies to consider, but a normal collection would have a large number of frequencies, especially if document length normalization is used. To alleviate this problem, it is possible to aggregate all of the frequencies into classes. Thus, all of the documents with zero frequency would be in one class, but for terms with positive term frequency, intervals $(0, f_1], (f_1, f_2], \ldots, (f_n, \infty)$ would be selected such that the intervals contain approximately equal numbers of terms. To obtain the weights, $P(d_i|R)$ and $P(d_i|N)$ are replaced by $P(d_i \in I_j|R)$ and $P(d_i \in I_j|N)$, respectively. $I_j$ is the $j^{\underline{th}}$ interval $(f_{j-2}, f_{j-1}]$. The weight becomes:

$$\log\left[\frac{P(d_i \in I_j|R)}{P(d_i \in I_j|N)}\right]$$

## 2.2.3 Poisson Model

Robertson and Walker developed a probabilistic model which uses a Poisson distribution to estimate probabilities of relevance and incorporate term frequency and document length [Robertson and Walker, 1994]. In the standard probabilistic model, the weighting is given by:

$$w = \log\frac{p(1-q)}{q(1-p)}$$

where $p$ is the probability that the term is present given that a document is relevant, and $q$ is the probability that the term is present given that a document is not relevant.

To incorporate the term frequencies, $p_{tf}$ is used. This indicates the probability that the term is present with frequency $tf$, given relevance and $q_{tf}$ is the corresponding probability for non-relevance. The subscript 0 denotes the absence of a term. The weighting then becomes:

$$w = \log\frac{(p_{tf})(q_0)}{(q_{tf})(p_0)}$$

The assumption is made that terms randomly occur within the document according to the Poisson distribution.

$$p(tf) = e^{-m}\frac{m^{tf}}{tf!}$$

The parameter $m$ differs according to whether or not the document is *about* the concepts represented by the query terms. This leads to the weighting:

$$w = \log \frac{(p' + (1-p')(\frac{\mu}{\lambda})^{tf}e^{j})(q'e^{(-j)} + (1-q'))}{(q' + (1-q')(\frac{\mu}{\lambda})^{tf}e^{j})(p'e^{(-j)} + (1-p'))}$$

where $\lambda$ is the Poisson mean for documents which are about the term $t$,
$\mu$ is the Poisson mean for documents which are not about the term $t$,
$j$ is the difference: $\lambda - \mu$,
$p'$ is the probability that a document is about $t$ given that it is relevant, and
$q'$ is the probability that a document is about $t$ given that it is not relevant.

The difficulty with this weight is in its application; it is unlikely that there will be direct evidence for any of the four parameters: $p', q', \lambda, \mu$. The shape of the curve is used, and simpler functions are found, based on the more readily observable quantities: term frequency and document length, that have similar shape. To incorporate term frequency, we use the function:

$$w' = w\frac{tf}{k_1 + tf}$$

where $w$ is the standard probabilistic weight, and $k_1$ is an unknown constant whose value depends on the collection and must be determined experimentally.

Document length is also taken into account. The simplest means to account for document length is to modify the equation given above for $w'$ by substituting:

$$k_1 = \frac{(k_1)(d)}{\Delta}$$

$$\begin{aligned} d &= \text{document length} \\ \Delta &= \text{average document length} \end{aligned}$$

The new equation for $w'$ is:

$$w' = w\left(\frac{tf}{\frac{(k_1)(d)}{\Delta} + tf}\right)$$

The symmetry between documents and queries is used to incorporate the query term frequency in a fashion similar to document frequency. A tuning parameter $k_1$ is used to scale the effect of document term frequency. Similarly, another parameter $k_3$ is used to scale the query term frequency ($qtf$). Finally, a closer match to the 2-Poisson estimate can be attempted with an additional term possessing a scaling factor of $k_2$. This term is:

$$k_2\left(|Q|\left(\frac{(\Delta - d)}{(\Delta + d)}\right)\right)$$

where $k_2$ is a constant that is experimentally determined, and $|Q|$ is the number of query terms. This term enables a high value of $k_2$ to give additional emphasis to documents that are shorter than average. These modifications result in the following similarity coefficient:

$$\begin{aligned} SC(Q, D_i) &= \sum_{j=1}^{t} \log\left(\frac{\frac{r}{(R-r)}}{\frac{(n-r)}{(N-n)-(R-r)}}\right)\left(\frac{tf_{ij}}{\frac{(k_1 dl_i)}{\Delta} + tf_{ij}}\right)\left(\frac{qtf_j}{k_3 + qtf_j}\right) + \\ &\quad \left((k_2)|Q|\left(\frac{(\Delta - dl_i)}{(\Delta + dl_i)}\right)\right) \end{aligned}$$

where:

| | | |
|---|---|---|
| $N$ | = | number of documents in the collection |
| $n$ | = | number of documents indexed by a given term |
| $R$ | = | number of relevant documents for the query |
| $r$ | = | number of relevant documents indexed by the given term |
| $tf_{ij}$ | = | term frequency of term $j$ in document $i$ |
| $qtf_j$ | = | term frequency of term $j$ in query Q |
| $dl_i$ | = | number of terms in document $i$ |
| $\|Q\|$ | = | number of terms in the query |
| $\Delta$ | = | average document length |
| $k_1, k_2, k_3$ | = | tuning parameters |

Small values for $k_1$ and $k_3$ have the effect of reducing the impact of term frequency and query term frequency. If either is zero, the effect is to eliminate that quantity. Large values of $k_1$ and $k_3$ result in significantly reducing the size of the first term.

Including a factor of $(k_1 + 1)$ and $(k_3 + 1)$ in the numerator does not affect the overall ranking because these factors apply equally to all documents.

However, it does allow for the use of large values of $k_1$ or $k_3$ without reducing the magnitude of the first term. Additionally, this normalizes the impact of the tuning parameters. The idea is that when the term frequency is one, there is no need to change the original weights.

To normalize for document length, the similarity measure also includes a denominator of $\Delta$ in the first term. This makes good sense if the only reason a document is long is because it is simply giving more detail about the topic. In this case, long documents should not be weighted any more than short documents. However, it could be that a document is long because it is discussing several unrelated topics. In this case, long documents should be penalized. A new tuning parameter, $b$, allows for tuning a query based on the nature of the document collection. This parameter is incorporated by substituting $K$ for $k_1$ in the factor involving $tf_{ij}$, where:

$$K = k_1\left((1-b) + b\left(\frac{dl_i}{\Delta}\right)\right)$$

Incorporating the tuning parameter $b$ and placing $(k_1 + 1)$ and $(k_3 + 1)$ in the numerator yields:

$$\begin{aligned} SC(Q, D_i) &= \sum_{j=1}^{t} \log\left(\frac{\frac{r}{(R-r)}}{\frac{(n-r)}{(N-n)-(R-r)}}\right)\left(\frac{(k_1+1)tf_{ij}}{K + tf_{ij}}\right)\left(\frac{(k_3+1)qtf_j}{k_3 + qtf_j}\right) \\ &\quad + \left((k_2)|Q|\frac{\Delta - dl}{\Delta + dl_i}\right) \end{aligned}$$

For the experiments conducted in [Robertson et al., 1995], these values were taken as ($k_1 = 1, k_2 = 0, k_3 = 8, b = 0.6$).

### 2.2.3.1 Example

Using the same documents and query as before, we previously computed $w_4$ as:

gold = -0.477
silver = 0.477
truck = 1.176
$avgdl = \frac{22}{3} = 7.33$

Using the same parameters for $k_1, k_2, k_3, b$, we compute values for $dl_i$:

$dl_1 = 7$
$dl_2 = 8$
$dl_3 = 7$

For $D_1$ the only match with the query is "gold" which appears with a $tf = 1$; so, the SC for $D_1$ is just the value for "gold" (Note the length of $dl$ for $D_1$ is seven).

$$K = 1\left((1-0.6) + \frac{(0.6)(7)}{7.33}\right) = 0.973$$

Now that we have the value of K, it is possible to compute the similarity coefficient for $D_1$. The coefficient is a summation for all terms, but only one term "gold" will have a non-zero value. We start with the value of $w_4 = -0.477$ which was obtained in Section 2.2.1.1.

$$SC(Q,D_1) = -0.477\left(\frac{(1+1)(1)}{0.973+1}\right)\left(\frac{8+1}{8+1}\right) = -0.477\left(\frac{2}{1.973}\right) = -0.484$$

For $D_2$ the terms that match the query "silver" and "truck" result in non-zero values in the summation.

$$K = 1\left(0.4 + \frac{(0.6)(8)}{7.33}\right) = 1.055$$

For "silver", $tf_{12} = 2$:

$$w_4 = 0.477\left(\frac{(1+1)(2)}{1.055+2}\right)\left(\frac{(8+1)(2)}{8+2}\right) = 1.124$$

For "truck", $tf_{22} = 1$:

$$w_4 = 1.176\left(\frac{(1+1)(1)}{1.055+1}\right)\left(\frac{(8+1)(1)}{8+1}\right) = 1.145$$

$$SC(Q,D_2) = 1.124 + 1.145 = 2.269$$

For $D_3$, $dl = 7$ so K = 0.973 (as in the case of $D_1$). We have two terms "gold" and "truck" that both appear once. For "gold", $tf_{13} = 1$.

$$w_4 = -0.477\left(\frac{(1+1)(1)}{0.973+1}\right)\left(\frac{(8+1)(1)}{8+1}\right) = -0.484$$

For "truck", $tf_{23} = 1$.

$$w_4 = 1.176\left(\frac{(1+1)(1)}{0.973+1}\right)\left(\frac{(8+1)(1)}{8+1}\right) = 1.192$$

$$SC(Q,D_3) = -0.484 + 1.192 = 0.708$$

Comparing the SC, with the term frequency, to the base SC, without the term frequency we see in Table 2.7 that again, the document ranking is the same: $D_2, D_3, D_1$.

*Table 2.7.* Poisson Model: Final Similarity Measure

| | No $tf$ | $tf$ |
|---|---|---|
| $D_1$ | -0.477 | -0.484 |
| $D_2$ | 1.653 | 2.269 |
| $D_3$ | 0.699 | 0.708 |

### 2.2.4 Term Components

A variation on the standard probabilistic model is given in [Kwok, 1990]. The premise of the algorithm is to rank documents based on the components of the document. For example, a document can be partitioned into multiple paragraphs, and a similarity coefficient can be computed for each paragraph. Once this is done, a measure is needed to combine the component similarity coefficients to develop a ranking for the entire document. Kwok proposes the use of a geometric mean (for $n$ numbers, the $n^{\underline{th}}$ root of their product) to effectively average the individual components.

The algorithm used to rank a given component certainly can vary, and the size of a component can also vary. If the whole document is used as a component, then we are back at traditional probabilistic information retrieval.

The basic weight for a given component is defined as the ratio of the probability of the component being relevant to the probability that it is not relevant. This is:

$$w_{ak} = \ln\left(\frac{r_{ak}}{(1 - r_{ak})}\right) + \ln\left(\frac{(1 - s_{ak})}{s_{ak}}\right)$$

$r_{ak}$ and $s_{ak}$ are weights that can be estimated in one of three different ways:

- **Initial estimate using self-relevance.** A component which is relevant to itself, results in:

$$r_{ak} = \frac{q_{ak}}{L_a}$$

$$s_{ak} = \frac{F_k}{N_w}$$

where $L_a$ is the number of components in the document, and $F_k$ is the number of occurrences of term $k$ in the collection. $N_w$ is the number of distinct components in the collection.

- **Inverse collection term frequency (ICTF).** The estimate of $s$ above is good because there are probably more non-relevant than relevant documents. Hence, a term that is infrequent in the entire collection has a low value of $s_{ik}$. Assume the initial estimate of $r_{ak}$ is poor and just use a constant $p$. Using $r_{ak}$ as a constant results in the whole weight being roughly equivalent to $s_{ik}$. $s_{ik}$ is estimated by removing the one relevant document, $d_{ik}$ from the estimates that use the whole collection. This is done by using the number of terms, $d_{ik}$, that match the assumed "relevant document." $s_{ik}$ is then computed as:

$$s_{ik} = \left(\frac{F_k - d_{ik}}{N_w - L_i}\right)$$

Using $p$ in our weight computation yields the following weight which is very close to the *idf*.

$$w_{ik} = \ln\left[\frac{p}{1-p}\right] + \ln\left[\frac{1-s_{ik}}{s_{ik}}\right]$$

- Essentially, weights are computed based on the use of feedback from the user. (Use of relevance feedback will be discussed in more detail in Section 3.1). Once the estimates are obtained, all that remains is to combine the component weights—in either query focused means, a document focused measure, or a combined measure. Using the query as focus, the query is given, and all the weights are computed as related to the query. The geometric mean is then computed for each of the components. This reduces to:

$$\sum_{i=1}^{k}\left(\frac{d_{ik}}{L_i}\right) w_{ak}$$

A document focused measure computes the components of the query and then averages them in relation to a given document. This reduces to:

$$\sum_{i=1}^{k}\left(\frac{q_{ik}}{L_i}\right) w_{ik}$$

The combined measure can then be obtained. This combined measure is simply the sum of the query focused and the document focused measures given as:

$$\sum_{i=1}^{k}\left(\frac{d_{ik}}{L_i}\right) w_{ak} + \sum_{i=1}^{k}\left(\frac{q_{ik}}{L_i}\right) w_{ik}$$

The component theory was shown to be comparable to term-based retrieval, and superior for retrieval in the presence of relevance feedback. The combination of query focused and document focused retrieval was almost always shown to be superior than either query focused or document focused retrieval.

## 2.2.5 Key Concerns with Probabilistic Models

Typically, probabilistic models must work around two fundamental problems: parameter estimation and independence assumptions. Parameter estimation refers to the problem that accurate probabilistic computations are based on the need to estimate relevance. Without a good training data set, it is often difficult to accurately estimate parameters. The second problem is the use of independence assumptions. It is clear that the presence of the term *new* increases the likelihood of the presence of the term *york* but many probabilistic models require this assumption even though it is not a realistic assumption.

### 2.2.5.1 Parameter Estimation

The need for good parameter estimation was clearly documented in the 1970's. Initial experiments with simple term weights partitioned the document collection into an *even* and an *odd* component. The even component was used as training data; after relevance information was obtained, it was used to retrieve data in the odd component. For many applications, the *a priori* relevance information is not known. A follow-up paper used reduced relevance information [Sparck Jones, 1979b]. The effect of using only the best one or two most relevant documents as training data, instead of using all relevant documents was measured. For the small Cranfield collection, results from using fewer relevant documents were comparable to using all relevant documents. Unfortunately, when the test was run on the larger UKCIS, the results with only two relevant documents were inferior to results using all relevant documents.

The initial model did not indicate how the process should start. Once relevance information is available (via a training set), it is possible to conduct new searches. In an on-line system where it is not possible to guess which queries will be asked in advance, it is not possible to use the weighting functions given above. They all require values for $r$ and $R$ which can only be obtained by running a query and examining the relevant documents. Certainly, it is possible to use another technique for the initial search, and then ask users for relevance information on the results of the initial search. This information can then be used for subsequent searches. This technique is called *relevance feedback* and is discussed in more detail in Section 3.1.

Using the probabilistic weights as a means of implementing relevance feedback relegates the probabilistic model to an interesting utility that can be applied to an arbitrary retrieval strategy. A cosine measure can be used to identify an initial ranking, and then the probabilistic weights can be used for relevance feedback.

Using the probabilistic model without any *a priori* relevance information creates problems which are addressed in [Croft and Harper, 1979]. In doing so, it becomes clear that the probabilistic model is a retrieval strategy that is capable of ranking documents without any other assistance. The key is that we assume (without any relevance information), the probability that a given term will induce relevance is equal for each term. Thus, the following similarity coefficient is obtained:

$$SC(Q, D_j) = C \sum_{i=1}^{t} q_i d_{ij} + \sum_{i=1}^{t} q_i d_{ij} \log \frac{N - n_i}{n_i}$$

$$\begin{array}{rcl}
N & = & \text{number of documents in the collection} \\
n_i & = & \text{number of documents indexed by term } i \\
d_{ij} & = & 1, \text{ if term } i \text{ appears in document } j \\
d_{ij} & = & 0, \text{ if term } i \text{ does not appear in document } j \\
q_i & = & 1, \text{ if term } i \text{ appears in the query} \\
q_i & = & 0, \text{ if term } i \text{ does not appear in the query}
\end{array}$$

C is a constant that can be varied to "tune" the retrieval. The term weight of $\frac{N-n_i}{n_i}$ is very close to the inverse document frequency of $\frac{N}{n_i}$ for large document collections (large values of N). Hence, the whole expression is very close to the *tf-idf* that was used in the vector space model.

The authors tested this SC against the cosine coefficient and a coefficient obtained by simply summing the *idf's* of each term. The new SC performed slightly better, but it is important to remember that the tests were run on the small Cranfield collection.

Recently, Croft and Harper's work on the problem of computing relevance weights with little or no relevance information was improved [Robertson and Walker, 1997]. They note that the weighting scheme of Croft and Harper can, under some circumstances, lead to negative weights.

In the original model by Robertson and Sparck Jones, two probabilities were used to determine the weighting. The first value, $p$, estimates for a given term the probability that a document containing the term will be relevant. The probability $q$ estimates the probability that a document containing the term will not be relevant. In previous models, $p$ and $q$ are assumed to be constant, but Robertson and Walker allow $p$ to vary as a function of known evidence of relevance.

Specifically, a weighting function that is developed with no information gives an inverse collection frequency weight (or some slight variation). At the other extreme, with a large amount of relevance information, the weighting function is determined by the relevance information. The equation from Croft and Harper can take on negative weights (when a term appears in over half of the document collection). Robertson and Walker developed new equations that are tunable and that estimate the weights of $p$ and $q$ independently. That is, information about relevance only influences the weight due to $p$ and information about non-relevance only influences the weight due to $q$.

The new weight is given by:

$$w = \frac{k_5}{k_5 + R}\left(k_4 + \log\frac{N}{N-n}\right) + \frac{R}{k_5 + R}\log\left(\frac{r + 0.5}{R - r + 0.5}\right)$$
$$-\frac{k_6}{k_6 + S}\log\left(\frac{n}{N-n}\right) - \frac{S}{k_6 + S}\log\left(\frac{(s + 0.5)}{(S - s + 0.5)}\right)$$

where

| | | |
|---|---|---|
| $R$ | = | number of relevant documents |
| $r$ | = | number of relevant documents indexed by the given term |
| $S$ | = | number of non-relevant documents |
| $s$ | = | number of non-relevant documents which contain the term |
| $k_4, k_5, k_6$ | = | tuning constants, where $k_4 \geq 0$ |

The first two terms give the component of the weight due to relevance information, and the last two terms give the weight due to non-relevance information. (Note that if there is no knowledge (R=S=0), then the equations reduce to $k_4 + \log\frac{N}{n}$). $k_4$ measures how good the query term should be, while $k_5$ and $k_6$ measure the sensitivity to relevance and non-relevance, respectively. A statistically-based argument can be made that, instead of using R and S to scale the terms in the equation, the square roots of R and S should be used.

#### 2.2.5.2 Independence Assumptions

The key assumption that provides for a simple combination of term weights to compute the probability of relevance is the assumption that the terms appear independent of one another. Because this assumption is false, it was suggested that the entire model is derived from a "faulty theory" [Cooper, 1991]. In fact, the inference network strategy and the logistic regression utility are both designed to work around the problem of independence assumptions. These are discussed in Sections 2.4 and 3.5, respectively.

Papers in the late 1970's and early 1980's start to address the failure of the independence assumption [Rijsbergen, 1977, Yu et al., 1983], but they all require co-occurrence information which is very computationally expensive to obtain. Van Rijsbergen suggests that related terms should be grouped together by using simple clustering algorithms and then the dependencies between groups can be obtained [Rijsbergen, 1977].

With increased computational speed, these approaches may soon be more tractable. To our knowledge, none of these modifications have been tried on a large test collection.

## 2.3 Language Models

A statistical language model is a probabilistic mechanism for "generating" a piece of text. It thus defines a distribution over all the possible word sequences. The simplest language model is the unigram language model, which is essentially a word distribution. More complex language models might use more context information (e.g., word history) in predicting the next word [Charniak, 1993, Rosenfeld, 2000].

Despite more than twenty years of using language models for speech recognition [Hodjat et al., 2003] and language translation, their use for information retrieval started only in 1998 [Ponte and Croft, 1998]. The core idea is that documents can be ranked on their likelihood of *generating* the query. Consider spoken document recognition, if the speaker were to utter the words in a document, what is the likelihood they would then say the words in the query. Formally, the similarity coefficient is simply:

$$SC(Q, D_i) = P(Q|M_{D_i})$$

where $M_{D_i}$ is the language model implicit in document $D_i$.

There is a need to precisely define what we mean exactly by "generating" a query. That is, we need a probabilistic model for queries. One approach (proposed in [Ponte and Croft, 1998]) is to model the presence or absence of any term as an independent Bernoulli event and view the generation of the whole query as a joint event of observing all the query terms and not observing any terms that are not present in the query. In this case, the probability of the query is calculated as the product of probabilities for both the terms in the query and terms absent. That is,

$$SC(Q, D_i) = \prod_{t_j \in Q} P(t_j|M_{D_i}) \prod_{t_j \notin Q} (1 - P(t_j|M_{D_i}))$$

The model $p(t_j|M_{D_i})$ can be estimated in many different ways. A straightforward method is:

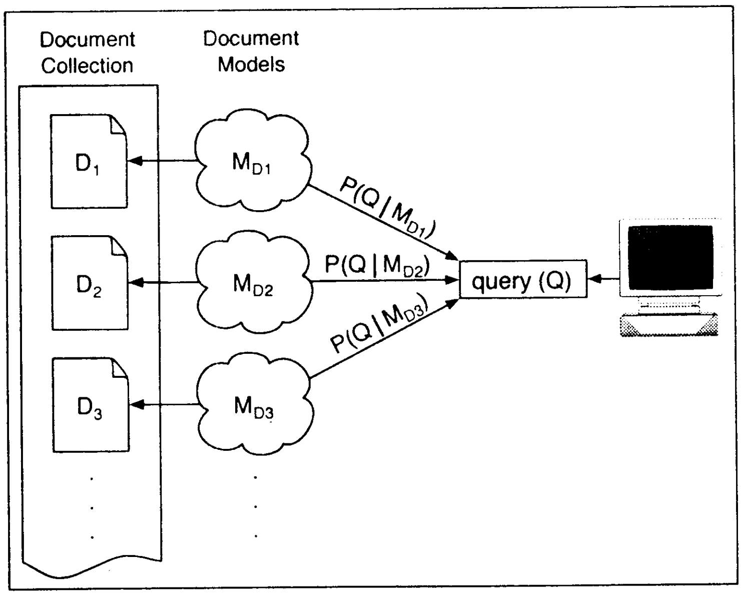
$$p(t_j|M_{D_i}) = p_{ml}(t_j|M_{D_i})$$

where $p_{ml}(t_j|M_{D_i})$ is the maximum likelihood estimate of the term distribution (i.e., the relative term frequency), and is given by:

$$p_{ml}(t_j|M_{D_i}) = \frac{tf(t_j, D_i)}{dl_{D_i}}$$

where $dl_{D_i}$ is the document length of document $D_i$.

*Figure 2.5.* Language Model



The basic idea is illustrated in Figure 2.5. The similarity measure will work, but it has a big problem. If a term in the query does not occur in a document, the whole similarity measure becomes zero. Consider our small running example of a query and three documents:

$Q$ : "gold silver truck"
$D_1$: "Shipment of gold damaged in a fire"
$D_2$: "Delivery of silver arrived in a silver truck"
$D_3$: "Shipment of gold arrived in a truck"

The term *silver* does not appear in document $D_1$. Likewise, *silver* does not appear in document $D_3$ and *gold* does not appear in document $D_2$. Hence, this would result in a similarity coefficient of zero for all three sample documents and this sample query.

Hence, the maximum likelihood estimate for

$$p_{ml}(silver|M_{D_i}) = \frac{tf(silver, D_i)}{dl_{D_i}} = 0$$

### 2.3.1 Smoothing

To avoid the problem caused by terms in the query that are not present in a document, various *smoothing* approaches exist which estimate non-zero values for these terms. One approach assumes that the query term could occur in this model, but simply at no higher a rate than the chance of it occurring in any other document. The ratio $\frac{cf_t}{cs}$ was initially proposed where $cf_t$ is the number of occurrences of term $t$ in the collection, and $cs$ is the number of terms in the entire collection. In our example, the estimate for *silver* would be $\frac{2}{22} = .091$.

An additional adjustment is made to account for the reality that these document models are based solely on individual documents. These are relatively small sample sizes from which to build a model. To use a larger sample (the entire collection) the following estimate is proposed:

$$p_{avg}(t) = \frac{\sum_{d(t \in d)} p_{ml}(t|M_d)}{df_t}$$

where $df_t$ is the document frequency of term $t$, which is also used in computing the idf as discussed in Section 2.1).

To improve the effectiveness of the estimates for term weights it is possible to minimize the risk involved in our estimate. We first define $\bar{f}_t$ as the mean term frequency of term $t$ in the document. This can be computed as $\bar{f}_t = p_{avg}(t) \times dl_d$. The risk can be obtained using a geometric distribution as:

$$R_{t,d} = \left(\frac{1.0}{1.0 + \bar{f}_t}\right) \times \left(\frac{\bar{f}_t}{1.0 + \bar{f}_t}\right)^{tf_{t,d}}$$

The first similarity measure described for using language models in information retrieval uses the smoothing ratio $\frac{cf_t}{cs}$ for terms that do not occur in the query and the risk function as a mixing parameter when estimating the values for $w$ based on small document models. The term weight is now estimated as:

$$P(t|M_{di}) = \begin{cases} p_{ml}(t,d)^{(1-R(t,d))} \times p_{avg(t)}^{R(t,d)} & \text{if } tf(t,d) > 0 \\ \frac{cf_t}{cs} & \text{otherwise.} \end{cases}$$

## 2.3.2 Language Model Example

We now illustrate the use of this similarity measure with our running example: There are three documents in our test collection. There are 22 tokens. So, $cs$=22. The total number of tokens in documents $D_1$, $D_2$ and $D_3$ are 7, 8, and 7, respectively. Table 2.8 contains values for $dl_d$.

*Table 2.8.* Term Occurrence

| | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| $dl_d$ | 7 | 8 | 7 |

$df_t$, the document frequency of $t$, is listed in Table 2.9. We use the same term ordering as described in Section 2.1.1.

*Table 2.9.* Document Frequency

| | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $df_t$ | 3 | 2 | 1 | 1 | 1 | 2 | 3 | 3 | 2 | 1 | 2 |

$cf_t$, the raw count of token $t$ in the collection is given in Table 2.10.

*Table 2.10.* Collection Frequency for Each Token

| | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $cf_t$ | 3 | 2 | 1 | 1 | 1 | 2 | 3 | 3 | 2 | 2 | 2 |

$tf_{t,d}$, the raw term frequency of term $t$ in document $d$ is given in Table 2.11. First, we need to calculate $p_{ml}(t|M_d)$, the maximum likelihood estimate of the probability of term $t$ under the term distribution for document $d$. For each term, $t$, $P_{ml}(t|M_d) = \frac{tf_{t,d}}{dl_d}$ is given in Table 2.12.

*Table 2.11.* Raw Term Frequency

| | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_1$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| $D_2$ | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 2 | 1 |
| $D_3$ | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

*Table 2.12.* Maximum Likelihood for Each Term

| $p_{ml}(t\|M_d)$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| a | 0.143 | 0.125 | 0.143 |
| arrived | 0 | 0.125 | 0.143 |
| damaged | 0.143 | 0 | 0 |
| delivery | 0 | 0.125 | 0 |
| fire | 0.143 | 0 | 0 |
| gold | 0.143 | 0 | 0.143 |
| in | 0.143 | 0.125 | 0.143 |
| of | 0.143 | 0.125 | 0.143 |
| shipment | 0.143 | 0 | 0.143 |
| silver | 0 | 0.250 | 0 |
| truck | 0 | 0.125 | 0.143 |

Second, we calculate the mean probability of term $t$ in documents which contain the term. The equation is:

$$P_{avg}(t) = \frac{\sum_{d(t \in d)} P_{ml}(t|M_d)}{df_t}$$

For the term "arrived", it only appears in $D_2$ and $D_3$, so:

$$P_{avg}(arrived) = \frac{P_{ml}(arrived|M_{D_2}) + P_{ml}(arrived|M_{D_3})}{df_{arrived}}$$

Using our previous estimates: we know that $P_{ml}(arrived|M_{D_2}) = 0.125$, $P_{ml}(arrived|M_{D_3}) = 0.143$ and $df_{arrived} = 2$. Thus, $P_{avg}(arrived) = (0.125 + 0.143)/2 = 0.134$. The remaining terms are given in Table 2.13.

Third, we calculate the risk for a term $t$ in a document $d$. To do that, $\bar{f}_t$, the mean term frequency of term $t$ in a document is computed by the following equation $\bar{f}_t = P_{avg}(t) \times dl_d$. $\bar{f}_t$ of each term $t$ is given in Table 2.14. We then use the following risk function to obtain Equation 2.3.

*Table 2.13.* Average Probability for Each Term

| | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $P_{avg}(t)$ | 0.137 | 0.134 | 0.143 | 0.125 | 0.143 | 0.143 | 0.137 | 0.137 | 0.143 | 0.250 | 0.134 |

*Table 2.14.* Mean Term Frequency for Each Term

| $f_t$ | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_1$ | 0.958 | 0.938 | 1.000 | 0.875 | 1.000 | 1.000 | 0.958 | 0.958 | 1.000 | 1.750 | 0.938 |
| $D_2$ | 1.096 | 1.071 | 1.143 | 1.000 | 1.143 | 1.143 | 1.096 | 1.096 | 1.143 | 2.000 | 1.071 |
| $D_3$ | 0.958 | 0.938 | 1.000 | 0.875 | 1.000 | 1.000 | 0.958 | 0.958 | 1.000 | 1.750 | 0.938 |

$$R_{t,d} = \left(\frac{1.0}{1.0 + \bar{f}_t}\right) \times \left(\frac{\bar{f}_t}{1.0 + \bar{f}_t}\right)^{tf_{t,d}} \tag{2.3}$$

The risk values per term per document are shown in Table 2.15.

*Table 2.15.* Risk for Each Term

| $R_{t,d}$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| a | 0.250 | 0.249 | 0.250 |
| arrived | 0.516 | 0.250 | 0.250 |
| damaged | 0.250 | 0.467 | 0.500 |
| delivery | 0.533 | 0.250 | 0.533 |
| fire | 0.250 | 0.467 | 0.500 |
| gold | 0.250 | 0.467 | 0.250 |
| in | 0.250 | 0.249 | 0.250 |
| of | 0.250 | 0.249 | 0.250 |
| shipment | 0.250 | 0.467 | 0.250 |
| silver | 0.364 | 0.148 | 0.364 |
| truck | 0.516 | 0.249 | 0.250 |

Now, we use the risk value as a mixing parameter to calculate $P(Q|M_d)$, the probability of producing the query for a given document model as mentioned before. It consists of two steps. Initially, we calculate $P(t|M_d)$ as shown in Equation 2.4. The italicized terms in Table 2.16 match a query term. For all other term occurrences the smoothing estimate, $P(t|M_d) = \frac{cf_t}{cs}$, is used.

$$P(t|M_d) = P_{ml}(t|M_d)^{(1.0-R_{t,d})} \times P_{avg}(t)^{R_{t,d}} \tag{2.4}$$

Finally, using the equation 2.5, we compute a final measure of similarity, $P(Q|M_d)$. The actual values are given in Table 2.17.

*Table 2.16.* Expected Probability for Each Term (with Smoothing)

| $P(t\|M_d)$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| a | 0.141 | 0.128 | 0.141 |
| arrived | 0.091 | 0.127 | 0.141 |
| damaged | 0.143 | 0.045 | 0.045 |
| delivery | 0.045 | 0.125 | 0.045 |
| fire | 0.143 | 0.045 | 0.045 |
| *gold* | ***0.143*** | 0.091 | ***0.143*** |
| in | 0.141 | 0.128 | 0.141 |
| of | 0.141 | 0.128 | 0.141 |
| shipment | 0.143 | 0.091 | 0.143 |
| *silver* | 0.091 | ***0.250*** | 0.091 |
| *truck* | 0.091 | ***0.127*** | ***0.141*** |

$$P(Q|M_d) = \prod_{t \in Q} P(t|M_d) \times \prod_{t \notin Q} (1.0 - P(t|M_d)) \tag{2.5}$$

*Table 2.17.* Similarity using Language Models

| | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| $P(Q\|M_d)$ | 0.000409 | 0.001211 | 0.000743 |

Hence, as with all other strategies that we describe, the final ranking is $D_2$, $D_3$ and $D_1$.

Although other terms in the query match document $D_1$ (e.g.; *gold*), the similarity measure results in zero for this document simply because one term does not occur. The model we have just presented is based on Ponte and Croft's original model, which models term presence and absence in the query but ignores *query term frequency* or the number of occurrences of a term in the query. Hence it does not matter if the term "silver" occurs once or twice in the query. Other language modeling work, including early work in TREC-7, models the occurrence of every query term with a unigram language model and thus incorporates term frequencies [Miller et al., 1999, Hiemstra and Kraaij, 1998].

Specifically, for a query, $Q = q_1, q_2, \ldots, q_m$, the probability $p(Q|D)$ now incorporates $tf(t, Q)$ to include the query term frequency (as shown in Equation 2.6).

$$p(Q|D) = \prod_{i=1}^{m} p(q_i|M_D) = \prod_{t \in Q} p(t|M_D)^{tf(t,Q)} \tag{2.6}$$

Such a multinomial model is more similar to the language models used in speech recognition. A discussion of this distinction can be found in [McCallum and Nigam, 1998, Song and Croft, 1999]. When using multinomial models, we reduce the retrieval problem to one of language model estimation, i.e., estimating $p(w|M_D)$.

Numerous smoothing functions are used to compute this estimate and detailed surveys are found in [Chen and Goodman, 1998, Manning and Schutze, 1999]. The Good-Turing estimate adjusts raw term frequency scores with the transformation given in equation 2.7:

$$tf^* = (tf + 1)\frac{E(N_{tf+1})}{E(N_{tf})} \tag{2.7}$$

This estimate was used to improve initial language models [Song and Croft, 1999]. Unfortunately, this estimate requires the count of words which have the same frequency in a document. This is a computationally expensive task.

A study of three efficient smoothing methods is given in [Zhai and Lafferty, 2001b]. The three methods were Jelinek-Mercer [Jelinek and Mercer, 1980], Bayesian smoothing using Dirichlet priors [MacKay and Peto, 1995], and Absolute Discounting [Ney et al., 1994]. For long queries, on average, the Jelinek-Mercer smoothing approach is better than the Dirichlet prior and absolute discounting approaches. For title only queries, experimentation demonstrated that Dirichlet prior is superior to absolute discounting and Jelinek-Mercer.

A general description of smoothing methods is to define them in terms of their ability to compute the probability of a term given the presence of a document: $P(t|d)$. Smoothing methods tend to reduce probabilities of terms that are observed in the text and boost the probabilities of terms that are not seen. As a last resort, unseen terms can simply be assigned a probability proportional to their probabilities according to the collection language model $p(t|C)$.

The Jelinek-Mercer method uses a linear interpolation of the maximum likelihood model and the collection model. The parameter $\lambda$ is used to control the influence of each model. More formally:

$$p_\lambda(t|d) = (1 - \lambda)p_{ml}(t|d) + \lambda P(w|C)$$

Absolute discounting simply lowers the probability of terms that have occurred by subtracting a simple constant from their frequency. More formally:

$$p_\delta(t|d) = \frac{max(tf(t,d) - \delta, 0)}{\sum_t tf(t,d)} + \sigma P(t|C)$$

where $\delta \in [0, 1]$ is a constant and

$$\sigma = \delta \frac{|d|_u}{|d|}$$

where $|d|_u$ is the number of unique terms in document $d$ and $|d|$ is the number of terms in the document.

For long queries, on average, the Jelinek-Mercer smoothing approach is better than the Dirichlet prior and absolute discounting approaches. For title only queries, experimentation demonstrated that Dirichlet prior is superior to absolute discounting and Jelinek-Mercer. In the Bayesian smoothing using Dirichlet priors, the model is given in Equation 2.8.

Returning now to our example, since our sample query contains only three terms, we treat it as a title query. Thus, in the following, we give a brief example illustrating the computation of relevance using the Dirichlet prior smoothing method.

$$p_\mu(t|d) = \frac{tf(t,d) + \mu P(t|C)}{\sum_t tf(t,d) + \mu} \tag{2.8}$$

where $tf(t,d)$ is the number of occurrences of term $t$ in document $d$. $\sum_t tf(t,d)$ is the total count of terms in document $d$. $P(t|C)$ is the collection language model and is calculated as the number of occurrence of term $t$ in the collection C divided by the total number of terms in C, namely, $\frac{\sum_d tf(t,d)}{cs}$.

Smoothing for terms that do not occur in the collection can be done with a Laplace method [Katz, 1987]. Hence, we have $P(t|C) = \frac{1+\sum_d tf(t,d)}{N+cs}$, where $N$ is the number of distinct terms in the collection. In our example, since all query terms appear in the collection, we simply use $\frac{\sum_d tf(t,d)}{cs}$. In general, smoothing of the collection language model has only an insignificant effect as $cs$ is typically quite large.

Finally, $\mu$ is an adjustable smoothing parameter. Experimentation has shown that the optimal prior seems to vary from collection to collection. However, it is frequently around 2000. Due to our desire to make the example readable, we will use a value of three in our example (this reduces the number of significant digits needed to present a useful example on this small three document test collection).

First, for each term $t$, we compute $\sum_d tf(t,d)$. The values are given in Table 2.18. In our sample collection, there are 3 documents. Totally, there are 22 tokens. So, $cs = 22$. By using $\frac{\sum_d tf(t,d)}{cs}$, we obtain $P(t|C)$, as shown in Table 2.19. Second, the total count of terms in document $d$ is, once again, given in Table 2.20. Third, $tf(t,d)$ is, once again, given in Table 2.21. We now compute the smoothed probabilities by using $p_\mu(t|d) = \frac{tf(t,d)+\mu P(t|C)}{\sum_t tf(t,d)+\mu}$. The new probabilities are shown in Table 2.22. A value of 3 is used for $\mu$ to reduce the number of significant digits needed for this example. A new similarity measure is computed as $P(Q|d) = \prod_t p_\mu(t|d)$, and the results are shown in Table 2.23. For example the similarity between document one and the query can now simply be computed as the product of the smoothed probabilities for the three terms in the query. For document one this is $(0.1378)(0.0091)(0.0091) = 0.0000114$.

*Table 2.18.* $\sum_d tf(t,d)$ for Each Term

| | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\sum_t tf(t,d)$ | 3 | 2 | 1 | 1 | 1 | 2 | 3 | 3 | 2 | 2 | 2 |

*Table 2.19.* $P(t|C)$ for Each Term

| | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| P(t\|C) | .136 | .091 | .045 | .045 | .045 | .091 | .136 | .136 | .091 | .091 | .091 |

*Table 2.20.* $\sum_t tf(t,d)$ for Each Document

| | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| $\sum_t tf(t,d)$ | 7 | 8 | 7 |

Hence, the ranking with this measure is $D_3$, $D_2$, and $D_1$. We now provide a brief description of two additional smoothing methods. The Jelinek-Mercer method is a linear interpolation of the maximum likelihood model with the collection model, using a coefficient $\lambda$ to control the influence of each model.

$$P_\lambda(t|d) = (1-\lambda)P_{ml}(t|d) + \lambda P(t|C).$$

*Table 2.21.* $tf(t,d)$ for Each Term

| $tf(t,d)$ | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| D1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| D2 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 2 | 1 |
| D3 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

*Table 2.22.* $p_\mu(t|d)$ for Each Term

| $p_\mu(t\|d)$ | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| D1 | 0.141 | 0.027 | 0.114 | 0.014 | 0.114 | 0.127 | 0.141 | 0.141 | 0.127 | 0.027 | 0.027 |
| D2 | 0.128 | 0.116 | 0.012 | 0.103 | 0.012 | 0.025 | 0.128 | 0.128 | 0.025 | 0.207 | 0.116 |
| D3 | 0.141 | 0.127 | 0.014 | 0.014 | 0.014 | 0.127 | 0.141 | 0.141 | 0.127 | 0.027 | 0.127 |

*Table 2.23.* Final Similarity Measure with Dirichlet Priors

| Document | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| SC(Q, $D_i$) | 0.0000114 | 0.0002590 | 0.0001728 |

P($t$|C) is the same as given using Dirichlet priors. For $\lambda$, we choose different optimal values for different queries. Experiments have shown that a small value of $\lambda$, around 0.1, works well for short queries and a higher value around 0.7 for long queries. The final similarity measure for our example is given in Table 2.24.

Note, as with all our ranking examples, document two is ranked higher than documents three which is ranked higher than document one. The absolute discounting method lowers the probability of seen words by subtracting a constant from their counts. The following equation shows how the discount is incorporated:

$$p_\delta(t|d) = \frac{max((tf(t,d) - \delta), 0)}{\sum_w tf(t,d)} + \sigma P(t|C)$$

where $\delta \in [0,1]$ and $\sigma = \frac{\delta |d|_u}{|d|}$. $|d|_u$ is the count of unique terms in document d and $|d|$ is the total number of terms in the document. A key difference from the Jelinek-Mercer smoothing is that the optimal value of $\delta$ is frequently around 0.7 according to experimental results. This is true for title queries as well as long queries. The final similarity coefficient for our example with absolute discounting is given in Table 2.25.

We briefly described the use of language models as a recent type of search strategy. Although it is an old technique from speech recognition, it was only recently applied to information retrieval. Our discussion is mostly based on Ponte and Croft's pioneering work, which models term presence or absence in the query. We also discussed an alternative multinomial model which treats a query as a sample drawn from a unigram language model (i.e., a multinomial word distribution). In each case, we show how to score a document based on the likelihood of generating the query using a model estimated from the document. The retrieval problem is reduced to the problem of estimating a document language model. A straightforward method of estimating the model using relative frequency has the obvious problem of assigning a zero probability to any word not observed in the document. Smoothing adjusts the estimate to avoid such a situation. This improves the accuracy of the estimated model. Many different smoothing methods are possible. The Dirichlet prior method has thus far performed well.

*Table 2.24.* Final Similarity Measure with Jelinek-Mercer

| | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| SC(Q, $D_i$) | 0.000314 | 0.000443 | 0.000381 |

Retrieval with these basic language models can be as efficient as retrieval using a traditional model such as the vector space model [Zhai and Lafferty, 2001b], and likewise they were shown to be as effective as, or more effective than, well-tuned traditional models. Perhaps the main advantage of using language models for retrieval lies in the potential for automatic tuning of parameters, which is demonstrated in [Zhai and Lafferty, 2002].

*Table 2.25.* Final Similarity Measure with Absolute Discounting

| | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| SC(Q, $D_i$) | 0.001215 | 0.021716 | 0.005727 |

Many more complex language models were studied. Some of them attempt to improve the basic language modeling approach (i.e., the query likelihood approach) by going beyond the unigram language models by capturing limited dependencies [Song and Croft, 1999], and by introducing translation models to capture word relatedness [Berger and Lafferty, 1999]. Some others use language models in a different way. For example, in [Lavrenko and Croft, 2001], language models are incorporated into the classic probabilistic information retrieval model, and a novel method is used for estimating the relevance model without relevance judgments.

Recent work focuses on reducing the size of these models [Hiemstra et al., 2004]. Another recent approach builds language models based on clusters (see Section 3.2) of documents instead of the entire document collection [Liu and Croft, 2004].

The basic idea is very similar to pseudorelevance feedback. In other work [Zhai and Lafferty, 2001b, Zhai and Lafferty, 2001a], both a query model and a document model are estimated and a model distance function (i.e., Kullback-Leibler divergence) is used to compute the distance between the two models. This is very similar to the vector space model, except that the representation of documents and the query are based on unigram language models. The main advantage of these alternatives and more sophisticated models is that they can handle relevance feedback (see Section 3.1) more naturally. Most of these alternative models were shown to outperform the simple basic query likelihood scoring method. Development of more accurate and more robust language models remains an active research area.

## 2.4 Inference Networks

Inference networks use evidential reasoning to estimate the probability that a document will be relevant to a query. They model the probabilistic retrieval strategy discussed in Section 2.2 and enhance that model to include additional evidence that a document may be relevant to a query. In this section, we first give a basic overview of inference networks. We then describe how inference networks are used for relevance ranking.

### 2.4.1 Background

The essence of an inference network is to take known relationships and use them to "infer" other relationships. This dramatically reduces the computational requirements needed to estimate the probability that an event will occur.

A binary inference network uses events where the event will have either a value of *true* or *false*. A prior probability indicates the likelihood of the event. Assume we know events A, B, C, D and E all occur with respective probabilities $P(A = true) = a$, $P(B = true) = b$, $P(C = true) = c$, $P(D = true) = d$ and $P(E = true) = e$. These events are independent—that is, the probability that all events will still occur is the same regardless of whether or not any of the other events occur. More formally, for all possible combinations of $b, c, d$ and $e$ then $P(A|b, c, d, e) = P(a)$. Assume we know that event F depends on events A, B, C, D and E, and we want to compute the probability that F occurs given the probability that A, B, C, D and E occur. Figure 2.6 illustrates this example inference network.

To do this without an inference network, the computation is exponential and requires consideration of all $2^5$ combinations for the events A, B, C, D and

E. Using notation given in [Greiff, 1996], let R be the set of all $2^5$ possible subsets of 1, 2, 3, 4, 5. Let $p_i$ indicate the probability that the $i^{\underline{th}}$ event is true—in our example events 1, 2, 3, 4 and 5 correspond to A, B, C, D and E. Let $P_i$ indicate the state value (either true or false) of the $i$th event—that is $P_1$ indicates whether or not A is true, $P_2$ indicates whether or not B is true, etc. Finally, the mere existence of an event or combination of events A, B, C, D or E changes the likelihood that F is true. This probability is called $\alpha_R$ and is defined as:

$$\begin{aligned}\alpha_R &= P(F = \mathtt{true}|P_1, \ldots, P_5)\\ &\quad \mathtt{where}\ i \in R \rightarrow P_i\ \mathtt{is\ true},\\ &\quad i \notin R \rightarrow P_i\ \mathtt{is\ false}\end{aligned}$$

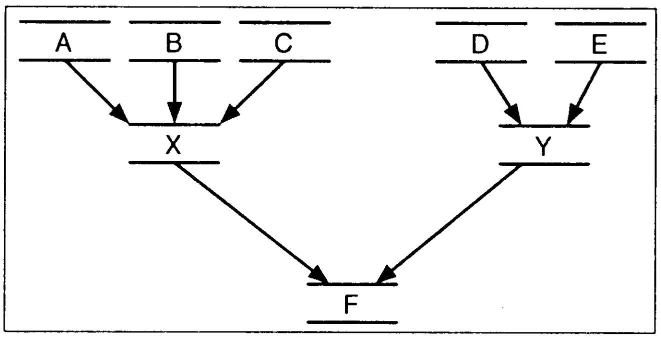
To compute $P(F = true)$, assuming, A, B, C, D and E are independent, the following equation is used:

$$P(F = true) = \sum_{R\subseteq\{1,\ldots,5\}} \alpha_R \prod_{i\in R} p_i \prod_{i\notin R}(1 - p_i) \tag{2.9}$$

For a simple problem with only five values, we end up with a 32-element computation. The exponential nature of this problem is addressed by inference networks with naturally occurring intermediaries. This enables the use of partial inferences to obtain a final inference.

Assume we know that events A, B, and C cause event X, and events D and E cause event Y. We can now use X and Y to infer F. Figure 2.6 illustrates this simple inference network .

*Figure 2.6.* Simple Inference Network



Consider an example where we are trying to predict whether or not the Salamanders will win a softball game. Assume this depends on which coaches are present and which umpires are present. At the top layer of the network might be nodes that correspond to a given coach or umpire being present. The Scott, David, and Catherine nodes (nodes A, B,and C) all correspond to coaches and the Jim and Manny nodes (nodes D and E) correspond to umpires. Now the event "good coach is present" (node X) depends on the Scott, David, and Catherine nodes and the event "good umpire is present" (node Y) depends on the Jim and Manny nodes. The event "Salamanders win" (node F) clearly depends on nodes X and Y.

In our example, the presence of an umpire in no way determines whether or not another umpire attends, but it certainly impacts whether or not the umpires are "friendly" to the Salamanders. Similarly, the presence of a given coach does not impact the presence of other coaches, but it does impact whether or not the whole coaching staff is present. Also, the presence or absence of a coach has no impact on whether or not the umpires will be favorable.

To compute F, equation 2.9 can be used, or we can use an inference network to take advantage of the logical groupings inherent in the events X and Y. First, we compute $P(X = true)$ using the three parents— A, B, and C — this requires $2^3$ computations. The impact of the parent nodes on the child node with the variable $\alpha$ and a binary subscript that indicates the likelihood that the child node is true, given various combinations of parent nodes being true. For example, $\alpha_{111}$ indicates the probability that the child node is true given that all three parents are true. Computing $P(X = true)$ we obtain:

$$\begin{aligned} P(X = true) \; = \; & \alpha_{111}abc + \alpha_{110}ab(1-c) + \alpha_{101}a(1-b)c + \\ & \alpha_{100}a(1-b)(1-c) + \alpha_{011}(1-a)bc + \\ & \alpha_{010}(1-a)b(1-c) + \alpha_{001}(1-a)(1-b)c + \\ & \alpha_{000}(1-a)(1-b)(1-c) \end{aligned}$$

and now we compute $P(Y = true)$ using the two parents D, E:

$$P(Y = true) = \alpha_{11}de + \alpha_{10}d(1-e) + \alpha_{01}(1-d)e + \alpha_{00}(1-d)(1-e)$$

Once the prior probabilities for X and Y are known, we compute the probability that F is true as:

$$P(F = true) = \alpha_{11}xy + \alpha_{10}x(1-y) + \alpha_{01}y(1-x) + \alpha_{00}(1-x)(1-y)$$

To compute F, it took eight additions for X, four for Y, and finally four for F. Therefore, we now require only sixteen instead of the thirty-two required without the inference network. The key is that F is independent of A, B, C, D and E given X, Y or:

$$P(F = true|a, b, c, d, e, x, y) = P(F = true|x, y)$$

Once the initial values of the top layer of the inference network are assigned (these are referred to as prior probabilities), a node on the network is *instantiated* and all of its links are followed to nodes farther down the network. These nodes are then activated. At this point, the node that is activated is able to compute the *belief* that the node is true or false. The belief is computed based on the belief that all of its parent nodes are true or false. At this point, we have assumed that all parents contributed equally to the belief that a node was present. Link matrices indicate the strength by which parents (either by themselves or in conjunction with other parents) affect children in the inference network.

## 2.4.2 Link Matrices

Another capability provided by the inference network is the ability to include the dependence of a child on the parents. Suppose we know that a particular umpire, Manny, is much more friendly to the Salamanders than any other umpire. Hence, the contribution of D, E to the value of Y may not be equal.

The link matrix contains an entry for each of the $2^n$ combinations of parents and (for a binary inference network in which nodes are either true or false) will contain only two rows.

In our example, the link matrix for the node Y that represents the impact of umpires D and E on Y is given as:

| | DE | $D\overline{E}$ | $\overline{D}E$ | $\overline{DE}$ |
|---|---|---|---|---|
| Y true | 0.9 | 0.8 | 0.2 | 0.05 |
| Y false | 0.1 | 0.2 | 0.8 | 0.95 |

This matrix indicates that the presence of the friendly umpire Manny (D) coupled with the absence of Jim ($\overline{E}$) results in an eighty percent contribution to the belief that we have friendly umpires for the game. We use the notation $L_i(Y)$ to indicate the value of the $i^{th}$ entry in the link matrix to identify whether or not Y is true and $\overline{L_i(Y)}$ to indicate the value to determine whether or not Y is false.

The link matrix entries are included as an additional element in the computation given above. The new equation to compute the belief that a given node N with $n$ parents using the previously used set R becomes:

$$P(N = true) = \sum_{R \subseteq \{1,...,n\}} L_i(N) \prod_{i \in R} p_i \prod_{i \notin R} (1 - p_i)$$

The link matrix for $p$ parents contains $2^p$ entries. Again, the link matrix measures the contribution of individual parents to a given inference. The link matrix can be selected such that a closed form computation is possible. The simplest matrix, the $L_{AND}$ for an arbitrary node N is of the form [Turtle, 1991]:

| | $P_{000}$ | $P_{001}$ | $P_{010}$ | $P_{011}$ | $P_{100}$ | $P_{101}$ | $P_{110}$ | $P_{111}$ |
|---|---|---|---|---|---|---|---|---|
| N true | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

The entries are given in binary such that a 1 or 0 is given for each of the possible $p$ parents (p=3 in this case). The computation results in zeros for all combinations of parents except when all $p$ parents exist. The value of P(N=true) will be $(p_1)(p_2)(p_3)$ where $p_i$ indicates the probability that the parent $p_i$ is true. Only a single element must be summed to obtain the final result instead of the worst case of $2^N$. Other closed form link matrices exist that essentially average the prior probabilities of parents.

To give an example with our existing inference network, assume there is a seventy percent chance A will attend, and a sixty percent chance B and C will attend (P(A) = 0.7, P(B) = 0.6, P(C) = 0.6). For umpires, assume P(D) = 0.8 and P(E) = 0.4. The links from A, B, C, D and E are followed and the probability that X and Y are true can now be computed. To compute $P(X = true)$, we need the link matrix for X. Let's assume the link matrix results in a closed form average of all parent probabilities

$$\frac{\sum_i^n p_i}{n}$$

$$P(X = true) = \frac{0.7 + 0.6 + 0.6}{3} = 0.633$$

Now to compute the probability for Y, the link matrix given above is used to obtain:

$$\begin{aligned} P(Y = true) &= L_{11}(Y)de + L_{10}(Y)d(1-e) + L_{01}(Y)(1-d)e + \\ &\quad L_{00}(Y)(1-d)(1-e) \end{aligned}$$

$$\begin{aligned} P(Y = true) &= (0.9)(0.8)(0.4) + (0.8)(0.8)(0.6) + (0.2)(0.2)(0.4) + \\ &\quad (0.05)(0.2)(0.6) \end{aligned}$$

$$P(Y = true) = 0.288 + 0.384 + 0.016 + 0.006 = 0.694$$

Now we have the belief that X and Y are true, assume we use the unweighted sum link matrix. This is a closed form link matrix that results in a simple average of the parent probabilities to compute the belief in F. The final value for F is:

$$P(F = true) = \frac{0.694 + 0.633}{2} = 0.6635$$

In this case, we had only three elements to sum to compute X, four to compute Y, and two to compute F. If we did not have a closed form link matrix we would have had $2^3$ for X, $2^2$ for Y, and $2^2$ for F or 16 elements—substantially less than the $2^5$ required without an inference network.

### 2.4.3 Relevance Ranking

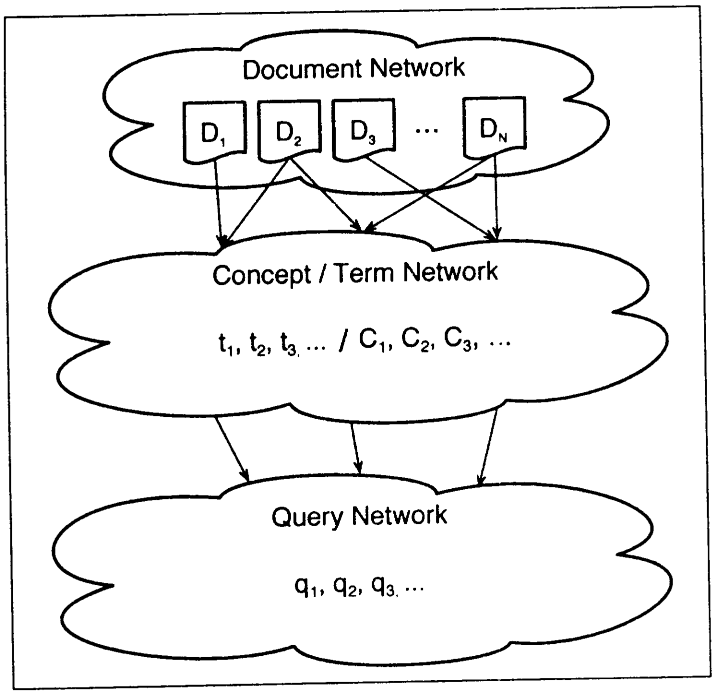
Turtle's Ph.D. thesis is the seminal work on the use of inference networks for information retrieval [Turtle, 1991]. The documents are represented as nodes on the inference network , and a link exists from each document to each term in the document. When a document is instantiated, all of the term nodes linked to the document are instantiated. A simple three-layered approach then connects the term nodes directly to query nodes. This three-layered network is illustrated in Figure 2.7. A link exists between a term node and a query node for each term in the query. Note that this is the most simplistic form of inference network for information retrieval. The three-layered approach consists of a document layer, a term layer, and a query layer. Note that the basic algorithm will work if a layer that contains generalizations of terms or *concepts* exists. This layer could sit between the term layer and the query layer. Links from a term to a concept could exist based on semantic processing or the use of a thesaurus (see Sections 3.6 and 3.7). Using a concept layer gives the inference network resilience to the matching problem because terms in the query need not directly match terms in the document; only the *concepts* must match.

An example of an inference network with actual nodes and links is given in Figure 2.8. A query and three documents are given along with the corresponding network. Links exist from the document layer to the term layer for each occurrence of a term in a document.

For our discussion, we focus on a three-layered inference network. Processing begins when a document is instantiated. By doing this, we are indicating that we believe document one ($D_1$) was observed. This instantiates all term nodes in $D_1$. We only instantiate the network with a single document at a time. Hence, the closed form for the link matrix for this layer will equal the weight for which a term might exist in a document. Typically, some close variant on $tf - idf$ is used for this weight.

Subsequently, the process continues throughout the network. All links emanate from the term nodes just activated and are instantiated, and a query node is activated. The query node then computes the belief in the query given $D_1$. This is used as the similarity coefficient for $D_1$. The process continues until all documents are instantiated.

*Figure 2.7.* Document-Term-Query Inference Network



### 2.4.4 Inference Network Example

We now use an inference network to compute a similarity coefficient for the documents in our example:

$Q$ : "gold silver truck"
$D_1$: "Shipment of gold damaged in a fire."
$D_2$: "Delivery of silver arrived in a silver truck."
$D_3$: "Shipment of gold arrived in a truck."

We need to evaluate our belief in the query given the evidence of a document, $D_i$. Assuming that our belief is proportional to the frequency within the document and inversely proportional to the frequency within the collection leads us to consider the term frequency, $tf$, and inverse document frequency, $idf$. However, both are normalized to the interval [0,1] by dividing $tf$ by the maximum term frequency for the document, and $idf$ by the maximum possible $idf$ (see Table 2.26).

In our three document collection, each term appears 1, 2 or 3 times. The total size of the collection is 3, so for:

*Figure 2.8.* Inference Network

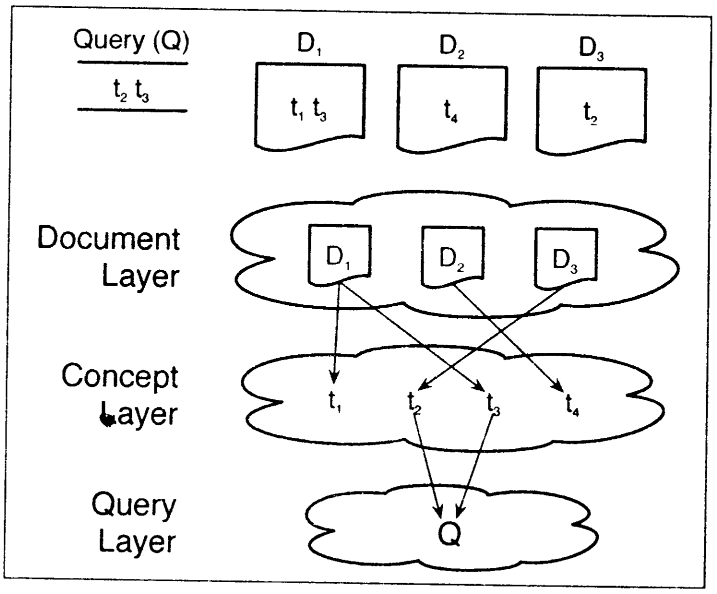


*Table 2.26.* Initial Values for Example Collection

| | a | arrived | damaged | delivery | fire | gold | in | of | shipment | silver | truck |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *idf* | 0 | 0.41 | 1.10 | 1.10 | 1.10 | 0.41 | 0 | 0 | 0.41 | 0.41 | 0.41 |
| *nidf* | 0 | 0.37 | 1 | 1 | 1 | 0.37 | 0 | 0 | 0.37 | 0.37 | 0.37 |
| $D_1$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| $D_2$ | 0.5 | 0.5 | 0 | 0.5 | 0 | 0 | 0.5 | 0.5 | 0 | 1 | 0.5 |
| $D_3$ | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

$tf = 1$, $idf = \ln \frac{3}{1} = 1.10$

$tf = 2$, $idf = \ln \frac{3}{2} = 0.41$

$tf = 3$, $idf = \ln \frac{3}{3} = 0$

Each term appears in a document either once or not at all, with the single exception of *silver* which appears twice in $D_2$. For each combination of term and document, we evaluate $P_{ij} = P(r_i = true|d_j = true)$. Turtle used the formula $P_{ij} = 0.5 + 0.5(ntf_{ij})(nidf_i)$ to compute the belief in a given term for a particular document. Instantiating a document provides equal support for all members of the assigned term nodes. Any node for which there is no support (no documents instantiated) has belief equal to zero.

For each term, a link matrix is constructed that describes the support by its parent nodes. For the concepts in the query, the link matrices are given below:

The link matrix for *gold* is:

| | $\overline{D_1 D_3}$ | $\overline{D_1} D_3$ | $D_1 \overline{D_3}$ |
|---|---|---|---|
| False | 1 | 0.315 | 0.315 |
| True | 0 | 0.685 | 0.685 |

The matrix indicates the belief of falsehood (first row) or truth (second row) given the conditions described in the column. When we instantiate a document, it is taken as true. Only one document is instantiated at a time. The number in the table is the value for $P_{ij}$, or the belief that term $i$ is true given document $j$ has been instantiated.

If $D_3$ is assigned a value of *true*, the belief is computed as:

$$P_{ij} = 0.5 + 0.5(ntf_{ij})(nidf_i) = 0.5 + 0.5(0.369)(1) = 0.685$$

This is found in the link matrix when $D_3$ is true. In this case, the link matrix has a closed form. Hence, it need not be stored or computed in advance. The matrix only accounts for three possibilities: both documents are false, $D_1$ is assigned a value of *true* and not $D_3$, $D_3$ is assigned a value of *true* and not $D_1$. Since *gold* does not appear in document two, there is no need to consider the belief when $D_2$ is assigned a value of *true* as there is no link from $D_2$ to the node that represents the term *gold*. Also, since documents are assigned a value of *true* one at a time, there is never a need to consider a case when $D_1$ and $D_3$ are true at the same time.

Similarly, the link matrix for *silver* can be constructed. *Silver* only appears in document $D_2$ so the only case to consider is whether or not $D_2$ is assigned a value of *true*. The link matrix computes:

$$P_{ij} = 0.5 + 0.5(ntf_{ij})(nidf_i) = 0.5 + 0.5(0.369)(1) = 0.685$$

Similarly, the link matrix for *truck* is constructed. *Truck* has two parents $D_2$ and $D_3$.

| | $\overline{D_2 D_3}$ | $\overline{D_2} D_3$ | $D_2 \overline{D_3}$ |
|---|---|---|---|
| False | 1 | 0.315 | 0.408 |
| True | 0 | 0.685 | 0.592 |

For $D_2$ true and $D_3$ false, we have:

$$P_{ij} = 0.5 + 0.5(ntf_{ij})(nidf_i) = 0.5 + 0.5(0.5)(0.369) = 0.592$$

We have now described all of the link matrices used to compute the belief in a term given the instantiation of a document. Now a link matrix for a query node must be developed.

There is quite a bit of freedom in choosing this matrix. The user interface might allow users to indicate that some terms are more important than others. If that is the case, the link matrix for the query node can be weighted accordingly. One simple matrix is given in Turtle's thesis [Turtle, 1991]. Using $g$, $s$, and $t$ to represent the terms *gold*, *silver*, and *truck*, it is of the form:

| | $\overline{gst}$ | g | s | gs | t | gt | st | gst |
|---|---|---|---|---|---|---|---|---|
| False | 0.9 | 0.7 | 0.7 | 0.5 | 0.5 | 0.3 | 0.3 | 0.1 |
| True | 0.1 | 0.3 | 0.3 | 0.5 | 0.5 | 0.7 | 0.7 | 0.9 |

The rationale for this matrix is that *gold* and *silver* are equal in value and *truck* is more important. Also, even if no terms are present, there is *some* small belief (0.1) that the document is relevant. Similarly, if all terms are present, there is some doubt (0.1). Finally, belief values are included for the presence of multiple terms.

We now instantiate $D_1$ which means Bel(gold)—the belief that *gold* is true given document $D_1$—is 0.685, Bel(truck) = 0.5, and Bel(silver) = 0.5. Note "Bel($x$)" represents "the belief in $x$" for this example.

At this point, all term nodes have been instantiated so the query node can now be instantiated.

Bel(Q | $D_1$) = 0.1(0.315)(1)(1) + 0.3(0.685)(1)(1) + 0.3(0.315)(0)(1) +
0.5(0.685)(0)(1) + 0.5(0.315)(1)(0) + 0.7(0.685)(1)(0) +
0.7(0.315)(0)(0) + 0.9(0.685)(0)(0) = 0.031 + 0.206 = 0.237.

This directly follows from the equation given in our prior examples using the link matrix entries $L_i(Q)$.

Instantiating $D_2$ gives Bel(gold) = 0, Bel(silver) = 0.685, and Bel(truck) = 0.592. The belief in $D_2$ is computed as:

Bel(Q | $D_2$) = (0.1)(1)(0.315)(0.408) + (0.3)(0)(0.315)(0.408) +
(0.3)(1)(0.685)(0.408) + (0.5)(0)(0.685)(0.408) +
(0.5)(1)(0.315)(0.592) + (0.7)(0)(0.315)(0.592) +
(0.7)(1)(0.685)(0.592) + (0.9)(0)(0.685)(0.592) =
0.013 + 0.084 + 0.093 + 0.283 = 0.473.

Assigning $D_3$ a value of *true* gives Bel(gold) = 0.685, Bel(silver) = 0, Bel(truck) = 0.685. The belief in $D_3$ is computed as:

$$\begin{aligned}\text{Bel}(Q \mid D_3) = {} & (0.1)(0.315)(1)(0.315) + (0.3)(0.685)(1)(0.315) + \\ & (0.3)(0.315)(0)(0.315) + (0.5)(0.685)(0)(0.315) + \\ & (0.5)(0.315)(1)(0.685) + (0.7)(0.685)(1)(0.685) + \\ & (0.7)(0.315)(0)(0.685) + (0.9)(0.685)(0)(0.685) = \\ & 0.01 + 0.065 + 0.108 + 0.328 = 0.511.\end{aligned}$$

In the link matrices throughout this example, we assume that each parent has an equal contribution to the child probability. The assumption is that if two parents exist, regardless of *which parents*, the child probability is greater. Recent work has described the potential to generate closed forms for link matrices in which the presence or absence of each parent is not equal [Greiff et al., 1997]. Only the surface has been scratched with regard to the topology of inference networks for relevance ranking. Potential exists to group common subdocuments in the inference network or to group sets of documents or clusters within the inference network. Also, different representations for the same document can be used. To our knowledge, very little was done in this area.

## 2.5 Extended Boolean Retrieval

Conventional Boolean retrieval does not lend itself well to relevance ranking because documents either satisfy the Boolean request or do not satisfy the Boolean request. All documents that satisfy the request are retrieved (typically in chronological order), but no estimate as to their relevance to the query is computed.

An approach to extend Boolean retrieval to allow for relevance ranking is given in [Fox, 1983a] and a thorough description of the foundation for this approach is given in [Salton, 1989]. The basic idea is to assign term weights to each of the terms in the query and to the terms in the document. Instead of simply finding a set of terms, the weights of the terms are incorporated into a document ranking. Consider a query that requests ($t_1$ OR $t_2$) that is matched with a document that contains $t_1$ with a weight of $w_1$ and $t_2$ with a weight of $w_2$.

If both $w_1$ and $w_2$ are equal to one, a document that contains both of these terms is given the highest possible ranking. A document that contains neither of the terms is given the lowest possible ranking. A simple means of computing a measure of relevance is to compute the Euclidean distance from the point $(w_1, w_2)$ to the origin. Hence, for a document that contains terms $t_1$ and $t_2$ with weights $w_1$ and $w_2$, the similarity coefficient could be computed as:

$$SC(Q, d_i) = \sqrt{(w_1)^2 + (w_2)^2}$$

For weights of 0.5 and 0.5, the SC would be:

$$SC(Q, d_i) = \sqrt{0.5^2 + 0.5^2} = \sqrt{0.5} = 0.707$$

The highest value of SC occurs when $w_1$ and $w_2$ are each equal to one. In this case we obtain $SC(Q, D_i) = \sqrt{2} = 1.414$. If we want the similarity coefficient to scale between 0 and 1, a normalization of $\sqrt{2}$ is added. The SC becomes:

$$SC(Q_{t_1 \bigvee t_2}, d_i) = \frac{\sqrt{(w_1)^2 + (w_2)^2}}{\sqrt{2}}$$

This coefficient assumes we are starting with a query that contains the Boolean OR: $(t_1 \bigvee t_2)$. It is straightforward to extend the computation to include an AND. Instead of measuring the distance to the origin, the distance to the point (1,1) is measured. The closer a query is to the point (1,1) the more likely it will be to satisfy the AND request. More formally:

$$SC(Q_{t_1 \bigwedge t_2}, d_i) = 1 - \frac{\sqrt{(1 - w_1)^2 + (1 - w_2)^2}}{\sqrt{2}}$$

### 2.5.1 Extensions to Include Query Weights

Consider again the same document that contains query terms $t_1$ and $t_2$ with weights $w_1$ and $w_2$. Previously, we assumed the query was simply a Boolean request of the form ($t_1$ OR $t_2$) or ($t_1$ AND $t_2$). We now add the weights $q_1$ and $q_2$ to the query. The new similarity coefficient that includes these weights is computed as:

$$SC(Q_{q_1 \bigvee q_2}, d_i) = \frac{\sqrt{q_1^2 w_1^2 + q_2^2 w_2^2}}{\sqrt{q_1^2 + q_2^2}}$$

$$SC(Q_{q_1 \bigwedge q_2}, d_i) = 1 - \left( \frac{\sqrt{q_1^2 (1 - w_1)^2 + q_2^2 (1 - w_2)^2}}{\sqrt{q_1^2 + q_2^2}} \right)$$

### 2.5.2 Extending for Arbitrary Numbers of Terms

For Euclidean distances in two-dimensional space, a 2-norm is used. To compute the distance from the origin in multi-dimensional space, an $L_p$ vector norm is used. The parameter, $p$, allows for variations on the amount of importance the weights hold in evaluating the measure of relevance. The new similarity coefficient for a query Q with terms $t_i$ and $t_j$ with weights $q_i$ and $q_j$ and a document $D_i$ with the same terms having weights of $w_i$ and $w_j$ is defined as:

$$SC(D, Q_{(q_i \bigvee q_j)}) = \left[\frac{q_i^p w_i^p + q_j^p w_j^p}{q_i^p + q_j^p}\right]^{\frac{1}{p}}$$

$$sim(D, Q_{(q_i \bigwedge q_j)}) = 1 - \left[\frac{q_i^p(1 - w_i^p) + q_j^p(1 - w_j^p)}{q_i^p + q_j^p}\right]^{\frac{1}{p}}$$

At $p$ equal to one, this is equivalent to a vector space dot product. At $p$ equal to infinity, this reduces to a normal Boolean system where term weights are not included. Initial tests found some improvement with the extended Boolean indexing over vector space (i.e., p = 2), but these tests were only done for small data collections and were computationally more expensive than the vector space model.

## 2.5.3 Automatic Insertion of Boolean Logic

Each of the retrieval strategies we have addressed do not require users to identify complex Boolean requests. Hence, with the use of OR, a query consisting only of terms can be used. Weights can be automatically assigned (using something like *tf-idf*) and documents can then be ranked by inserting OR's between each of the terms. The conventional vector space model, implicitly computes a ranking that is essentially an OR of the document terms. Any document that contains at least one of the terms in the query is ranked with a score greater than 0.

Conversely, a more sophisticated algorithm takes a sequence of terms and automatically generates ANDs and ORs to place between the terms [Fox, 1983a]. The algorithm estimates the size of a retrieval set based on a worst-case sum of the document frequencies. If term $t_1$ appears in 50 documents and term $t_2$ appears in 100 documents, we estimate that the query will retrieve 150 documents. This will only happen if $t_1$ and $t_2$ never co-occur in a document.

Using the worst-case sum, the terms in the query are ranked by document frequency. The term with the highest frequency is placed into a REMOVED set. This is done for the two highest frequency terms. Terms from the REMOVED set are then combined into pairs, and the pair with the lowest estimated retrieval set is added. The process continues until the size of the retrieval set is below the requested threshold.

## 2.6 Latent Semantic Indexing

Matrix computation is used as a basis for information retrieval in the retrieval strategy called Latent Semantic Indexing [Deerwester et al., 1990]. The premise is that more conventional retrieval strategies (i.e., vector space, probabilistic and extended Boolean) all have problems because they match directly on keywords. Since the same concept can be described using many different keywords, this type of matching is prone to failure. The authors cite a study in which two people used the same word for the same concept only twenty percent of the time.

Searching for something that is closer to representing the underlying semantics of a document is not a new goal. Canonical forms were proposed for natural language processing since the early 1970's [Winograd, 1983, Schank, 1975]. Applied here, the idea is not to find a canonical knowledge representation, but to use matrix computation, in particular Singular Value Decomposition (SVD). This filters out the noise found in a document, such that two documents that have the same semantics (whether or not they have matching terms) will be located close to one another in a multi-dimensional space.

The process is relatively straightforward. A term-document matrix A is constructed such that location $(i, j)$ indicates the number of times term $i$ appears in document $j$. A SVD of this matrix results in matrices $U \sum V^T$ such that $\sum$ is a diagonal matrix. $A$ is a matrix that represents each term in a row. Each column of $A$ represents documents. The values in $\sum$ are referred to as the singular values. The singular values can then be sorted by magnitude and the top $k$ values are selected as a means of developing a "latent semantic" representation of the $A$ matrix. The remaining singular values are then set to 0. Only the first $k$ columns are kept in $U_k$; only the first $k$ rows are recorded in $V_k^T$. After setting the results to 0, a new $A'$ matrix is generated to approximate $A = U \sum V^T$.

Comparison of two terms is done via an inner product of the two corresponding rows in $U_k$. Comparison of two documents is done as an inner product of two corresponding rows in $V_k^T$.

A query-document similarity coefficient treats the query as a document and computes the SVD. However, the SVD is computationally expensive; so, it is not recommended that this be done as a solution. Techniques that approximate $\Sigma$ and avoid the overhead of the SVD exist. For an infrequently updated document collection, it is often pragmatic to periodically compute the SVD.

### 2.6.1 LSI Example

To demonstrate Latent Semantic Indexing, we once again use our previous query and document example:

$Q$: "gold silver truck"
$D_1$ "Shipment of gold damaged in a fire."
$D_2$: "Delivery of silver arrived in a silver truck."
$D_3$: "Shipment of gold arrived in a truck."

The $A$ matrix is obtained from the numeric columns in the term-document table given below:

| | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| a | 1 | 1 | 1 |
| arrived | 0 | 1 | 1 |
| damaged | 1 | 0 | 0 |
| delivery | 0 | 1 | 0 |
| fire | 1 | 0 | 0 |
| gold | 1 | 0 | 1 |
| in | 1 | 1 | 1 |
| of | 1 | 1 | 1 |
| shipment | 1 | 0 | 1 |
| silver | 0 | 2 | 0 |
| truck | 0 | 1 | 1 |

This step computes the singular value decompositions (SVD) on $A$. This results in an expression of $A$ as the product of $U \sum V^T$. In our example, $A$ is equal to the product of:

$$\begin{bmatrix} -0.4201 & 0.0748 & -0.0460 \\ -0.2995 & -0.2001 & 0.4078 \\ -0.1206 & 0.2749 & -0.4538 \\ -0.1576 & -0.3046 & -0.2006 \\ -0.1206 & 0.2749 & -0.4538 \\ -0.2626 & 0.3794 & 0.1547 \\ -0.4201 & 0.0748 & -0.0460 \\ -0.4201 & 0.0748 & -0.0460 \\ -0.2626 & 0.3794 & 0.1547 \\ -0.3151 & -0.6093 & -0.4013 \\ -0.2995 & -0.2001 & 0.4078 \end{bmatrix} \begin{bmatrix} 4.0989 & 0 & 0 \\ 0 & 2.3616 & 0 \\ 0 & 0 & 1.2737 \end{bmatrix} \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & -0.2469 \\ -0.5780 & -0.2556 & 0.7750 \end{bmatrix}$$

However, it is not the intent to reproduce $A$ exactly. What is desired, is to find the best rank $k$ approximation of $A$. We only want the largest $k$ singular values ($k < 3$). The choice of $k$ and the number of singular values in $\Sigma$ to use is somewhat arbitrary. For our example, we choose $k = 2$. We now have $A_2 = U_2 \sum_2 V_2^T$. Essentially, we take only the first two columns of $U$ and the first twc rows of $\sum$ and $V^T$.

This new product is:

$$\begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \begin{bmatrix} 4.0989 & 0 \\ 0 & 2.3616 \end{bmatrix} \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & -0.2469 \end{bmatrix}$$

To obtain a $k \times 1$ dimensional array, we now incorporate the query. The query vector $q^T$ is constructed in the same manner as the original $A$ matrix. The query vector is now mapped into a 2-space by the transformation $q^T U_2 \sum_2^{-1}$.

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \begin{bmatrix} 0.2440 & 0 \\ 0 & 0.4234 \end{bmatrix} = \begin{bmatrix} -0.2140 & -0.1821 \end{bmatrix}$$

We could use the same transformation to map our document vectors into 2-space, but the rows of $V_2$ contain the co-ordinates of the documents. Therefore:

$D_1$ = (-0.4945 -0.0688)
$D_2$ = (-0.6458 0.9417)
$D_3$ = (-0.5817 1.2976)

Finally, we are ready to compute our relevance value using the cosine similarity coefficient. This yields the following:

$$D_1 = \frac{(-0.2140)(-0.4945) + (-0.1821)(0.6492)}{\sqrt{(-0.2140)^2 + (-0.1821)^2)}\sqrt{(-0.4945)^2 + (0.6492)^2)}} = -0.0541$$

$$D_2 = \frac{(-0.2140)(-0.6458) + (-0.1821)(-0.7194)}{\sqrt{(-0.2140)^2 + (-0.1821)^2)}\sqrt{(-0.6458)^2 + (-0.7194)^2)}} = 0.9910$$

$$D_3 = \frac{(-0.2140)(-0.5817) + (-0.1821)(-0.2469)}{\sqrt{(-0.2140)^2 + (-0.1821)^2)}\sqrt{(-0.5817)^2 + (-0.2469)^2)}} = 0.9543$$

### 2.6.2 Choosing a Good Value of k

The value $k$ is the number of columns kept after the SVD, and it is determined via experimentation. Using the MED database of only 1,033 documents and thirty queries, the average precision over nine levels of recall was plotted for different values of $k$. Starting at twenty, the precision increases dramatically up to values of around 100, and then it starts to level off.

### 2.6.3 Comparison to Other Retrieval Strategies

A comparison is given between Latent Semantic Indexing (LSI) with a factor of 100 to both the basic *tf-idf* vector space retrieval strategy and the extended Boolean retrieval strategy. For the MED collection, LSI had thirteen percent higher average precision than both strategies. For the CISI collection of scientific abstracts, LSI did not have higher precision. Upon review, the authors found that the term selection for the LSI and *tf-idf* experiments was very different. The LSI approach did not use stemming or stop words. When the same terms were used for both methods, LSI was comparable to *tf-idf*. More work was done with LSI on the TIPSTER collection [Dumais, 1994]. In this work, LSI was shown to perform slightly better than the conventional vector space model, yielding a 0.24 average precision as compared to 0.22 average precision.

### 2.6.4 Potential Extensions

LSI is relatively straightforward, and few variations are described in the literature. LSI focuses on the need for a semantic representation of documents that is resilient to the fact that many terms in a query can describe a relevant document, but not actually be present in the document.

### 2.6.5 Run-Time Performance

Run-time performance of the LSI approach is clearly a serious concern. With the vector space or probabilistic retrieval strategy, an inverted index is used to quickly compute the similarity coefficient. Each document in the collection does not need to be examined (unless a term in the query appears in every document). With LSI, an inverted index is not possible as the query is represented as just another document and must, therefore, be compared with all other documents.

Also, the SVD itself is computationally expensive. We note that several parallel algorithms were developed specifically for the computation of the SVD given here [Berry, 1992]. For a document collection with $N$ documents and a singular value matrix $\sum$ of rank $k$, an $O(N^2k^3)$ algorithm is available. A detailed comparison of several parallel implementations for information retrieval using LSI is given in [Letsche and Berry, 1997].

## 2.7 Neural Networks

Neural networks consist of nodes and links. Essentially, nodes are composed of output values and input values. The output values, when activated, are then passed along links to other nodes. The links are weighted because the value passed along the link is the product of the sending nodes output and the link weight. An input value of a node is computed as the sum of all incoming weights. Neural networks can be constructed in layers such that all the data the network receives are activated in phases, and where an entire layer sends data to the next layer in a single phase. Algorithms that attempt to learn based on a training set, modify the weights of the links in response to training data. Initial work with neural networks to implement information retrieval was done in [Belew, 1989]. This work used only bibliographic citations, but it illustrates the basic layered approach used by later efforts.

For ad hoc query retrieval, neural nets were used to implement vector space retrieval and probabilistic retrieval. Additionally, relevance feedback can be implemented with neural networks.

Using a neural network to implement vector space retrieval can, at first, appear to be of limited interest. As we have discussed (see Section 2.1), the model can be implemented without the use of neural networks. However, neural networks provide a learning capability in which the network can be changed based on relevance information. In this regard, the network adapts or learns about user preferences during a session with an end-user.

Section 2.7.1 describes a vector space implementation with a neural network. Section 2.7.2 describes implementation of relevance feedback. Section 2.7.3 describes a learning algorithm that can be used with a neural network for information retrieval. Subsequently, we describe a probabilistic implementation in Section 2.7.4. Section 2.7.5 describes how term components can be used within neural networks. Section 2.7.6 uses weights derived from those used for vector space and probabilistic models.

### 2.7.1 Vector Space

To use a neural network to implement the vector space model, we establish a network of three types of nodes: QUERY, TERM, and DOCUMENT [Crouch et al., 1994]. The links between the nodes are defined as query-term links and document-term links. A link between a query and a term indicates the term appears in the query. The weight of the link is computed as *tf-idf* for the term. Document-term links appear for each term that occurs in a given document. Again, a tf-idf weight can be used.

A feed-forward network works by activating a given node. A node is active when its output exceeds a given threshold. To begin, a query node is activated by setting its output value to one. All of its links are activated and subsequently new input weights for the TERM nodes are obtained. The link sends a value of $(tf)(idf)(1)$ since it transmits the product of the link weight with the value sent by the transmitting node (a one in this case). The weight, *tf-idf* in this case, is received by the term node. A receiving node computes its weight as the sum of all incoming links. For a term node with only one activated query, one link will be activated. The TERM node's output value will be a tf-idf weight. In the next phase, the TERM nodes are activated and all of the links that connect to a document node are activated. The DOCUMENT node contains the sum of all of the weights associated with each term in the document. For a collection with $t$ terms, the DOCUMENT node associated with document $j$ will now have the value:

$$DOC_j = \sum_{i=1}^{t}(tf_{ij})(idf_j)$$

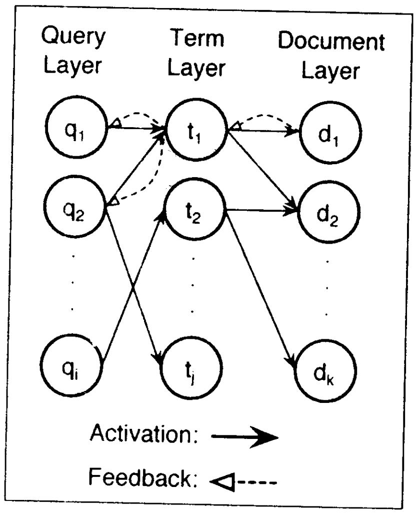
The DOCUMENT node now has a weight associated with it that measures the relevance of the document to a given query. It can easily be seen that this weight is equivalent to a simple dot product similarity coefficient as given in Section 2.1.

### 2.7.2 Relevance Feedback

To implement relevance feedback, a new set of links are added to the network. The new links connect DOCUMENT nodes back to TERM nodes. The document-term link is activated after the initial retrieval. Figure 2.9 illustrates this process along with a sample query and three documents. Links are fed into a newly defined input site on the TERM node, and their input is added to the value found in the existing query site of the TERM node. Without relevance feedback, the network operates in two phases. The first phase sends information from the QUERY nodes to the TERM nodes. The second phase sends information from the TERM nodes to the DOCUMENT nodes.

If relevance feedback is used, processing continues. The third phase sends information from the DOCUMENT nodes to the TERM nodes for the documents that are deemed relevant. The relevant documents are identified manually, or the top $n$ documents can be deemed relevant. Finally, in the fourth phase, the TERM nodes are activated, if they exceed a threshold parameter. The TERM-DOCUMENT links are used to send the newly defined weights

*Figure 2.9.* Neural Network with Feedback



obtained during the relevance feedback phase to the DOCUMENT nodes. At this point, the DOCUMENT nodes are scored with a value that indicates the effect of a single iteration of relevance feedback.

Initial experiments with the MEDLARS and CACM collection found an improvement of up to fifteen percent in average precision for MEDLARS and a degradation of eleven percent for CACM. As mentioned before, these collections are very small. Using a residual evaluation, in which documents found before relevance feedback are no longer considered, the average precision for CACM reached twenty-one percent and MEDLARS was as high as sixty percent.

## 2.7.3 Learning Modifications

The links between the terms and documents can be modified so that future queries can take advantage of relevance information. Typical vector space relevance feedback uses relevance information to adjust the score of an individual query. A subsequent query is viewed as a brand new event, and no knowledge from any prior relevance assessments is incorporated.

To incorporate relevance information into subsequent queries, the document nodes add a new signal called the learning signal. This is set to one if the user judges the document as relevant, zero if it is not judged, and negative one if it is judged as non-relevant. Term-document links are then adjusted based on the difference between the user assessment and the existing document weight. Documents with high weights that are deemed relevant do not result in much change to the network. A document weighted 0.95 will have a $\delta$ of $1 - 0.95 = 0.05$, so each of its term-document links will be increased by only five percent. A document with a low weight that is deemed relevant will result in a much higher adjustment to the network.

Results of incorporating the learning weight were not substantially different than simple relevance feedback, but the potential for using results of feedback sets this approach apart from traditional vector space relevance ranking.

## 2.7.4 Probabilistic Retrieval

Standard probabilistic retrieval based on neural networks is described in [Crestani, 1994]. The standard term weight given in Sparck Jones and described in more detail in Section 2.2.1 is used. This weight is:

$$\log \frac{r_i(N - n_i - R + r_i)}{(R - r_i)(n_i - r_i)}$$

where:

$N$ = number of documents

$R$ = relevant documents

$r_i$ = number of relevant documents that contain term $i$

$n_i$ = number of documents (relevant or non-relevant) that contain term $i$

The weight is a ratio of how often a term appears in relevant documents to the number of times it occurs in the whole collection. A term that is infrequent in the collection, but appears in most of the relevant documents is given a high weight. These weights are used as the weight of the query-term links, the term-document links, and essentially replaces the *tf-idf* weights used in the vector space model. The sum operation to combine links takes place as before, and results in a value that is very similar to the weight computed by the standard probabilistic retrieval model.

The training data are used, and a standard back propagation learning algorithm is used to re-compute the weights. Once training is complete, the top ten terms are computed using the neural network, and the query is modified to include these terms.

Using the Cranfield collection, the neural network-based algorithm performed consistently worse than the *News Retrieval Tool*, an existing probabilistic relevance feedback system [Sanderson and Rijsbergen, 1991]. The authors cite the relative lack of training data as one problem. Also, the authors note that the large number of links in the neural network makes the network cumbersome and consumes a substantial computational resource.

### 2.7.5 Component Based Probabilistic Retrieval

A component-based retrieval model using neural networks is given in [Kwok, 1989]. A three-layered network is used as before, but the weights are different. Query-term links for query $a$ are assigned a weight of $w_{ka} = \frac{q_{ak}}{L_q}$, where $q_{ak}$ is the frequency of term $k$ in query $a$. Document term links for document $i$ are assigned $w_{ki} = \frac{d_{ik}}{L_i}$, where $d_{ik}$ indicates the term frequency of term $k$ in document $i$. $L_q$ is the number of terms in the query and $L_i$ is the number of terms in document $i$. Term-query links are weighted $w_{ak}$ and document-term links are weighted $w_{ik}$. The query-focused measure is obtained by activating document nodes and feeding forward to the query. The document–focused measure is obtained by activating the query nodes and feeding forward to the documents.

Kwok extended his initial work with neural networks in [Kwok, 1995]. The basic approach is given along with new learning algorithms that make it possible to modify the weights inside of the neural network based on a training collection. Learning algorithms that added new terms to the query based on relevance feedback were given. Other algorithms did not require any additional query terms and simply modified the weights. The algorithms were tested on a larger corpus using more than three hundred megabytes of the *Wall Street Journal* portion of the TIPSTER collection. Kwok goes on to give learning algorithms based on no training collections, training based on relevance information, and query expansion.

Without a training collection, some initial data estimates can be assigned a constant (see Section 2.2.4). Training with relevance information proceeds using document-term links and term-document links as described in Section 2.7.2.

### 2.7.6 Combined Weights

A similar input-term-document-output layered neural network was used in [Boughanem and Soule-Depuy, 1997]. To our knowledge, this is the first report of the use of a neural network on a reasonably large document collection.

The key weight, which is used to represent the occurrence of a term in a document, is based on the pivoted document length normalization developed for the vector space model and the document length normalization developed for the probabilistic model (See Section 2.1.2 and Section 2.2.3).

The weight of the link from term $t_i$ to document $D_j$ is:

$$w_{ij} = \frac{(1 + \log(tf_{ij})) * (h_1 + h_2 * \log \frac{N}{df_i}))}{h_3 + h_4 * \frac{d_j}{\Delta}}$$

where:

$$\begin{aligned} tf_{ij} &= \text{weight of term } i \text{ in document } j \\ df_i &= \text{number of documents that contains term } i \\ d_j &= \text{length in terms (not included stop terms) of document } j \\ \Delta &= \text{average document length} \end{aligned}$$

Tuning parameters, $h_1$, $h_2$, $h_3$, and $h_4$, were obtained by training on the TREC-5 collection. Relevance feedback was also incorporated with the top twelve documents assumed relevant and used to supply additional terms. Documents 500-1000 were assumed non-relevant. An average precision of 0.1772 was observed on the TREC-6 data, placing this effort among the top performers at TREC-6.

### 2.7.7 Document Clustering

A neural network algorithm for document clustering is given in [Macleod and Robertson, 1991]. The algorithm performs comparably to sequential clustering algorithms that are all hierarchical in nature. On a parallel machine the neural algorithm can perform substantially faster since the hierarchical algorithms are all inherently sequential.

The algorithm works by first associating a node in the network for each cluster. Each node then computes (in parallel) a measure of similarity between the existing document and the centroid that represents the cluster associated with the node. First, a similarity coefficient is computed between the incoming document X and the existing cluster centroids. The input nodes of the neural network correspond to each cluster. If the similarity coefficient, $s_1$, is higher than a threshold, $s_{1_{avg}}$, the input node is activated. It then loops back to itself after a small recalculation to participate in a competition to add X to the cluster. Nodes that are not sufficiently close enough to the incoming document are deactivated.

A new pass then occurs for all of the nodes that won the first round, and the similarity coefficient is computed again. The process continues until only one cluster passes the threshold. At this point, a different similarity coefficient is computed, $s_2$, to ensure the winning cluster is reasonably close to the incoming document. If it is close enough, it is added to the cluster, and the centroid for the cluster is updated. Otherwise, a new cluster is formed with the incoming document.

The algorithm performed comparably to the single linkage, complete linkage, group average, and Ward's method which are described in Section 3.2. Given that this algorithm is non-hierarchical and can be implemented in parallel, it can be more practical than its computationally expensive hierarchical counterparts.

## 2.8 Genetic Algorithms

Genetic algorithms are based on principles of evolution and heredity. An overview of genetic algorithms used for information retrieval is given in [Chen, 1995]. Chen reviews the following steps for genetic algorithms:

- Initialize Population
- Loop
  - Evaluation
  - Selection
  - Reproduction
  - Crossover
  - Mutation
- Convergence

The initial population consists of possible solutions to the problem, and a fitness function that measures the relative "fitness" of a given solution. Note that the similarity coefficient is a good fitness function for the problem of finding relevant documents to a given query. Some solutions are selected (preferably the ones that are most fit) to survive, and go on to the next generation. The solutions correspond to chromosomes, and each component of a solution is referred to as a gene.

The next generation is formed by selecting the surviving chromosomes. This is done based on the fitness function. A value $F$ is computed as the sum of the individual fitness functions:

$$F = \sum_{i=1}^{population} fitness(V_i)$$

where *population* is the number of initial solutions. Consider a case where the initial population has a fitness function as given in Table 2.27:
The aggregate sum'of the fitness function, $F$, is 100, and the population size is five. To form the next generation, five values are chosen randomly, with a bias based on their corresponding portion of $F$.

In Table 2.28, the proportions of the total for each entry in the population are presented. To form a new generation of size five, five random values are selected between zero and one. The selection interval used for each member

*Table 2.27.* Simple Fitness Function

| i | *fitness(i)* |
|---|---|
| 1 | 5 |
| 2 | 10 |
| 3 | 25 |
| 4 | 50 |
| 5 | 10 |

of the population is based on its portion of the fitness function. If the random number is 0.50 it falls within the (0.40, 0.90] interval, and member four is selected. The magnitude of the fitness for a given member plays a substantial role in determining whether or not that member survives to the next generation. In our case, member four's fitness function is one half of the fitness function. There is a $1 - (\frac{1}{2})^5 = \frac{31}{32}$ chance of selecting this member into the next round.

*Table 2.28.* Selection Interval

| i | fitness(i) | $\frac{fitness(i)}{F}$ | Selection Interval |
|---|---|---|---|
| 1 | 5 | 0.05 | [0,0.05) |
| 2 | 10 | 0.10 | [0.05,0.15) |
| 3 | 25 | 0.25 | [0.15,0.40) |
| 4 | 50 | 0.50 | [0.40,0.90) |
| 5 | 10 | 0.10 | [0.90,1.0] |

Two types of changes can occur to the survivors—a crossover or a mutation. A crossover occurs between two survivors and is obtained by swapping components of the two survivors. .Consider a case where the first survivor is represented as 11111 and the second is 00000. A random point is then selected, (e.g., three). After the crossover, the two new children are: 11100 and 00011. This first child is derived from the first three one's from the first parent and the last two zero's of the second parent. The second child is derived from the first three zero's of the second parent with the last two one's of the first parent.

Subsequently, mutations occur by randomly examining each gene of each survivor. The probability of mutation is a parameter to the genetic algorithm. In implementations of genetic algorithms for information retrieval, the genes are represented as bits and a mutation results in a single bit changing its value. In our example, a random mutation in the second bit of the second child results in the second child changing its value from zero to one giving 01011.

The process continues until the fitness function for a new generation or sequence of generations is no better than it was for a preceding generation. This is referred to as *convergence*. Some algorithms do not attain convergence and are stopped after a predetermined number of generations.

### 2.8.1 Forming Document Representations

An initial foray into the use of genetic algorithms for information retrieval is given in [Gordon, 1988]. The premise being that a key problem in information retrieval is finding a good representation for a document. Hence, the initial population consists of multiple representations for each document in the collection. Each representation is a vector that maps to a term or phrase that is most likely selected by some users. A fixed set of queries is then identified, and a genetic algorithm is used to form the best representation for each document.

The query representation stays fixed, but the document representation is evaluated and modified using the genetic algorithm. The Jaccard similarity coefficient is used to measure the fitness of a given representation. The total fitness for a given representation is computed as the average of the similarity coefficient for each of the training queries against a given document representation. Document representations then "evolve" as described above by crossover transformations and mutations. Overall, the average similarity coefficient of all queries and all document representations should increase. Gordon showed an increase of nearly ten percent after forty generations.

First, a set of queries for which it was known that the documents were relevant were processed. The algorithm was then modified to include processing of queries that were non-relevant. Each generation of the algorithm did two sets of computations. One was done for the relevant queries and another for the non-relevant queries, against each representation. Survivors were then selected based on those that maximized the increase of the average Jaccard score to the relevant queries and maximized a decrease of the average Jaccard score for the non-relevant queries. After forty generations, the average increase was nearly twenty percent and the average decrease was twenty-four percent. Scalability of the approach can not be determined since the queries and the documents came from an eighteen document collection with each document having eighteen different description collections. These results must be viewed as somewhat inconclusive.

It should be noted, however, that although we referred to this as a document indexing algorithm, it is directly applicable to document retrieval. Different automatically generated representations, such as only terms, only phrases, different stemming algorithms, etc., could be used. After some training and evolution, a pattern could emerge that indicates the best means of representing documents. Additionally, we note that this strategy might be more applicable for document routing applications than for ad hoc query processing.

### 2.8.2 Automatic Generation of Query Weights

A genetic algorithm that derives query weights is given in [Yang and Korfhage, 1994]. It was tested on the small Cranfield collection. Tests using the DOE portion of the TIPSTER collection (and associated modifications that were necessary to scale to a larger collection) are given in [Yang and Korfhage, 1993]. Essentially, the original query is taken without any weights. The initial population is simply composed of randomly generating ten sets of weights for the terms in the original query. In effect, ten queries then exist in the population.

The genetic algorithm subsequently implements each of the queries and identifies a fitness function. First, the distance from the query to each document is computed, and the top $x$ documents are retrieved for the query ($x$ is determined based on a distance threshold used to determine when to stop retrieving documents, with an upper limit of forty documents). The fitness function is based on a relevance assessment of the top $x$ documents retrieved:

$$\text{fitness(i)} = 10R_r - R_n - N_r$$

where:

$$\begin{aligned} R_r &= \text{number of relevant retrieved} \\ R_n &= \text{number of non-relevant retrieved} \\ N_r &= \text{number of relevant not retrieved} \end{aligned}$$

Basically, a good fitness value is given for finding relevant documents. Since it is difficult to retrieve any relevant documents for larger collections, a constant of ten is used to give extra weight to the identification of relevant documents. Selection is based on choosing only those individuals whose fitness is higher than the average. Subsequently, reproduction takes place using the weighted application of the fitness value such that individuals with a high fitness value are most likely to reproduce. Mutations are then applied with some randomly changed weights. Crossover changes occur in which portions of one query vector are swapped with another. The process continues until all relevant documents are identified. The premise is that the original queries will find some relevant documents and, based on user feedback, other relevant documents will be found.

Tests on the Cranfield collection showed improved average precision, after feedback, to be twenty percent higher than previous relevance feedback approaches. In the follow-up paper using the DOE collection, the authors indicate that the genetic algorithm continues to find new relevant documents in each pass. This is interesting because the only thing that changes in the query are the query weights. No new terms are added or removed from the query.

Yang and Korfhage did use a relatively large collection (the DOE portion of the TIPSTER document collection) but only tested two queries.

### 2.8.3 Automatic Generation of Weighted Boolean Queries

Genetic algorithms to build queries are given in [Kraft et al., 1994, Petry et al., 1993]. The idea is that the perfect query for a given request can be evolved from a set of single query terms. Given a set of documents known to be relevant to the query, all of the terms in those documents can be used as the initial population for a genetic algorithm. Each term is then a query, and its fitness can be measured with a similarity coefficient (Kraft et al., used the Jaccard coefficient). Mutations of the query terms resulted in weighted Boolean combinations of the query terms. Three different fitness functions were proposed. The first is simple recall:

$$E_1 = \frac{r}{R}$$

where $r$ is the number of relevant retrieved and $R$ is the number of known relevant.

The second combines recall and precision as:

$$E_2 = \alpha(recall) + \beta(precision)$$

where $\alpha$ and $\beta$ are arbitrary weights.

The results showed that either of $E_1$ or $E_2$ fitness functions were able to generate queries that found all of the relevant documents (after fifty generations). Since $E_2$ incorporated precision, the number of non-relevant documents found decreased from an average of thirty-three (three different runs were implemented for each test) to an average of nine.

This work showed that genetic algorithms could be implemented to generate weighted Boolean queries. Unfortunately, it was only done for two queries on a small document collection (CACM collection with 483 documents), so it is not clear if this algorithm scales to a larger document collection.

## 2.9 Fuzzy Set Retrieval

Fuzzy sets were first described in [Zadeh, 1965]. Instead of assuming that an element is a member in a set, a membership function is applied to identify the degree of membership in a set. For information retrieval, fuzzy sets are useful because they can describe what a document is "about."

A set of elements where each element describes what the document is about is inherently fuzzy. A document can be about "medicine" with some oblique references to lawsuits, so maybe it is slightly about "medical malpractice." Placing "medical malpractice" as an element of the set is not really accurate, but eliminating it is also inaccurate. A fuzzy set is a membership in which the strength of membership of each element is inherently more accurate. In our example, the set of concepts that describe the document appears as:

$$C = \{(medicine,\ 1.0),\ (malpractice,\ 0.5)\}$$

The set C is a fuzzy set since it has degrees of membership associated with each member. More formally, a fuzzy set including the concepts in $C = \{c_1,\ c_2, \ldots, c_n\}$ is represented as:

$$A = (c_1,\ f_{A(c_1)}),\ (c_2,\ f_{A(c_2)}), \ldots,\ (c_n,\ f_{A(c_n)})$$

where $f_A : C \rightarrow [0,1]$ is a membership function that indicates the degree of membership of an element in the set.

For finite sets, the fuzzy set A is expressed as:

$$A = \left\{ \frac{f_{A(c_1)}}{c_1}, \frac{f_{A(c_2)}}{c_2}, \ldots, \frac{f_{A(c_n)}}{c_n} \right\}$$

Basic operations of intersection and union on fuzzy sets are given below. Essentially, the intersection uses the minimum of the two membership functions for the same element, and union uses the maximum of the two membership functions for the same element.

The following definitions are used to obtain intersection, union, and complement.

$$f_{A \bigcap B}(c_i) = Min(f_{A(c_i)}, f_{B(c_i)}), \forall c_i \in C.$$

$$f_{A \bigcup B}(c_i) = Max(f_{A(c_i)}, f_{B(c_i)}), \forall c_i \in C.$$

$$f_{A'}(c_i) = 1 - f_{A(c_i)}, \forall c_i \in C.$$

### 2.9.1 Boolean Retrieval

Fuzzy set extensions to Boolean retrieval were developed in the late 1970's and are summarized in [Salton, 1989]. A Boolean similarity coefficient can be computed by treating the terms in a document as fuzzy because their membership is based on how often they occur in the document.

Consider a set D that consists of all documents in the collection. A fuzzy set $D_t$ can be computed as the set D that describes all documents that contain the term $t$. This set appears as: $D_t = \{(d_1, 0.8), (d_2, 0.5)\}$. This indicates that $d_1$ contains element $t$ with a strength of 0.8 and $d_2$ contains $t$ with a strength of 0.5.

Similarly, a set $D_s$ can be defined as the set of all documents that contain term $s$. This set might appear as: $D_s = \{(d_1, 0.5), (d_2, 0.4)\}$

Computing $(s \bigvee t)$ requires $D_s \bigcup D_t$ and $(s \bigwedge t)$ $D_s \bigcap D_t$. These can be computed using the maximum value for union and the minimum for intersection. Hence:

$$(s \bigvee t) = D_s \bigcup D_t = \{(d_1, 0.8), (d_2, 0.5)\}$$

$$(s \bigwedge t) = D_s \bigcap D_t = \{(d_1, 0.5), (d_2, 0.4)\}$$

More complex Boolean expressions are constructed by applying the results of these operations to new expressions. Ultimately, a single set that contains the documents and their similarity coefficient is obtained.

One problem with this approach is that the model does not allow for the weight of query terms. This can be incorporated into the model by multiplying the query term weight by the existing membership strength for each element in the set. Another problem is that terms with very low weight dominate the similarity coefficient. Terms that have a very low membership function are ultimately the only factor in the similarity coefficient. Consider a case where document one contains term $s$ with a membership value of 0.0001 and term $t$ with a membership value of 0.5. In a query asking for $s \bigwedge t$, the score for document one will be 0.0001. Should the query have many more terms, this one term dominates the weight of the entire similarity coefficient. A remedy for this is to define a threshold $x$ in which the membership function becomes zero if it falls below $x$.

#### 2.9.1.1 Fuzzy Set Example

We now apply fuzzy set Boolean retrieval to our example. Our query "gold silver truck" is inadequate as it is designed for a relevance ranking, so we change it to the Boolean request: "gold OR silver OR truck." We take each document as a fuzzy set. To get a strength of membership for each term, we take the ratio of the term frequency within the document to the document length. Hence, our collection of documents becomes a collection of fuzzy sets:

$D_1$ = {(a, 0.143), (damaged, 0.143), (fire, 0.143), (gold, 0.143), (in, 0.143), (of, 0.143), (shipment, 0.143)}

$D_2$ = {(a, 0.125), (arrived, 0.125), (delivery, 0.125), (in, 0.125), (of, 0.125), (si!ver, 0.25), (truck, 0.125)}

$D_3$ = {(a, 0.143), (arrived, 0.143), (gold, 0.143), (in, 0.143), (of, 0.143), (shipment, 0.143), (truck, 0.143)}