INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Books Retrieval by Hierarchical Linear Subspace Method

PEDRO ARTEAGA GUERRA

Dissertação para obtenção do Grau de Mestre em

ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Júri

Orientador: Prof. Andreas Wichert

Novembro de 2008

Acknowledgements

I would like to start by thanking my aunts, my grandfather, my father and specially my mother, to whom I deeply thank the endless patience and infinite support.

To Prof. Andreas Wichert, to whom I truly thank the investigation skills, motivation and dedication. His knowledge and the capacity to show me the bright side, even when I thought things were not going so well, make him an important reference to me and to this work.

I also want to dedicate a very special word to all my friends, specially Nuno Matos and Guido Varatojo, who give me the privilege of being my friends.

To all that are close to me to the ones mentioned above, a truthful "Obrigado" to every contribution, each one in its own way, that helped developing this work.

Abstract

Nowadays, with the explosion of digital information availability, the need for fast methods to store, organize and quickly access that information as become an important issue. Written content does no longer appear to the consumer exclusively as physical books and magazines. The existence of a myriad of new devices like Personal Digital Assistants, laptops, tablet pc's, among others, and the concomitant access to the internet, have made a multitudinous number of contents available to every user.

The task of manually search or compare this books can be long and painful. Moreover, to quantify the similarity between some books and rank them accordingly, is rather impractical. Systems using *keyword* matching (also known as *boolean*) are widely used but are only practical for small documents, although good results can be achieved with more complex systems. Motivated by the results in *content-based image retrieval* [55], this thesis consists on experiments to assess whether or not the *hierarchical linear subspace method*, used with images, works as well with books. The benefits of this method rely not only in its simplicity when compared to other solutions, but also on the good results shown with image retrieval.

The major novelty of this work regards the usage of a new method (the *hierarchical linear subspace method*) to solve the books retrieval problem, while using the existing *vector-space model* to represent the collection of books in an high-dimensional vector-space.

To evaluate the viability of this method regarding books retrieval, several tests were carried out to attest the quality of the results, and of course, the performance. Although the results showed that the method was not a breakthrough comparing with the *list matching method*, they also led us to alternative paths where this method might work as with image retrieval.

Keywords: Hierarchical linear subspace method; list matching method; vector-space; books.

Resumo

Actualmente, devido ao explosivo aumento de informação digital disponivel, a necessidade de métodos eficazes para guardar, organizar e rapidamente aceder a essa informação tornou uma questão relevante. Os conteúdos escritos não nos chega apenas através dos tradicionais livros ou revistas. Na verdade, o fácil acesso a uma miríade de dispositivos electrónicos como os PDA's, computadores portáteis, *tablet pc's*, entre outros, bem como a concomitante facilidade de acesso à internet, tornaram possível a consulta de inúmeros conteúdos escritos.

A tarefa de procurar ou consultar manualmente estes livros pode ser longa e extenuante. No entanto, pode considerar-se verdadeiramente impraticável quantificar a semelhança entre vários livros e classificá-los de acordo com o grau de semelhança. Os sistemas baseados em procura por palavra-chave (usando a lógica booleana) ainda são muito usados apesar de serem práticos apenas para pequenos documentos, apesar de se obterem bons resultados através de métodos mais complexos, como *expert systems* por exemplo. Motivado pelo trabalho desenvolvido na área de *content-based image retrieval* [55], esta tese consiste em experiências para atestar se o *método de subespaço hierárquico* usado nas imagens, produz os mesmos resultados com livros. Os beifícios deste método baseiam-se não só na sua simplicidade quando comparado com outros métodos, mas também nos bons resultados obtidos com imagens.

A inovação deste trabalho reside na utilização de um novo método (o *método de subespaço hierárquico*) como solução para o problema de procurar livros, representando a colecção de livros em causa, num espaço vectorial de grande dimensão (*vector-space model*).

Por forma a avaliar a viabilidade deste método no contexto dos livros, foram levados a cabo diversos testes para averiguar a qualidade dos resultados, bem como, incontornavelmente, a performance. Apesar dos resultados terem revelado que o novo método não supera o *método de procura linear*, mostraram também novos caminhos que poderão produzir resultados semelhantes aos das imagens, usando o *método de subespaço hierárquico*.

Palavras-chave: Método de subespaço hierárquico; método de procura linear; espaço vectorial; livros.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Objectives . . . 2

1.3 Structure . . . 3

2 State of the Art **5**

2.1 Introduction - Structuring the Problem . . . 5

2.2 Boolean Model . . . 6

2.2.1 A First Approach . . . 6

2.2.2 Inverted Index . . . 7

2.2.3 Commercial Boolean Search: Westlaw [34] . . . 9

2.3 Vector Space Retrieval . . . 9

2.3.1 Tokenization . . . 9

2.3.2 Scoring and Term Weighting . . . 11

2.3.3 Definition of Similarity . . . 12

2.3.4 Efficient Scoring . . . 13

2.3.5 Comparison . . . 14

2.4 Dimensionality Reduction . . . 15

2.4.1 The Random Projection Method . . . 15

2.4.2 PCA, SVD and LSI . . . 16

2.5 Conclusions - Final Remarks . . . 17

3 Architecture **19**

3.1 Theoretical Model . . . 19

3.1.1 Vector-Space Model . . . 19

3.1.2 Generic Multimedia Indexing Method . . . 21

3.1.3 Hierarchical Linear Subspace Indexing Method . . . 23

3.2 System's Overview . . . 25

3.3 Document Indexing . . . 27

3.3.1 Building the Dictionary . . . 27

3.3.2 Creating the Original Vector-Space . . . 29

3.3.3 Vector-Space Hierarchy Computation . . . 30

3.4 Query Document Representation . . . 32

3.4.1 Reseting the Dictionary . . . 32

3.4.2 Query Document Creation . . . 34

3.4.3 Represent the Query Document in the Subspaces . . . 35

3.5 Query Processing . . . 35

3.5.1 Retrieving Top Importance Terms . . . 35
3.5.2 Document Retrieval . . . 35
3.5.3 Computing Characteristics Files . . . 38
3.6 The Prototype . . . 40
3.6.1 Programing Language . . . 40
3.6.2 Frameworks and Libraries . . . 41

4 Performance Evaluation **43**
4.1 Objectives and Methodology . . . 43
4.2 Quality . . . 44
4.3 Performance . . . 45
4.3.1 List Matching . . . 45
4.3.2 Speed Up . . . 47
4.3.3 Conclusions . . . 49

5 Conclusions and Future Work **55**
5.1 Conclusions . . . 55
5.2 Future Work . . . 56

Appendix A **59**

Appendix B **63**

Appendix C **67**

List of Figures

2.1 The dictionary stores the terms and their total frequency in the collection. The postings lists store the list of documents in which each term occurs, and may store other information such as the term frequency in each document and, optionally, the position(s) of the term in the document. 8

3.1 For example, the orthogonal projection of points $\vec{x} = (x_1, x_2) \in \mathbf{R^2}$ on the bisecting line $U = \{(x_1, x_2) \in \mathbf{R^2} | x_1 = x_2\} = \{(x_1, x_1) = \mathbf{R^1}\}$ corresponds to the mean value of the projected points. $\vec{a} = (2, 4)$ is mapped into $P(\vec{a}) = 3$, and $\vec{b} = (7, 5)$ into $P(\vec{b}) = 6$. 24
3.2 Example of characteristics plotting for an images database. The horizontal axis represents the number of images and the vertical axis corresponds to the distance. 25
3.3 Overview of the system's architecture. 26
3.4 Architecture of the indexing module. 28
3.5 DTD of the dictionary XML file. 30
3.6 DTD of the document XML file. 30
3.7 Architecture of the query document representation module. 33
3.8 Architecture of the query processing module. 36
3.9 Screenshot of a distances file used to compute the characteristics. 39
3.10 High-level application's class diagram. 40

4.1 Characteristics plotting for a 1000 books normalized collection. 49
4.2 Characteristics plotting for a 80 books normalized collection. 50
4.3 Characteristics plotting for a 80 books normalized collection. 50
4.4 Characteristics plotting for a 500 books normalized collection. 51
4.5 Characteristics plotting for a 500 books, not normalized collection. 51
4.6 Characteristics plotting for a 500 books normalized collection. 52
4.7 Characteristics plotting for a 500 books, not normalized collection. 52

1 Example of properties files using Windows and Mac OS X file paths. 63
2 Screenshot of the indexing phase. 64
3 Screenshot of a terms of importance retrieval. 64
4 Screenshot of a book retrieval query. 65
5 Screenshot of the raw dictionary XML file. 67
6 Screenshot of the reduced dictionary XML file. 68
7 Screenshot of a document XML file. 68
8 XML file of an auxiliary dictionary used in the creation of the query document. 69
9 Screenshot of a query with the book *The Hound of the Baskervilles*. 70
10 Screenshot of a query with the book *Pride and Prejudice*. 71
11 Screenshot of a query performed using the *linear search method*. 72

List of Tables

2.1 A term-document incidence matrix. Matrix element (i,j) is 1 if the play in column j contains the word in row i, and is 0 otherwise [34]. 7
2.2 Collection frequency (cf) and document frequency (df) behave differently [34]. 12

4.1 Results obtained in two of the performed quality queries, with outputs pictured in Appendix C. 45
4.2 This table shows for each query book, the time and number of operations taken with the *list matching method* and the 76 books collection. 46
4.3 This table shows for each query book, the time and number of operations taken with the *list matching method* and the 500 books collection. 46
4.4 This table shows for each query book, the time and number of operations taken with the *hierarchical subspace method* and the 76 books collection. 47
4.5 This table shows for each query book, the time and number of operations taken with the *hierarchical subspace method* and the 500 books collection. 48
4.6 Description of testing circumstances for each of the resulting graphics showed in the Figures below. 48

List of Acronyms

CBIR	Content-based Image Retrieval
DTD	Document Type Definition
GEMINI	GEneric Multimedia INdexIng
IDF	Inverse Document Frequency
CBIR	Content-based Image Retrieval
IR	Information Retrieval
LSI	Latent Semantic Indexing
PCA	Principal Components Analysis
RP	Random Projection
SAM	Spacial Access Method
SVD	Singular Value Decomposition
TF	Term Frequency
WF	Weighted Frequency
XML	eXtended Markup Language

Chapter 1

Introduction

This chapter's objective is to provide an overview of this thesis' scope. It starts by presenting a brief resume of the developed work, motivating this its topic and structure its objectives, and finally presents the thesis's organization.

1.1 Motivation

The exponential growth of digital information has increased the need for an automated method to store, organize and quickly access that information. For instance, written contents are no longer brought to us exclusively by books and magazines, as it was for a long time. Nowadays, one has access to a myriad of new systems, like several types of personal digital assistants (PDA), laptops, tablet pc's, among others, and most of them can easily access the internet, widening the amount of collections of contents available, like e-books, magazines or articles.

The search of written content is still greatly dependent on key word matching, with some operators (*boolean operators* such as AND, OR and NOT) which allow to perform more complex and accurate requests (*queries*). Since this method relies on the presence or absence of each key word, the queries have to traverse all documents (e.g. books, articles or magazines) in the collection, making the retrieval time, consequently, dependent on their size. Although this method is still practical for quick and simple searches, some necessities have been arising in order to improve the quality of the retrieved results as well as the retrieval time.

Among the performance issues that usually lead to the improvement of any technic (in this case, the retrieval of textual data in large collections), the need to perform queries using an example document or to retrieve documents similar to the given query, emerged not only as interesting but also as absolutely essential features.

In order to address this issue, a model has been created, and its basis consists of representing the data objects (can be textual or visual data, for instance) in a vector-space, hence the name *vector-space model*. This model adds new important features to this field, however, for larger documents, such as books, it becomes impractical to perform queries in a reasonable library, e.g. one thousand books, since the dimension of the space becomes very large, as the reader will see further.

During this work, several systems were studied to get a better understanding of what could be achieved with current methods. The objective of this work is to improve the extremely expensive *linear search*

method[1], however none of the systems seemed to present good results, particularly when dealing with larger books' collections. Nevertheless, some of the studied systems presented good results, using semantic interpretation, neural networks, statistic inference or complex heuristics. The ones that presented the best results were the expert systems, but they were subject dependent (vide [54], [57], [51], [38], [9]). The objective was to find a *simple* method that could effectively dismiss false candidates and find the most similar books to the query. By *simple* was meant that it would not require complex, and computationally expensive methods, like semantic interpretation or subject dependent heuristics and expert systems.

There was one system described in [55], which presented an innovative indexing method for large collections, in the field of *content-based image retrieval*. This indexing method, called *hierarchical linear subspace method*, motivated the development of the *Hierarchical Retrieval*, a system to test the application of this groundbreaking method to books retrieval. This system uses the vector-space method to represent the books, assigning an importance measure to each of its words. Then, using the *hierarchical linear subspace method*, a hierarchy of spaces is created to gradually eliminate the false hits to the query.

Further, the reader shall see in more detail that this method makes use of the dimension of the lowest spaces in the hierarchy, to achieve computational savings when comparing the query book against the books in the collection.

The hierarchy just mentioned represents the basis for our query engine, which will allow to retrieve the the *top important terms/words* of the collection, or a list of books representing *the most similar* books to the query. The later can make use of two distinct methods:

1. the *hierarchical linear subspace method*;
2. the *linear search method*.

Implementing this two methods was crucial to compare them since the main objective of this work is to prove whether or not the *hierarchical linear subspace method* is faster then the *linear search method*, in books retrieval.

The work developed in *Content Based Image Retrieval* [55] motivated the experiments described in this thesis. In the image context, this method has proven to overcome the high dimensionality issue, and provide excellent retrieval results. The advantage of having such a system for book retrieval, either in local collections or in the internet, is thought to be rather promising, motivating the experiments carried out in this work.

1.2 Objectives

There are many systems that are still based in the boolean retrieval model, and some are using the vector-space model, however, still facing the dimensionality problem. Therefore, the purpose of this thesis is to assess whether or not the *hierarchical subspace method* has identical behavior with textual contents, as with image contents.

To this end, the development of an application capable of indexing a considerably large collection of books was crucial to test the viability of the *hierarchical subspace method* applied to textual contents. This thesis targets the creation of a new indexing method to retrieve books from a large collection.

[1] This method corresponds to the regular comparison between every data element without previous elimination of false hits. It is all known as list matching.

1.3 Structure

This thesis is divided in seven chapters, starting by this one introducing the subject of the work.

Chapter 2 presents a review on the book indexing problem, presenting the evolution of the used models and the main problems that arise today in the context of book retrieval.

Chapter 3 intends to thoroughly describe the theoretical model behind vector-space representation and retrieval, starting by explain the GEMINI model and how it evolves to the *hierarchical subspace method*.

Chapter 4 presents the developed system's architecture, divided in core modules, as well as an in-depth description of the algorithms used to test the theoretical model presented in the previous chapter.

Chapter 5 is exclusively dedicated to the application itself, motivating the choice of the programming language, as well as presenting the libraries and frameworks used, and an high-level class diagram. This chapter intents to provide sufficient information for the correct usage of the prototype.

Chapter 6 presents the structure and the results of the tests carried out to assess the viability of the proposed model.

Finally, chapter 7 is dedicated to the conclusions and suggestions for future work.

Chapter 2

State of the Art

This chapter intends to give a general idea of the state of the art in information retrieval, specifically in what concerns to book indexing. First, we introduce the Boolean model of information retrieval, and the inverted index building process. Then, we briefly present the tokenization process, mentioning some special cases that often occur. Next, we describe the concepts of term frequency and weighting, in order to introduce the *inverse document frequency* scoring method. Finally, we present the vector-space retrieval method, and the definitions of similarity and dimensionality reduction that are involved.

2.1 Introduction - Structuring the Problem

Trying to find a specific document or book given a certain topic or subject, has been a real problem, perhaps since libraries exist. With the increasing amount of stored information, it has become a serious problem to face, and led to the creation of a myriad of indexing methods, such as to alphabetically order the books in an library, having an index built by alphabet letter and topic. Given that nowadays, one has an enormous amount of digital information arbitrarily deployed, *information retrieval* emerged not only as a necessity, but also as a science to efficiently manage and search digital content.

The definition of *information retrieval* proposed by *Manning* [34] (IR) (also vide [52]) is "finding material (usually documents) of an unstructured nature[1] (usually text) that satisfies an information need from within large collections (usually on local computer servers or on the internet)." Nowadays, there is an everyday use of information retrieval by means of search engines or emails.

A simple approach for selecting documents according to some criteria is, given a set of keywords and operators, find the documents in a collection which contain all or one of the keywords (depending on which operator is used in the request; the former would use an AND, as the later would use an OR). Although the *boolean model* is widely applied, it has an important limitation: it does not allow requests such as *find documents like this one*, because it is based in an exact retrieval method (either the word is present in a document, or not). In order to overcome this problem, one started to assign weights (like an importance measure) to each term/word in a document, and then represent each document in the collection as a vector, having the term weights as its components. Hence, the collection of documents could now be represented in a vector-space with as many dimensions as the number of different terms in all the documents.

[1] Here, unstructured, meaning a semantically not understandable structure for a computer.

Having the collection represented in a vector-space, one should now have a way to compare the documents, preferentially one that ranks the similarity between documents. To this end, one can use the Euclidean distance between two vector representations of documents, or a similarity measure called *cosine similarity*. Both of them will be explained further in this chapter.

Now that the documents are in the vector-space and there is a method to compare them, it is possible to answer the previously mentioned request: *find documents like this one*. A threshold number, k, of results is defined and the answer contains the k documents that are most similar to the query document, according to the chosen similarity measure. Recall that the number of dimensions corresponds to the number of different terms in all the collection. For a reasonably sized collection, one can have 50.000 dimensions for instance, which is quite a large number. In order to compute the similarity function between a query document and all collection's documents is quite an expensive operation, which is not practical for large (common/usual) collections. A new problem arises: how to speed up the query processing?

A possible solution to this issue is the *dimensionality reduction*, through which the dimension of the vector-space is reduced to achieve computational savings. There are several methods to accomplish this goal and there are going to be explained some of them, giving special attention to the most efficient, the *random projection method*. *Principal component analysis* (PCA), *singular value decomposition* (SVD) and *latent semantic indexing* (LSI) are also going to be briefly explained, and compared with the *random projection method*.

Throughout this chapter, we will present a theoretical description of the boolean model, the vector-space retrieval and the dimensionality reduction methods. Then, we will describe two vector-space retrieval systems, and finally some remarks on the efficiency of the presented methods.

2.2 Boolean Model

The simplest way to retrieve information from a set of documents is to go through each of them and see whether they contain all or some of the words of the query, for instance. To this end, regular expressions can be used for pattern matching. In spite of being practical for small collections of documents, in most cases one needs to quickly process large sets of documents, to provide more flexible operations, and also to allow ranked retrieval. As a consequence of these needs, document indexing[2] emerged as an alternative to linear scan.

2.2.1 A First Approach

As a first solution we can build an incidence matrix with terms and documents (either of them can be rows or columns). The queries correspond to simple boolean operations (like AND, OR, NOT) over certain rows and columns of the matrix.

As we can see in Table 2.1, if we want to answer the query Brutus AND Caesar AND NOT Calpurnia[3], we just have to take the vectors for Brutus, Caesar and Calpurnia, complement the last, and then perform a bitwise AND: 110100 AND 110111 AND 101111 = 100100. Thus, the plays which according to Table 2.1 answer the query are *Anthony and Cleopatra*, and *Hamlet*. Although this model's simplicity, for a more realistic context of IR, for instance a corpus with one million documents, this incidence matrix becomes extremely sparse, not mentioning its enormous size. A considerably better approach is to record only things that do occur, that is, the 1 positions (pointer encoding).

[2]This indexing phase takes place before the retrieval.
[3]This query represents all plays that make reference to Brutus and Caesar but do not mention Calpurnia.

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Table 2.1: A term-document incidence matrix. Matrix element (i,j) is 1 if the play in column j contains the word in row i, and is 0 otherwise [34].

2.2.2 Inverted Index

In order to overcome the drawbacks of the previous model, the concept of inverted index was developed, and consists of a vocabulary or dictionary of terms, each of them having a list, called a postings list, that records in which documents that term occurs (each item of the list is called a posting). This index will be used further in the vector-space retrieval model.

In order to take advantage of the speed benefits of indexing at retrieval time, the index must be built in advance, and the general steps to achieve it are the following:

1. Collect documents to be indexed;
2. Tokenize the text, resulting in a list of tokens per document;
3. Linguistic pre-processing or token normalizing, which is the hardest phase because it involves semantic interpretation;
4. Finally, the indexing step from which results the inverted index, consisting of a lexicon and postings lists.

For now, let us assume we have already chosen the documents and that no linguistic pre-processing is required.

Building the inverted index In this phase we start by assigning each document an identifier (docID). The first step consists in tokenizing the documents, that is, given a corpus of documents we get a list of normalized tokens that correspond to pairs (term, docID):

- the first element of the pair is the term itself;
- the second element of the pair is the docID of the document in which the corresponding term appeared.

In the second step the list previously obtained is sorted alphabetically, and in the third step, repeated terms are merged, and for each of them it is added information[4] about their frequency. In the fourth and final step we split the previous list into a dictionary (each item corresponds to the term and its frequency) and postings lists (a postings list corresponds to a list of documents in which each term occurs, the frequency of the term in a specified document, and optionally the position in the document). Figure 2.1 shows the final layout of the index. More information about indexing in [3].

[4]Initially this information is not used by the basic boolean search engine.

Doc 1
I did enact Julius Caesar: I was killed i′ the Capitol; Brutus killed me.

Doc 2
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

term	docID	freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i′	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

$\Longrightarrow$

term	coll. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	3	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
I	2	→	1
i′	1	→	1
it	1	→	2
julius	1	→	1
killed	2	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Figure 2.1: The dictionary stores the terms and their total frequency in the collection. The postings lists store the list of documents in which each term occurs, and may store other information such as the term frequency in each document and, optionally, the position(s) of the term in the document.

Query Processing In the basic boolean retrieval method a query is processed in two phases:

1. Retrieve the postings for each term in the query;

2. Merge the postings according to the boolean operations in the query.

In this merging step, we must assure that the postings are ordered by docID so that we can traverse each retrieved postings list and compare the docIDs. When computing an AND query with two terms (their postings lists have lengths x and y), we would take O(x+y) to merge the two postings lists.

2.2.3 Commercial Boolean Search: Westlaw [34]

Extended boolean retrieval models (usually including a proximity operator) were the main, if not only, search option available until, approximately, the arrival of the **World Wide Web** in the early 1990s. As an example of this search method let us consider **Westlaw**, a service that started in 1975, which is the largest legal search service (in terms of paying subscribers), and mainly uses boolean queries as default search method.

Usually in this system, queries are incrementally developed until they appear to give a suitable result for the user. Boolean queries are still preferred by many users, essentially professionals, who often find them more reliable and precise: either a document is a match or not. This system has also the capability to present ranked results, consisting of returning the documents in reverse chronological order.

Studies driven by **Westlaw's** reference librarians (for more information vide [34]) shown that natural language queries returned better results that the boolean queries developed by those librarians.

2.3 Vector Space Retrieval

After building the inverted index we are now able to use it in another model: the *vector-space model* [45], [34], [1], [46]. This method involves the representation of a set of documents in a common vector-space. Each document is represented by a vector in that space, and each of its components corresponds to a term/dimension in that document/space.

In this section, we will start by giving a more detailed explanation of the tokenization process, presenting some problems that this phase arises and some ways to reduce the dictionary size. Then, we introduce the concept of weighting measure to build a more consistent idea of the *wf-idf* weighting measure. Finally, we are able to introduce a similarity measure, some methods to achieve efficient scoring and compare this model with some other retrieval methods.

2.3.1 Tokenization

The documents we have to work with are often in formats other than ASCII, and in those cases the decoding process can become complex. For instance, in writing systems such as the Arabic, the conversion to a sequence of characters is not straightforward. Nevertheless, there is an underlying phonetic sequence which allows us to define a linear structure. These correspond to some of the issues that often arise in this phase.

It is worth mentioning that, among many other decisions to be made in the analysis process [47] (the topics of this section introduce some of these aspects), we must decide what corresponds to a unit document for indexing. For instance, we can consider each file in the collection as a document. As an alternative, and depending on the desired granularity on the search, we can consider each paragraph or a certain set of documents as a pseudo-document corresponding to the unit for indexing.

The Tokenization Phase

The process of tokenization can be defined as the separation of a document (considering a character sequence grouped in the defined unit) into pieces (we can think of them as terms, called tokens) removing some characters like white spaces and punctuation. According to [34] "a token is an instance of a character sequence in some particular document" whereas "a type is the class of all tokens containing the same character sequence".

It is also worth mentioning that it is crucial to employ the same tokenization method for the documents and the queries posed to those documents, in order to assure that a certain sequence of characters in a document will always match the same sequence in a query.

Special Cases During the process of tokenization some problems might arise with particular cases[5] like *o'neill* or *aren't* or even *students'*. Each of these cases require a particular method, and a choice must be made to decide how to separate them, that is, how to convert them into a token.

Character sequences such as email addresses, URLs, IP addresses, page tracking numbers, or monetary amounts should be recognized as a single token. One possible solution is to omit these tokens as they contribute to the growth of the dictionary, although their omission restricts, in a large scale, the terms of the search, resulting in a loss of information content of the document.

Hyphenation can also be a problem as it can be used:

- to split vowels in words, like *co-education*;
- to join nouns as name, like *Hewlett-Packard*;
- to show word grouping, like *hold-him-back-and-throw-him-away-maneuver*.

White spaces may also arise some confusion in names (*Los Angeles*), foreign phrases (*au fait* or *en passant*), dates, phone numbers, and words that sometimes have a space between them and sometimes do not, like *whitespace* and *white space*.

Another special case are *stop words* which consist of words that are extremely common and have little or no semantic meaning, such as articles and prepositions. This set of words in a particular indexing context is called *stop list*, and it is often hand-filtered by their semantic context. In spite of considerably reducing the size of the postings, this method disproportionately affects some kinds of queries, such as "President of the United States" which is more precise then "President" AND "United States".

Although it did not represent a main subject of this research, it is worth mentioning the several limitations brought by each language. According to [34], "each language presents some new features" and consequently new special cases to take into account[6].

[5] Examples taken from [34].
[6] One can think of Japanese or Arabic, using completely different phonetics and writing systems than ours.

Stemming and lemmatization are two methods used to reduce inflectional and variant forms of a certain word to a base form. In *stemming* is used a raw heuristic process to drop the suffixes and prefixes from words in order to achieve the correct form. For instance, car, cars, car's and cars' all are reduced to car. The *lemmatization* process is more complex as it makes use of a dictionary to make a morphological analysis of words, in order to achieve the dictionary or base form of a word.

2.3.2 Scoring and Term Weighting

Until now we described the indexing processes that led to an inverted index which supports Boolean queries. We also mentioned some problems related to the tokenization process. Now let us explore the concept of scoring with the objective of ranking the documents to order them in terms of importance/relevance.

Parametric and Zone Indexes

The documents we work with every day, besides being just a sequence of terms, they are also structured in some way; they have an author, a title, some sections or chapters, paragraphs, etc. Hence, to make use of this structure and to support queries like "find documents authored by Shakespeare in 1601, containing the phrase **alas poor Yorick**" (example from [34]), we have to build *parametric indexes*. These, consist of inverted indexes, as we previously saw, referring to each kind of meta-data (as the author or the date, for instance). Each of those kinds of meta-data or fields, will correspond to an inverted index whose dictionary consists of all distinct values occurring in that field, and postings pointing to documents with that field value.

Besides *parametric indexes* we can also have *zone indexes*. A zone is similar to a field, except its contents can be thought of as an arbitrary amount of text, instead of a relatively small set of values.

Term Frequency and Weighting

So far, we were restricted to whether or not a query term was present in a document. The following step consists in assigning more importance to the document in which the term occurs more often. In order to allow more flexible queries, the concept of *free text queries* is introduced. This is a query in which its terms are typed without any search operators, such as boolean operators.

Now, the idea is to assign to each term in a document, a weight for that term in that specified document, depending on the number of occurrences. First, let us follow a simple approach saying that the weight corresponds to the number of occurrences in the specified document. This approach is called term frequency of term t in document d, $tf_{t,d}$. One common weighting function is the logarithmic function given by

$$wf_{t,d} = \begin{cases} 1 + log(tf_{t,d}) & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

In the upper equation, we add 1 to distinguish the $tf_{t,d}$ values of one and zero, since logarithm of 1 is 0. Given a document d, one can consider the set of weights as a vector, with one component for each distinct term, hence representing document d as a vector of $tf_{t,d}$ values. The logarithmic function is used to avoid an exaggerated growth of the frequency function.

Inverse Document Frequency There are words in a document that do not define its subject. The idea is to give them less importance when calculating the weights. There are two concepts that will help to explain this idea:

- **Collection frequency** (cf) corresponds to the total number of occurrences of a term in the collection/corpus.
- **Document frequency** (df) is defined as the number of documents in the collection/corpus in which the term occurs.

Usually one prefers *document frequency* instead of *collection frequency*, since the former reflects more precisely the importance of the term, as can be intuitively seen in Table 2.2.

Word	cf	df
ferrari	10422	17
insurance	10440	3997

Table 2.2: Collection frequency (cf) and document frequency (df) behave differently [34].

If N is the total number of documents in the corpus, the *inverse document frequency* (idf_t) [42] [40] [12] of a term t is given by

$$idf_t = log\frac{N}{df_t} \tag{2.2}$$

The idf_t value is high for a rare term and low for a frequent one.

tf-idf weighting This idea intends to combine the *term frequency* and *inverse document frequency* concepts to produce a more accurate weighting measure, considering its importance in each document (tf), and its relevance in a collection's point of view (idf). **tf-idf** [4] weighting assigns a certain weight to term t in document d and is given by

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_{t,d} \tag{2.3}$$

Just as before, it can be generalized considering a weighting function instead of raw term frequency,

$$wf\text{-}idf_{t,d} = wf_{t,d} \times idf_{t,d} \tag{2.4}$$

The values of $tf\text{-}idf_{t,d}$ and $wf\text{-}idf_{t,d}$ represent the weight of a term t in a document d:

- highest value, when t occurs within a small set of documents;
- lower, when t occurs fewer times in a document or in many documents;
- lowest value, when t occurs virtually in all documents.

At this phase, we can already view a document as a vector with each component corresponding to each term. Each term then corresponds to the values given by equation 3 or 4.

There are other variants of weighting functions that will not be mentioned here, as well as several other topics under the scoring subject, that are thoroughly described and can be consulted in [48], [34], [41], [49] and [43].

2.3.3 Definition of Similarity

At this time, one can view a document as a vector of $wf\text{-}idf$ values, each of them corresponding to a term. Hence, we have a vector-space where documents are represented as vectors, and where terms correspond to axes. Note that, even applying the stemming process previously mentioned, this space can go up to 50,000 dimensions. The idea is that if two documents have similar vector representations they

discuss the same topics. One should note that this representation loses the relative ordering of the terms in each document. This representation also allows *query-by-example*, that is, finding documents similar to a given query document.

Let us now describe the concept of similarity between two documents. In a first approach, one might consider the magnitude of the difference vector between the two vectors/documents, that is, if one has two documents, say d_1 and d_2, the distance between them corresponds to the length of the vector $|d_1 - d_2|$, using simple Euclidean distance. The main drawback of this measure is that it doesn't take into account the length of the documents, hence bigger documents are always similar to each other by virtue of length, and not the topic as one wants.

A method of overcoming this problem is by taking into account the angles instead of the length[7]. Usually the standard method to quantify the similarity between two documents, is to compute the *cosine similarity* of their vector representations:

$$sim(d_1, d_2) = \frac{\vec{V}(d_1).\vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|} \tag{2.5}$$

This similarity measure, based on the angle between the two documents, is equivalent to computing the Euclidean distance between the normalized vector representations of the documents. Nevertheless, *similarity* is not *distance*, since we lose the real length of the vectors. The numerator corresponds to the inner product of the vector representations of the two documents and the denominator corresponds to the simple product of their lengths. $sim(d_1, d_2)$ can also be considered as the inner product of the normalized vector representations of the documents. The problem of finding the documents most similar to a given document d is equivalent to find the documents with the highest inner products (sim values) $\vec{v}(d).\vec{v}(d_i)$. We can then view a corpus of n documents (set of vectors) as a term-document matrix: that is, an $m \times n$ matrix whose rows represent the m terms (dimensions) of the n documents (columns). One should note that unless a stemming process is driven (like previously mentioned), words/terms like *jealous* and *jealousy* are considered as two different dimensions. One should also note that the cosine similarity is not a distance measure; it only represents a *similarity* concept.

Queries as vectors One compelling reason to use the vector representation of documents is to view queries as vectors. The idea is to convert the query into the corpus coordinate system and compute the inner product between the query vector and each document, and choose the one with the highest similarity value.

Through this approach a query is seen as a very small document, which makes it possible to use the cosine similarity as the scoring method for each pair of documents. These scorings can then be used to elect the k most relevant documents for that particular query. Note that the vector that represents the query document is very sparse; it has zeros in many dimensions[8].

2.3.4 Efficient Scoring

Until now, we were assuming that given a query, the exact k most relevant documents are returned, without thinking of the high cost of this operation. We will now present some methods to reduce this cost, by returning a set of k documents that are likely to be among the k most relevant ones.

[7] Which implicitly represents a normalization.
[8] Recall that each dimension is a term.

Here are some **heuristics** to reduce the number of documents to compute the cosine similarity (index elimination):

1. Let us consider only the documents whose *idf* values exceed a preset threshold. We only traverse the postings from terms with high *idf* because, low *idf* terms are considered stop words and represent little or no contribution to the score; but mostly, the postings lists with low *idf* are generally longer, and hence take more time to traverse.

2. Only take into account the documents containing many, or all, query terms. The idea is to consider the query as conjunctive query.

Champion Lists The idea of champion lists is to compute in advance, for each term in the dictionary, the set of M documents where they appear with the highest score. Given a term t, its set of M documents with the highest weights is called the *champion list* for term t. The next step is to make the union of the champion lists of each term in a certain query, and restrict the cosine computation to the documents in this union. We must note that M is application dependent, and must be chosen in advance.

Cluster pruning consists of separating the corpus (with n elements) in several sets, specifically in $\sqrt{n}$ sets, having each of them a representative leader. This is done in a pre-processing phase as follows:

- Select randomly $\sqrt{n}$ documents to be the leaders;

- Compute the nearest leader for every document which is not a leader (those are called the followers). For each leader, the number of followers is approximately $\sqrt{n}$.

Given a query the process goes as follows:

- Find the leader which is closest to the query, by computing the cosine similarity with each of the $\sqrt{n}$ leaders;

- Compute the cosine similarity for all documents in the candidate set, composed by the previously chosen leader and its followers.

One might think: what if the number of followers of a certain cluster is smaller than the number of pretended results? It can be adopted a general form of cluster pruning:

- In preprocessing, for each follower compute its a closest leaders;

- In query processing, consider the b closest leaders to the query.

2.3.5 Comparison

Here, will be presented the relation between vector-space and other retrieval methods [28], in terms of expressiveness of queries and index that supports the evaluation of the various retrieval methods. It is important to note that, in contrast with the other retrieval methods, the vector-space retrieval supports free-text retrieval.

Boolean retrieval Vector space index can be used to answer boolean queries (as long as, whenever a term is present, its weight is non zero) but not vice versa. Actually, in practice vectors and boolean queries do not fit quite well. In a vector-space, the proximity/similarity is selected by spheres. On the other hand, boolean queries select their results by rectangles[9]. Vector space queries can be viewed

[9] Or hyper-rectangles, in case of higher dimensions.

as evidence accumulating, as a document's score increases if there are more query terms present in that document. On the other hand, boolean queries imply the verification of a formula for selecting documents through the presence or absence of specific keywords.

Wildcard queries are also worth mentioning. Just as vector-space, it can be implemented using postings and a dictionary but require different indexes.

In contrast to wild card queries, due to the lossy representation of documents as vectors, as it looses the relative order of terms, the index built for vector-space retrieval cannot be used for phrase queries.

As a final reference, query parsing and composite scoring can be used to map the user-specified query into a compatible query for the existing indexes.

2.4 Dimensionality Reduction

The idea behind reducing the dimension of the space is to speed up the selection of the k most relevant documents for a given query, consequently reducing the cosine computations. A statistically optimal way to reduce the dimension of the space is to project the data onto a lower-dimension orthogonal subspace that reflects the original data distribution in the space, as reliably as possible. The *principal component analysis* (PCA), in spite of being widely used, is extremely expensive for high-dimensional data.

With little distortion[10] of the actual relation between data elements, the *random projection method* (RP) makes use of randomly generated matrices in order to project the original high-dimensional space, to a lower-dimensional one.

2.4.1 The Random Projection Method

The idea of random projection is to project the m-dimensional collection onto a k-dimensional subspace ($k << m$) through the origin, using a $k \times m$ matrix R whose columns have unit lengths. Given that $X_{m \times n}$ (n is the number of documents in the collection) is the matrix corresponding to the original collection in the m-dimensional space,

$$X^{RP}_{m \times n} = R_{k \times m} X_{m \times n} \tag{2.6}$$

represents the projection of the collection onto the k-dimensional subspace. The central idea of this method is based on the Johnson-Lindenstrauss lemma [27]: if points in a vector-space are projected onto a randomly selected subspace of suitably high dimension, then the distances between the points are approximately preserved. A simple proof can be found in [15], [20].

Algorithm The computation of the random projection method goes as follows:

- Project n vectors from m dimensions to k *dimensions* ($k << m$):
  - Start with $m \times n$ matrix of *terms* $\times$ *documents*, A;
  - Find random $k \times m$ orthogonal projection matrix, R, through the following steps:
    - choose a random direction x_1 in the vector-space;
    - for $i = 2$ to k, choose a random direction, x_i, which must be orthogonal to $x_1, x_2, ..., x_{i-1}$;
    - project each vector representation of a document into the subspace spanned by $x_1, x_2, ..., x_k$.

[10] There will be presented some experiments showing the actual magnitude of distortion.

- Compute the $k \times n$ matrix $W = R \times A$.

- The jth column of W is the vector corresponding to document j, however, now it is represented in $k << m$ dimensions.

According to [39], random projection is computationally simple: generating the random matrix R and projecting the original vector-space $m \times n$ matrix X into k dimensions can be achieved in *O(mkn)*, and if the X matrix is sparse, considering c nonzero entries per column, the complexity is *O(ckn)*. For a more detailed information vide [34], [15], [21], [6], [5], [8], [14], [26], [29], [30], [31], [53], [17], [18], [25], [32].

To be more precise, the projection equation presented earlier is actually not a projection, since R is generally not orthogonal. Such a linear mapping can cause significant distortions in the data if R is not orthogonal, however orthogonalizing R is extremely expensive in terms of computational resources. Nevertheless, we can rely on a result presented by Hecht-Nielsen [24]: in a high dimensional space, there exists a much larger number of *almost orthogonal* than *orthogonal* directions. Thus vectors having random directions might be sufficiently close to orthogonal, and equivalently $R^T R$ would approximate an identity matrix. According to experiments in [11], the mean squared difference between $R^T R$ and an identity matrix is approximately $1/k$ per element.

As seen earlier, text documents are usually compared using the cosine of the angle between the vector representations of those documents[11]. If two document vectors are normalized, this corresponds to the inner product of those vectors. Now, let us assume we use the Euclidean distance to compare two document vectors, x_1 and x_2, and that their diference is represented by $||x_1 - x_2||$. When the random projection is applied, this distance can be approximated by the scaled Euclidean distance of the vectors in the lower dimensional space,

$$\sqrt{m/k}||Rx_1 - Rx_2|| \tag{2.7}$$

where m and k correspond to the dimensions of the original and subspace, respectively. The scaling term $\sqrt{m/k}$ takes into account the decrease in the dimensionality of the represented data. As we can see in the Johnson-Lindenstrauss lemma [27], the expected norm of a projection of a unit vector onto a random subspace through the origin is $\sqrt{k/m}$.

One of the crucial points of the random projection method is the choice of the random matrix R. Usually, the elements $r_{i,j}$ of the matrix obey a Gaussian distribution, however this is not a strict rule. In fact, there is a much simpler distribution proposed by Achlioptas [2] that can result in further computational savings:

$$r_{i,j} = \sqrt{3}. \begin{cases} +1 & \text{with probability } \frac{1}{6} \\ 0 & \text{with probability } \frac{2}{3} \\ -1 & \text{with probability } \frac{1}{6}. \end{cases} \tag{2.8}$$

2.4.2 PCA, SVD and LSI

We are now going to briefly present three other common methods to accomplish dimensionality reduction: *principal component analysis* (PCA), *singular value decomposition* (SVD) and *latent semantic indexing* (LSI).

[11] Cosine similarity

PCA In PCA, the computation of the eigenvalues of the collection's covariance matrix follows $E\{XX^T\} = EUE^T$, where the columns of matrix E correspond to the eigenvectors of the collection's covariance matrix $E\{XX^T\}$, and U is a diagonal matrix built from the respective eigenvalues. To achieve dimensionality reduction of the collection, it must be projected onto a subspace spanned by the most representative eigenvectors,

$$X^{PCA} = E_k^T X \tag{2.9}$$

where the $m \times k$ matrix E_k consists of the eigenvectors corresponding to the k largest eigenvalues[12]. In spite of being an optimal way of data projection, since the error introduced is minimized over all projections onto a k-dimensional space [11], the eigenvalue decomposition of the collection's covariance matrix (a $m \times m$ matrix for a m-dimensional data) is extremely expensive to compute. Although there are less expensive methods [44] [50] to find only a few eigenvectors and eigenvalues of a large matrix, the average complexity of PCA is $O(m^2n) + O(m^3)$ [22].

SVD *Singular value decomposition* is closely related to PCA: $X = USV^T$ where orthogonal matrices U and V contain the left and right singular values of X. Using SVD, the dimensionality of the collection can be reduced by projecting it onto the space spanned by the left singular vectors corresponding to the k largest singular values,

$$X^{SVD} = U_k^T X \tag{2.10}$$

where U_k is a $m \times k$ matrix and contains these k singular vectors. Just as PCA, SVD is also quite expensive to compute. However, there are specific routines to speed up SVD, like the Lanczos method [10], which make the SVD more appropriate than PCA for sparse text data. Given a sparse data matrix $X_{m \times n}$ with c nonzero entries per column, the computational complexity of SVD method id $O(mcn)$ [39].

LSI *Latent semantic indexing* [39] [16] is a dimensionality reduction method only used with text document data (whereas PCA and SVD can also be used with image data). Using LSI, the document data is presented in a lower-dimensional "topic" space: the documents are characterized by some underlying (latent, hidden) concepts referred to by the terms. LSI can be computed either by PCA or SVD of the data matrix of n d-dimensional document vectors.

Some other systems for vector-space retrieval and dimensionality reduction exist and must be referenced, although their description falls of the scope of this work. For a detailed information about those systems vide [54], [57], [51], [38], [9].

2.5 Conclusions - Final Remarks

At the end of this research, one can easily conclude that the boolean model has proven to be suitable for simple exact queries, in spite of lacking in flexibility, as features like *near* or *similar* are not available. Nevertheless, many people are used to boolean queries, and so it is still widely used.

One might also conclude that the tokenization phase is not straightforward, specially if one wants to preform semantic analysis: stemming and lemmatization are complex processes, the later more than the former [34].

[12] The ones that are the most representative of the original space.

It is also important to mention that the *inverse document frequency* is a good measure of the importance/relevance of the document, since it is based on the rarity of each term, that is, a document is relevant in what concerns to a certain term if the number of occurrences of that term in the corpus, is small compared to the total number of documents.

The vector representation of a document consists on a major breakthrough, which makes it possible to perform queries like "find documents near/like" a given query document (query by example). This is done using the cosine similarity method. Since the computation of the similarities between the query document and all the corpus' documents is quite expensive, one reduces the dimension of the vector-space where the documents are represented, in order to reduce the number of calculations.

The dimensionality reduction problem has several solutions each of which with its drawbacks. From the three presented methods, the random projection method is the one with the best results (distortion to the data and the method's complexity, were the criteria according to [11]). The consulted study indicates that random projection consistently preserves the similarities between the document vectors, even when the data is projected to a considerably lower-dimensional space; the projection is yet fast to compute. It has proved to be a computationally simple method of dimensionality reduction, while still preserving, to a high degree, the similarity between the vectors. One problem that is left open is the number, k, of dimensions of the new lower-dimensional space. Although the Johnson-Lindenstrauss result [27] [20] [15] gives us bounds for this value, these are a worst case scenario. One can achieve good results using fewer dimensions but it is still a matter of study the properties that make it possible.

We conclude that the random projection is a good alternative to traditional, statistically optimal methods of dimensionality reduction that are computationally infeasible for high dimensional data. Random projection does not suffer from the curse of dimensionality , quite contrary to the traditional methods. Nevertheless, this method presents a significant number of false dismissals and a lack of precision in exact queries as a result of the *almost* orthogonal random selection of the dimensions to consider. The purpose of this work is to eliminate the false dismissal factor and still maintain the computational effectiveness of the process. This was intended to give the reader the notion of some of the possible methods, while also introducing the approach taken as reference for this work's experiments.

Chapter 3

Architecture

The purpose of this chapter is to describe the developed system, named *Hierarchical Retrieval*. The theoretical model of the system is presented, beginning with the description of the structure to represent the books followed by the indexing method used to retrieve them. Afterwards, the architecture of the *Hierarchical Retrieval* is described from an implementation standpoint, divided in three core modules with their respective functional description. Each of the presented diagrams is separated in contexts of process flow which are pictured by the coloured boxes. Then, each context encapsulates a set operations (the grey boxes) that are described in the proper sections. Next chapter will then provide a description of the application, in what concerns to the choice of the programming language as well as other options, e.g. regarding libraries and frameworks, and installation and user's guides.

3.1 Theoretical Model

The *Hierarchical Retrieval* system can divided in two theoretical concepts:

- the representation;
- the indexing.

These concepts will now be explained and integrated in the theoretical implementation of the *Hierarchical Retrieval*. Recall from the *State of the Art* the vector-space model which will be here contextualized to support the representation of the books. With the represented books, an indexing method is required to retrieve them, and so, the *GEneric Multimedia INdexIng* (**GEMINI**) method will motivate the evolution to the the *hierarchical linear subspace method*, after whom the system is named. For a better understanding of this indexing method, its application to content-based image retrieval will be briefly presented as an example of success.

3.1.1 Vector-Space Model

The reader might remember the *vector-space model* presented in the *State of the Art*, used to represent textual data in high dimensional vector-space. In the *Hierarchical Retrieval* system, the representation of the books' collection makes use of this model. Let us see how this model works in this case.

Tokenization First of all, one must choose the books which will form the collection. After the collection is gathered, the representation phase starts by tokenizing the books and creating a list of all different words in the collection. The approach taken in this system does not involve any linguistic preprocessing, word normalization or semantic interpretation. For instance, words like *jealous* and *jealousy* are considered as

completely different. This list of words or terms is then organized in a *dictionary + postings* manner, as showed in the Figure 2.1.

Scoring As we can see through the previous example, we now have a list of all different terms in the dictionary, and their respective frequency concerning each of the books they appear in (this information is given by the postings). The next step is to assign a score to each term in each of the books it appears in. This will then allow the vector representation of each document, since each of these scores/weights will correspond to each of the coordinates of the document vector.

There are two main aspects to consider when assigning a importance/relevance measure to a term:

- the importance of the term concerning the book it appears in;
- the importance of the term concerning the all collection.

The former can be achieved by modulating the term frequency (the number of occurrences of the term) with the following equation (already presented in 2.1) which avoids an exaggerated growth of the frequency function through the logarithmic function:

$$wf_{t,d} = \begin{cases} 1 + log(tf_{t,d}) & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The later is represented by the *inverse document frequency* measure given by the following equation (already presented in 2.2):

$$idf_t = log\frac{N}{df_t} \tag{3.2}$$

Here, N corresponds to the number of books in the collection and df_t is the number of documents in which the term t can be found.

Combining these two measures, we get the $wf\text{-}idf_{t,d}$ weighting measure defined by:

$$wf\text{-}idf_{t,d} = wf_{t,d} \times idf_{t,d} \tag{3.3}$$

which takes into account the two previously mentioned aspects of a term's importance. The values computed by this equation will correspond to the importance of a certain term in a book, also concerning all the collection. In the vector-space model, each of these values correspond to a coordinate of the vector representation of a certain book.

Similarity Measure One compelling reason to represent books in a vector-space is to handle queries as vectors, addressing the *query by example* problem, converting the query book into the collection's vector-space and comparing it with the other books.

The comparison process aims to quantify how different two given books are, in other words, since we are dealing with their vector representation, we want to know how far they are from each other. This distance can be obtained through the magnitude of the difference between the two vectors however, bigger books will always be similar to each other by virtue of length instead of topic.

In order to eliminate relative size of the vectors/books, they are all normalized to length one. As a consequence, the similarity measure is based on the angle between the vectors, allowing the distance between the books to computed through the *euclidean distance*.

The **Euclidean distance** or **Euclidean metric** is the "ordinary" distance between two points that one would measure with a ruler, which can be proven by repeated application of the *Pythagorean theorem*. The **Euclidean distance** between points $P = (p_1, p_2, ..., p_n)$ and $Q = (_1, q_2, ..., q_n)$, in an euclidean n-dimensional space, is defined as:

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{3.4}$$

We now have a representation of the books' collection which allows *query by example*, although the comparison in these high dimensional spaces through linear search is too expensive. The idea is to have an idexing method which quickly eliminates a significant amount of false candidates and leaves the high dimensional comparisons to few left false hits.

3.1.2 Generic Multimedia Indexing Method

This section is intends to present a theoretical overview of the GEMINI method based on [56] and also on [55], where an in-depth can be found.

Motivation

Recall from last chapter the need to perform queries by example or involving similarity thresholds. The approach to that subject was in the context of textual data retrieval, however, the spectrum of application can be extended to plagiarism detection, or finding similar medical records, market fluctuations analysis or even face recognition. The idea of this indexing method is to speed up searching in multimedia databases, in the contexts just mentioned.

The first thing to do is to quantify the distance between to objects by defining a distance function. Given two objects, O_1 and O_2, the distance (here also meaning *similarity measure*) between them is denoted by $d(O_1, O_2)$.

Similarity queries can fall in one of two categories:

- Whole match: Given a collection of S objects, $O_1...O_S$, and a query object Q, find the objects that are within a distance ε from Q[1].
- Sub-pattern match: In this case, the query corresponds only to part of an object from the collection. Given a collection of S objects, $O_1...O_S$, a query (sub-)object Q and a tolerance ε, identify the parts of the objects that match the query.

Some other types of queries are mentioned in [56], although they fall off the scope of this work.

Both the above mentioned categories of queries are desired to meet the following requisites:

- **Fast**: sequential scanning and distance calculation with each and every object is too slow for large databases;
- **Correct**: it should not have any false dismissals, that is, it should return all the qualifying objects (false hints are accepted since they can be discarded further);
- **Space**: it should require a small space overhead;
- **Dynamic**: easy to insert, delete and update objects.

[1] The query and the collection's objects must be of the same type.

The GEMINI Method

To introduce the basic idea of the method, let us focus on whole match queries. Thus, the problem is structured as a collection of objects with a distance/similarity function defined as $d(O_1, O_2)$, and a user specified tolerance ε and query object Q. An option is to use sequential scanning, however it might be too slow:

- the distance computation is expensive (e.g. the editing distance in DNA strings);
- the size S database might be enormous.

Hence, the approach sugested in [56] consists of two steps:

1. a *quick and dirty* test to rapidly discard the great majority of non-qualifying objects;
2. the use of spacial access methods, (**SAM**), (R-Trees, Hilbert-Curve, etc.) to achieve searches faster than sequential.

The idea is to have a feature extraction function, $F()$, which will map the high dimensional objects into a lower dimensional space, and preserve the relations between them, in other words, objects that are very dissimilar in the feature space, must also be very dissimilar in the original space. The mapping $F()$ from the original objects to k-dimensional points should not distort the distances. In fact, it would be perfect to have a total preservation of distances, nevertheless, in practice is already enough to have no false dismissals. This mapping represents the solution to improve the sequential scanning issue by lowering the cost of distances computation.

To assure the mapping really works, one should expect that if the distances in the feature space are always smaller or equal than the distances in the original space, one can determine a bound valid in both spaces. Formally speaking, this corresponds to the *Lower Bounding Lemma* which states that if the distance of similar objects in the original space is smaller or equal to ε, then it is also smaller or equal to ε in the feature space:

$$d_{feature}(F(O_1), F(O_2)) \leq d(O_1, O_2) \leq \varepsilon \tag{3.5}$$

In practice, this means that no object will be missed (*false dismissals*) in the feature space, although there will be some objects which are not similar in the original space (*false hits*). It is guaranteed to have selected in the feature space, all the objects we wanted plus some false hits, which will then be filtered from the set of resulting objects through comparison in the original space.

Concluding, and according to [56], the approach to indexing multimedia objects for fast similarity searching is, in resume:

1. Determine a distance function, $d()$, to compare the objects;
2. Find one or more numerical feature extraction functions to provide a quick-and-dirty test;
3. Prove that the distance in feature space lower-bounds the actual distance $D()$, to guarantee correctness;
4. Choose a SAM and use it to manage the k-D feature vectors.

In [56], the reader can find a thorough description of the GEMINI method as well as how it can be applied to 1-D time series and 2-D color images.

3.1.3 Hierarchical Linear Subspace Indexing Method

After describing GEMINI it is important to refer some drawbacks of this method. Note that the metric index trees only operate efficiently when the number of dimensions is small. The growth of the number of dimensions has a negative impact in the performance, since it fails to reduce the time in sequential scanning. This is explained by the fact that the volume of a sphere with constant radius, grows exponentially as the dimension increases.

As a general case of the GEMINI method, a direct consequence of the extension of the *lower bounding lemma* 3.5 gives place to the *hierarchical linear subspace indexing method* [55]. In this method, V is an m-dimensional vector-space and $F()$ is a linear mapping that obeys the *lower bounding lemma* from the vector-space V into an f-dimensional subspace U. The *extended lower bounding lemma* is stated in [55] as follows:

Let O_1 and O_2 be two objects and $F()$ the mapping of the objects into an f-dimensional subspace U. $F()$ should satisfy the following formula for all objects, where $d()$ is a distance function in the space V and d_U in the subspace U:

$$d_U(F(O_1), F(O_2)) \leq d(F(O_1), F(O_2)) \leq d(O_1, O_2) \tag{3.6}$$

The linear subspace sequence corresponds to a sequence of subspaces where $V = U_0$, and for every space $U_0, U_1, U_2, ..., U_n$:

- $U_0 \supset U_1 \supset U_2 \supset ... \supset U_n$
- $dim(U_0) > dim(U_1) > dim(U_2) > ... > dim(U_n)$
- and for the *extended lower bounding lemma* $d(U_n(x_1), U_n(x_2)) \leq ... \leq d(U_2(x_1), U_2(x_2)) \leq d(U_1(x_1), U_1(x_2)) \leq d(U_0(x_1), U_0(x_2))$.

Orthogonal Projection

In order to produce a linear subspace sequence, the original space has to be subsequently projected. In this section, it will be presented the projection method used in this thesis (and also proposed in [55]) to create the subspaces in sequence.

If the underlying vector-space is endowed with an inner product, orthogonality and its attendant notions (such as the self-adjointness of a linear operator) become available. An orthogonal projection operator in R^2 or R^3 is any operator that maps each vector into its orthogonal projection on a line or plane through the origin [7]. In Figure 3.1 from [55], the reader has a simple example of the projection method used in this thesis to create the subspaces. Note that in practice the dimensions are much higher, however, the process is exactly the same.

In fact, the distance d between the projected points in R^m corresponds to the distance d_U in the orthogonal subspace U, with dimension f, multiplied by a constant (proof in [55]):

$$c = \sqrt{\frac{m}{f}} \tag{3.7}$$

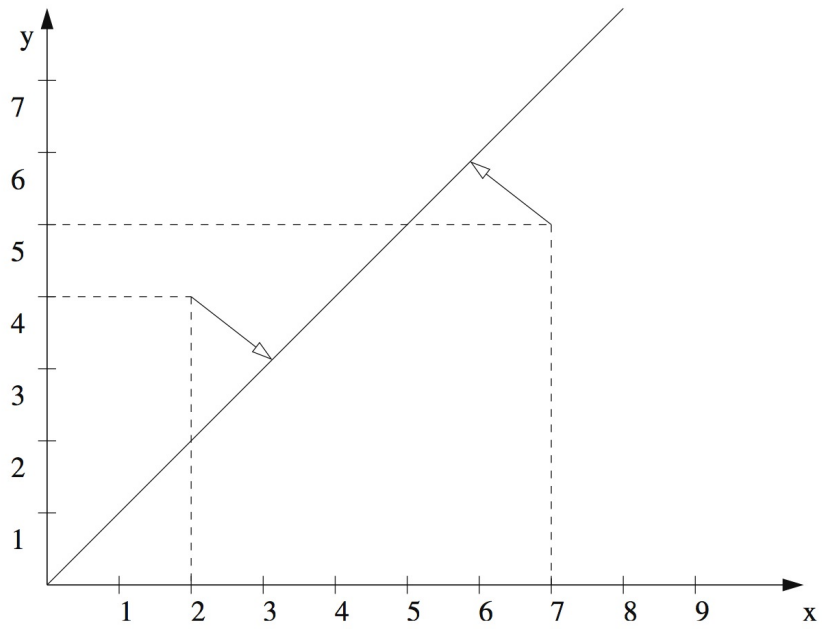


Figure 3.1: For example, the orthogonal projection of points $\vec{x} = (x_1, x_2) \in \mathbf{R^2}$ on the bisecting line $U = \{(x_1, x_2) \in \mathbf{R^2} | x_1 = x_2\} = \{(x_1, x_1) = \mathbf{R^1}\}$ corresponds to the mean value of the projected points. $\vec{a} = (2, 4)$ is mapped into $P(\vec{a}) = 3$, and $\vec{b} = (7, 5)$ into $P(\vec{b}) = 6$.

Content-based Image Retrieval

In order to provide an overview of the application of the *hierarchical linear subspace method*, it is worth mention the work developed in the field of *content-based image retrieval* (CBIR). The full description of the results and details of the method are in [55].

First, it is important to describe how the feature extraction is accomplished for images. The naive features are the scaled RGB images. The average color of an image $x = (R_{avg}, G_{avg}, B_{avg})^T$ corresponds to an orthogonal projection. As a consequence, the following equations from the *lower bounding lemma* are valid:

- $d_{avg}(F_{avg}(\vec{x_1}), F_{avg}(\vec{x_2})) \leq d(\vec{x_1}, \vec{x_2})$;
- $d_{avg}(F_{avg}(\vec{x_1}), F_{avg}(\vec{x_2})) \leq d(F_{avg}(\vec{x_1}), F_{avg}(\vec{x_2})) \leq d(\vec{x_1}, \vec{x_2})$.

As stated in [55], lowering an image's resolution represents an orthogonal projection in rectangular windows, which define sub-images of that image. In other words, the image is tiled in rectangular windows in which the mean value of the color feature is computed. This method is denominated *averaging filter*. The arithmetic mean value computation in a window corresponds to an orthogonal projection of these values onto a bisecting line. As consequence, the several resolutions of an image correspond to a sequence of subspaces which meet the requirements of the *lower bounding lemma*. The result, a set of images represented in several resolutions, corresponds to a structure called "image pyramid" [55].

The results of the CBIR work reveal that the process becomes less complex, and the mean retrieval computation costs, which are dependent on the number of images to be retrieved, are considerably lower than list matching.

Defining the Best Hierarchy

The reader might already asked himself how the best hierarchy can be found. It will be briefly described how this value is determined and the images case will be used to help the description. In fact, the number of subspaces depends not only on the number of elements but also on the threshold distance ε. It represents the maximum distance an object can be from the query to be considered a possible result. As a consequence, it also determines the maximum number of results that can be retrieved. To estimate ε and its dependency on the number of retrieved images, a mean sequence of retrieved images is defined which describes the *characteristics* of an image database.

In Figure 3.2 taken from [55], the four crescent curves correspond to each of the spaces in the hierarchy, and the straight horizontal line represents an estimate of ε. The idea is to find a line which intersects all curves in such a way that, the intersection with the highest curve (line 1) corresponds to a sufficiently high distance to retrieve a significant amount of images, and the intersection with the lowest curve (line 4) is sufficiently low to eliminate as many false hits as it is possible in the lowest dimension subspace. The formal description of this method can be found in [55].

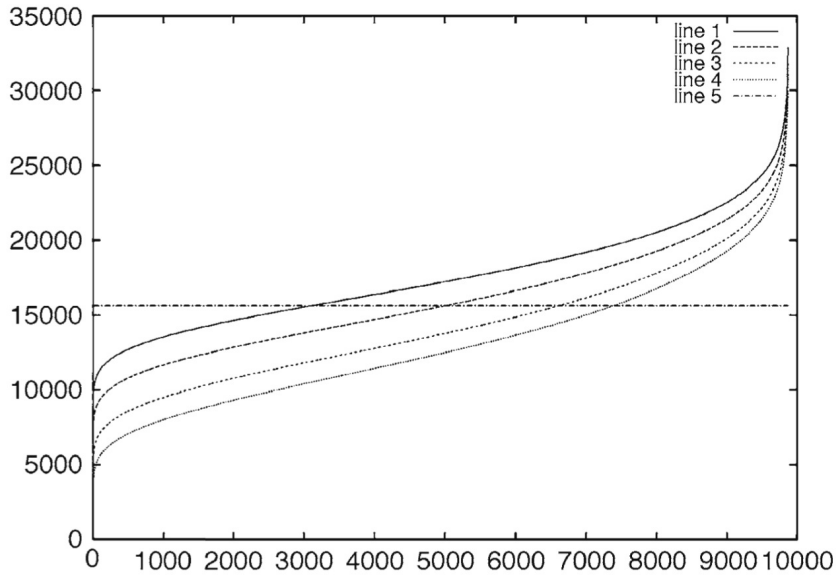


Figure 3.2: Example of characteristics plotting for an images database. The horizontal axis represents the number of images and the vertical axis corresponds to the distance.

This section intended to describe the theoretical fundaments of the *Hierarchical Retrieval*, starting with the vector-space model, then explaining the need of an indexing method and presenting the evolution of the model from GEMINI to *hierarchical linear subspace method*, and providing an overview of its application in the field of *content-based image retrieval*. Next section will present the architecture of the developed application as well as the algorithms used to support each of the prototype's functionalities.

3.2 System's Overview

The system developed to test the application of the *hierarchical subspace method* to documents, is far from being an actual software application. The reader can think of it more as a laboratory tool in which the model was tested. As so, an overview of its architecture will now be presented.

The document indexing and vector-space representation are based on the solution proposed by Manning [34] for dictionary representation and weighting measures, as well as the basis of the similarity function used to compare the documents[2]. The major difference to this thesis consists on the method used in distance computations saving, which is the *hierarchical subspace method* explained in the previous chapter.

As shown in Figure 3.3, the system consists of three core modules:

- Document indexing;
- Query document representation;
- Query processing.

There is also a XML Storage module, which is not considered a core module because it was created only as a simple solution to some performance issues. Its purpose would be better fulfilled with a simple data base.

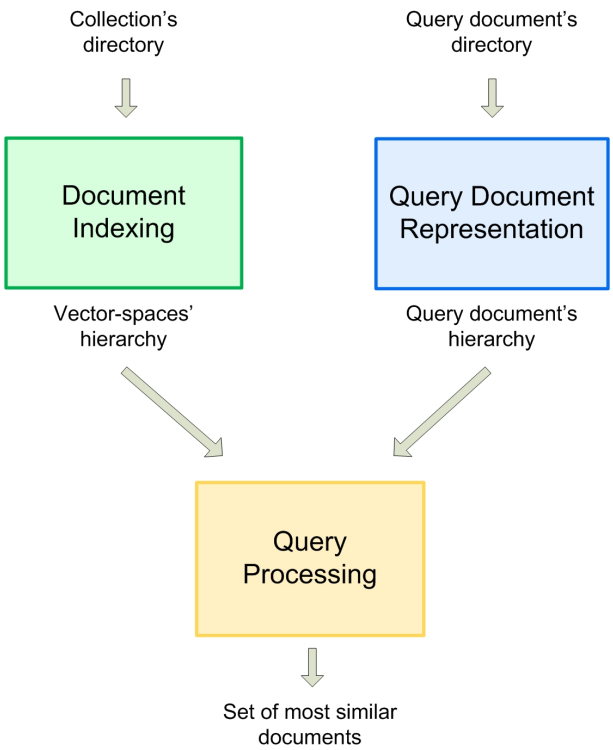


Figure 3.3: Overview of the system's architecture.

The workflow of this architecture can be easily briefly stated as follows:

1. **Document indexing**: from a set of text documents, this module creates a dictionary of terms and then converts it to a vector-space representation, in which each document is represented by a vector and each dimension corresponds to a term in the dictionary. Afterwards, the dimensions of the subspaces are computed and through subsequent projections of the original space, the space hierarchy is created.

2. **Query document representation**: in order to represent the query document in the same vector-spaces than the collection, the dictionary is reseted (further it will be explained how) and the process previously described is repeated for the query document. Here, the reseted dictionary is updated

[2]This structure is also described in the *state of the art* chapter.

exclusively with the terms of the query document that already exist in the reseted dictionary. Then vector-space representation of the query document is created and it is projected to the same subspaces of the collection's hierarchy.

3. **Query processing**: given the vector-spaces' hierarchy and the representations of the query document, the query processing consists of retrieving the most similar documents, taking into account a threshold distance to select the documents which are nearer to the query.

Now that the system is presented in separate core modules, each of them will be explained in more detail. In each of the following sections, a core module will be thoroughly described, starting by the *document indexing* which is the basis for the application of the *hierarchical subspace method*.

3.3 Document Indexing

As the reader can see in Figure 3.4, the indexing phase is composed by several steps during which, the processed data is exchanged between the Indexer and Dictionary contexts.

3.3.1 Building the Dictionary

In this section, the process of building the dictionary will be described, which in Figure 3.4 corresponds to the *Build dictionary* box, and the ones involved until the finished dictionary is returned. This process' input is the collection of text documents to be indexed. In this work, only plain text documents are supported and all the books were provided by the free electronic books producer *Project Gutenberg* [23].

In the indexer context, a list of all text files in the collection's directory is created (including subfolders), and one at a time, they are sent to be parsed in the dictionary context. Next, the steps applied in dictionary context will be described one by one, until the process of building the dictionary is finished.

Document Parsing

In the dictionary context, as soon as a new document file arrives it is parsed and added to the dictionary. In detail, this is treated as follows:

- the file is separated in lines and each line is tokenized using the white space as a separator (each token will correspond to a word in the document);
- anything that is neither a letter (majuscule or minuscule) nor a number, is removed from the token ([ˆa-zA-Z0-9], corresponds to the regular expression used to make this selection) and then all its characters are converted to minuscules;
- if the resulting token is not an empty string, the dictionary's list of terms is updated with the new token; if the token/term already exists, a new posting is created (that is, a new association between this term and the document under analysis), otherwise, a new term is created with the corresponding posting;

This process is repeated for each document file in the collection. It is worth mentioning that to each document file is associated a document identifier *docID*, to unequivocally distinguish each document. The association is made through an hash table (from Java SE 1.5 [37]) in which the key is the *docID* and the value is the *file absolute path*.

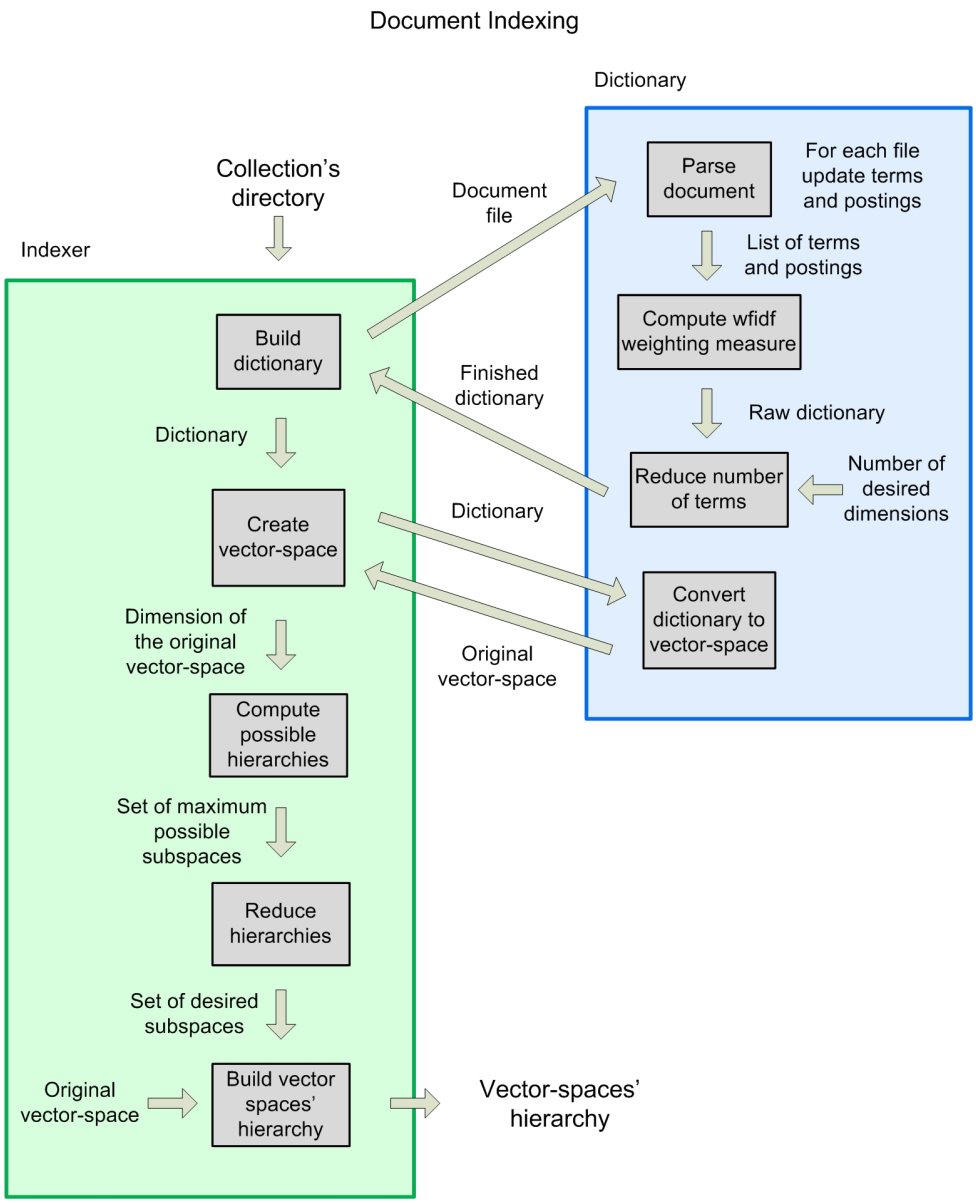


Figure 3.4: Architecture of the indexing module.

Compute *wf-idf* Weighting Measure

Once the dictionary is created and all the documents are included, we are able to compute the weights of the terms in each document. As previously explained in the *State of the Art* chapter, *wfidf* weighting consists in the product of two other weighting measures, the *weighted term frequency* (*wf*) and the *inverse document frequency* (*idf*). In this thesis, *weighted term frequency* is computed like in equation 2.1, as the logarithmic function prevents the *term frequency* value from growing exaggeratedly. The *inverse document frequency* is then computed as in equation 2.2. In this phase of the indexing, for each term of the dictionary the *idf* value is computed and it is used to compute the *wfidf* measure for each of the term's postings. It is important to mention that before computing the *wfidf* values, the logarithmic function is applied to the simple *tf* measure, the one that only represents the number of occurrences of a term in a specified document.

Reduce the Number of Terms

The raw dictionary obtained in the previous steps is most likely, too big to be directly converted to a vector-space and, as proven before, there are terms which do not define the subject of the document, and consequently, the main subjects of the collection. Hence, in this phase the number of representative terms is reduced to a number empirically defined. For instance, in a collection of 1000 books (one of the testing collections) there were 1088219 different terms that were reduced to 60000. This represents not only a reduction in the computation costs, but also the elimination of unnecessary non-representative terms. To this end, a max-heap [13] is created with all the terms and the root element (always contains the most important[3] term in the collection, obeying the max heap configuration) is removed as many times as the desired dimension for the new dictionary.

By now, the dictionary is finished and ready to be converted to the original vector-space. Before that, it is relevant to mention that both raw and reduced dictionary are stored, each of them in a XML file, for future computations, since the indexing phase is quite expensive in terms of time costs (in Appendix C the reader can find examples of these files). For a better understanding of the XML files it is important to mention that the postings are stored as a string of characters only to save memory. Each group of three values corresponds to a posting, where the first value is the *docID*, the second is the *term frequency* and third and last one is the *tfidf* weighting measure. The DTD of the dictionary's XML file is shown in 3.5.

Depending on the computer used in this phase as well as the implementation, the costs of indexing can vary, nevertheless, this process is expensive, specially in what concerns to time costs. For instance, it took about 48 hours to process the 1000 documents collection with the implementation developed in this thesis, in an Apple iMac 2.8 GHz Intel Core 2 Duo, with 2 GB SDRAM.

3.3.2 Creating the Original Vector-Space

With the final dictionary already built, now it has to be converted to the original vector-space in order to represent the documents as vectors, as described in section 2.3. The representation of a document in the dictionary is not explicit, since it is organized by terms. Therefore, to obtain the explicit representation of the documents, the information concerning their terms' weights must be gathered, by going through all the postings. The vector-space is built from the dictionary, document by document:

[3] A term is considered as more important than another one, if its mean *tfidf* value if greater than the other.

```
<!ELEMENT Dictionary (Term)*>
<!ATTLIST Dictionary
    numberOfTerms CDATA #IMPLIED
    currentDocID CDATA #IMPLIED
    corpusSize CDATA #IMPLIED
  >

<!ELEMENT Term (PostingsList)*>
<!ATTLIST Term
    name CDATA #IMPLIED
    inverseDocumentFrequency CDATA #IMPLIED
    documentFrequency CDATA #IMPLIED
    collectionFrequency CDATA #IMPLIED
  >

<!ELEMENT PostingsList EMPTY>
<!ATTLIST PostingsList
    postings CDATA #IMPLIED
    numberOfPostings CDATA #IMPLIED
  >
```

Figure 3.5: DTD of the dictionary XML file.

- for each document in the dictionary (this information is given by the number of documents in the collection, since in this case we only need the document identifier):
  - for each term in the dictionary:
    - go through all postings and, if the *docID* of the posting is the *docID* of the document that is being created, set the *tfidf* value previously computed, in the same position of the index of the term;

```
<!ELEMENT Document (documentVector)*>
<!ATTLIST Document
    spaceID CDATA #IMPLIED
    numberOfHierarchies CDATA #IMPLIED
    docID CDATA #IMPLIED
    dimension CDATA #IMPLIED
  >

<!ELEMENT documentVector EMPTY>
<!ATTLIST documentVector
    weights CDATA #IMPLIED
  >
```

Figure 3.6: DTD of the document XML file.

After each document is created, it is normalized and then, it is written in a XML file (vide Appendix C), with the name *vs<spaceID><docID>.xml*, where *<spaceID>* refers to the identifier of the vector-space in which the document is (starts in 1 for the original vector-space), and *<docID>* corresponds to its document identifier. The DTD of the document's XML file is shown in 3.6. This process runs in $O(n \times m)$, being n the number of documents and m the number of terms in the dictionary.

3.3.3 Vector-Space Hierarchy Computation

Before projecting the original space to create the hierarchy, one must first know how many hierarchies are needed, and what is the dimension of the subspaces. In order to obtain this information only the dimension of the original space[4] is required. Recall from the previous chapter that the projection method used to create the hierarchy is the *orthogonal projection*. As so, each projection consists on

[4]This dimension was chosen to be even to make the division operations easier, as it will be explained further.

gathering/grouping a specified number of coordinates from a document vector, and compute their mean value. Suppose the vector $v =< 1, 2, 3, 4, 5, 6 >$ represents a document in a $6 - dimensional$ space. If the objective is to project it into a $2 - dimensional$ space, our grouping would be of three coordinates (6 original dimensions divided by 2 dimensions for the subspace), and the resulting projected vector would be $v_p =< 2, 5 >$.

Now, back to obtaining the required information for the hierarchy. The process consists of two main steps:

1. compute all possible subspaces, that is, the largest hierarchy considering the given original space;
2. reducing the previously obtained hierarchy to match the desired number of subspaces.

In the first step, given the dimension of the original space, *spaceDim*, this value is divided by the smallest possible integer, which corresponds to *groupingSize*, until *spaceDim* is sufficiently small (dimension 2, 3, or smaller or equal to *groupingSize*), or the result of the division is a prime number greater than *groupingSize*. After each division, the dimension of the new subspace and the coordinate grouping necessary to the projection are stored in *subspaces* and *groupings* collections, respectively. The following pseudocode shows in more detail the explained algorithm.

Compute hierarchy spaces' dimensions Creates the dimensions and groupings collections for the hierarchy construction.

input:$spaceDim$

```
aux = 0;
groupingSize = 2;
while(spaceDim ≠ 2 & spaceDim ≠ 3 & spaceDim > groupingSize)
{
  aux = spaceDim/groupingSize;
  if((spaceDim mod groupingSize) == 0 & isPrime(aux) & aux > groupingSize)
  {
    spaceDim = spaceDim/aux;
    groupings.add(aux);
    subspaces.add(spaceDim);
    break;
  }
  if{(spaceDim % groupingSize)! = 0)
    groupingSize + +;
  }
  else
  {
    spaceDim = spaceDim/groupingSize;
    groupings.add(groupingSize);
    subspaces.add(spaceDim);
    groupingSize = 2;
  }
}
```

Reduce hierarchy's size From the groupings and subspaces obtained by computeSpaceHierarchy(), reduce the hierarchy size according to the experimental parameter hierarchySize, which will usually be 4.

input:$hierarchySize$

$i = 0;$
$size = groupings.size();$
$for(i = 0; i < (size - hierarchySize); i++)$
$\{$
$\quad former = groupings.get(size - i - 2);$
$\quad later = groupings.get(size - i - 1)$
$\quad groupings.remove(size - i - 1);$
$\quad groupings.set(size - i - 2, former * later);$
$\quad subspaces.set(size - i - 2, this.subspaces.get(size - i - 1));$
$\quad subspaces.remove(size - i - 1);$
$\}$

In the second step, reducing the hierarchy, the objective is to shrink the number of subspaces to match *hierarchySize*. To this end, the two last grouping values are merged (multiplied) and the subspace corresponding to that new grouping, is the last subspace in *subspaces*. For a more detailed understanding, the respective pseudocode pictures the complete algorithm.

Compute space hierarchy Once the information about the desired subspaces is computed, the space hierarchy can now be built. Therefore, each document in the original space is projected to the space specified in *subspaces* collection. The projection operation takes the dimension of the subspace (from *subspaces*) and the corresponding grouping value (from *groupings*), and for each *grouping value* number of elements (this elements are the weights in the document vector) compute their mean value, which will correspond the value of the coordinate in the new subspace. After a vector-space is projected, the same process is done for the subspace that was just created, using the information about the next subspace to be computed, until the hierarchy is completed. For a simpler example of the projection process vide 3.1.3.

3.4 Query Document Representation

The creation of the query document representation is illustrated in Figure 3.7 and, as one can easily observe, has some similarities with the indexing phase, since it uses some of the structures previously created, as will be shown next.

3.4.1 Reseting the Dictionary

Representing the query document in the same vector-spaces of the collection is nothing more than representing another document in the collection, only in this case, the collection cannot be modified. The fact of using the previously created structures must not alter values that are collection dependent, like the *inverse document frequency* or the number of terms (the original space's dimension), otherwise the collection was being updated instead of just being queried. To this end, the dictionary reseting operation consists of reseting each of its terms, maintaining both the number of terms and the number of documents in the collection (the query document must not be considered an extra document in the

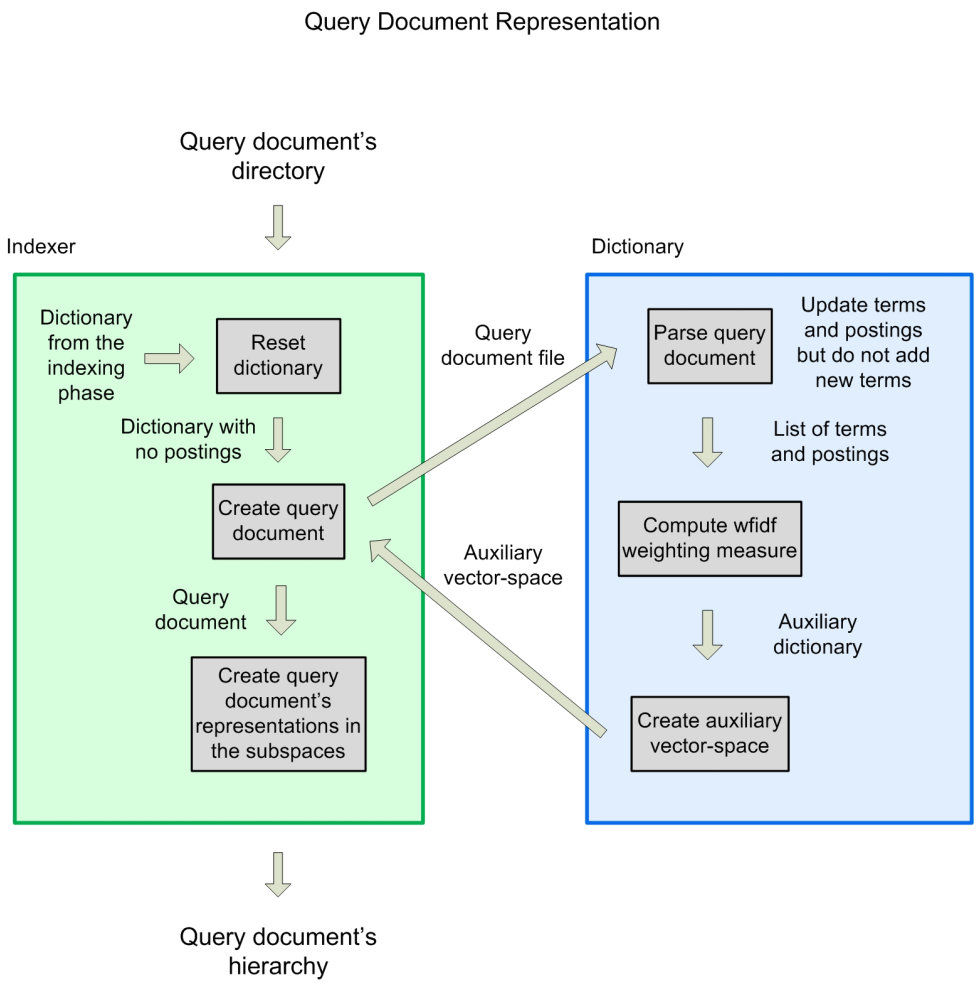


Figure 3.7: Architecture of the query document representation module.

collection, otherwise, each a query was preformed all the weighting measures should be recomputed, and also it would not correspond to the concept of a query). The process of reseting a term consists of eliminating all its postings as well the *collection frequency* and *document frequency* values. The result of this process is a dictionary with no postings (vide Appendix C).

3.4.2 Query Document Creation

Just as in the indexing phase, the creation of the query document representation consists of two major steps:

- creating the dictionary with the document;
- creating the representation in the original vector-space.

Here, the main objective is to obtain the representation of the query document in the original vector-space. Although there are several similarities, the following description gives a perspective of the algorithms used in the indexing phase, focused on the creation of the query document.

Parsing the Query Document

The process of parsing the query document is no different from parsing the documents from the collection, except for the fact that the dictionary must not be altered, that is, no terms shall be added during this process. As explained for the indexing phase, the query document file is separated in lines which are then tokenized using the white space as a separator. Each token/word is normalized, that is, all its characters are converted to minuscules and any present punctuation is removed. Finally, if the resulting token is not an empty string the dictionary is updated with the word. In contrast with the collection processing, here if the word does not exist in the dictionary it is discarded, only new postings are added or updated.

Computing Weighting Measure

After the document file is parsed, the *wfidf* weighting measure must be computed. Recall that this weighting measure is the product of the *weighted term frequency* and the *inverse document frequency* (vide 2.4). It is also important to remember that the *idf* value is collection dependent (vide 2.2), therefore the value used to compute the *wfidf* of the query document is the one computed in the indexing phase (that is why it is left intact when reseting the dictionary). Hence, the query document's *wfidf* values are computed with the collection's *idf* and its own *wf* values, making it possible to compare the importance of the terms of the query document within the collection's universe of terms.

Create the Auxiliary Vector-Space

As the reader can remember from the indexing phase, after the dictionary is created the vector-space can be built. Here the conversion is quite similar. For each term and for every existing posting, store the *wfidf* value of that posting in the document vector position corresponding to the current term. If there is not a posting for that term, that position of the document vector if simply set to zero. In this case, it is not necessary to write the query document in the XML file in contrast to the indexing phase.

Once the auxiliary vector-space is created, and to complete the objective of this phase, the query document is removed from the vector-space (the space is not needed anymore) and is normalized in the same way the collection's documents were.

3.4.3 Represent the Query Document in the Subspaces

By now, one already has the representation of the query document in the original vector-space, however, the representation in the subspaces is still needed. To this end, using the information present in the groupings and subspaces collections, the query document is projected into each of the subspaces of the hierarchy, creating a hierarchy of query document representations. The projection operation is exactly the same as the one described in the creation of the vector-space hierarchy.

3.5 Query Processing

At this time, the collection is represented in the vector-space hierarchy which supports the query processing. The detailed process is pictured in Figure 3.8, and as the reader can observe, in this process there is already user intervention to specify some operations. The query processing module will now be described by functionality. First, the *top importance terms retrieval* is presented, and afterwards the *document retrieval method*. The later is then separated according to the two used methods:

- linear search method;
- hierarchical subspace method.

Finally, the characteristics computation is explained.

3.5.1 Retrieving Top Importance Terms

The first functionality to be presented is the simplest, within the query processing, and yet, quite important as the reader shall see.

In the indexer context, the user begins by selecting the *preform query* operation and in the query engine context, selects the *retrieve top importance terms* operation followed by the number of terms to be retrieved, k. Recall the process of reducing the size of the dictionary (vide 3.3.1) where, using a max-heap data-structure, the most representative terms of the dictionary/collection were retrieved to create the final dictionary. By now, the reader is probably guessing the similarities between this two processes. In fact, to retrieve the desired top importance terms, no vector-space hierarchy is needed. Just like in the dictionary reduction, a max-heap is created from the dictionary, and k elements are retrieved from the root of the heap (in a max-heap, it always contains the element with the highest value [13]), which are, finally, returned to the user.

3.5.2 Document Retrieval

The document retrieval also begins with the selection of the *preform query* operation, however, the process is a bit more complex. As previously stated, the idea of document retrieval is to preform queries with documents, that is, given a query document q and a collection of documents S, find the set of documents most similar to the query. Here, will be presented two methods to achieve this result set:

- linear search method;
- hierarchical subspace method.

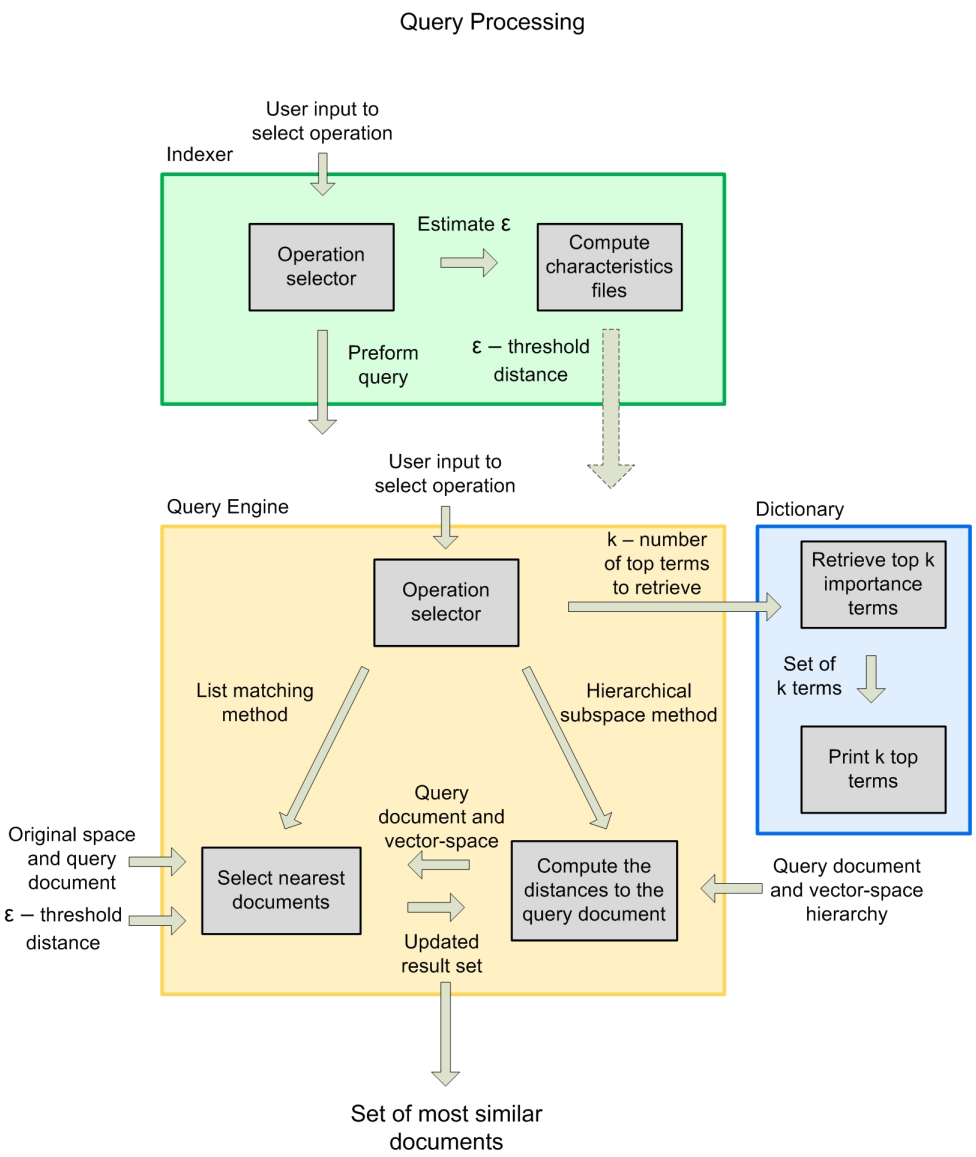


Figure 3.8: Architecture of the query processing module.

Linear Search Method

The linear (also called list matching) search method is the simplest *brute-force* method to compare a given element against a set of elements of the same kind, since it has to compare all the elements in the highest dimension, that is, in the original vector-space.

After selecting the *preform query* operation, the user selects the *list matching* method and is asked to input a threshold distance ε, which will be explained further (vide 3.5.3) how to be determined. Recall from the previous chapter that threshold distance represents the maximum distance a collection document can be from the query document, to be considered a possible result.

In the query engine context, as soon as the document retrieval is initiated, the set of result documents is initialized with all the documents in the collection as it is initially assumed that they are all plausible candidates. The selection of the nearest documents has, in this case, the original vector-space and the query document as inputs, as showed in the following pseudocode. Note that the distance function used to compare the documents is the *euclidean distance*, as described in 3.4.

Select nearest documents Given a vector space, a query document, removes from the result set the documents of the vector space which are not within the specified threshold distance from the query document.

input:$vectorSpace, queryDoc$

```
resultSetSize = resultSet.size();
c = sqrt(originalSpaceDim/vectorSpace.getDim())
foreach document in resultSet
{
  currentDoc = vectorSpace.get(document);
  distance = computeEuclideanDistance(queryDoc, currentDoc) * c;
  if(distance >= epsilon)
  {
    resultSet.remove(document);
  }
}
```

Hierarchical Subspace Method

In contrast to the previously presented method, the hierarchical subspace method intends to make use of the vector-spaces' hierarchy to gradually eliminate the false candidates from the result set. This process starts, just as before, by initializing the result set with all the documents in the collection. Then, starting with the lowest dimension subspace in the hierarchy, select the nearest documents to the query. This selection is repeated for each of the subspaces and finally, with the original vector-space. The objective of this process is, as explained in the previous chapter, to gradually eliminate from the result set the documents that are far from the query, according to the distance function and the threshold distance. To this end, the algorithm presented before (select nearest documents 3.5.2) is executed with all the subspaces of the hierarchy and the respective representation of the query document. Through this process, each time the algorithm finishes the result set is updated, eliminating some of its false hints, and consequently, the next execution has less documents to compare with. Therefore, when the selection

of the documents is to be done in the original space, there is only a small number of documents in the result set which are to be removed (vide 3.1.3). The following pseudocode shows the detailed algorithm.

Retrieve most similar documents Given a vector space hierarchy and the query document hierarchy, find the set of most similar documents to the query.

input:$vectorSpacesHierarchy, queryDocsHierachy$

```
resultSetSize = resultSet.size();
foreach vectorSpace in vectorSpacesHierarchy
{
  queryDoc = queryDocsHierarchy.getRepresentation(vectorSpace.getDimension());
  selectNearestDocuments(vectorSpace, queryDoc); /*vide 3.5.2*/
}
```

Once this process is finished the information about the documents in the result set is returned to the user.

3.5.3 Computing Characteristics Files

At this moment, it has already been explained the query processes and, as the reader might remember, it was mentioned that a threshold distance ε, was asked as an input. This value is estimated in the module that will now be described, and this shall be done before proceeding to the document retrieval, in order to get an accurate ε value.

In the indexer context (Figure 3.8), the user selects the *estimate* ε operation which will start the computation of the characteristics files. This process consists of computing, for each vector-space in the hierarchy, the distances between all the documents, as showed in the following pseudocode.

Compute characteristics files

input:$vectorSpacesHierarchy$

```
foreach vectorSpace in vectorSpacesHierarchy
{
  foreach document in vectorSpace
  {
    distances = computeDistances(document);
    distances.sort();
    writeFile(distances);
  }
}
```

Compute distances Given a document in a certain dimension, computes the distance between the given document and all the documents in the collection.

input:$document, vectorSpace$

```
c = sqrt(originalSpaceDim/vectorSpace.getDim())
docDistances = new List();
foreach currentDoc in vectorSpace
{
  distance = computeEuclideanDistance(document, currentDoc) * c;
  docDistances.add(distance);
}
return docDistances;
```

In the *compute characteristics files* 3.5.3 routine, for each document in each vector-space, call the *compute distances* 3.5.3 routine. The later will compute the distance[5] between the input document (a document from the current vector-space) and all the other documents in the current vector-space, including itself. The former will take the set of distances produced by the later, sort it, and write it to a text file which will produce a plotting like the one in Figure 3.9. The names of the files are assigned as $distances < docID >< spaceID > .txt$. The process is repeated for every document in every subspace, resulting in a set of files like the one presented.

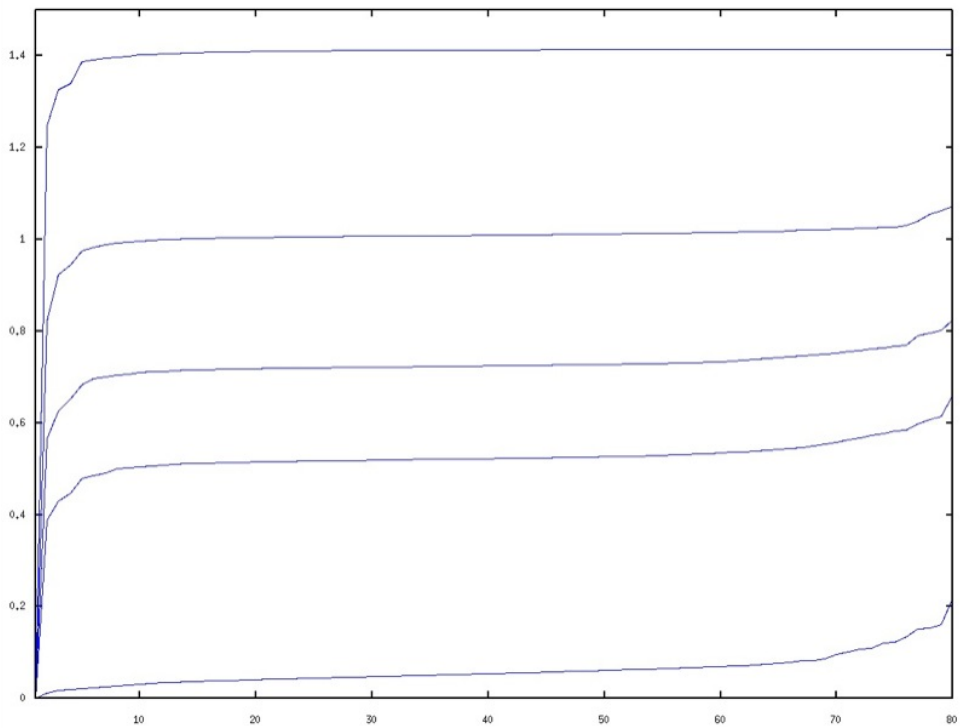


Figure 3.9: Screenshot of a distances file used to compute the characteristics.

After the files are created, and to finally estimate ε, an application is used to plot the graphics that will allow an estimation. The plotting is done as follows:

- for each vector-space:
  - compute the mean of all distance files of all documents

[5] Just as before, the euclidean distance function is used, and the the resulting distance is projected in the original space when multiplied by the constant c (vide 3.7).

- each vector-space will have a curve corresponding to the distribution of the documents in that space.

The dotted arrow in Figure 3.8 means that determining ε depends on the user. All the details of the estimation process are explained in the previous chapter.

This chapter intended to provide the reader with a thorough description of the entire process of document indexing and query processing, explaining all modules and algorithms developed to support the prototype's purpose. It should also give an understanding of the documents representation's evolution, as well as the differences between the methods used to retrieve the most similar documents.

Next chapter will describe the prototype in detail, justifying the choice of the programing language, presenting an application high-level architecture diagram and frameworks and libraries used.

3.6 The Prototype

The objective of this section is to motivate the choice of Java as the programming language and to briefly present the frameworks and libraries used in the development of this prototype. But first, Figure 3.10 presents a simplified application's high-level class diagram, for the reader to get an idea of how the architecture was implemented.

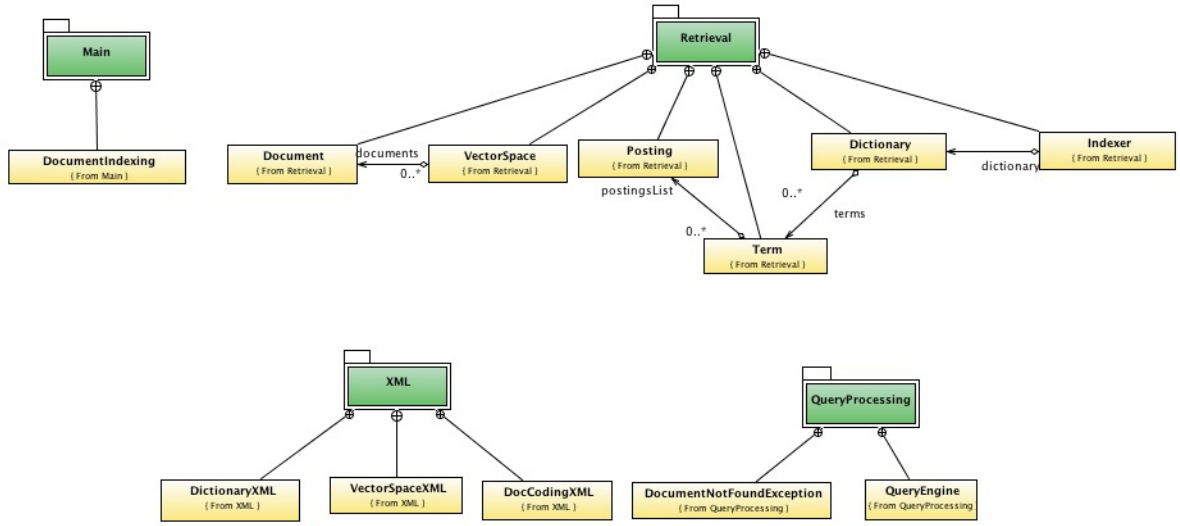


Figure 3.10: High-level application's class diagram.

3.6.1 Programing Language

As the developed prototype was never intended to be an actual application, the choice of the Internal Development Environment (IDE) was more a mater of the developer's habit. NetBeans IDE [35] is a software development product for computer programmers, sufficiently versatile to assure a myriad of the developer's needs, supporting more than one programming language, particularly Java applications to run on platforms supported by the several Java frameworks [36].

Once the IDE was chosen, the next decision was related to the programming language. As a consequence of choosing NetBeans, C++ and Java arose as the main candidates, although C had also been considered

an alternative. The two main reasons for choosing Java were the ease of migration between operating systems and mostly, the developer's experience in this programming language comparing with any of the alternatives.

3.6.2 Frameworks and Libraries

During the development of this prototype only one library was used, the **commons-collections-3.2** from Apache Software Foundation [19]. Among several data-structures, the purpose of this library was to provide a heap data-structure for dictionary reduction and the retrieval of the top importance terms, as explained in 3.3.1 and 3.5.1, respectively.

In Appendix B the reader can find a simple guide to install and use the developed system and in Appendix C there are some screenshots of the interaction with the *Hierarchical Retrieval*.

Chapter 4

Performance Evaluation

After the full description of the *Hierarchical Retrieval*, and since this thesis represents an experiment to assess the performance of the described model, this chapter aims to present and interpret the results of the evaluation study carried out in this work.

We begin by presenting the objectives and the methodology of this tests. The results presentation starts with a quality evaluation of the implemented model. Then the performance results are organized as follows:

- First, the retrieval results with the list matching method;
- Second, the results of with the implemented speed up method, the *hierarchical linear subspace method*;
- Finally, the conclusions taken from the presented results.

4.1 Objectives and Methodology

Objectives These tests were intended to evaluate the *hierarchical linear subspace method's* as a speed up in what concerns to books retrieval. To this end, an evaluation study was conceived with two main objectives:

- Evaluate the correctness/quality of the model's application to books using an element of the collection as a query, i.e. given a query book from the collection, the presented results' list should correspond to the query book in the first place with distance zero, and all the other books in decreasing order of similarity.
- Assess if the *hierarchical linear subspace method*, when applied to books, represents a breakthrough in terms of performance when compared to the *linear search method*, just as in the *content-based image retrieval* field [55].

Methodology In order to test the correctness of the *hierarchical linear subspace method's* application to books, a small collection from known authors was selected from *Project Gutenberg* [23] (vide Appendix A for the full list of books). The indexing structure corresponding to these documents was built and several queries were made using collection's books as the query. In the application the selected option was always the *hierarchical subspace method*. The presented results were, then, observed and it was checked if the distance from the query document to its copy in the the collection was zero. Finally, the rest of

the returned books were submitted to a subjective evaluation by inspection to see if the ones which were nearer, had similar subjects[1].

In what concerns to the performance of the *hierarchical linear subspace* method, the *characteristics* method described in 3.1.3 was used to assess whether or not it is applicable to books, in the sense that, just as with images [55], it represents a breakthrough in terms of performance when compared to the *linear search* method. Therefore, one thousand books were randomly selected from the *Project Gutenberg DVD: The July 2006 Special* (with about 17,500 titles [23]), to represent the test collection. This collection was tested in 7 different situations, using the complete collection once, and subsets on the remaining situations. According to the *characteristics* method, one attempted to estimate the optimal number of subspaces (e.g. the size of the hierarchy) and the threshold distance ε, in each of the testing scenarios.

These experiments it were carried out in an *iMac* 2.8 GHz Intel Core 2 Duo, with 2 GB of RAM memory, running Mac OS X 10.5.4.

4.2 Quality

The subject of this test was the correctness of the model, in other words, if the similarity measure used to compare the books had the desired semantic meaning. This semantic meaning refers to all explicit and implicit aspects of a textual document that characterizes it, which can go from the subject or author, to the writing style in what concerns to the vocabulary used[2].

In order to perform these testings, some known authors were selected and form each of them, some books were selected (some of them were known, others not). In Appendix A, the reader can find the complete list of 76 books from the selected authors:

- Jane Austen;
- Friedrich Nietzsche;
- William Shakespeare;
- Sir Arthur Conan Doyle.

When a query is performed the user wants to actually retrieve the collection's books which are most similar to the query. In Appendix C can be found two examples of the queries performed to test this item. The output is organized in the following way:

- the number of retrieved books;
- the number of performed operations;
- for each retrieved book shows the book identifier (*docID*), the respective distance from the query book and the absolute file path of the result.

For a simpler understanding of the example results, Table 4.1 shows the same queries but only with the books' titles in decreasing order of importance (the order of the output).

In both query results, one can see that the distance from the query document to its copy in the collection is zero. By observation of the remaining results, one can also notice that not only the most

[1] Note that the results are displayed in decreasing order of similarity.

[2] Note that this only refers to the different words used in the document, since their order is irrelevant as the vector representation does not take that in consideration.

Query Book	*The Hound of the Baskervilles*	*Pride and Prejudice*
Results	*The Hound of the Baskervilles*	*Pride and Prejudice*
	The Adventure of the Dying Detective	*Sense and Sensibility*
	The Adventure of the Devil's Foot	*Northanger Abbey*
	The Adventure of the Cardboard Box	*Mansfield Park*
	The Adventures of Sherlock Holmes	*Persuasion*
	The Adventure of Wisteria Lodge	*Emma*
	The Disappearance of Lady Frances Carfax	*Love and Friendship*
	The Adventure of the Bruce-Partington Plans	*Lady Susan*
	Danger! and Other Stories	*Victorian Short Stories of Troubled Marriages*
	The Return of Sherlock Holmes	*The Great Boer War*
	Tales of Terror and Mystery	*A Duet*
	The Green Flag	*The Captain of the Polestar*
	My Friend The Murderer	*The Hound of the Baskervilles*
	The Adventure of the Red Circle	*Beyond the City*
	Sign of the Four	*A Lover's Complaint*
	His Last Bow	*Through the Magic Door*
	A Duet	*Tales of Terror and Mystery*
	The Adventures of Gerard	*The Refugees*
	Northanger Abbey	*The Poison Belt*
		The Return of Sherlock Holmes
		The Firm of Girdlestone
		The Cabman's Story
		The Tragedy of Antony and Cleopatra

Table 4.1: Results obtained in two of the performed quality queries, with outputs pictured in Appendix C.

similar books are from the same author, but also, they fall under the same genre (*detective* for emphThe Hound of the Baskervilles; *love/drama* for *Pride and Prejudice*). Hence, one can conclude that the model is correct, that is, the expected books are returned and so the similarity measure is correct.

4.3 Performance

After attesting the quality of the *Hierarchical Retrieval*, one must now evaluate its performance. The idea is to compare the time and number of operations of the *list matching method* and the *hierarchical linear subspace method*, and see if the latter represents a speed up when compared to the latter.

The collection used to test the performance of the system was a random selection from the *Project Gutenberg DVD: The July 2006 Special*. As the reader shall see, in each of the testing scenarios a subset of this collection was used.

4.3.1 List Matching

The currently known method for retrieving high dimensional data is the *list matching method*, which consists on sequentially compare the query with all the high dimensional data objects. In the *Hierarchical Retrieval* system this method was also implemented, in order to have a more accurate comparison. To this end, two collections of books were selected to register the retrieval performance with this method:

- the collection of books used in the quality testing (vide Appendix A);
- a subset of 500 books randomly selected from the 1000 books collection.

Query Book	Time	Number of operations
The Tragedy of Antony and Cleopatra	35s	76
The Antichist	35s	76
The Hound of the Baskervilles	31s	76
The Adventure of the Dying Detective	15s	76
His Last Bow	10s	76
Pride and Prejudice	58s	76
The Return of Sherlock Holmes	58s	76
The Tragedy of Romeo and Juliet	16s	76
Sense and Sensibility	46s	76
Thus Spake Zarathustra	45s	76
Mean Value	34.9 s	76
Standard Deviation	17.2591 s	0

Table 4.2: This table shows for each query book, the time and number of operations taken with the *list matching method* and the 76 books collection.

Query Book	Time	Number of operations
Loues Labour's lost	33s	500
The Argonautica	59s	500
Detective Stories from Real Life	58s	500
Julius Caesar	25s	500
The Ethics	29s	500
The Diary of a U-boat Commander	49s	500
The Reconciliation of Races and Religions	42s	500
The Analysis of Mind	61s	500
The Burgess Animal Book for Children	58s	500
I and My Chimney	18s	500
Mean Value	43.2 s	500
Standard Deviation	16.0264 s	0

Table 4.3: This table shows for each query book, the time and number of operations taken with the *list matching method* and the 500 books collection.

To perform the queries, two testing query sets were randomly selected from each of the collections. Table 4.2 shows the time and number operations of the results of the queries to the small collection, and their respective mean value and standard deviation. As expected, the number of operations remains constant for all queries, since the *list matching method* compares all elements in the collection in the high dimensional space, hence the number of operations (76) corresponds to the number of books in the collection.

In table 4.3 are listed the results of the queries in the larger collection. As one can see, in this case the number of operations (500) is also constant and equal o the number of books in the collection, meaning the same number of books is compared in all the queries. Just as before, all books in the collection are compared with the query since the same method is used.

4.3.2 Speed Up

For a small collection with not a very high dimension (in the queries made before the dimension was 10000) the required time to perform a query is not exaggerated. However, if one is dealing with bigger collections (with more than 1000 books, where the dimension easily reaches 50000 or 60000), the cost of the number of operations reflects itself in the time taken to perform a query. In order to overcome the high dimensionality problem, the *Hierarchical Retrieval* creates a hierarchy of spaces and uses the lower dimensional spaces to gradually eliminate false candidates.

Let us now see how the *hierarchical subspace method* behaves with this collections. In Table 4.4 is shown the performance values for the small collection. Since there are five spaces in the hierarchy, the number of operations (380) corresponds to the number of books (76) multiplied by the number of spaces. This means that for every space in the hierarchy, the query is compared against all the collection, hence the false hits were not excluded in the lowest dimensional space.

Table 4.4 shows the results of this test for the larger collection. Exactly as before, there are five spaces in the hierarchy that result in the 2500 operations, reflecting the comparisons with the all 500 books in the different spaces of the hierarchy. Again, this is a consequence of not discarding the false hits in the lowest dimensional spaces.

Query Book	Time	Number of operations
The Tragedy of Antony and Cleopatra	51s	380
The Antichist	51s	380
The Hound of the Baskervilles	48s	380
The Adventure of the Dying Detective	33s	380
His Last Bow	16s	380
Pride and Prejudice	77s	380
The Return of Sherlock Holmes	77s	380
The Tragedy of Romeo and Juliet	29s	380
Sense and Sensibility	77s	380
Thus Spake Zarathustra	75s	380
Mean Value	53.4 s	380
Standard Deviation	22.5792 s	0

Table 4.4: This table shows for each query book, the time and number of operations taken with the *hierarchical subspace method* and the 76 books collection.

Query Book	Time	Number of operations
Loues Labour's lost	39s	2500
The Argonautica	65s	2500
Detective Stories from Real Life	71s	2500
Julius Caesar	38s	2500
The Ethics	40s	2500
The Diary of a U-boat Commander	71s	2500
The Reconciliation of Races and Religions	54s	2500
The Analysis of Mind	85s	2500
The Burgess Animal Book for Children	72s	2500
I and My Chimney	31s	2500
Mean Value	56.6 s	2500
Standard Deviation	18.6261 s	0

Table 4.5: This table shows for each query book, the time and number of operations taken with the *hierarchical subspace method* and the 500 books collection.

Although the differences between the query times are not exaggerated (due to the fact that the dimension of the original space is not very high), the number of operations transmit an accurate cost in the worst case scenario, which reveals that no false hits are discarded in the lowest dimensional spaces. As so, it will now be presented a proof of this fact by analyzing the characteristics plotting.

The graphics shown below are presented in chronological order, since each of them represents major change to the application, during the test phase. Table 4.6 describes the circumstances under which each of the tests were done. Each line corresponds to a different graphic, and the columns correspond to the criteria used to describe each test situation according to (exactly by this order) the number of books used in the testing collection, if the original space is normalized or not, if the term-frequency measure makes use of the logarithmic function, which precision floating-point is used (single or double), the dimension of the original space, and finally, the number of spaces in the hierarchy.

In each of the graphics, just as described before for pictures in the theoretical chapter, the vertical axis represents the distances and the horizontal axis corresponds to the number of books.

	Books	Normalized	log(tf)	Floating-point	Original space's dim.	Hierarchy size
Figure 4.1	1000	yes	no	single	60000	5
Figure 4.2	80	yes	no	double	60000	5
Figure 4.3	80	yes	no	double	60000	8
Figure 4.4	500	yes	no	double	60000	5
Figure 4.5	500	no	no	float	60000	5
Figure 4.6	500	yes	yes	float	10000	5
Figure 4.7	500	no	yes	float	10000	5

Table 4.6: Description of testing circumstances for each of the resulting graphics showed in the Figures below.

Recall from the theoretical chapter, in order to estimate ε and choose the best hierarchy, one must find an horizontal line which intersects the characteristics curves as shown in the image retrieval example 3.2. As the reader can see in the graphics from the books' experiments none of them permits the definition of such a line.

First, as the experiments done with the large collection resulted in the graphic from Figure 4.1, one thought it could be a consequence of an error introduced by rounding and consequently, all the calculations

were changed to use a double precision floating-point, resulting in the graphics shown in 4.2, 4.3 and 4.4.

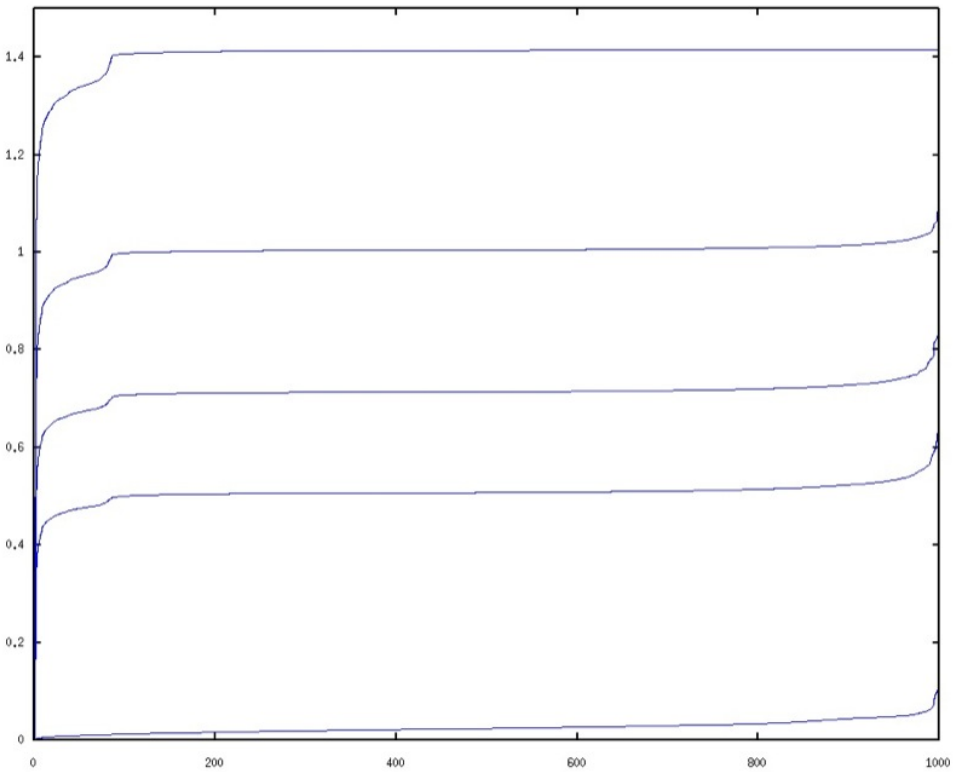


Figure 4.1: Characteristics plotting for a 1000 books normalized collection.

Again, a proper intersection line could not be found, even experimenting with a different hierarchy configuration 4.3. After these tests one concluded it was not from the rounding and all the computations were again changed to single precision floating-point, since the computations were a lot faster (almost half of the time taken with double precision). The hypothesis was that the problem consisted in the normalization of the original space. Thus, the same collection was tested without normalizing the original space resulting in the graphic shown in 4.5 with still no improvement.

The final hypothesis consisted in the idea that the original space was too sparse. This was motivated by the fact that the characteristics graphic, a representation of the distribution of the objects in the space, is not represented by a smooth curve. Also the same fact was stated by *Magalhães* and *Ruger* in [33]. As so, the dimension of the original space was reduced to 10000 terms and another two tests were performed, with and without normalizing the original space, although this time the *term frequency measure* was computed using the logarithmic function as in 2.1.

4.3.3 Conclusions

Based on the performed tests and since one still was not able to define a line to estimate ε, as the reader can see in 4.6 and 4.7, using the methods and algorithms of this thesis, the *hierarchical linear subspace method* does not represent a breakthrough in terms of performance when compared to the *linear search method*. Moreover, one can conclude from the fact of the space being too sparse, that the *hierarchical subspace indexing method* does not works for books using the orthogonal projection method.

Another important fact is the number of operation previously seen. In each query that was performed, using the *hierarchical subspace indexing method* all the books in the collection were compared to the

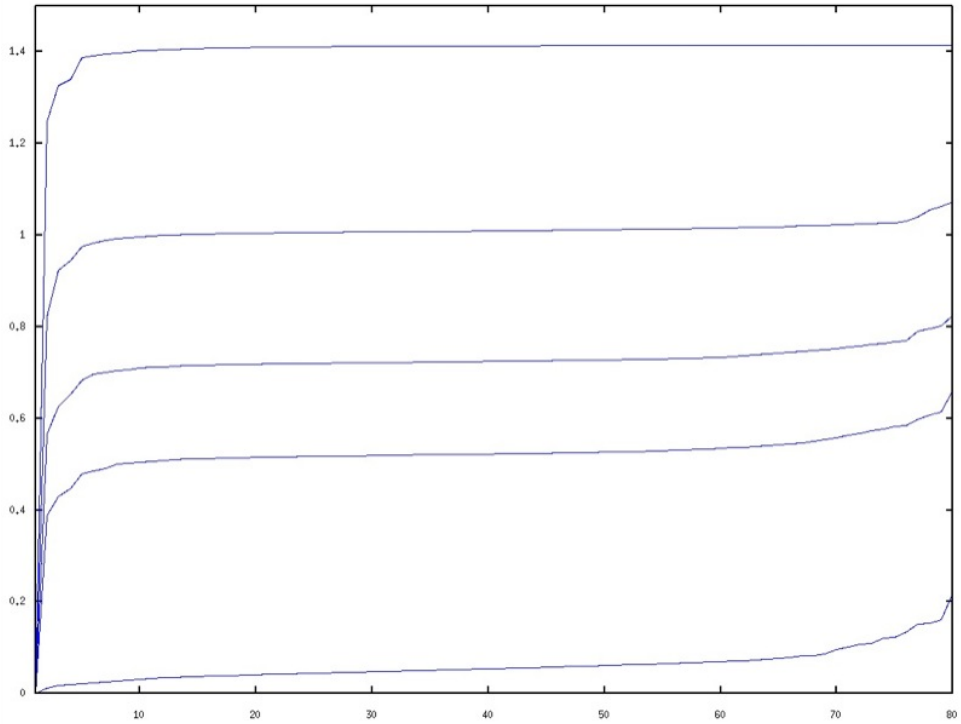


Figure 4.2: Characteristics plotting for a 80 books normalized collection.

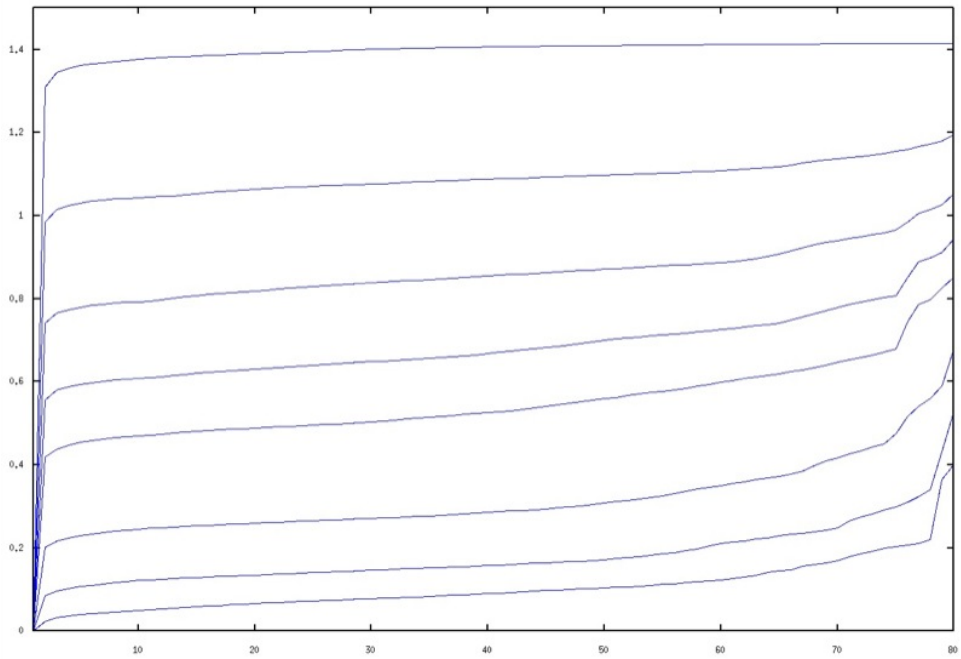


Figure 4.3: Characteristics plotting for a 80 books normalized collection.

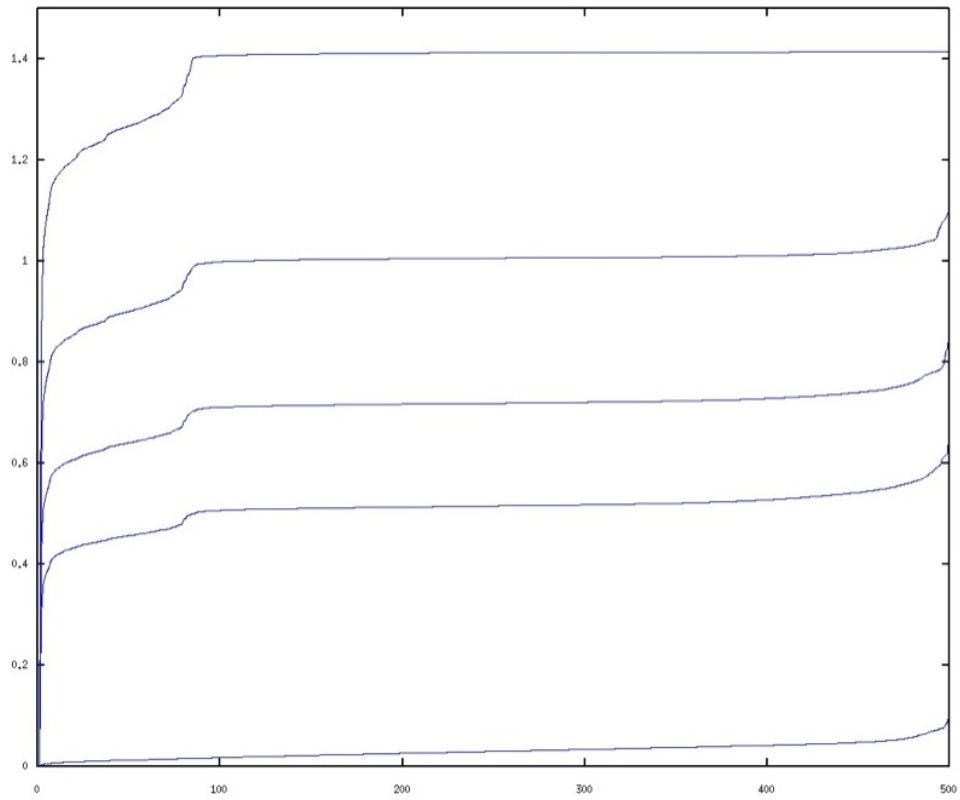


Figure 4.4: Characteristics plotting for a 500 books normalized collection.

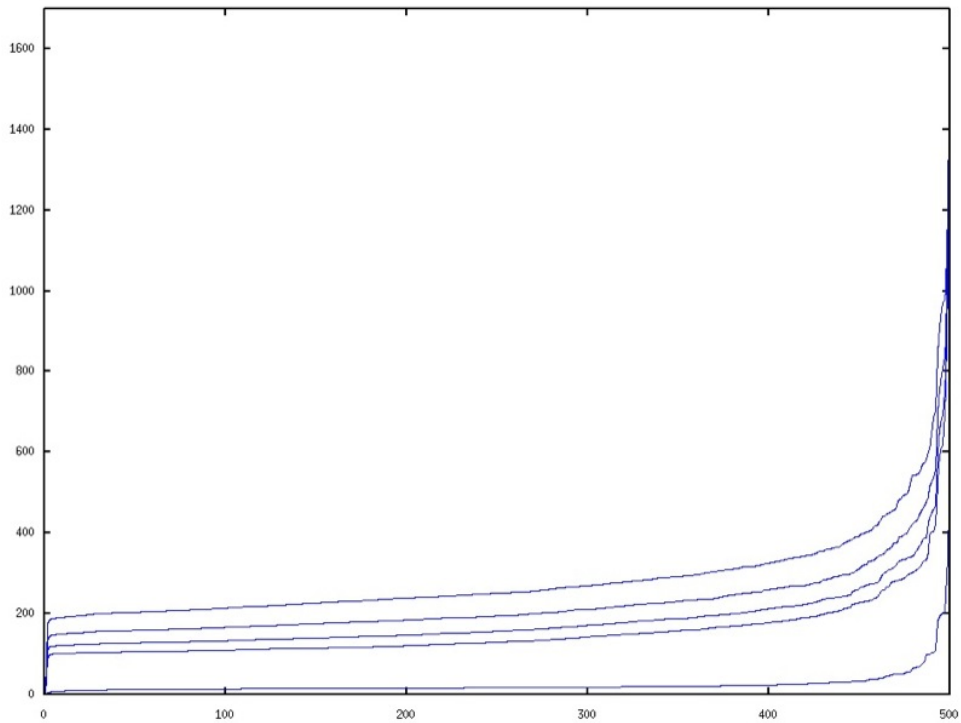


Figure 4.5: Characteristics plotting for a 500 books, not normalized collection.

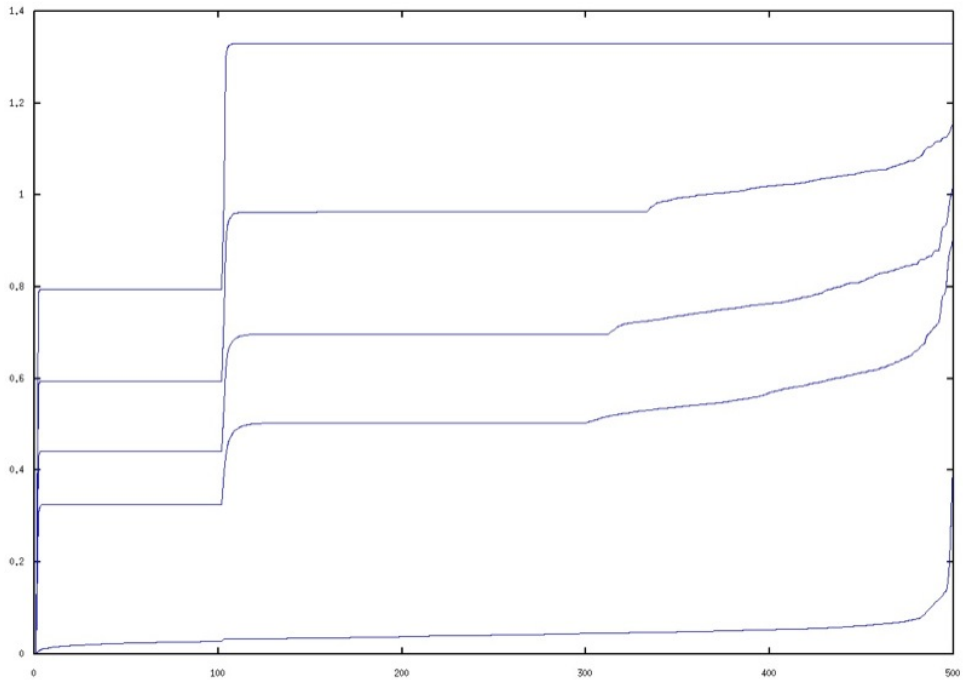


Figure 4.6: Characteristics plotting for a 500 books normalized collection.

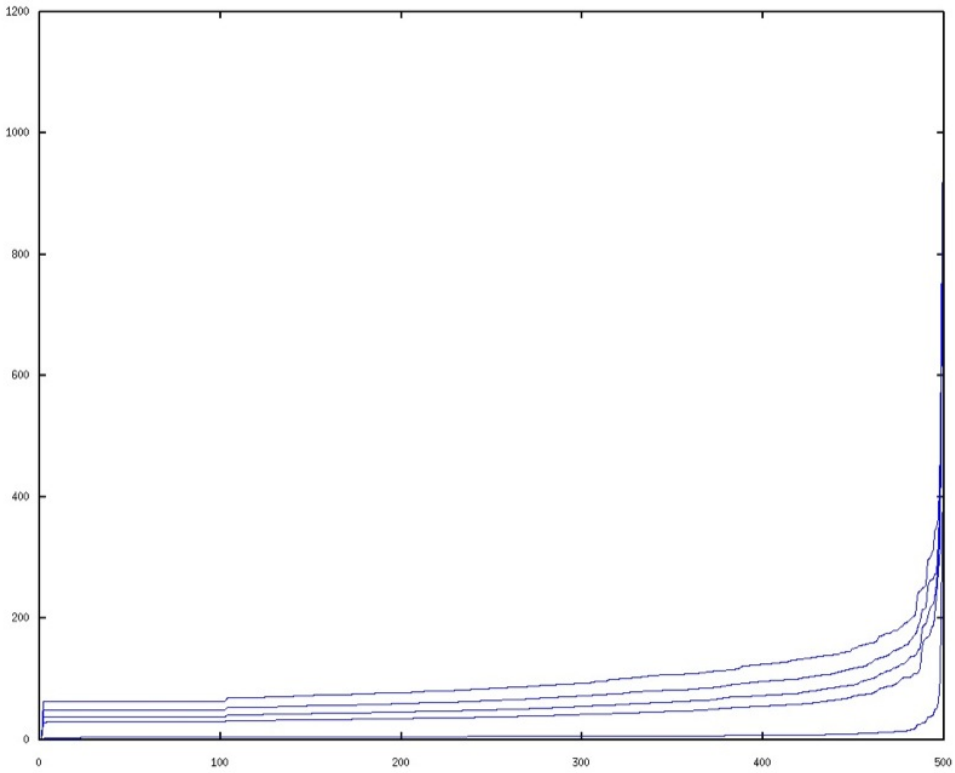


Figure 4.7: Characteristics plotting for a 500 books, not normalized collection.

query in the several spaces of the hierarchy, leading to an high number of operations which slow the retrieval for high dimensional collections.

Finally, by observation of the contrast between the normalized and not normalized plotting, one can infer that the normalization of the original space (in addition to using the orthogonal projection) distorts the relative distribution of the books in the space. For instance, comparing graphics 4.6 and 4.7, the former has an abrupt change in the distance distribution near 100 in the horizontal axis, with does not appear in the later representation. This change means that the first one hundred books are very similar to each other, and very different from the rest of the collection, which becomes more evident when the space is normalized.

This chapter presented the performance evaluation of the *Hierarchical Retrieval* by means of a study to evaluate if the model was correct, and if it represented a breakthrough when compared to the *linear search* solution. Hence, this chapter intended to follow a thorough analysis of the obtained results.

Chapter 5

Conclusions and Future Work

Finalizing this work, this chapter presents a brief summary of this work, some conclusions on the usage of the *hierarchical linear subspace indexing method* in textual data, and some prospect of future work.

5.1 Conclusions

Motivated by the need of a fast indexing method, capable of outrunning the *simple list matching search method*, this thesis was driven in order to apply the *hierarchical linear subspace method* to books retrieval.

After reviewing the document indexing problem, and analyzing the evolution of today's models, we were able to establish a list of issues that arise as consequences of the current methods' limitations. Therefore, the *Hierarchical Retrieval* system was developed, using the *vector-space model* to represent the collection of books. It is important to notice that this system does not correspond to a commercial application, but can be thought as a laboratory to experiment this model. Then, over this representation the *hierarchical linear subspace method* was used to index the documents and create a hierarchy of subspaces. Just as with the *content-based image retrieval* [55] case, this hierarchy is used to gradually eliminate the false candidates when a query is performed to the books' collection.

Once the system was complete, two main aspects were tested:

- the quality of the results;
- and the performance of the system.

In the quality of the results, it was easy to observe that the *Hierarchical Retrieval* actually gave the expected books as answer to the test queries. By observation, one concluded that the results were given in order of importance, with significant semantic meaning.

In what concerns to the performance tests, the reader could observe that with the *hierarchical linear subspace method* the retrieval had no improving. Actually, in terms of number of operations/comparisons it was worse than the *list matching method*, since it had to compare the query book with all the books of the collection.

After analyzing the characteristics of the collection (that is, the distribution of the documents in the spaces) in the several testing scenarios, we concluded that the *hierarchical linear subspace method* cannot be used with the *orthogonal projection method*. As sustained in the experiments, this is a consequence of the fact that a vector-space representing textual data is too sparse [33].

This investigation work can be considered innovative in the sense that it implements *book retrieval* using the *vector-space model* to represent the books, and the euclidean distance as a similarity measure. When compared with the majority of current methods, this work relies on the simple idea that the difference between two books can be determined by the euclidean distance between their vector representations. Moreover, the use of the *hierarchical linear subspace method* is also innovative since this method has only been used in *content-based image retrieval* by [55], where it has proven to be successful when dealing with the high dimensional image data.

At the beginning of this work, it was expected to obtain similar results than those with images, nevertheless it was a different type of data to deal with. After performing several tests and not obtaining a result similar to the images field, an article from [33] enlightened and confirmed the suspicion that the vector-space was too sparse. This issue was due to the dimensionality reduction method that was being used, the *orthogonal projection method*, which was not able to smoothen the distribution of the books in the space, by contrary, emphasizing the sparseness of the vector-space.

Although textual data revealed not to behave the same way image data did, this work constitutes a contribution to the *information retrieval* field, opening a vast scope of new paths following from this work. The immediate sequel of this thesis would be the use of a new projection method to overcome the high dimensionality problem, however, several other improvements can be carried out based on this work, making it a true investigation.

5.2 Future Work

Despite the results obtained in the evaluation study, other alternatives to approach this problem arose as possible. Here are presented some improvement suggestions.

Weighting Measure Besides term frequency and inverse document frequency, some different weighting measures could be analyzed as alternatives to the one used in this work, if better results could be achieved.

Document Indexing The module that represents the collection in the vector-space could be more efficient, nevertheless it falls a bit off the scope of this thesis. Although it gives the correct results, the tests took a long time as a result of wanting to have a proper collection to test with. As eventual future work, this module could be restructured and optimized.

The Projection Method As a possible solution to the fact of the space being too sparse, the use of an alternative projection method arose as plausible. In this thesis, the projection method used to create the hierarchy was the orthogonal projection. A possible alternative is the *Principal Components Analysis*, which was referred when structuring the problem. This method is an optimal way to project data in the mean-square sense: the squared error introduced in the projection is minimized over all projections onto a k-dimensional space. Another method that also should be taken into consideration is the *Singular Value Decomposition* suggested in [11] as a better solution than PCA for sparse textual data.

Performance Evaluation Regarding the tests carried out in this work, more could be done in the future, particularly, a more vast and well chosen collection could be selected. This would allow more accurate results, since more documents were being object of analysis and a deeper knowledge of those documents was present.

Concluding, there is still a lot to be done in the Document Indexing field, namely using the *hierarchical linear subspace indexing method*. Some other weighting measures should be studied as well as alternative projecting methods, in order to address the issue of textual data vector-spaces being too sparse.

Appendix A

This annex contains the list of books used to to evaluate the algorithm's correctness. The books are organized by author and they are included in the application's package.

Jane Austen

- *Emma*;
- *Lady Susan*;
- *Love and Friendship*;
- *Mansfield Park*;
- *Northanger Abbey*;
- *Persuasion*;
- *Pride and Prejudice*;
- *Sense and Sensibility.*

Friedrich Nietzsche

- *The Antichist*;
- *Beyond Good and Evil*;
- *Homer and Classical Philology*;
- *The Case of Wagner*;
- *Thus Spake Zarathustra.*

William Shakespeare

- *All's Well That Ends Well*;
- *The Tragedy of Antony and Cleopatra*;
- *As You Like It*;
- *The Tragedy of Coriolanus*;
- *Cymbeline*;
- *Hamlet, Prince of Denmark*;

- *The Tragedy of Julius Caesar*;
- *The Life of Timon of Athens*;
- *A Lover's Complaint*;
- *Love's Labour's Lost*;
- *The Tragedy of Macbeth*;
- *Measure for Measure*;
- *The Merchant of Venice*;
- *The Merry Wives of Windsor*;
- *A Midsummer Night's Dream*;
- *Much Ado About Nothing*;
- *The Tragedy of Othello, Moor of Venice*;
- *The Rape of Lucrece*;
- *The Tragedy of Romeo and Juliet*;
- *The Comedy of Errors*;
- *The Passionate Pilgrim*;
- *The Tempest*;
- *Venus and Adonis*;
- *The Winter's Tale.*

Sir Arthur Conan Doyle

- *The Hound of the Baskervilles*;
- *Beyond the City*;
- *The Great Boer War*;
- *The Exploits Of Brigadier Gerard*;
- *The Adventure of the Bruce-Partington Plans*;
- *The Captain of the Polestar*;
- *The Adventure of the Cardboard Box*;
- *Danger! and Other Stories*;
- *A Desert Drama*;
- *The Adventure of the Devil's Foot*;
- *A Duet*;
- *The Adventure of the Dying Detective*;

- *The Firm of Girdlestone*;
- *The Adventures of Gerard*;
- *The Great Shadow and Other Napoleonic Tales*;
- *The Green Flag*;
- *His Last Bow*;
- *The Disappearance of Lady Frances Carfax*;
- *The Lost World*;
- *My Friend The Murderer*;
- *The Doings Of Raffles Haw*;
- *The Adventure of the Red Circle*;
- *The Return of Sherlock Holmes*;
- *The Adventures of Sherlock Holmes*;
- *Sir Nigel*;
- *A Study In Scarlet*;
- *Tales of Terror and Mystery*;
- *The Cabman's Story*;
- *The Parasite*;
- *The Poison Belt*;
- *The Refugees*;
- *Sign of the Four*;
- *The Stark Munro Letters*;
- *The Vital Message*;
- *Through the Magic Door*;
- *Victorian Short Stories of Troubled Marriages*;
- *The Valley of Fear*;
- *The White Company*;
- *The Adventure of Wisteria Lodge.*

Appendix B

This annex intends to give the reader an installation and user's guide.

Installation Guide The installation of the application is straightforward. The only requirement is the **Java J2SE Framework 1.5.0** available at [36], which is quite simple to install. After J2SE Framework is installed, the following steps must be taken:

1. extract the application package to a desired directory;
2. open one of the properties files in the package and fill the paths with the desired directories; each of the files are an example depending on the operating system, as the reader can see in the following figures.

```
documentIndexing.documentsDirectory=/Users/pedro/IST/Tese/Documents
documentIndexing.XMLDirectory=/Users/pedro/IST/Tese/DocumentIndexing/xmls/
documentIndexing.queryDirectory=/Users/pedro/IST/Tese/DocumentIndexing/query/
documentIndexing.characteristicsDirectory=/Users/pedro/IST/Tese/DocumentIndexing/characteristics/
```

```
documentIndexing.documentsDirectory=C:/Users/Pedro/Documents/Tese/Documents
documentIndexing.XMLDirectory=C:/Users/Pedro/Documents/Tese/DocumentIndexing/xmls/
documentIndexing.queryDirectory=C:/Users/Pedro/Documents/Tese/DocumentIndexing/query/
documentIndexing.characteristicsDirectory=C:/Users/Pedro/Documents/Tese/DocumentIndexing/characteristics
```

Figure 1: Example of properties files using Windows and Mac OS X file paths.

Each of the fields in the properties file will now by briefly explained:

- **documentsDirectory** corresponds to the directory where the documents' collection is;
- **XMLDirectory** is where the XML files are stored during the execution of the program;
- **queryDirectory** is where the user places the query document in order to perform a query;
- **characteristicsDirectory** corresponds to the directory where the distances files to compute the characteristics are stored.

Usage Description Now it will be presented a brief description intended to cover the aspects of the usage of the prototype.

To run the application initialize the command line or terminal available in the operating system, go to the directory where the extracted *jar* file is, and write *java -jar DocumentIndexing.jar*. The first input asked is the properties file path, and after that the indexing phase begins. As soon as the indexing as finished, it will be asked to type which operation to perform. The following pictures are an example of these phases, respectively.

```
Write 'win' if you are using Windows or 'mac' if you are using Mac OS X! Write 'quit' to exit!
mac
Properties uploaded.
documentsDirectory - /Users/pedro/IST/Tese/Documents
XMLDirectory - /Users/pedro/IST/Tese/DocumentIndexing/xmls/
queryDirectory - /Users/pedro/IST/Tese/DocumentIndexing/query/
characteristicsDirectory - /Users/pedro/IST/Tese/DocumentIndexing/characteristics/
Nao existe XML. Vou construir o dicionario.
Adicionei o doc 1
Adicionei o doc 2
Adicionei o doc 3
Adicionei o doc 4
Adicionei o doc 5
Adicionei o doc 6
Adicionei o doc 7
Adicionei o doc 8
Adicionei o doc 9
Adicionei o doc 10
Adicionei o doc 11
```

Figure 2: Screenshot of the indexing phase.

```
Type 1 for TERMS OF IMPORTANCE RETRIEVAL, 2 for DOCUMENT RETRIEVAL or 3 to generate the CHARACTERISTICS files!
1
Updating term weights...
Updating term weights...DONE!
TERMS OF IMPORTANCE RETRIEVAL: How many top terms to return?
10
THE 10 RETRIEVED TERMS
TERM: boers CF: 691 DF: 1 IDF: 4.3307333
Postings list: <40,691,32.64567>
TERM: elinor CF: 616 DF: 1 IDF: 4.3307333
Postings list: <8,616,32.1481>
TERM: ezra CF: 397 DF: 1 IDF: 4.3307333
Postings list: <50,397,30.245565>
TERM: edmund CF: 362 DF: 1 IDF: 4.3307333
Postings list: <4,362,29.845873>
TERM: knightley CF: 336 DF: 1 IDF: 4.3307333
Postings list: <1,336,29.52309>
TERM: girdlestone CF: 332 DF: 1 IDF: 4.3307333
Postings list: <50,332,29.471224>
TERM: elton CF: 317 DF: 1 IDF: 4.3307333
Postings list: <1,317,29.271>
TERM: bennet CF: 293 DF: 1 IDF: 4.3307333
Postings list: <7,293,28.930046>
TERM: catinat CF: 286 DF: 1 IDF: 4.3307333
Postings list: <68,286,28.825327>
TERM: rosalind CF: 264 DF: 1 IDF: 4.3307333
Postings list: <16,264,28.478682>
BUILD SUCCESSFUL (total time: 39 seconds)
```

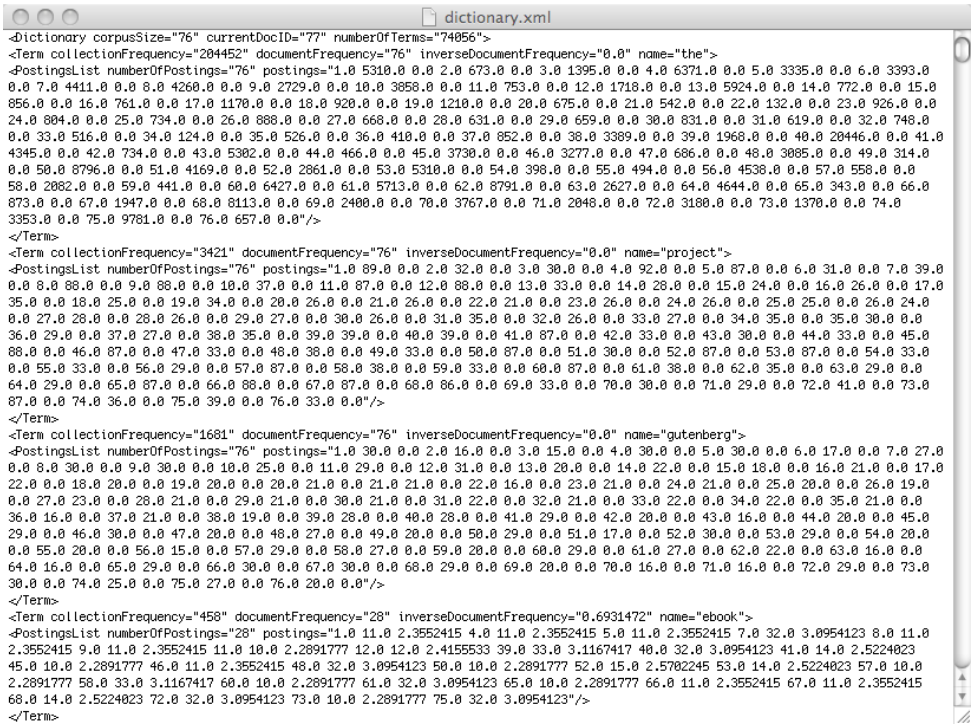
Figure 3: Screenshot of a terms of importance retrieval.

```
Type 1 for TERMS OF IMPORTANCE RETRIEVAL, 2 for DOCUMENT RETRIEVAL or 3 to generate the CHARACTERISTICS files!
2
Vou construir a hierarquia dos documentos query.
Vou construir o documento query.
Adicionei o doc -1
Ja construi o documento query!
Query document:
/Users/pedro/IST/Tese/DocumentIndexing/query/baskervilles.txt
Numero de representacoes do query doc 4
Ja construi a hierarquia dos documentos query!
DOCUMENT RETRIEVAL: Specify the distance threshold epsilon.
1.4
Type 1 for simple list matching ou 2 for the hierarchical subspace method!
2
RESULT SET
Number of results: 19
Number of operations: 380
docID - 38 distance - 0.0
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/baskervilles.txt
docID - 49 distance - 1.3765212
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/dyingDetective.txt
docID - 47 distance - 1.3782672
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/devilsFoot.txt
docID - 44 distance - 1.3848819
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/cardboardBox.txt
docID - 61 distance - 1.385254
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/sherlockHolmes.txt
docID - 76 distance - 1.3870152
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/wisteriaLodge.txt
docID - 55 distance - 1.3871522
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/ladyFrancesCarfax.txt
docID - 42 distance - 1.3889942
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/BrucePartington.txt
docID - 45 distance - 1.3892171
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/dangerAndOther.txt
docID - 60 distance - 1.3905452
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/returnOfSherlockHolmes.txt
docID - 64 distance - 1.3932855
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/talesOfTerrorAndMistery.txt
docID - 53 distance - 1.3937602
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/greenFlag.txt
docID - 57 distance - 1.3946218
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/myFriendTheMurderer.txt
docID - 59 distance - 1.3951147
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/redCircle.txt
docID - 69 distance - 1.3952378
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/theSignOfTheFour.txt
docID - 54 distance - 1.3971082
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/hisLastBow.txt
docID - 48 distance - 1.3974262
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/duet.txt
docID - 51 distance - 1.3981647
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/gerard.txt
docID - 5 distance - 1.398376
path - /Users/pedro/IST/Tese/Documents/JaneAusten/northangerAbbey.txt
BUILD SUCCESSFUL (total time: 50 seconds)
```

Figure 4: Screenshot of a book retrieval query.

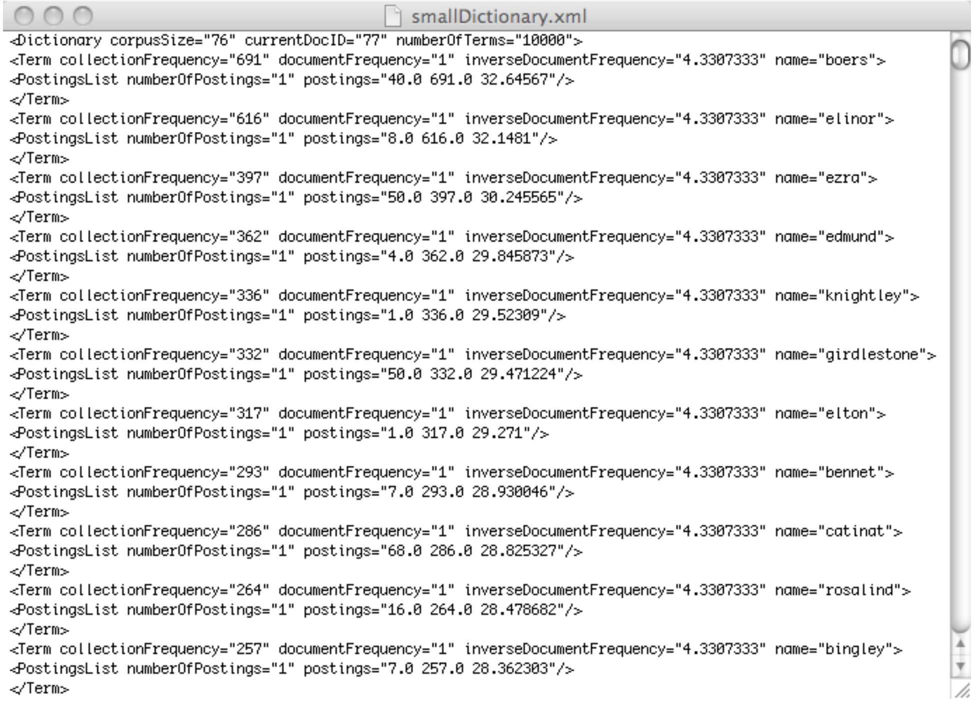
Appendix C

This annex contains screenshots of parts of the XML files generated by the prototype, as well as the output resulting of the interaction with the program in the indexing phase or when performing a query, for instance.



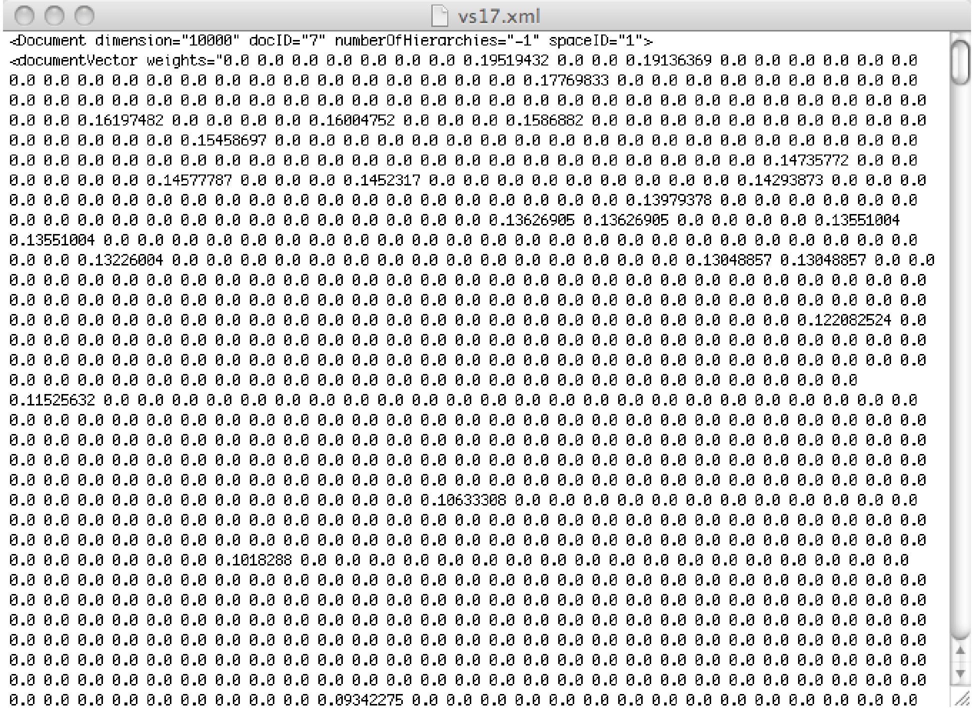
```
dictionary.xml
<Dictionary corpusSize="76" currentDocID="77" numberOfTerms="74056">
<Term collectionFrequency="204452" documentFrequency="76" inverseDocumentFrequency="0.0" name="the">
<PostingsList numberOfPostings="76" postings="1.0 5310.0 0.0 2.0 673.0 0.0 3.0 1395.0 0.0 4.0 6371.0 0.0 5.0 3335.0 0.0 6.0 3393.0
0.0 7.0 4411.0 0.0 8.0 4260.0 0.0 9.0 2729.0 0.0 10.0 3858.0 0.0 11.0 753.0 0.0 12.0 1718.0 0.0 13.0 5924.0 0.0 14.0 772.0 0.0 15.0
856.0 0.0 16.0 761.0 0.0 17.0 1170.0 0.0 18.0 920.0 0.0 19.0 1210.0 0.0 20.0 675.0 0.0 21.0 542.0 0.0 22.0 132.0 0.0 23.0 926.0 0.0
24.0 804.0 0.0 25.0 734.0 0.0 26.0 888.0 0.0 27.0 668.0 0.0 28.0 631.0 0.0 29.0 659.0 0.0 30.0 831.0 0.0 31.0 619.0 0.0 32.0 748.0
0.0 33.0 516.0 0.0 34.0 124.0 0.0 35.0 526.0 0.0 36.0 410.0 0.0 37.0 852.0 0.0 38.0 3389.0 0.0 39.0 1968.0 0.0 40.0 20446.0 0.0 41.0
4345.0 0.0 42.0 734.0 0.0 43.0 5302.0 0.0 44.0 466.0 0.0 45.0 3730.0 0.0 46.0 3277.0 0.0 47.0 686.0 0.0 48.0 3085.0 0.0 49.0 314.0
0.0 50.0 8796.0 0.0 51.0 4169.0 0.0 52.0 2861.0 0.0 53.0 5310.0 0.0 54.0 398.0 0.0 55.0 494.0 0.0 56.0 4538.0 0.0 57.0 558.0 0.0
58.0 2082.0 0.0 59.0 441.0 0.0 60.0 6427.0 0.0 61.0 5713.0 0.0 62.0 8791.0 0.0 63.0 2627.0 0.0 64.0 4644.0 0.0 65.0 343.0 0.0 66.0
873.0 0.0 67.0 1947.0 0.0 68.0 8113.0 0.0 69.0 2400.0 0.0 70.0 3767.0 0.0 71.0 2048.0 0.0 72.0 3180.0 0.0 73.0 1370.0 0.0 74.0
3353.0 0.0 75.0 9781.0 0.0 76.0 657.0 0.0"/>
</Term>
<Term collectionFrequency="3421" documentFrequency="76" inverseDocumentFrequency="0.0" name="project">
<PostingsList numberOfPostings="76" postings="1.0 89.0 0.0 2.0 32.0 0.0 3.0 30.0 0.0 4.0 92.0 0.0 5.0 87.0 0.0 6.0 31.0 0.0 7.0 39.0
0.0 8.0 88.0 0.0 9.0 88.0 0.0 10.0 37.0 0.0 11.0 87.0 0.0 12.0 88.0 0.0 13.0 33.0 0.0 14.0 28.0 0.0 15.0 24.0 0.0 16.0 26.0 0.0 17.0
35.0 0.0 18.0 25.0 0.0 19.0 34.0 0.0 20.0 26.0 0.0 21.0 26.0 0.0 22.0 21.0 0.0 23.0 26.0 0.0 24.0 26.0 0.0 25.0 25.0 0.0 26.0 24.0
0.0 27.0 28.0 0.0 28.0 26.0 0.0 29.0 27.0 0.0 30.0 26.0 0.0 31.0 35.0 0.0 32.0 26.0 0.0 33.0 27.0 0.0 34.0 35.0 0.0 35.0 30.0 0.0
36.0 29.0 0.0 37.0 27.0 0.0 38.0 35.0 0.0 39.0 39.0 0.0 40.0 39.0 0.0 41.0 87.0 0.0 42.0 33.0 0.0 43.0 30.0 0.0 44.0 33.0 0.0 45.0
88.0 0.0 46.0 87.0 0.0 47.0 33.0 0.0 48.0 38.0 0.0 49.0 33.0 0.0 50.0 87.0 0.0 51.0 30.0 0.0 52.0 87.0 0.0 53.0 87.0 0.0 54.0 33.0
0.0 55.0 33.0 0.0 56.0 29.0 0.0 57.0 87.0 0.0 58.0 38.0 0.0 59.0 33.0 0.0 60.0 87.0 0.0 61.0 38.0 0.0 62.0 35.0 0.0 63.0 29.0 0.0
64.0 29.0 0.0 65.0 87.0 0.0 66.0 88.0 0.0 67.0 87.0 0.0 68.0 86.0 0.0 69.0 33.0 0.0 70.0 30.0 0.0 71.0 29.0 0.0 72.0 41.0 0.0 73.0
87.0 0.0 74.0 36.0 0.0 75.0 39.0 0.0 76.0 33.0 0.0"/>
</Term>
<Term collectionFrequency="1681" documentFrequency="76" inverseDocumentFrequency="0.0" name="gutenberg">
<PostingsList numberOfPostings="76" postings="1.0 30.0 0.0 2.0 16.0 0.0 3.0 15.0 0.0 4.0 30.0 0.0 5.0 30.0 0.0 6.0 17.0 0.0 7.0 27.0
0.0 8.0 30.0 0.0 9.0 30.0 0.0 10.0 25.0 0.0 11.0 29.0 0.0 12.0 31.0 0.0 13.0 20.0 0.0 14.0 22.0 0.0 15.0 18.0 0.0 16.0 21.0 0.0 17.0
22.0 0.0 18.0 20.0 0.0 19.0 20.0 0.0 20.0 21.0 0.0 21.0 21.0 0.0 22.0 16.0 0.0 23.0 21.0 0.0 24.0 21.0 0.0 25.0 20.0 0.0 26.0 19.0
0.0 27.0 23.0 0.0 28.0 21.0 0.0 29.0 21.0 0.0 30.0 21.0 0.0 31.0 22.0 0.0 32.0 21.0 0.0 33.0 22.0 0.0 34.0 22.0 0.0 35.0 21.0 0.0
36.0 16.0 0.0 37.0 21.0 0.0 38.0 19.0 0.0 39.0 28.0 0.0 40.0 28.0 0.0 41.0 29.0 0.0 42.0 20.0 0.0 43.0 16.0 0.0 44.0 20.0 0.0 45.0
29.0 0.0 46.0 30.0 0.0 47.0 20.0 0.0 48.0 27.0 0.0 49.0 20.0 0.0 50.0 29.0 0.0 51.0 17.0 0.0 52.0 30.0 0.0 53.0 29.0 0.0 54.0 20.0
0.0 55.0 20.0 0.0 56.0 15.0 0.0 57.0 29.0 0.0 58.0 27.0 0.0 59.0 20.0 0.0 60.0 29.0 0.0 61.0 27.0 0.0 62.0 22.0 0.0 63.0 16.0 0.0
64.0 16.0 0.0 65.0 29.0 0.0 66.0 30.0 0.0 67.0 30.0 0.0 68.0 29.0 0.0 69.0 20.0 0.0 70.0 16.0 0.0 71.0 16.0 0.0 72.0 29.0 0.0 73.0
30.0 0.0 74.0 25.0 0.0 75.0 27.0 0.0 76.0 20.0 0.0"/>
</Term>
<Term collectionFrequency="458" documentFrequency="28" inverseDocumentFrequency="0.6931472" name="ebook">
<PostingsList numberOfPostings="28" postings="1.0 11.0 2.3552415 4.0 11.0 2.3552415 5.0 11.0 2.3552415 7.0 32.0 3.0954123 8.0 11.0
2.3552415 9.0 11.0 2.3552415 11.0 10.0 2.2891777 12.0 12.0 2.4155533 39.0 33.0 3.1167417 40.0 32.0 3.0954123 41.0 14.0 2.5224023
45.0 10.0 2.2891777 46.0 11.0 2.3552415 48.0 32.0 3.0954123 50.0 10.0 2.2891777 52.0 15.0 2.5702245 53.0 14.0 2.5224023 57.0 10.0
2.2891777 58.0 33.0 3.1167417 60.0 10.0 2.2891777 61.0 32.0 3.0954123 65.0 10.0 2.2891777 66.0 11.0 2.3552415 67.0 11.0 2.3552415
68.0 14.0 2.5224023 72.0 32.0 3.0954123 73.0 10.0 2.2891777 75.0 32.0 3.0954123"/>
</Term>
```


Figure 5: Screenshot of the raw dictionary XML file.



```
<Dictionary corpusSize="76" currentDocID="77" numberOfTerms="10000">
<Term collectionFrequency="691" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="boers">
<PostingsList numberOfPostings="1" postings="40.0 691.0 32.64567"/>
</Term>
<Term collectionFrequency="616" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="elinor">
<PostingsList numberOfPostings="1" postings="8.0 616.0 32.1481"/>
</Term>
<Term collectionFrequency="397" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="ezra">
<PostingsList numberOfPostings="1" postings="50.0 397.0 30.245565"/>
</Term>
<Term collectionFrequency="362" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="edmund">
<PostingsList numberOfPostings="1" postings="4.0 362.0 29.845873"/>
</Term>
<Term collectionFrequency="336" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="knightley">
<PostingsList numberOfPostings="1" postings="1.0 336.0 29.52309"/>
</Term>
<Term collectionFrequency="332" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="girdlestone">
<PostingsList numberOfPostings="1" postings="50.0 332.0 29.471224"/>
</Term>
<Term collectionFrequency="317" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="elton">
<PostingsList numberOfPostings="1" postings="1.0 317.0 29.271"/>
</Term>
<Term collectionFrequency="293" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="bennet">
<PostingsList numberOfPostings="1" postings="7.0 293.0 28.930046"/>
</Term>
<Term collectionFrequency="286" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="catinat">
<PostingsList numberOfPostings="1" postings="68.0 286.0 28.825327"/>
</Term>
<Term collectionFrequency="264" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="rosalind">
<PostingsList numberOfPostings="1" postings="16.0 264.0 28.478682"/>
</Term>
<Term collectionFrequency="257" documentFrequency="1" inverseDocumentFrequency="4.3307333" name="bingley">
<PostingsList numberOfPostings="1" postings="7.0 257.0 28.362303"/>
</Term>
```

Figure 6: Screenshot of the reduced dictionary XML file.



```
<Document dimension="10000" docID="7" numberOfHierarchies="-1" spaceID="1">
<documentVector weights="0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.19519432 0.0 0.0 0.19136369 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.17769833 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.16197482 0.0 0.0 0.0 0.0 0.16004752 0.0 0.0 0.0 0.1586882 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.15458697 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.14735772 0.0 0.0
0.0 0.0 0.0 0.0 0.14577787 0.0 0.0 0.0 0.1452317 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.14293873 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.13979378 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.13626905 0.13626905 0.0 0.0 0.0 0.0 0.13551004
0.13551004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.13226004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.13048857 0.13048857 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.122082524 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.11525632 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.10633308 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.1018288 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.09342275 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Figure 7: Screenshot of a document XML file.

dummyDictionary.xml


```xml
<Dictionary corpusSize="76" currentDocID="-1" numberOfTerms="10000">
<Term collectionFrequency="0" documentFrequency="0"
inverseDocumentFrequency="4.3307333" name="boers">
<PostingsList numberOfPostings="0" postings=""/>
</Term>
<Term collectionFrequency="0" documentFrequency="0"
inverseDocumentFrequency="4.3307333" name="elinor">
<PostingsList numberOfPostings="0" postings=""/>
</Term>
<Term collectionFrequency="0" documentFrequency="0"
inverseDocumentFrequency="4.3307333" name="ezra">
<PostingsList numberOfPostings="0" postings=""/>
</Term>
<Term collectionFrequency="0" documentFrequency="0"
inverseDocumentFrequency="4.3307333" name="edmund">
<PostingsList numberOfPostings="0" postings=""/>
</Term>
<Term collectionFrequency="0" documentFrequency="0"
inverseDocumentFrequency="4.3307333" name="knightley">
<PostingsList numberOfPostings="0" postings=""/>
</Term>
<Term collectionFrequency="0" documentFrequency="0"
inverseDocumentFrequency="4.3307333" name="girdlestone">
<PostingsList numberOfPostings="0" postings=""/>
</Term>
<Term collectionFrequency="0" documentFrequency="0"
inverseDocumentFrequency="4.3307333" name="elton">
<PostingsList numberOfPostings="0" postings=""/>
</Term>
```


Figure 8: XML file of an auxiliary dictionary used in the creation of the query document.

```
Type 1 for TERMS OF IMPORTANCE RETRIEVAL, 2 for DOCUMENT RETRIEVAL or 3 to generate the CHARACTERISTICS files!
2
Vou construir a hierarquia dos documentos query.
Vou construir o documento query.
Adicionei o doc -1
Ja construi o documento query!
Query document:
/Users/pedro/IST/Tese/DocumentIndexing/query/baskervilles.txt
Numero de representacoes do query doc 4
Ja construi a hierarquia dos documentos query!
DOCUMENT RETRIEVAL: Specify the distance threshold epsilon.
1.4
Type 1 for simple list matching ou 2 for the hierarchical subspace method!
2
RESULT SET
Number of results: 19
Number of operations: 380
docID - 38 distance - 0.0
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/baskervilles.txt
docID - 49 distance - 1.3765212
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/dyingDetective.txt
docID - 47 distance - 1.3782672
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/devilsFoot.txt
docID - 44 distance - 1.3848819
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/cardboardBox.txt
docID - 61 distance - 1.385254
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/sherlockHolmes.txt
docID - 76 distance - 1.3870152
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/wisteriaLodge.txt
docID - 55 distance - 1.3871522
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/ladyFrancesCarfax.txt
docID - 42 distance - 1.3889942
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/BrucePartington.txt
docID - 45 distance - 1.3892171
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/dangerAndOther.txt
docID - 60 distance - 1.3905452
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/returnOfSherlockHolmes.txt
docID - 64 distance - 1.3932855
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/talesOfTerrorAndMistery.txt
docID - 53 distance - 1.3937602
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/greenFlag.txt
docID - 57 distance - 1.3946218
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/myFriendTheMurderer.txt
docID - 59 distance - 1.3951147
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/redCircle.txt
docID - 69 distance - 1.3952378
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/theSignOfTheFour.txt
docID - 54 distance - 1.3971082
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/hisLastBow.txt
docID - 48 distance - 1.3974262
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/duet.txt
docID - 51 distance - 1.3981647
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/gerard.txt
docID - 5 distance - 1.398376
path - /Users/pedro/IST/Tese/Documents/JaneAusten/northangerAbbey.txt
BUILD SUCCESSFUL (total time: 50 seconds)
```

Figure 9: Screenshot of a query with the book *The Hound of the Baskervilles*.

```
Type 1 for TERMS OF IMPORTANCE RETRIEVAL, 2 for DOCUMENT RETRIEVAL or 3 to generate the CHARACTERISTICS files!
2
Vou construir a hierarquia dos documentos query.
Vou construir o documento query.
Adicionei o doc -1
Ja construi o documento query!
Query document:
/Users/pedro/IST/Tese/DocumentIndexing/query/prideAndPrejudice.txt
Numero de representacoes do query doc 4
Ja construi a hierarquia dos documentos query!
DOCUMENT RETRIEVAL: Specify the distance threshold epsilon.
1.409
Type 1 for simple list matching ou 2 for the hierarchical subspace method!
2
RESULT SET
Number of results: 23
Number of operations: 380
docID - 7 distance - 0.0
path - /Users/pedro/IST/Tese/Documents/JaneAusten/prideAndPrejudice.txt
docID - 8 distance - 1.3442986
path - /Users/pedro/IST/Tese/Documents/JaneAusten/senseAndSensibility.txt
docID - 5 distance - 1.3525617
path - /Users/pedro/IST/Tese/Documents/JaneAusten/northangerAbbey.txt
docID - 4 distance - 1.354711
path - /Users/pedro/IST/Tese/Documents/JaneAusten/mansfieldPark.txt
docID - 6 distance - 1.3595792
path - /Users/pedro/IST/Tese/Documents/JaneAusten/persuasion.txt
docID - 1 distance - 1.361106
path - /Users/pedro/IST/Tese/Documents/JaneAusten/emma.txt
docID - 3 distance - 1.377508
path - /Users/pedro/IST/Tese/Documents/JaneAusten/loveAndFriendship.txt
docID - 2 distance - 1.3868961
path - /Users/pedro/IST/Tese/Documents/JaneAusten/ladySusan.txt
docID - 73 distance - 1.4018732
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/troubledMarriages.txt
docID - 40 distance - 1.4046099
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/boerWar.txt
docID - 48 distance - 1.4054437
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/duet.txt
docID - 43 distance - 1.4059148
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/captainPolestar.txt
docID - 38 distance - 1.4060098
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/baskervilles.txt
docID - 39 distance - 1.4062989
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/beyondTheCity.txt
docID - 22 distance - 1.4068666
path - /Users/pedro/IST/Tese/Documents/Shakespeare/loversComplaint.txt
docID - 72 distance - 1.4071968
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/throughTheMagicDoor.txt
docID - 64 distance - 1.4076473
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/talesOfTerrorAndMistery.txt
docID - 68 distance - 1.407662
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/theRefugees.txt
docID - 67 distance - 1.4078678
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/thePoisonBelt.txt
docID - 60 distance - 1.4079111
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/returnOfSherlockHolmes.txt
docID - 50 distance - 1.4080398
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/firmOfGirdlestone.txt
docID - 65 distance - 1.408534
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/theCabmansStory.txt
docID - 15 distance - 1.4088702
path - /Users/pedro/IST/Tese/Documents/Shakespeare/anthonyAndCleopatra.txt
BUILD SUCCESSFUL (total time: 1 minute 22 seconds)
```

Figure 10: Screenshot of a query with the book *Pride and Prejudice*.

```
Type 1 for TERMS OF IMPORTANCE RETRIEVAL, 2 for DOCUMENT RETRIEVAL or 3 to generate the CHARACTERISTICS files!
2
Vou construir a hierarquia dos documentos query.
Vou construir o documento query.
Adicionei o doc -1
Ja construi o documento query!
Query document:
/Users/pedro/IST/Tese/DocumentIndexing/query/baskervilles.txt
Numero de representacoes do query doc 4
Ja construi a hierarquia dos documentos query!
DOCUMENT RETRIEVAL: Specify the distance threshold epsilon.
1.4
Type 1 for simple list matching ou 2 for the hierarchical subspace method!
1
RESULT SET
Number of results: 19
Number of operations: 76
docID - 38 distance - 0.0
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/baskervilles.txt
docID - 49 distance - 1.3765212
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/dyingDetective.txt
docID - 47 distance - 1.3782672
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/devilsFoot.txt
docID - 44 distance - 1.3848819
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/cardboardBox.txt
docID - 61 distance - 1.385254
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/sherlockHolmes.txt
docID - 76 distance - 1.3870152
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/wisteriaLodge.txt
docID - 55 distance - 1.3871522
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/ladyFrancesCarfax.txt
docID - 42 distance - 1.3889942
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/BrucePartington.txt
docID - 45 distance - 1.3892171
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/dangerAndOther.txt
docID - 60 distance - 1.3905452
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/returnOfSherlockHolmes.txt
docID - 64 distance - 1.3932855
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/talesOfTerrorAndMistery.txt
docID - 53 distance - 1.3937602
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/greenFlag.txt
docID - 57 distance - 1.3946218
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/myFriendTheMurderer.txt
docID - 59 distance - 1.3951147
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/redCircle.txt
docID - 69 distance - 1.3952378
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/theSignOfTheFour.txt
docID - 54 distance - 1.3971082
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/hisLastBow.txt
docID - 48 distance - 1.3974262
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/duet.txt
docID - 51 distance - 1.3981647
path - /Users/pedro/IST/Tese/Documents/SirArthurDoyle/gerard.txt
docID - 5 distance - 1.398376
path - /Users/pedro/IST/Tese/Documents/JaneAusten/northangerAbbey.txt
BUILD SUCCESSFUL (total time: 1 minute 22 seconds)
```

Figure 11: Screenshot of a query performed using the *linear search method*.

Bibliography

[1] IJsbrand Jan Aalbersberg and Frans Sijstermans. *High-Quality and High-Performance Full-Text Document Retrieval: the Parallel InfoGuide System.* Philips Research Laboratories, IEEE, 1991.

[2] D. Achlioptas. *Database-friendly random projections.* Proc. Acm Symp. on the Principles of Database Systems, 2001.

[3] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. *A new method for similarity indexing of market basket data.* ACM SIGMOD Int. Conf. on Management of data, pages 407-418, 1999.

[4] A. Aizawa. *An information-theoretic perspective of tf-idf measures.* Information Processing and Management Vol. 39, pages 45-65, 2003.

[5] Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. *Vector-space ranking with effective early termination.* 24th Annual International ACM SIGIR Conference, 2001.

[6] Vo Ngoc Anh and Alistair Moffat. *Pruned query evaluation using pre-computed impacts.* SIGIR Conference, 2006.

[7] Howard Anton and Chris Rorres. *Elementary linear algebra: applications vertion.* Wiley, 2000.

[8] R. I. Arriaga and S. Vempala. *An algorithmic theory of learning: robust concepts and random projection.* 40th Annual Symp. on Foundations of Computer Science, pages 616-623, IEEE Computer Society Press, 1999.

[9] Eduardo Fernandes Barbosa, Gonzalo Navarro, Ricardo Baeza-Yates, Chris Perleberg, and Nivio Ziviani. *Optimized binary search and text retrieval.* Springer Berlin / Heidelberg, 1995.

[10] M. W. Berry. *Large-scale sparce singular value computations.* International Journal of Supercomputer Applications, 1992.

[11] Ella Bingham and Heikki Mannila. *Random projection in dimensionality reduction: Applications to image and text data.* Laboratory of Computer and Information Science, Helsinki University of Technology, 2001.

[12] K. Church and W. Gale. *Inverse Document Frequency (IDF): a measure of deviations from Poisson.* Third Workshop on very large corpora, ACL, MIT, pages 121-130, 1995.

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, second edition.* MIT Press, 2005.

[14] S. Dasgupta. *Experiments with random projection.* Uncertainty in Artificial Intelligence, 2000.

[15] Sanjoy Dasgupta and Anupam Gupta. *An elementary proof of the Johnson-Lindenstrauss Lemma.* International Computer Science Institute, 1999.

[16] S. Deerwester, S. T. Dumais, G. W. Furnas, and T. K. Landauer. *Indexing by latent semantic analysis.* Journal for the American Society for information Science, 1990.

[17] I. S. Dhillon and D. M. Modha. *Concept Decompositions for Large Sparse Text Data using Clustering.* Machine Learning, 2001.

[18] S. T. Dumais. *LSI meets TREC: A Status Report.* Text REtrieval Conference, 1992.

[19] The Apache Software Foundation. *http://www.apache.org/.* The Apache Software Foundation, Wednesday, July 23, 2008.

[20] P. Frankl and H. Maehara. *The Johnson-Linderstrauss lemma and the sphericity of some graphs.* Journal of Combinational Theory, 1988.

[21] Alan M. Frieze, Ravi Kannan, and Santosh Vempala. *Fast Monte-Carlo Algorithms for finding low-rank approximations.* IEEE Sysposium on Foundations of Computer Science, 1998.

[22] G. H. Golub and C. F. van Loan. *Matrix Computations.* North Oxford Academic, Oxford, UK, 1983.

[23] Michael Hart. *http://www.gutenberg.org/wiki/Main-Page.* Project Gutenberg, Wednesday, July 16, 2008.

[24] R. Hecht-Nielsen. *Context vectors: general purpose approximate meaning representations self-organized from raw data.* IEEE Press - Computational Intelligence: Imitating Life, 1994.

[25] T. Hofmann. *Probabilistic Latent Semantic Indexing.* Proceedings of SIGIR, 1999.

[26] P. Indyk and R. Motwani. *Approximate nearest neighbors: towards removing the curse of dimensionality.* 30th Symp. on Theory of Computing, pages 604-613, ACM, 1998.

[27] W. B. Johnson and J. Lindenstrauss. *Contemporary Mathematics - Extensions of Lipshitz mapping into Hilbert space.* American Math. Society - Conference in modern analysis and probability, 1984.

[28] K. Sparck Jones, S. Walker, and S. E. Robertson. *A probabilistic model of information retrieval: development and comparative experiments.* Information Processing and Management Vol. 36, pages 779-808 (part 1) and 809-840 (part 2), 2000.

[29] S. Kaski. *Dimensionality reduction by random mapping.* Int. Joint Conf. on Neural Networks, 1998.

[30] E. J. Keogh and M. J. Pazzini. *A simple dimensionality reduction technique for fast similarity search in large time series databases.* 4th Pacific-Asia Conf. on knowledge Discovery and Data Mining, 2000.

[31] J. M. Kleinberg. *Two algorithms for nearest-neighbor search in high dimensions.* 29th ACM Symp. on Theory of Computing, pages 599-608, 1997.

[32] J. Lin and D. Gunopulos. *Dimensionality Reduction by Random Projection and Latent Semantic Indexing.* Proceedings of SDM03 Conference, Text Mining Workshop, 2003.

[33] João Magalhães and Stefan Ruger. *Information-Theoretic Semantic Multimedia Indexing.* ACM, 2007.

[34] Chistopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *An Introduction to Information Retrieval (preliminary draft).* Cambridge University Press, 2006.

[35] Sun Microsystems. *http://www.netbeans.org/.* Sun Microsystems, Wednesday, August 13, 2008.

[36] Sun Microsystems. *http://java.sun.com/.* Sun Microsystems, Wednesday, July 16, 2008.

[37] Sun Microsystems. *http://java.sun.com/j2se/1.5.0/docs/api/*. Sun Microsystems, Wednesday, July 16, 2008.

[38] Assadaporn Nuchprayoon and Robert R. Korfhage. *GUIDO: Visualizing Document Retrieval*. IEEE Computer Society, 1997.

[39] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. *Latent semantic indexing: A probabilistic analysis*. Proc. 17th ACM Symp. on the Principles of Database Systems, 1998.

[40] Kishore Papineni. *Why inverse document frequency?* NAACL 2, 2001.

[41] S.E. Robertson and K. Sparck Jones. *Relevance weighting of search terms*. Journal of the American Society for Information Science, 1976.

[42] Stephen Robertson. *Understanding inverse document frequency: on theoretical arguments for IDF*. Journal of Documentation Volume 60 Number 5, pages 503-520, 2004.

[43] Stephen E. Robertson, Steve Walker, Micheline Hancock-Beaulieu, Aarron Gull, and Marianna Lau. *Okapi at TREC*. Text REtrieval Conference, 1992.

[44] S. Roweis. *EM algorirthms for PCA and SPCA*. Neural Information Processing Systems 10, 1997.

[45] G. Salton and M. J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1983.

[46] G. Salton, A. Wong, and C. S. Yang. *A Vector Space Model for Automatic Indexing*. ACM, 1975.

[47] Gerard Salton. *AutomaticText Processing: The Transformation, Analysis, and Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison Wesley, 1989.

[48] Gerard Salton and Chris Buckley. *Information Processing and Management (Term weighting approaches in automatic text retrieval - Technical Report TR87-881)*. Department of Computer Science, Cornell University, 1996.

[49] Amit Singhal, Chris Buckley, and Mandar Mitra. *Pivoted document length normalization*. ACM SIGIR, 1996.

[50] L. Sirovich and R. Everson. *Management and analysis of large scientific datasets*. Int. Journal of Supercomputer Applications, 1992.

[51] Bernd H. Storb and Raul S. Wazlawick. *Fuzzy Document Retrieval for Portuguese Language*. IEEE Computer Society, 1999.

[52] C. J. van Rijsbergen. *Information retrieval*. Butterworks, London, U.K. (Second Edition), 1979.

[53] S. Vempala. *Random projection: a new approach to VLSI layout*. 39th Annual Symp. on Foundations of Computer Science, IEEE Computer Society Press, 1998.

[54] Vishwa Vinay, Ingemar J. Cox, Ken Wood, and Natasa Milic-Frayling. *A Comparison of Dimensionality Reduction Techniques for Text Retrieval*. IEEE, 2005.

[55] Andreas Wichert. *Content-based image retrieval by hierarchical linear subspace method*. Springer Berlin / Heidelberg, 2007.

[56] Carlo Zaniolo, Stefano Ceri, Christos Faloutsos, Richard T. Snodgrass, V. S. Subrahmanian, and Roberto Zicari. *Advanced Database Systems*. Morgan Kaufmann, 1997.

[57] Taizo Zushi, Kosuke Takano, and Yasushi Kiyoki. *A Vector Space Retrieval Method with Causal Relationship Computation Functions for Event Data.* IEEE Computer Society, 2005.