2014

Search Queries in an Information Retrieval System for Arabic-Language Texts

Zainab Majeed Albujasim
University of Kentucky, zalb222@g.uky.edu

Recommended Citation
Albujasim, Zainab Majeed, "Search Queries in an Information Retrieval System for Arabic-Language Texts" (2014). *Theses and Dissertations--Computer Science*. 23.
https://uknowledge.uky.edu/cs_etds/23

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Zainab Majeed Albujasim, Student

Dr. Jerzy W. Jaromczyk, Major Professor

Dr. Mirosław Truszczyński, Director of Graduate Studies

SEARCH QUERIES IN AN INFORMATION RETRIEVAL SYSTEM FOR ARABIC-LANGUAGE TEXTS

THESIS

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in the
College of Engineering
at the University of Kentucky

By Zainab, M. Albujasim

Lexington, Kentucky

Director: Dr. Jerzy W. Jaromczyk Associate Professor of Computer Science
Lexington, Kentucky
2014

ABSTRACT OF THE THESIS

SEARCH QUERIES IN AN INFORMATION RETRIEVAL SYSTEM FOR ARABIC-LANGUAGE TEXTS

Information retrieval aims to extract from a large collection of data a subset of information that is relevant to user's needs. In this study, we are interested in information retrieval in Arabic-Language text documents. We focus on the Arabic language, its morphological features that potentially impact the implementation and performance of an information retrieval system and its unique characters that are absent in the Latin alphabet and require specialized approaches. Specifically, we report on the design, implementation and evaluation of the search functionality using the Vector Space Model with several weighting schemes. Our implementation uses the ISRI stemming algorithms as the underlying stemming technique and the general Arabic stop word list for building inverted indices for Arabic-language documents. We evaluate our implementation on a corpus consisting of selected technical papers published in Arabic-language journals. We use the Open Journal Systems (OJS) from the Public Knowledge Project as a repository for the corpus used in the evaluation. We evaluate the performance of our implementation of the search using a classic recall/precision approach and compare it to one of the default multilingual search functions supported in the OJS. Our experimental analysis suggests that stemming is an effective technique for searches in Arabic-language texts that improves the quality of the information retrieval system.

Keywords: Open Journal, Arabic language, Vector Space Model, Information retrieval,

Ranking schemes

Zainab, M. Albujasim

30/July 2014

SEARCH QUERIES IN AN INFORMATION RETRIEVAL SYSTEM FOR ARABIC-LANGUAGE TEXTS

By

ZAINAB ALBUJASIM

Dr.Jerzy W. Jaromczyk______________

Director of Thesis

Dr. Mirosław Truszczyński_________

Director of Graduate Studies

7/ 30 /2014

Data

To my husband and family

Acknowledgments

I would like to express my deepest appreciation to my adviser, Dr. Jerzy, Jaromczyk, for his persistent support since I first joined the Department of Computer Science. I could not have finished this work without his guidance and encouragement. Secondly, I would like to thank my committee members, Dr. Mirosław Truszczyński and Dr. Zomming,Fei for their dedication.

I would like to thank my family and my friends, for their support and prayers, and especially my caring husband Akram for his endless devotion, enthusiasm, and patience. Without his unconditional support and love, I could not have completed this study. Thank you.

TABLE OF CONTENTS

List of Figures

List of tables

1 Chapter One: Introduction

This study is concerned with the search component of Information Retrieval (IR) systems for Arabic texts. Finding relevant information in an ever-growing amount of Arabic data is important, but this process presents unique challenges that IR systems based on other languages have not addressed. One of the goals of all IR systems is to make locating relevant information as accurate and efficient a process as possible. The existing solutions depend on the data collection's character (structured or unstructured), type (text, image, video, music), format (digital or non-digital, alphabet, encoding, language), volume (artifact, document, journal, library or the Internet), and purpose (information that provides evidence, relations and context, or data used in inference).

Research in IR has intensified over the last three decades. A lot of researchers' efforts have been devoted to developing more powerful IR models. The focus on IR has become even more prominent since the arrival of the Internet, which led to an exponential growth in the sheer amount of digital documents. Many of these studies, however, have focused exclusively on English documents, while IR that focuses on Arabic documents has remained a relatively neglected field. In the last decade, however, interest in Arabic IR has grown tremendously, leading to the development of new techniques and algorithms. Several factors have contributed to this recent interest in supporting technologies for the Arabic Language. Arabic is a major world language: it is the official language for twenty countries and the mother language of more than 300 million people [Farghaly, and Khaled *,Shaalan,2009*]. For that reason, IT companies are interested in attracting Arabic users to help build a global online communication environment that supports multilingualism. Clearly, economic, cultural and security issues are all contributing to the international interest in the development of Arabic-related technologies.

The general goal of this thesis is to compare a stemming-based approach to search queries with the language-agnostic search tools that are available in the Open Journal System (OJS) Project, the open source system for collecting and editing scientific journals. Our hope is that this research will lead to better methods of organizing collections of articles written in the Arabic language and will provide users with the best utilities for building inverted indexes for search queries.

We build a search and indexing component for an IR system for Arabic-language texts using the vector space model. We use the *inverted index* technique that is useful in several other techniques such as stemming and stop word removal. We use the Information Science Research Institute's (ISRI) stemmer module to stem the Arabic words. Finally, we evaluate the proposed system using the precision and recall method, and we compare these results to those of the OJS's default search function for Arabic content.

In our development, we have used *Python* version 3.3 as the programming language. Python is an open source and general purpose language that has many modules that support the specific tasks of our study. We use *UTF_8* encoding for our dataset and queries. The Python environment (version 3 and above) supports UTF_8 encoding. In our implementation we have used the default dictionary data structure for building the *inverted index*. (`defaultdict`) is a specialized data type container in the collection module. (`defultdict`) is a hash table data structure. We have also used the `nltk.stem.isri` stemming module in Python as an Arabic stemmer. The `nltk.stem.isri` module is one of the `nltk` package modules. We have used the regular expression module (`re`) for the normalization process (removing vowel signs from an Arabic word).

The following section starts with a brief description of the Arabic language and focuses especially on features of the language that are potentially relevant to information retrieval systems. We then describe the basic models for supporting search queries.

1.1 The Characteristics of the Arabic Language

Arabic is one of the most ancient languages in the world and remains a major language today. It belongs to the Semitic language family, which includes Akkadian, Aramaic, Ethiopic, Hebrew, Phoenician, Syriac, and Ugaritic [*Moukdad, 2004]*. Arabic is the official language in the Arab world as well as the religious language for Muslims throughout the globe, since it is the language of Islam’s holy book, the Quran. In addition, it is one of six major languages represented by the United Nations.

The Arabic language can be classified into Classical Arabic (CA) and Modern Standard Arabic (MSA). CA refers to the ancient form of Arabic used in the Quran and other religious books. MSA is the common writing form and the official language for communication in the Arab world today. It is used in schools, media, magazines, scholarly journals, and newspapers [Al-Maimani, Naamany, and Abu Bakar, 2011]. CA is a stable language that has its own style and vocabulary and has proven immune to major changes for over fifteen centuries. Arabic people can comprehend CA with relative ease [Farghaly and Khaled, 2009]. At the same time, Arabic is highly diglossic: each Arabic country has three or more variants of spoken Arabic that are used in informal situations. Each informal dialect has its own word pronunciations and vocabularies [Albalooshi, , Nader, and Al-Jaroodi, 2011].

The Arabic Alphabet contains 29 letters counting “Alhmeza” (pronounced like the “a” in” apple”), which, in some cases, behaves as a diacritic. Table 1-1 shows the Arabic letters and their names. An Arabic text is written horizontally from right to left; however, numbers in Arabic are written from left to right as shown in figure 1-1 below. Arabic letters undergo slight modifications when they are combined within a word. Some letters may have one or more written forms depending on its position within a word, while other letters may have only one form. For example, “Alhmza” is written in the same form regardless of its position in a word. Some letters have two, three, or four forms. For instance, the letter “ع” has four forms (ع, ـع, عـ, ـعـ) [Molijy, Ismail, and Izzat, 2011].

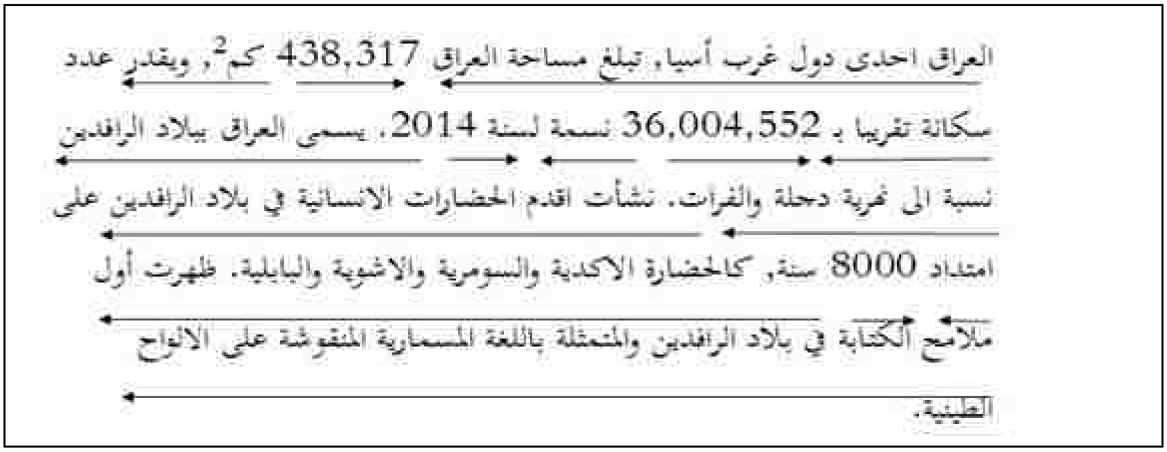
The Arabic alphabet primarily consists of consonants. Vowel signs are written above or under the consonant letters. The diacritic characters such as “Damma,” “Fathah,” ”kasra,” and “shaddah” control the pronunciation of a word. Any misuse of these diacritics can produce a word with a different denotation. The process of writing a vowel sign over or under a consonant letter to indicate the correct pronunciation is called vocalization. In

Modern Standard Arabic, vocalization is dropped from text, because an Arab reader can easily understand the meaning of a word based on the context. Figure 1-1 shows an example of MSA, figure 1-3 shows an example of Arabic text with vocalization (classical Arabic), and figure 1-2 shows the English translation of the text in figure 1-1 and figure 1-3.

Table 1-1 The Arabic letters and their names and the respective English letter.

(Notice: some letters have the same shape but are distinguished by dots.)

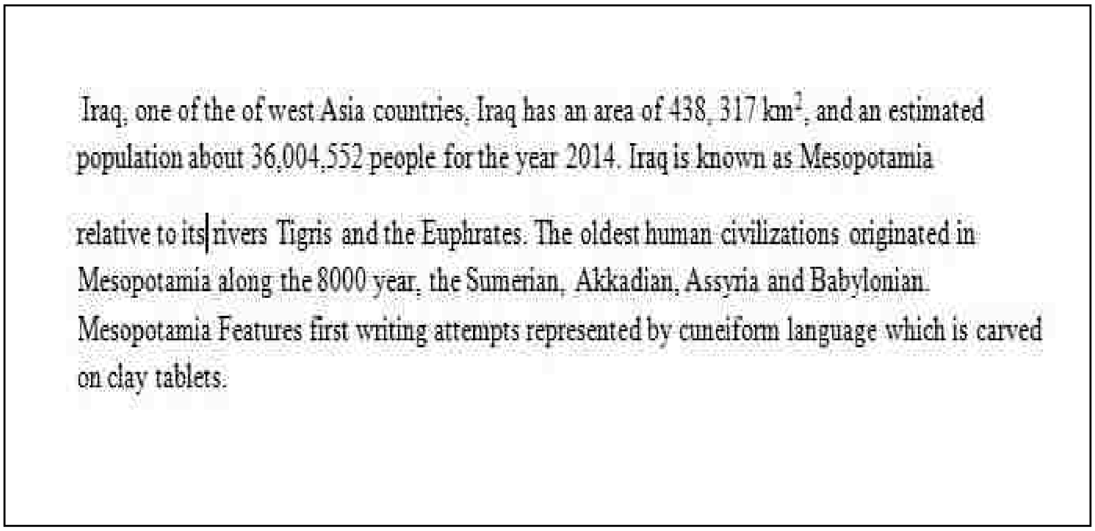
ر	ذ	د	خ	ح	ج	ث	ت	ب	أ
/raa/ r	/thal/ th	/dal/ D	/khaa/ kh	/haa/ h	/jeem/ j	/thaa/ th	/taa/ t	/baa/ b	/alif/ a
ف	غ	ع	ظ	ط	ض	ص	ش	س	ز
/faa/ f	/ghain/ gh	/`ain/ `	/thaa/ d	/taa/ t	/thad/ d	/sahad/ s	/sheen/ sh	/seen/ s	/zaa/ z
	ء	ي	و	ه	ن	م	ل	ك	ق
	/a/ a	/yaa/ Y	/waw/ o	/haa/ h	/noon/ n	/meem/ m	/laam/ l	/kaaf/ k	/qaaf/ q



العراق احدى دول غرب أسيا, تبلغ مساحة العراق 438,317 كم^2, ويقدر عدد
سكانة تقريبا بـ 36,004,552 نسمة لسنة 2014. يسمى العراق ببلاد الرافدين
نسبة الى نهرية دجلة والفرات. نشأت اقدم الحضارات الانسانية في بلاد الرافدين على
امتداد 8000 سنة, كالحضارة الاكدية والسومرية والاشورية والبابلية. ظهرت أول
ملامح الكتابة في بلاد الرافدين والمتمثلة باللغة المسمارية المنقوشة على الالواح
الطينية.

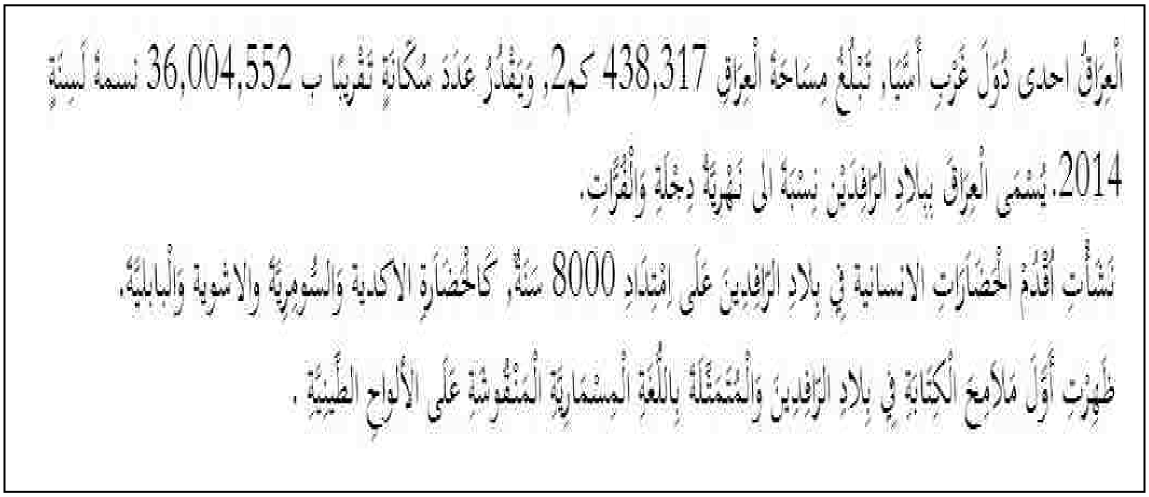
Figure 1-1 Modern Standard Arabic text with writing direction indicated.

(Note: the direction of the written text moves from right to left, while numbers are written from left to right.



Iraq, one of the of west Asia countries, Iraq has an area of 438, 317 km^2, and an estimated population about 36,004,552 people for the year 2014. Iraq is known as Mesopotamia relative to its rivers Tigris and the Euphrates. The oldest human civilizations originated in Mesopotamia along the 8000 year, the Sumerian, Akkadian, Assyria and Babylonian. Mesopotamia Features first writing attempts represented by cuneiform language which is carved on clay tablets.

Figure 1-2 The English translation of the text in figures 1-1 and figure 1-3



اُلْعِرَاقُ احدى دُوَلُ غَرْبِ أَسْيَا, تَبْلُغُ مِسَاحَةُ اُلْعِرَاقِ 438,317 كم2, وَيُقَدَّرُ عَدَدُ سُكَّانِهِ تَقْرِيبًا ب 36,004,552 نسمة لِسَنَةِ
2014. يُسَمَّى اُلْعِرَاقُ بِبِلَادِ الرَّافِدَيْنِ نِسْبَةً الى نَهْرَيْهِ دِجْلَةِ وَاُلْفُرَاتِ.
نَشَأَتِ أَقْدَمُ اُلْحَضَارَاتِ الانسانية فِي بِلَادِ الرَّافِدَيْنِ عَلَى اِمْتِدَادِ 8000 سَنَةٌ, كَاُلْحَضَارَةِ الاكدية وَالسُّومَرِيَّةِ والاشوية وَاُلْبَابِلِيَّةُ.
ظَهَرَتِ أَوَّلُ مَلَامِحِ اُلْكِتَابَةِ فِي بِلَادِ الرَّافِدَيْنِ وَاُلْمُتَمَثِّلَةُ بِاللُّغَةِ اُلْمِسْمَارِيَّةِ اُلْمَنْقُوشَةِ عَلَى الأَلْوَاحِ الطِّينِيَّةِ .

Figure 1-3 The same text in figure 1-1 but with vocalization (Classic Arabic).

Arabic is a rich and flexible language. Due to the morphological characteristics, tens or hundreds of words can be derived from the same root. This is why Arabic has three times as many words as English, with approximately five million words that originate from around 11,400 roots. However, only 1,200 roots are typically used in Modern Standard Arabic (MSA) [Al-Maimani, Naamany, and Abu Bakar, 2011].

Arabic speech mainly consists of verbs and nouns, with the verb typically coming before the noun. Arabic verbs have two tenses, past and non-past, and two voices, active and passive. Nouns in Arabic have three grammatical cases (nominative, genitive and accusative) and two genders (feminine and masculine). For example, the feminine form of the Arabic word for teacher is “ المعلمة“ or “mualma,” and the masculine word for teacher is “المعلم” or ”mualm.” Nouns also have three numbers: singular, dual and plural.

These and other characteristics of Arabic pose new challenges that demand new solutions for information retrieval systems and applications. For example, conjunctions and prepositions are written as a continuous stream and linked to a word. The Arabic phrase for “and she said,” for instance, is indexed as a one string: “ وقالت“ or "wa kalt.” Ideally, the prepositional letter (“و”) (“waw”) should be treated as a stop word and removed from the text, but due to the continuous Arabic script, it is considered one string [Mukdad, 2001]. Likewise, articles, like the definite article “ الـ" or “al,” are often combined with other words.

Another difficulty with the Arabic language is that most nouns do not follow the basic rules for pluralization, which are adding the suffix (“ون”) (ūn) for a masculine noun or the suffix (“ات”) (“āt”) for a feminine noun. Many plural nouns are irregular, and some words are reshaped to assume the plural form. The plural form can be derived by adding or removing letters from words or by adding suffixes and prefixes. For example, the plural form of “كتاب” (book) is “كتب”. Note that the third letter ("ا") (“alif”) is removed in the plural form. In another example, the plural form of the word “قلب” (heart) is “قلوب.” Note that a letter (“و") (“waw”) is added after the second letter to make the word plural.

The Arabic language has its own character set; these characters can be represented using several encodings such as ISO-8859-6, Windows-1256, and Unicode. Diacritics may be encoded as a letter in an integrated encoding, which can be difficult to handle, or as a separated encoding, which is considerably easier to manage. However, the encoding alone does not solve the issues that accompany valid representations of Arabic text, because the form of each letter changes with regard to its position within a word due to the direction of the text and the glyph phenomenon. The digital appearance of Arabic text is determined by the “engine rendering” used to display the appropriate format [Wikipedia].

1.2 Arabic IR Challenges and Techniques.

Several IR techniques have been developed over the last three decades. This section gives a brief description of some of the challenges that face Arabic IR and techniques that could improve the output of Arabic IR.

One of the challenges that researchers face when they deal with Arabic text is normalization. MAS has some inconsistency regarding diacritics within a word. A word may have one or two vocalization marks. In the past, the placement of these diacritics was inconsistent with no set standard. Thus, it has become necessary to convert the processed text into a united form [Farghaly, and Khaled, 2009]. For example, the letter “Alif” has different forms (“أ”,”إ”,”آ”) based on the “Alhmza” mark’s position below or above the letter. Without normalization, these forms are treated differently within a word, especially in information retrieval models with different text formats. Some text uses the verb for “read” (“أقرا” “aqraa”) in different formats: “أقرأ” ,"إقرأ” ,”اقرا.” Without normalization, an IR system is unable to retrieve all of the different formats of the same word. The result will only match the format of the user’s query. This could lessen the accuracy of an IR system. On the other hand, normalization increases the level of ambiguity. For instance, with normalization, a lot of words are treated the same, even if they hold totally different meanings [Farghaly and Khaled, 2009]. The word “الشَعر” or “alshaar” (“hair”), for instance, is difficult to distinguish from the word “الشِعر ” or “alshiru’’ (“poem”). IR systems that implement normalization will retrieve all documents that contain this ambiguous word, regardless of its meaning. In another example, the word “كتب” or “katab” holds two meanings. It could be the past tense of the verb “write,” or it could be the plural form of the noun “book.” As a result, the ambiguity of many Arabic words poses a difficult challenge to IR systems. Still, the meaning of normalized words can be inferred from the context. One of the techniques that has been proposed to help solve the problem of ambiguity is the *Word Sense Disambiguation* technique. This technique uses dictionaries and synonyms to extract meaning from surrounding words [Al-Maimani, Naamany, and Abu Bakar, 2011].

Secondly, Arabic is regulated by a complex morphology that has a significant effect on the performance of an IR system. Therefore, stemming must be implemented in every Arabic IR system [Darwish, and Douglas, 2007]. Stemming simplifies a word's format by removing prefixes and suffixes and mapping a word to its root. Basically, there are four types of roots in the Arabic language: tri-literal, quad-literal, penta-literal, and hexa-literal roots. Several stemming Algorithms have been proposed. Light stemming and root-based stemming (called aggressive stemming) [Al-Maimani, Naamany, and Abu Bakar, 2011] are the most popular algorithms. The light stemming approach works by eliminating prefixes and suffixes from a word, while root-based stemming maps a word to its root. For instance, the word " لاستعمالاتهم", which means "for usage," is mapped to the root "استعمل", which means "used," by using light stemming; however, aggressive stemming maps it to the root "عمل", which means "to do" [Al-Maimani, Naamany, and Abu Bakar, 2011].

Stop word removal is another technique that may be used with an IR system. Stop words are generally common words that appear most frequently in texts. They don't usually add significant meaning to the user's query and are thus considered irrelevant to the search. Stop word classification is based on the characteristics of a language; usually, articles, prepositions, and adverbs are considered stop words. Generally, removing stop words reduces the size of the index by 20% -30%. Removing stop words could also improve the performance of the retrieval process [El-Khair, 2006]. Stop word lists can be classified into two groups: dependent and independent. There are three ways to create a stop word list: by basing the list on the characteristics of the language, by using statistical information about the corpus at hand (dependent approach), or by a combination of the two means. The first stop word list, created by Fox, includes 421 words that were suggested based on English word usage [Fox, 1995]. Stop words should not be selected randomly; they should be selected based on an intimate knowledge of the language.

2 Chapter Two: Information Retrieval Models

2.1 The Definition of Information Retrieval

Information retrieval refers to the extraction of user-specified information from documents and files, ranging from books to online blogs, journals, and academic articles[Manning, Prabhakar, and Hinrich 2008]. The primary objective of IR is quickly and precisely retrieving from a collection a subset of information related to the user's interests [Pierre, Paolo and Padhraic, 2003]. According to Hiemstra, IR technology is a "combination of experiments and theory." Experiments are required to assess how the technology deals with the rapid growth of information and documents, and theoretical models help researchers avoid deductive reasoning during such experiments.

The importance of information retrieval has vastly increased since the appearance of the World Wide Web (www) and its expansive volume of electronic documents. IR has become a part of many people's daily lives. While ordinary users may not be familiar with the term IR, they are certainly well-acquainted with web-based search engines like Google, Yahoo, Ask, etc. Today, IR is very prominent, and it has become an exciting research field because of its pervasive presence in day-to-day life. [Liddy, 2005].

The goal of any IR system is to respond to user-requested information by providing reference documents that meet the desired criteria. Different factors determine whether or not a document is relevant to a user's query. One of these factors is the documents themselves—the scope of their content, how they are written, etc. The user is also an important factor (e.g. user knowledge, the reason for the search, etc.) [*Raghavan, and Michael, 1986*]. Because a document's relevance to a user's interest is subjective and depends solely on the user's judgment, IR systems may not be 100 percent accurate. What an IR system can do, however, is propose methods that can estimate the efficiency of the results and whether they meet the user's information needs based on his or her query [Raghavan, and Michael, 1986]. Relevancy is difficult to measure. Many factors can determine a result's relevance, but one of the most important factors is user satisfaction. If the user "likes" a document, the document will be considered relevant; if not, it will be considered irrelevant. Therefore, researchers are constantly trying to find new methods for defining the relevance of documents to a user's query. There are many theories related to document relevance called formal models (Hiemstra, Djoerd 2009). IR models differ from one another based on the various ways they compute the weight of matching documents. Each IR model serves some purposes better than others. Despite these differences, every IR system should maintain three basic operations: document representation, query formulation, and a process for matching queries with documents [Hiemstra, Djoerd 2009]. Figure 2-1 illustrates the basic operation of IR. The square shapes represent data, and the oval shapes represent processes.

First, the process for document representation is known as the *indexing process*. Typically, the indexing process runs offline, and it does not require user interaction. Second, the query formulation process transforms a user's information needs into a format that is recognized by the system (Boolean format, free text format, etc.). A user may try several queries until he/she gets satisfactory results. This process is called feedback. Finally, the matching process finds and ranks documents that contain the user's query terms. Usually, statistical information is used to rank documents from the highest to the lowest matching score, hopefully reducing the time that the user spends sifting through all of the returned documents [Hiemstra, Djoerd, 2009].

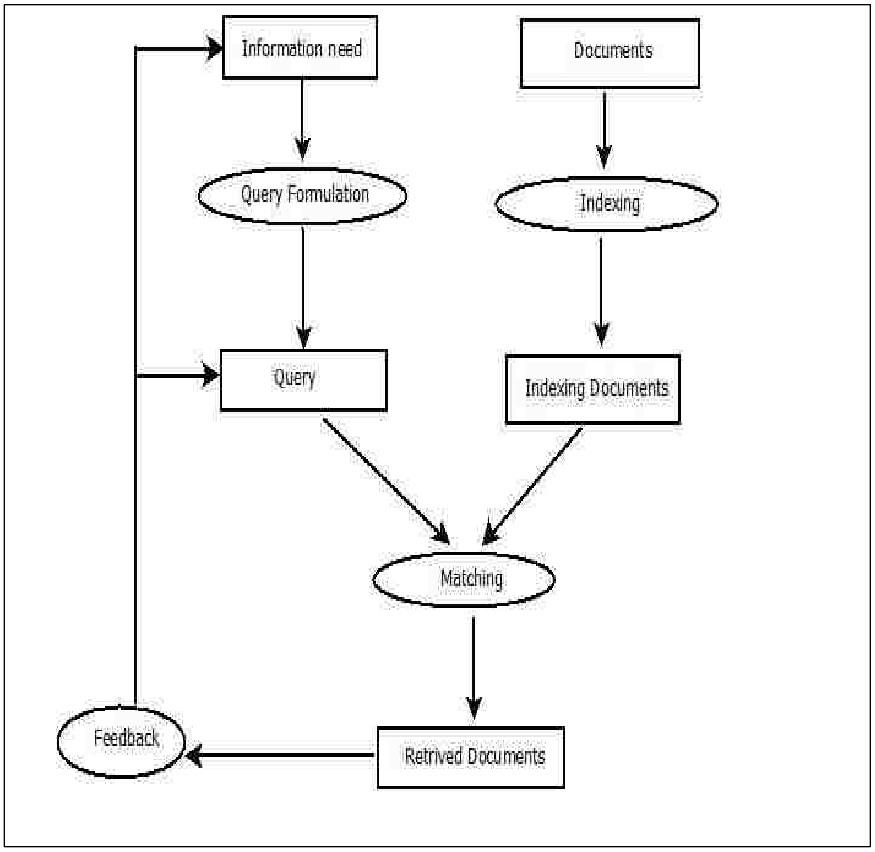


Figure 2-1 General information retrieval process. (Hiemstra, Djoerd, 2009)

2.2 Information Retrieval Models

In this section we give a brief description of well-known information retrieval models.

2.2.1 The Boolean Model

The Boolean model is the first IR model that requires structural language for a query. It uses the logical operations "AND," "OR," and "NOT." The Boolean model is categorized as an exact matching model. An exact matching model retrieves either all matching documents or no matching documents. In a large document collection the results might exceed one thousand documents, or there might be zero results if no matching documents are found [Manning, Prabhakar, and Hinrich 2008]. The Boolean model requires that the user have some knowledge of Boolean operations; therefore, not all users are able to attain satisfying results with this model. Additionally, the Boolean model has no way of favoring one document over another. In other words, it does not support document ranking. For that reason, it is less popular than other models. It is also time consuming, since it retrieves all matching documents, which in a large data collection could mean thousands of results. On the contrary, statistical models provide a score that indicates how well a document matches a query. In essence, the Boolean model builds a *matrix*; the rows of the *matrix* are the key terms, and the columns are the documents themselves. Each cell in the matrix contains either a zero or a one. Zero indicates that the term does not occur in the specified document, and one indicates that the term is present [Signal, 2001].

2.2.2 The Probabilistic Model

The Probabilistic model was originally proposed by Maron and Kuhans in 1960 [Baldi, Paolo, and Padhraic, 2003]. Since then, the proposed model has undergone several improvements and refinements. Today, several versions of the probabilistic model are available. The first version of the model, `BM1`, was introduced by Robertson-Sparck Jones in 1976. In 1998, version `BM5` was proposed by Robertson et al as a refinement of the first version [Baldi, Paolo, and Padhraic, 2003]. The basic idea of this approach is to define the question of a document's relevance to a query as a probabilistic problem. The model uses query terms and documents to measure relevance. Bayes' theory is used to compute the probability of a document's relevance or irrelevance. This requires an initial set of predetermined relevant and non-relevant documents to calculate the probability of a new document's relevance; thus, it treats relevancy as an a priori problem. Estimating the probability of a document's relevance using a set of predefined documents, however, is not a practical solution, since these predefined sets may not adapt well to the demands of new queries. Therefore, computing the occurrence of query terms is important to estimating the probability of a retrieved document's relevance. [Baldi, Paolo, and Padhraic, 2003].

The measure of probability can be defined as query q containing $t_1, t_2, \ldots, t_n$ words and document d.

$$score(q, d_i|t_1, t_2, t_n) = product\ (p(R|t_1) + p(R|t_i) + p(R|t_n) \quad \text{................} \quad \textit{(2-1)}$$

In the probabilistic model, the parameter $(p(R|t_i)$ needs to be enumerated. There are two ways of estimating the p quantity. The first method is to use a large data collection that includes millions of queries and millions of documents in order to compute the probability of each term occurring in a relevant document. The other method is to estimate the probability for a particular query based on which document is relevant and which one is not.

The positive aspects of this model are its strong theoretical foundation and its ability to rank documents by their probable relevance. On the other hand, the negative aspects of the model are that it treats probability as a binary condition, its term independency assumption is abstract, and it initially lacks relevance data [Liddy, 2005].

2.2.3 **The Language Model**

The language model was proposed by Ponte & Croft in 1998. The basic idea of this model is that it estimates a document's probability of generating a query instead of estimating the probability of a document's relevance. This model is also known as the Likelihood retrieval model. The language model uses other IR models and techniques such as probabilistic models and *n-gram* analysis. Over the years, several refinements and improvements have been developed to support and enrich the language model [Liddy*, 2005*].

2.2.4 **The Vector Space Model**

Salton (1971) proposed a new statistical model that has proven more effective and flexible than other IR models in some regards. In this model, a voluminous collection of documents is represented in multi-dimensional space, with thousands or millions of dimensions for a large collection. The terms of the documents become the dimensions, while the documents themselves are represented by points in space [Singh and Sanjay, 2012]. A document and a user's query are together represented as vectors (see figure 2-2). Since the number of terms in a document is limited, the vector of the documents can be very sparse. This sparseness helps reduce memory storage. Using the cosine similarity measurement of the angle between query and document vectors, this model measures the similarity between a query and a document. Experiments prove that the cosine similarity measurement is the optimal coefficient when compared to other similarity measurements such as Euclidean distance [Manning, Prabhakar, and Hinrich 2008] which could generate long distances between document vectors of different lengths, even if they share the same terms. On the contrary, the similarity in a cosine measurement is 1.0 for identical vectors and 0.0 for orthogonal vectors. Thus, the cosine is the best fit for measuring the similarity between two vectors. In addition, we could use the inner product of two vectors as another similarity measurement, but this would require that all vectors have identical lengths. The inner product of vectors of the same length is the same as the cosine measurement [Singhal, 2003].

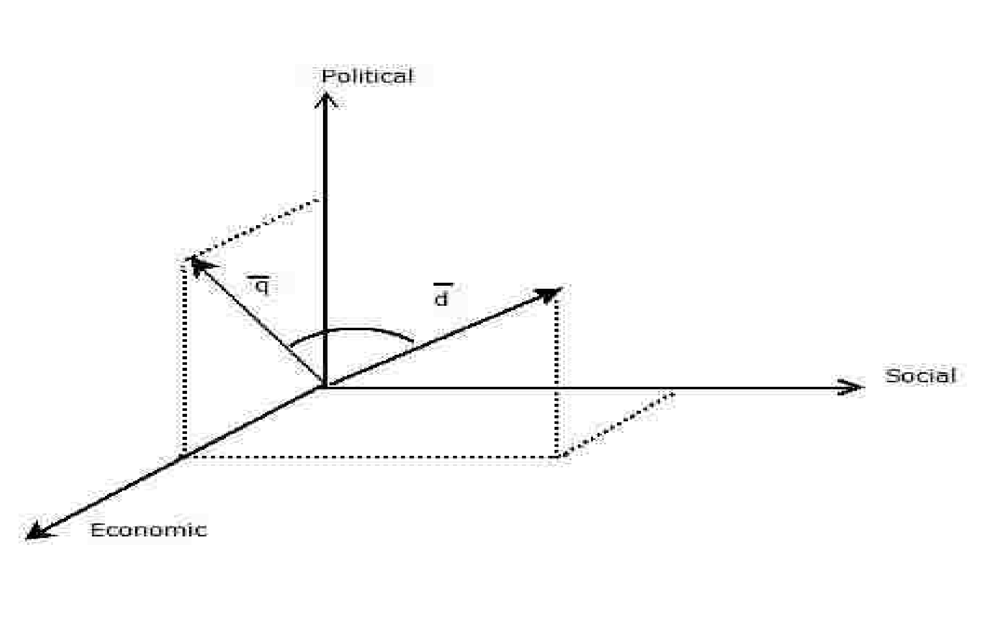


Figure 2-2 Representation of a query and document in the vector space model.

(Hiemstra, Djoerd, 2009).

2.2.4.1 Assigning Term Weight in the Vector Space Model

The term-weighted vector space model computes the frequency of a term within a document. Several weighting schemas have been developed to compute the weight of terms in a document.

- Term – count model (TF only) (naïve approach).
- TF-IDF (term frequency-inverse document frequency).
- Vector space model based on normalization.

2.2.4.1.1 Term – Count Model (TF Only) (Naïve Approach)

In Boolean vector space, a term in a document vector is represented as a 1 if it is present in a document; otherwise it is represented as a 0. But this approach to vector representation does not take into account the fact that some terms appear more frequently in a document than others. If a query term appears multiple times in a document, then it can be assumed that this document is more relevant to a user's query than others. For this reason, term frequency (TF) is important in the representation of a document vector. TF helps rank a document with multiple occurrences of a query term over a document with a single occurrence. A document with 10 occurrences of the term is more relevant than a document with one occurrence of the same term. However, it is not simply ten times more relevant; therefore, various methods have been used to modify TF [Manning, Prabhakar, and Hinrich 2008]

$$w_{t,d} = \begin{cases} 1 \log_{10} tf_{t,d} & , if \ tf_{t,d} > 0 \\ 0 & , otherwise \end{cases} \quad \ldots\ldots\ldots\ldots \text{(2-1)}$$

where the ($w_{t,d}$) is the weight of a term t in a document d.

2.2.4.1.2 IDF: Inverse Document Frequency

Inverse document frequency (IDF) was introduced to solve the problem of term equivalence. IDF is the proportion of the total number of documents in a collection over the number of documents containing a term [Manning, Prabhakar, and Hinrich 2008]. Naturally, not all terms in a document have the same importance regarding a user's query. Common words, for example, are usually excluded from a user's query. On the other hand, rare terms in a query are considered very important. IDF solves the problem of term equivalence by giving a higher weight to more meaningful words and a lower weight to more common words. For instance, a weight of 0 might be given to a stop word. As a result, IDF offers another potential advantage, in that it can be used as a filter for stop words in a document collection [Manning, Prabhakar, and Hinrich 2008].

$$idf = log_{10} \frac{N}{df_t} \text{ .... } (2\text{-}2)$$

Where *df* is the document frequency
N is the number of document in the collection
df <=N.

2.2.4.1.3 TF-IDF

TF-IDF combines two quantities: the document frequency (df) (the number of documents in which a specified term appears) and the inverse document frequency. IDF gives more attention to rare terms than to common words. Combining TF and IDF, therefore, increases the weight of the common terms in a document, while at the same time increasing the weight of rare terms in the collection. IDF decreases as the number of documents that contain a specified term increases [Manning, Prabhakar, and Hinrich 2008].

$$sore(q, d) \sum_{t \in q \cap d} tf.idf_{t,d} \text{................}(2\text{-}3)$$

$$w_{t,d} = \log(1 + tf_{t,d}) * \log_{10} N/df_t \text{ .... } (2\text{-}4)$$

2.2.4.1.4 Vector Space Model Based on Normalization

Salton and McGill (1983) first proposed a vector space model that is based on normalization [Pierre B., F. Paolo and S. Padhraic, 2003]. This model measures the cosine of the angle between two vectors in m-dimensional space. A vector can be normalized (given a length of 1) by dividing each of its components by its length. Here we use the *L_2* norm.

$$|x| = \sqrt{\sum_i {x_i}^2} \ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(2-5)}$$

$$\cos(\ q^{\rightarrow},d^{\rightarrow}) = \frac{q^{\rightarrow}.d^{\rightarrow}}{|q^{\rightarrow}||q^{\rightarrow}|} = \frac{\sum_{t=1}^{|v|} q_t d_t}{\sqrt{\sum_{t=1}^{|v|} {q_t}^2}\sqrt{\sum_{t=1}^{|v|} {d_t}^2}} \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(2-6)}$$

3 Chapter Three: Implementation of IR System Using Vector Space Model

In the previous chapter, we described the vector space model and the possible weighting schemes that we can use as a similarity measurement in implementing our IR system. Figures 3-1and 3-2 show the architecture of our implementation. In the next sections, we describe the elements of our design. Section 3.1 describes the steps of Arabic text processing; Section 3.1.1 describes the tokenization process; Section 3.1.2 describes the normalization process; Section 3.1.3 describes the general Arabic stop list, its content, and the stop word removal process; Section 3.1.4 goes through the stemming process; Section 3.1.4.1 gives a brief description of the available Arabic stemming techniques and algorithms; Section 3.1.4.2 provides details about the ISRI stemming algorithm; Section 3.2 describes indexing data structures; Section 3.5.1 describes the inverted indexing technique; Section 3.6 describes query processing; and Section 3.6.1 describes the type phase query that works with the vector space model.

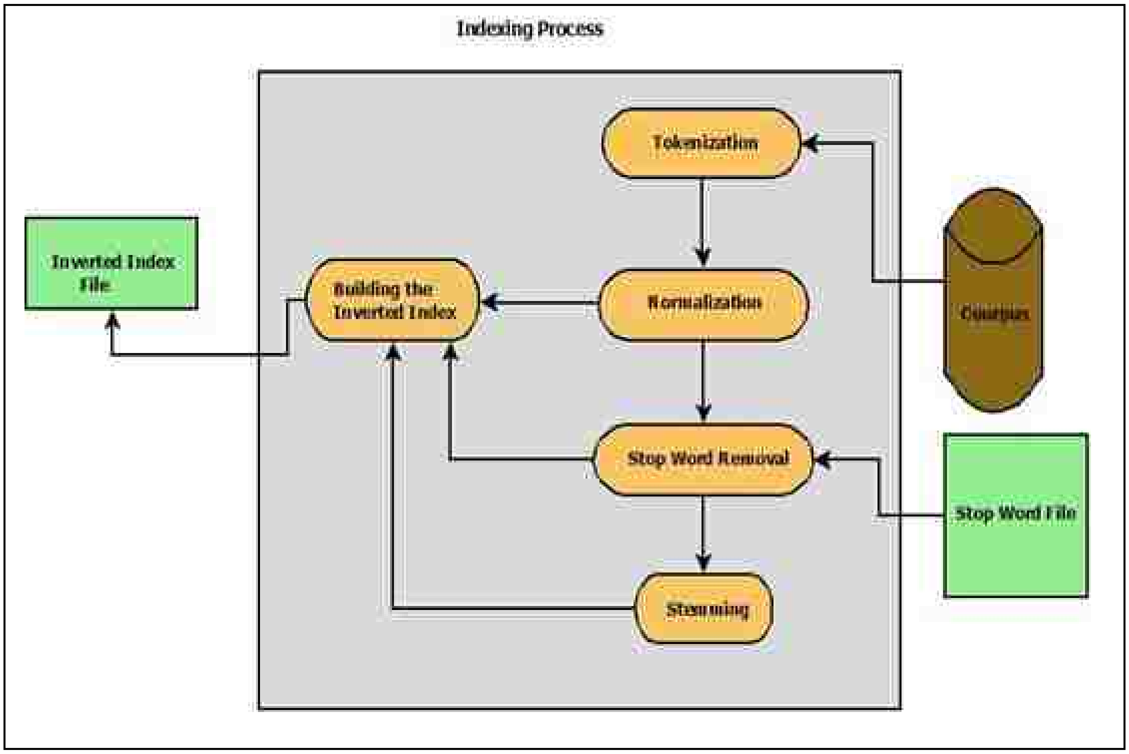


Figure 3-1 The architecture design of indexing process.

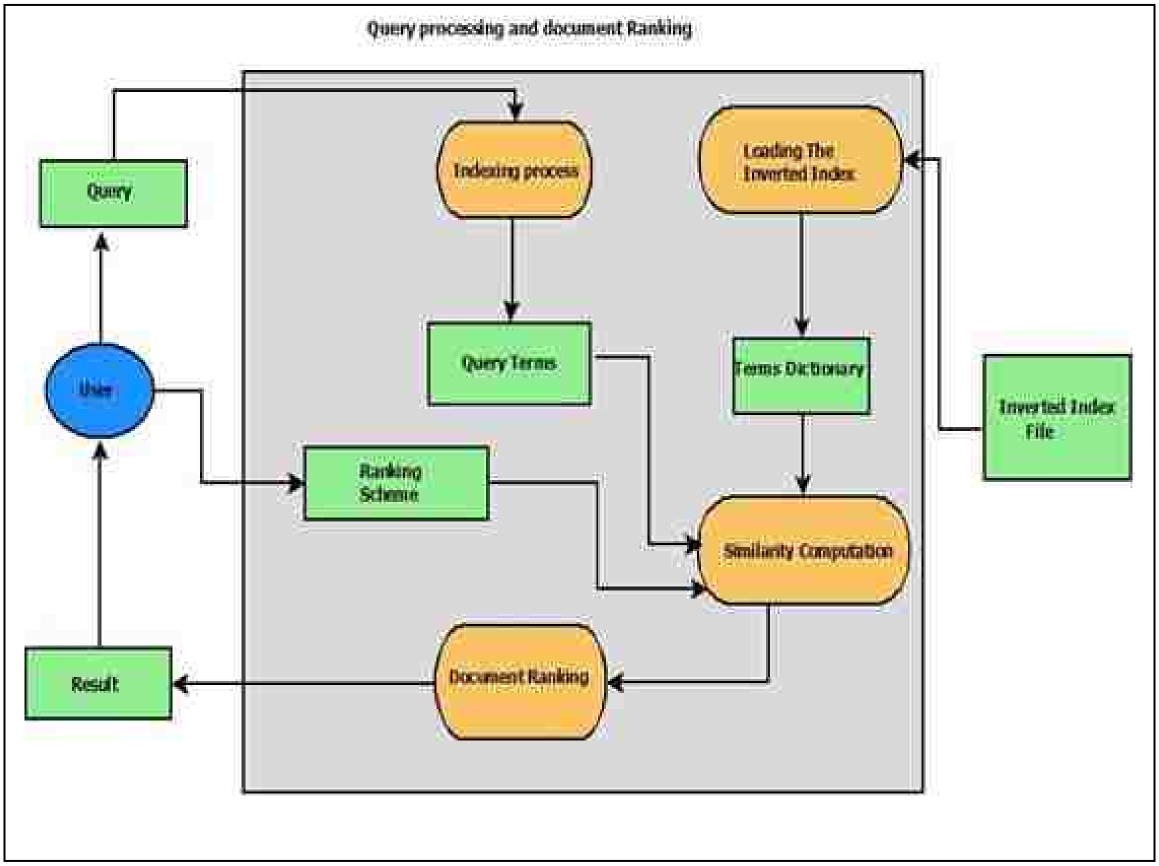


Figure 3-2 The architecture design of query processing and document ranking.

(Note: we do not draw the index process steps for the user's query, since it is the same in the indexing process figure 3-1)

3.1 Tools

3.1.1 The Programming Language

We use Python as the programming language because of its outstanding support for natural language processing (NLP). Python has a variety of packages and models that support our specific tasks. For example, the Natural Language Toolkit (NLTK) is one of the packages that supports NLP with various functions, from tokenization to advanced NLP techniques like clustering and textual classification. The NLTK was originally developed by the Department of Information Science at the University of Pennsylvania and has been further developed by many other contributors. Python (version 3 and above) is also ideal for our project because it supports Unicode encoding for the "str" type container and uses Unicode in its source code. These features make Python more suitable for representing Arabic characters. Unicode assigns a number called a "code point" to each character. In Python, each code point is represented as "/uxxxx", where "xxxx" is a four digit hexadecimal number [Bird, Ewan, and Edward, 2009].

Despite the fact that the Unicode string in Python is manipulated, storing the Unicode string in a text file or displaying it on a terminal requires that it be encoded as a stream of bytes. There are two ways of encoding a stream of bytes in Python. The first involves ASCII and Latin2, but this method can only cover a small range of Unicode characters. The other way to encode a string of bytes in Python is to use a multibyte method, such as UTF_8, which covers a full range of Unicode characters. This creates an additional problem when we try to process an Arabic string using different Python models: we cannot get the original format of the string. Some models display the string as Unicode numbers, while others display a string in an invalid format [Bird, Ewan, and Edward, 2009].

Python provides two mechanisms for decoding and encoding that help deal with text files and terminals. Each text file has its own encoding; in the decoding process, Python converts text file encoding to Unicode, while the encoding process converts Unicode to the proper encoding when the text is stored in a text file.

3.1.2 **Arabic Keyboards**

Arabic keyboards support both Arabic and Roman characters. Each key controls two characters: an Arabic letter and a Roman letter. Figure 3-3 shows an Arabic keyboard.

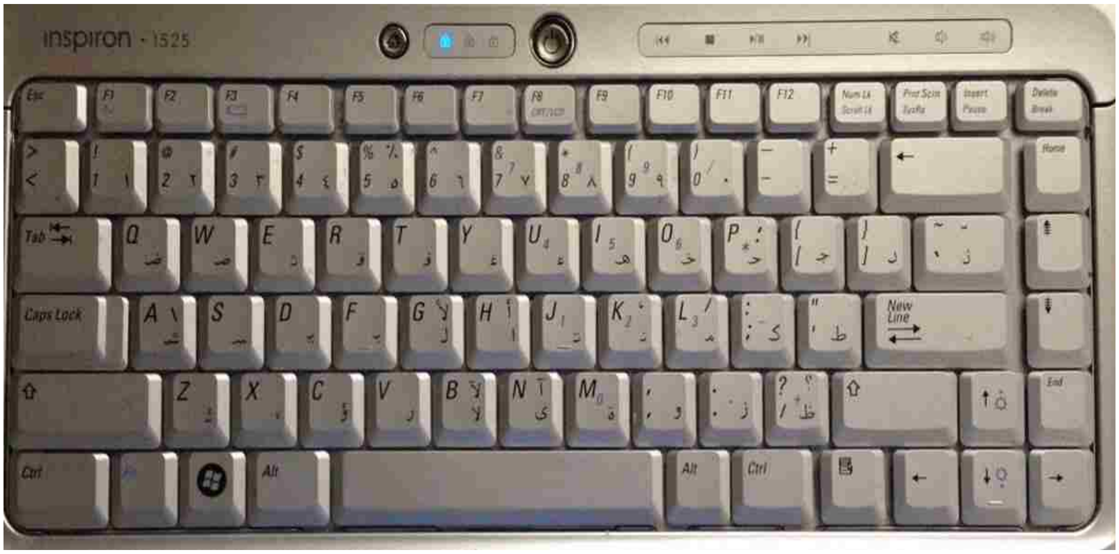


Figure 3-3 Our Arabic keyboard. On each key is marked two characters: an Arabic letter and a Roman letter.

3.2 Data Set Collection

The dataset we have used in our implementation of the Arabic IR is a collection of 100 Arabic documents with an average size of 12 KB and an average number of 5 pages. The articles were collected from three different Arabic magazines. They are mainly academic articles that cover several disciplines, including Arabic literature, philosophy, psychology, engineering, biology, art, geography and physics.

The name and the websites for these magazines are:

1- Babylon Journal of Applied and Pure Sciences.

Link: http://repository.uobabylon.edu.iq/applicable.aspx

2- Babylon Journal for Humanities.

Link: http://repository.uobabylon.edu.iq/humanities.aspx

3- Lisaan Al-Arab Magazine

http://lisaanularab.blogspot.com/.

3.3 Lexical Processing

In this section we describe the steps for indexing the document collection and storing the indexer as an inverted index

in a text file. Let D= {d1, d2, d3…, dn} be the collection of the documents, where N is the number of documents in the collection. Documents are stored as text files using UTF_8 encoding.

3.3.1 Tokenization

Tokenization is the process of extracting terms and keywords from documents, eliminating punctuation and special characters. In Arabic, a word is a set of characters that are linked together through complex mutation and are separated from other words by a white space [Manning, Prabhakar, and Hinrich 2008]. Although white spaces are generally used as string separators within an Arabic text, they also carry other implications, because prepositions and pronouns are written in a continuous string with the words they modify, producing complex tokens. This problem could be solved using a good stemming algorithm.

3.3.2 Text Normalization

Normalization is a preprocessing stage that employs NLP. The general aim is to clean up a text by removing punctuation and numbers. In English documents, normalization converts capital letters to lowercase letters. Therefore, normalization in English is relatively easy, especially considering how many packages and tools there are that support English and are available in most programming languages. Unfortunately, this is not the case for Arabic. In Arabic, normalization eliminates vocalization marks and converts the text into a more unified form. There are few premade tools available that support Arabic normalization.

In our implementation we have used the regular expression (re) Python model to remove the vocalization marks from the text. We have tried to normalize the different forms (,أ ,إ ا ,آ ,ٱ) of the letter "ا" “alif” to the standard form (“ا”). Also, the final letter”ى” is changed to the form (“ي”) and the letter ("ه") to (“ة”).

3.3.3 Stop Word Removal

In our implementation we have used the Arabic general stop list without any additions. The Arabic general list was created by Abu El-khair (2006) and is based on the structure and characteristics of the Arabic language. It contains all possible words and articles that may be considered stop words. The list also contains many categories: adverbs, conditional pronouns, interrogative pronouns, prepositions, referral names, relative pronouns, transformative verbs, verbal pronouns, and others. Due to Arabic's rich morphological characteristics, the general Arabic stop word list is triple the size of the English stop word list [El-Khair, 2006]. In Arabic, pronouns may have more than one form, whether it is feminine, masculine, singular, dual, or plural. For example, the pronoun "these" has six forms in Arabic: "هاتان" "hatan" for feminine nominative, "هاتين" "hatean" for feminine genitive, "هذين" "hethan" for masculine nominative, "هذان" for masculine genitive/accusative, and either "هؤلاء" "haa'ulaa" or "اولئك"" uulaa'ika" for feminine-masculine. Secondly, pronouns and prepositions are combined [*Chen, and Fredric, 2002*]. Another example of a stop word in Arabic is the Arabic phrase for "on you," which in Arabic is "عليك"; this phrase is one string, and it too has several forms based on the pronoun being used: the form "عليك" "aluka" for the singular masculine, the form "عليكي" "alukee" for the singular feminine, the form "عليكما" "alukma" for the dual masculine, the form "عليكم" "alukm" for the plural masculine, and the form "عليكن" "alukn" for the plural feminine.

The high frequency of Arabic stop words can adversely affect the weighting process, which makes the removal of stop words all the more critical. Stop word removal significantly reduces document length, which is reflected in the weighting scheme [El-Khair, 2006]. According to El-Khair, removing Arabic stop words could reduce the indexing process by 30-50% in a large collection. Therefore, using stop word removal in the indexing process is an essential step in any Arabic IR system.

3.3.4 **Stemming Process**

Stemming is a technique that is common to most search engines. Stemming is the process whereby morphological variants of words are mapped onto a single root. In the early days of IR, when memory was more limited, stemming helped reduce the size of an index. To this day, stemming continues to prove its effectiveness on the performance of IR systems [Liddy, 2005].

3.3.4.1 Arabic Stemming Algorithms

Unlike the English language, which has straightforward stemming rules that are easily implemented, the Arabic language is morphologically complex, with many suffixes, infixes, and prefixes that are difficult to remove. Indeed, removing suffixes has proven unhelpful in Arabic stemming algorithms. Arabic stemming is a challenging issue, due to the language’s complex morphological rules. Words in Arabic are derived from roots. For example, the root verb “كتب” ”katab” (“wrote”) has a long list of derivatives. The word “مكتبة” “maktaba” (“library”), ”كاتب” “katb” (“writer”), “مكتوب” “maktop” (“letter”), and “كتب” “kotop” (books) are all based on the same root word. Therefore, it is important that any Arabic IR system includes a stemming algorithm that maps these different word forms to one indexing entry. Several stemming algorithms have been proposed to accomplish this, each using different approaches such as manual stemming, which uses dictionaries, light stemming, which is based on removing suffixes and prefixes, aggressive stemming, which is based on morphological analyses of root words, and the clustering approach, which groups similar words in one cluster [Atwan, Ghassan, and, 2013]. In our implementation we have used The *ISRI* stemming method, which is part of `nltk.stem` package from the Python distributions.

3.3.4.2 The ISRI Arabic Stemming Algorithms

The *IRSI* Arabic stemmer is an Arabic stemmer algorithm that was developed by Taghva, Elkoury, and Coombs (2005) at the Information Science Research Institute at the University of Nevada. It follows the strategy of Khojo’s Arabic stemmer, with one important exception: it does not map the obtained root to the dictionary of existing Arabic roots. Because several modifications have been developed to improve this particular algorithm, ISRI is considered a refinement of Khojo’s original stemmer.

In short, the ISRI Arabic stemmer works as follows:

As a preprocessing step, two tables are defined; Table 3-1 shows the affixes and diacritics that need to be removed from a word as preprocessing steps. The algorithm also works on normalized “alhamza” and “Alalf” letters, since these two letters have different written forms based on their position in a word. All of these letter forms ("ئـ",“ء”,“ئ”,“ أ”) normalize to one form, which is (“أ”). In addition, the letter “و” “waw” is removed if it precedes a word. After these preprocessing steps, the algorithm stems a word to its two or three roots. Figure 3-1 shows the Arabic word patterns and roots with some examples of the each pattern. The algorithm follows a particular procedure on finding a word’s root based on a word’s length. The algorithm then tries to match a word with a pattern (shown in Table 3-1 using the length of a word as a key. If the pattern matches the word, the relevant root will be returned. If the algorithm does not recognize the pattern of the word, it will try to eliminate suffixes or prefixes, one character at a time, and repeat the matching procedure until a matching pattern is found. The algorithm attempts to return a root with a length of at least three characters. Although the algorithm works well, it fails to find the correct roots for some words, especially words that have been adopted from other languages and thus do not have an Arabic root. In these cases, the algorithm still tries to find a root.

Table 3-1 Affix sets (Taghva, Elkoury, and Coombs (2005))

Set	Description	Examples
D	Vocalizations	صُ,صَ,صّ,صِ,صْ,صٌ,صً,صٍ
P3	Prefixes of length three	ولل,وال,كال,بال
P2	Prefixes of length two	ال,لل
P1	Prefix of length one	ى,ت,ن,ا,ل,ب,ف,س,و
S3	suffixes of length three	تمل,همل,تان,تين,كمل
S2	suffixes of length two	ني,وا,ما,هم,ون,ات,ان,ين,تن,كم,هن,نا,يا,ها,تم,كن
S1	Suffixes of length one	ة,ه,ي,كـ,ت,ا,ن

(Note: This table shows the diacritic marks and affix characters that are removed in the preprocessing stages of the IRSI algorithm. The letter "ص" in the first row of the table is not a diacritic sign; however, it is written as an example of a consonant letter that has a diacritic mark on it [Taghva, Elkoury, and Coombs 2005].

Table 3-2 Arabic patterns and roots (Taghva, Elkoury, and Coombs (2005))

Set	Description	Examples
PR4	Length four patterns	فاعل,فعول,فعلة,فعال,فعيل,مفعل
PR53	Length five pattern and length four roots	تفاعل,افتعل,افعال,فعالة,فعلان,فعولة,تفعلة,تفعيل,مفعلة ,مفعول,فاعول,فواعل,مفعال,مفعيل,افعله ,فعائل,منفعل,مفتعل,فاعلة,مفاعل,فملاع,بفتعل,تفتعل,ف عالي,انفعل
PR54	Length five pattern and length four roots	تفعلل,افعلل,مفعلل,فعلله,فعلان,فعالل
PR63	Length six pattern and length three roots	استفعل,مفعاله,افتعال,افعوعل,انفعل,مستفعل
PR64	Length six pattern Length four roots	افنلل,افعلال,متفعل

3.4 Data and File Structure of Information Retrieval

The effectiveness of text retrieval depends largely on having the appropriate data structure in which to store a text [Baldi, Paolo, and Padhraic, 2003]. Different methods have been proposed, including clustering (Salton 1971) and *signature files* (Faloutsos and

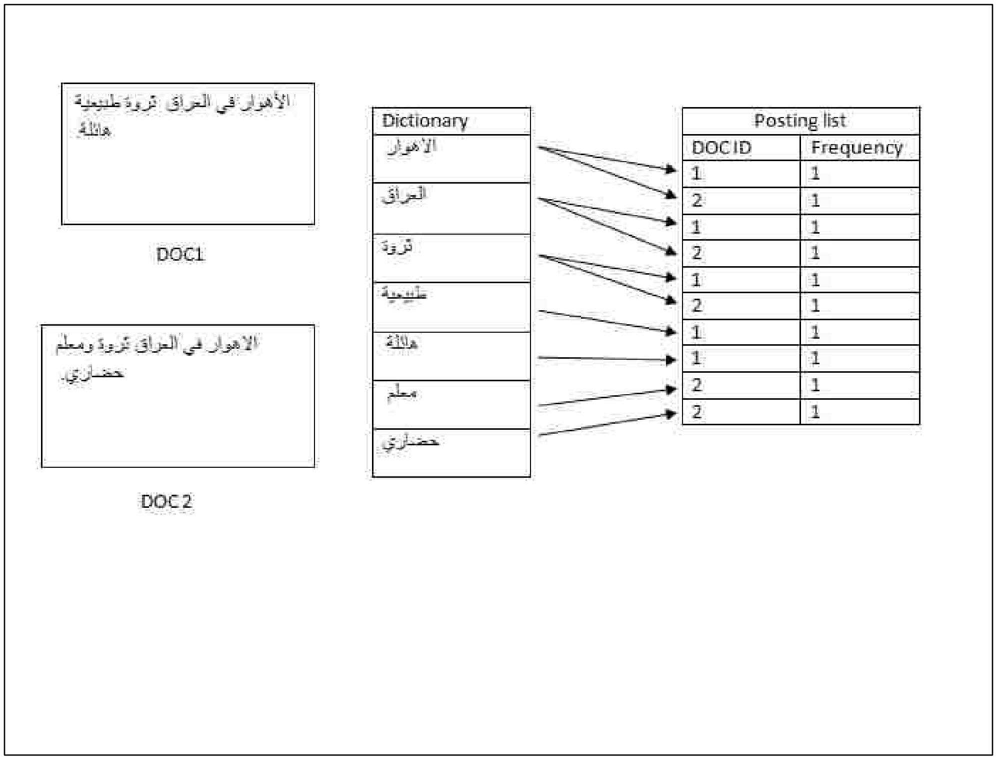
Christodoulakis 1984). The *Suffix tree* and *suffix array* were introduced by Manber and Myers (1990), but these were not capable of storing a large document collection. The method that has proven most successful in IR systems uses a data structure called an inverted index. The inverted index was proposed by Berry and Browne (1999) and Witten et al. (1999) (Baldi, Paolo, and Padhraic, 2003*).* In this method, each term in a collection is mapped to its occurrences in the collection. The set of terms is called a vocabulary (V). Each term (t) in the inverted index has a pointer p (t) that points to the posting list or bucket, which is a list of all the occurrences of a term (t). The size of the *inverted index* is Ω (|V |), and it can be stored in the main memory [Baldi, Paolo, and Padhraic, 2003]. An inverted index can be implemented using a hash table; in this case, the expected time is independent of the size of the vocabulary. Posting lists and documents should be stored on disk. After obtaining the posting lists for each term, we can combine the posting lists using one of the set operations corresponding to the Boolean operation in the query. In free text query, the intersection set operation is used to obtain a final posting list for all query terms [Manning, Prabhakar, and Hinrich 2008].

3.4.1 **Inverted Index**

An inverted index is an optimized data structure that can be used for information retrieval. An inverted index is like a conventional index found in the back of a book that maps a key term to a page number [Manning, Prabhakar, and Hinrich 2008]. The basic idea for building an inverted index is to keep a dictionary of the unique terms in the collection. For each term in the collection, we maintain a list of documents (by document IDs) in which the term occurs as well as a number for the term’s frequency in the specified document. This list is called a *posting list*. The posting list is stored in the secondary storage, while the dictionary is stored in main memory [Manning, Prabhakar, and Hinrich 2008]. Figure 3-4 shows the structure of an inverted index of two documents. In this study we use a Python data structure called *defaultdict* to store the dictionary, and we use Python’s default dictionary as a data structure for storing the posting list. The

Python dictionary is a *hash table* data structure, which means the lookup operation in the Python dictionary is O (1).

The text should undergo several preprocessing operations before it can be stored in an inverted index. Text is typically stored as bytes in a digital document, and bytes need to have the correct encoding schemas. For example, the typical encoding schema for English text is *ASCII*; however, Arabic encoding is more complicated. Arabic text thus requires *multibyte* encoding like Unicode UTF-8 [Manning, Prabhakar, and Hinrich 2008].


الأهوار في العراق ثروة طبيعية هائلة
DOC1
الاهوار في العراق ثروة ومعلم حضاري.
DOC 2
Dictionary
الأهوار
العراق
ثروة
طبيعية
هائلة
معلم
حضاري

Posting list	
DOC ID	Frequency
1	1
2	1
1	1
2	1
1	1
2	1
1	1
1	1
2	1
2	1

Figure 3-4 Inverted Index of two Arabic documents.

As shown in Figure 3-4, the terms “العراق”,”الاهوار”,”ثروة”appear in both documents. Also, the stop word “في”is removed, and the other words ”طبيعية”, "هائلة"appear in doc1; only “معلم”,”حضاري” appear in doc2. Other useful information may be extracted from an inverted index. For example, the number of documents that contain a term is the size of the posting list for a term.

3.5 Query Processing

Most IR systems redo all document indexing steps for each user query, including tokenization, stop word removal, and stemming. The only difference between document indexing and query indexing is that query indexing should be run in real time while a user is waiting for the results [Liddy, 2005].

3.6 Phrase Query

The vector space model, by definition, does not support bi-word queries, because it represents documents as vectors, which in turn causes the relative order of terms within a document to be lost. The computing of the IDF quantity should therefore be extended to include bi-word queries. Since indexing cannot be used for phrase query, the vector space model offers an alternative for accurately identifying a document with frequently occurring terms, only it does not retain the sequential order of the terms [Manning, Prabhakar, and Hinrich 2008].

4 Chapter 4: Experiment and Evaluation

4.1 Building the Inverted Index

In our implementation we provide several options for building the inverted index*:* the basic (without stemming and stop word removal), with stemming (stm), with stop word removal (rsw), and with stemming and stop word removal (rsw-stm). The goal of having different indexing options is to explore the effect of these indexing techniques on the *indexing process* in terms of time and space. Further, we investigate their effects on the retrieval process.

Figure 4-1 shows the running time for indexing 100 documents with approximately 91,900 strings. The basic method (without stemming and stop word removal) was the fastest one; it took 2.5 minutes to build the inverted index for the data sets, while the rsw-stm took 3.6 minutes for the same data sets.

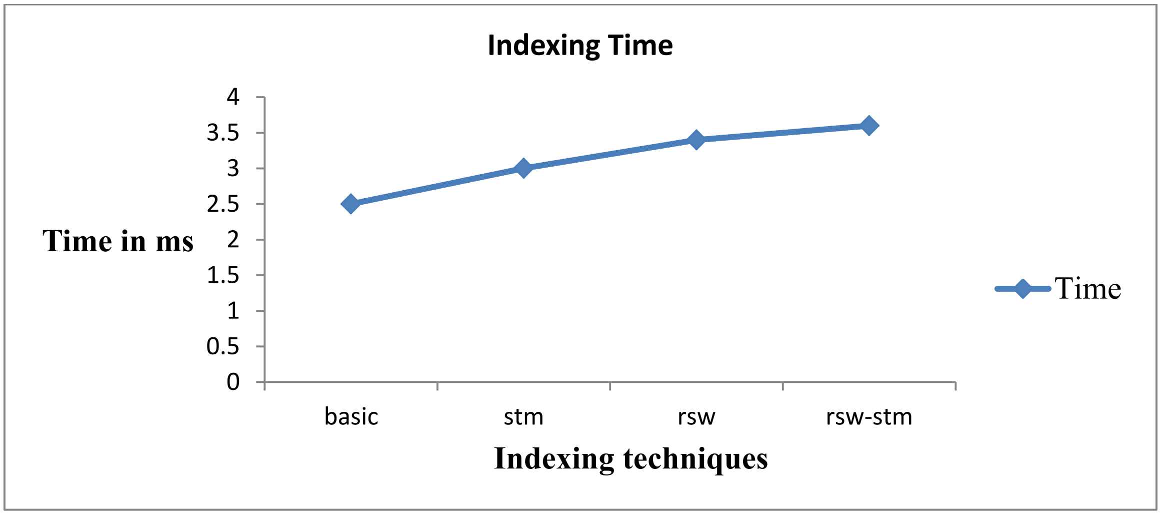


Figure 4-1 Time for indexing 100 documents with different indexing techniques

.

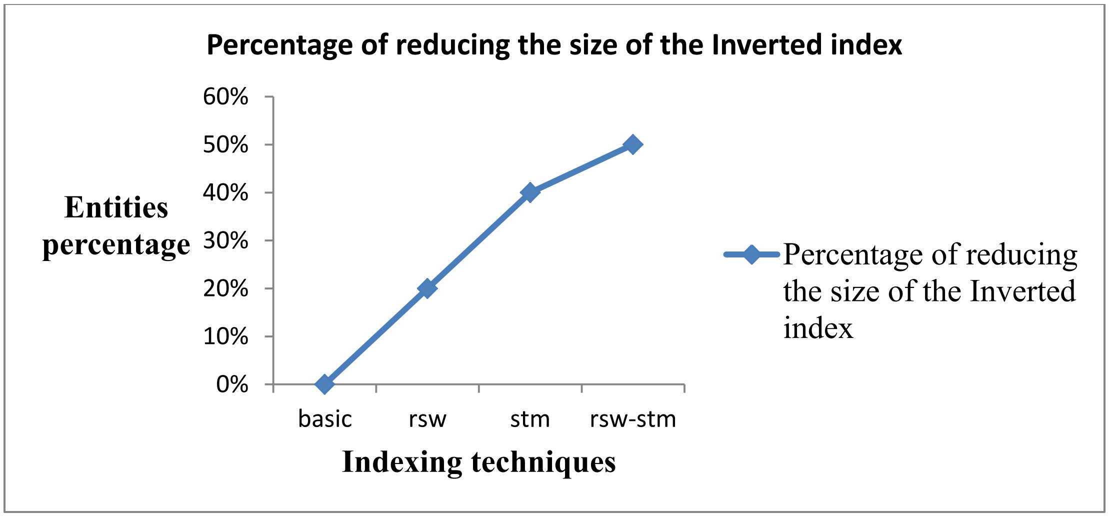


Figure 4-2 The percentage of reduction in the number of entries for the inverted index for 100 documents using different indexing techniques.

The experiment shows that the stemming technique reduces the size of the inverted index by approximately 40%. The stop word removal technique reduces the size by 20 %. Using both techniques reduces the size to less than half of the original number of entities.

4.2 Evaluation methodology

In this study we use a traditional precision /recall method of evaluation. Precision is the number of relevant documents retrieved divided by the total number of retrieved documents. Recall is the number of relevant documents retrieved divided by the total number relevant documents in the database [*M.Greenwood, 2001]*.

$$Precision = \frac{number\ of\ retrived\ that\ are\ relevent}{tota\ l\ number\ of\ retrived\ documents} \ldots\ldots\ldots\ldots\ldots \text{(4-1)}$$

$$Recall = \frac{number\ of\ retrived\ that\ are\ relevent}{total\ number\ relevent\ in\ the\ collection} \ldots\ldots\ldots\ldots\ldots\ldots \text{(4-2)}$$

In this method, we use the relevance and non-relevance notation to evaluate the performance of the system [M.Greenwood, 2001*]*. The relevance notation plays a central role in the evaluation of the IR system [Teufel, Simone, 2007]. IR system evaluation is called “laboratory style” in most modern IR systems. This style of performance is easy to control. It consists of three main components: dataset, relevance decision, and queries [Teufel, Simone, 2007]. Usually, the document is considered relevant to the query if it contains more occurrences of the user's query terms, but in reality the document should be considered relevant if it satisfies a user's information needs.

In the evaluation of our experiment we used free text query (query with two words or more). One word query makes the output ambiguous, especially with a morphologically complex language like Arabic. We prefer free text because it helps eliminate some of the unexpected results that are not related to the query search. Free text query also makes it easier for the system to determine the user's information needs. We evaluated the system with different weighting schemes and different indexing options.

4.3 Results

In the experiment we use nine queries that have a number of relevant documents in our data collections. We calculate the percentage of recall and precision for each output. In the next sections we list the results as tables for each approach.

4.3.1 **Precision – Recall Percentage Tables**

- Term-count (TF) (Term Frequency) method (without stop word removal and stemming).

Table 4-1 Shows precision and recall percentage for TF weighting scheme (without stop word removal / stemming)

Query	Precision	Recall
البيئة الصحية	50%	20%
الفلسفة الواقعية	100%	20%
الفطريات المرضية	100%	30%
الشعر الحر	60%	40%
التلوث البيئي	70%	40%
المسرح القومي	100%	50%
سكان الاهوار	70%	100%
علم النفس	20%	50%
مسجد الرسول	70%	100%

- Term – count (TF) method with (stop word removal /stemming) indexing.

Table 4-2 Shows precision and recall percentage for TF weighting scheme (with stop word removal/stemming).

Query	Precision	Recall
البيئة الصحية	70%	90%
الفلسفة الواقعية	30%	100%
الفطريات المرضية	100%	100%
الشعر الحر	50%	90%
التلوث البيئي	80%	70%
المسرح القومي	80%	60%
سكان الاهوار	50%	100%
علم النفس	40%	80%
مسجد الرسول	100%	100%

- (TF-IDF-basic) method without stemming /stop word removal.

Table 4-3 Shows precision and recall percentage for TF-IDF weighting scheme (without stop word removal/stemming).

Query	Precision	Recall
البيئة الصحية	70%	50%
الفلسفة الواقعية	100%	20%
الفطريات المرضية	50%	30%
الشعر الحر	60%	40%
التلوث البيئي	100%	50%
المسرح القومي	100%	50%
سكان الاهوار	50%	50%
علم النفس	30%	70%
مسجد الرسول	70%	100%

- TF-IDF method with stemming /stop word removal indexing**.**

  Table 4-4 Shows precision and recall percentage for TF-IDF weighting scheme (with stemming/stop word indexing).

Query	Precision	Recall
البيئة الصحية	70%	60%
الفلسفة الواقعية	30%	100%
الفطريات المرضية	100%	100%
الشعر الحر	50%	90%
التلوث البيئي	50%	70%
المسرح القومي	80%	90%
سكان الاهوار	50%	100%
علم النفس	10%	100%
مسجد الرسول	100%	100%

- TF-IDF based on normalization method without stemming /stops word removal indexing.

  Table 4-5 Shows precision and recall percentage for TF-IDF based on normalization weighting scheme (without stemming/stop word indexing)

Query	Precision	Recall
البيئة الصحية	50%	30%
الفلسفة الواقعية	100%	20%
الفطريات المرضية	60%	30%
الشعر الحر	60%	40%
التلوث البيئي	60%	40%
المسرح القومي	100%	80%
سكان الاهوار	100%	50%
علم النفس	40%	80%
مسجد الرسول	70%	100%

- TF-IDF based on normalization method with stemming /stops word removal indexing.

Table 4-6 Shows precision and recall percentage for TF-IDF based on normalization weighting scheme (with stemming/stop word indexing).

Query	Precision	Recall
البيئة الصحية	70%	60%
الفلسفة الواقعية	100%	100%
الفطريات المرضية	100%	100%
الشعر الحر	50%	100%
التلوث البيئي	100%	70%
المسرح القومي	80%	90%
سكان الاهوار	50%	50%
علم النفس	20%	100%
مسجد الرسول	100%	100%

4.3.2 The Effect of Stemming Vs. Non- Stemming

Figure 4-3 shows the number of retrieved documents with stemming and without stemming for the same queries. The stemming technique significantly increases the number of retrieved document. At the same time, the number of irrelevant documents also increases, since Arabic words that are derived from the same root often hold different meanings. For example, the word “الجامعة” “aljamcka” (“university”) stems from the root “جمع” (“add”). Several other words are likewise derived from this root. Therefore, when a user tries to search for the term “aljamcka,” a long list of documents is returned. Table 4-7 shows a sample of the Arabic words that stem from the root "جمع". When the recall increases, the ambiguity of the output increases as well, which leads to unexpected and often irrelevant results for the user. This is the only negative aspect of using the stemming technique; otherwise, the experiment proves that stemming improves the performance of the Arabic IR system. We observed, however, that the *ISRI* stemming algorithm is not always accurate. There are some cases in which the algorithm fails to find the valid Arabic root. In particular, the *ISRI* tends to suggest incorrect roots for borrowed words.

Table 4-7 shows a number of Arabic words that have different denotations, even though they stem from the same root, which is "جمع" ("add").

The Arabic word	Meaning (in English)	Root (in Arabic)	Meaning (in English)
جامعة	university	جمع	add(v)
مجموعة	set	جمع	add(v)
مجمع	composed	جمع	add(v)
تجمع	be accumulated	جمع	add(v)
اجتماع	meeting	جمع	add(v)
اجتماعي	socially	جمع	add(v)
اجماع	group	جمع	add(v)
جماعة	Team	جمع	add(v)
تجميعي	assembly	جمع	add(v)
جامع	mosque	جمع	add(v)
اجمع	agreed	جمع	add(v)
جمع	add	جمع	add(v)
جمعة	Friday	جمع	add(v)
جمعية	institution	جمع	add(v)

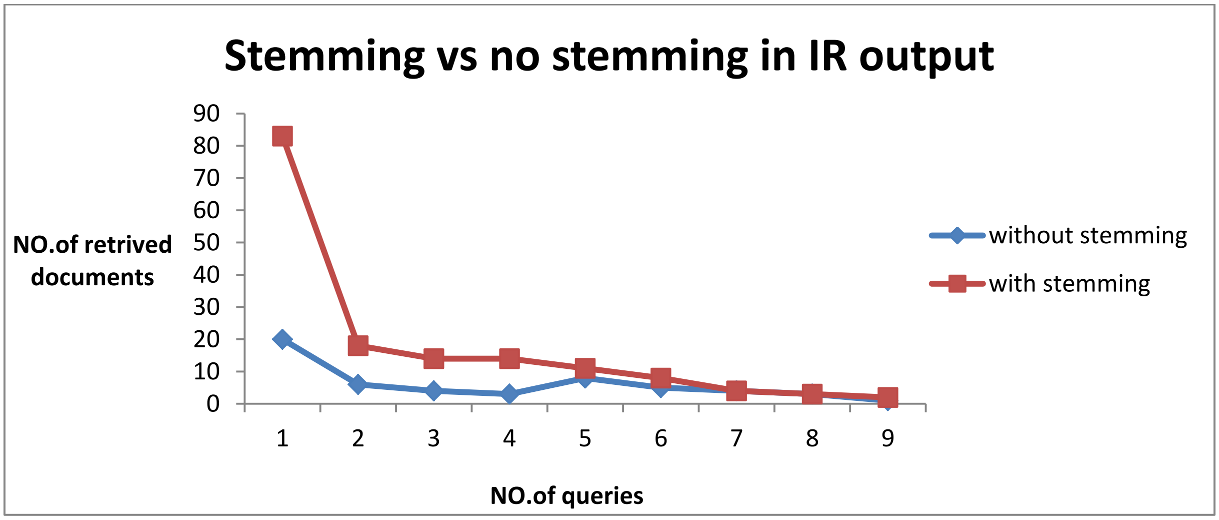


Figure 4-3 The number of retrieved documents for the same queries with stemming and without stemming.

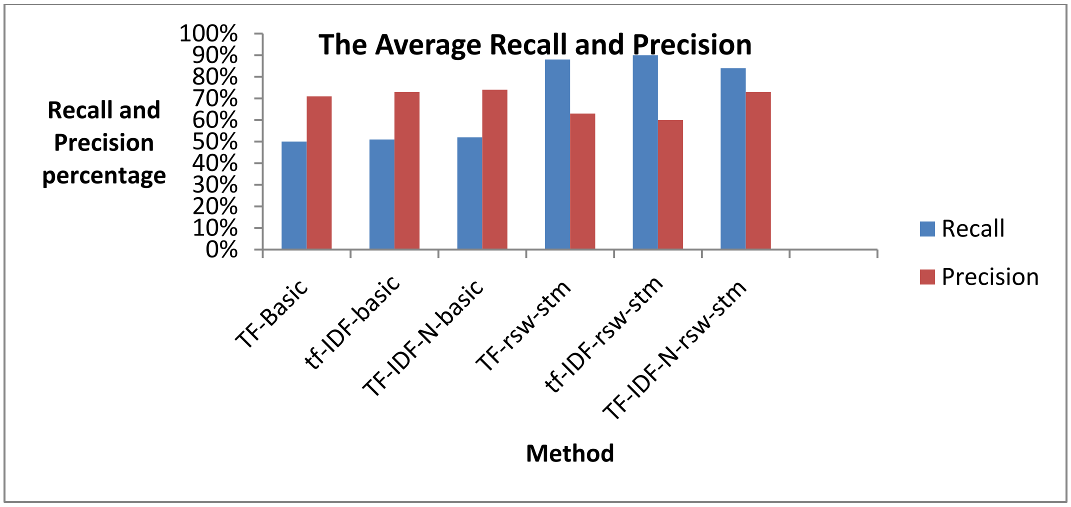


Figure 4-4 The average recall and precision of the different ranking schemes with a combination of different indexing techniques (basic, stemming, stop word removal).

The above figure shows that the three (TF-basic, TF_IDF-basic, TF_IDF_basic) weighting schemes have a high precision, while the average recall is low for the three methods. On the other hand, the figure shows an increase in the recall for the TF-rsw-stm, TF-IDF_rsw-stm, and TF_IDF-N-rsw-stm schemes. The TF-IDF-N-rsw-stm method seems to produce the optimal result, yielding both a high recall and a high precision. The TF-IDF-N method outperforms the other methods with different indexing options.

5 Chapter Five: The Default Search in Open Journal System

5.1 Public Knowledge Project (PKP)

The Public Knowledge Project (PKP) is a non-profit organization devoted to investigating how technology can help develop professional and public knowledge. PKP was founded by Dr. Jhon Willinsky in 1998 through his work in education and publishing in the College of Education at the University of British Columbia. PKP attempts to make scholarly knowledge available to the public through online access. The aim is to connect scholars in different fields with public librarians in order to enhance the quality of education using technology and online environments [Smecher, Alec, 2008].

5.2 Open Journal Systems (OJS)

Open Journal Systems (OJS) is one of PKP's open access management systems developed at the University of British Columbia. The main goal of OJS is to make academic knowledge available to the public through online environments. OJS provides support throughout the entire editorial process, from uploading submissions, to editing, to peer reviewing, to publishing. This comprehensive tool saves precious time and energy for users [Smecher, Alec, 2008]. OJS is a multilingual system and is used to publish journals in several languages. Originally, OJS only supported English, French, Spanish and Portuguese, but it now supports more than 33 languages according to the PKP website. OJS translations are provided by the community of developers. Because OJS designers try to separate the translation process from the source code, all text in OJS is extracted and stored in XML files called *locale files*. OJS was built using model-view - controller software architecture. This model gives OJS more security and flexibility at the same time. Also, the design of OJS provides a rich online reading environment by supporting tools that improve active reading. More than 5,000 journals have used OJS since it launched in 2002 [Smecher, Alec, 2008].

5.3 Open Journal Systems as Repository

Open Journal Systems (OJS) is an open source publishing and management system. The goal of OJS and other PKP projects is to enhance the quality of public education and knowledge through online environments. Figure 5-1 shows the main OJS webpage, and figure 5-2 shows our OJS journal.

OJS can be downloaded for free from the PKP website (All OJS downloads can be found at http://pkp.sfu.ca/ojs_download, and detailed installation steps can be found at http://pkp.sfu.ca/ojs/docs/userguide/2.3.1/systemAdministrationInstallProcess.html. OJS provides the facilities to create multiple journals with one installation.

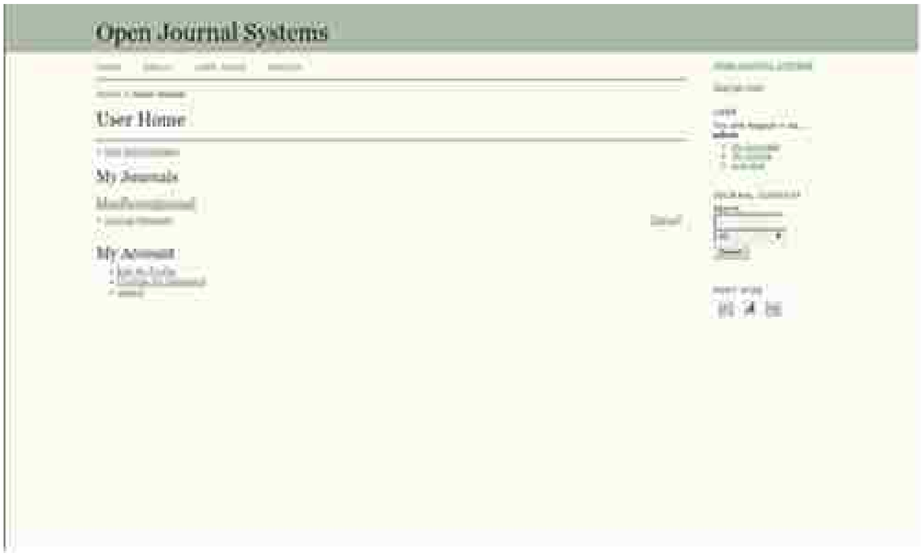


Figure 5-1 The home page of Open Journal System

Figure 5-2 The home page of our journal

5.3.1 **OJS Submission and Editorial Process**

OJS is designed to streamline the editorial process for journals, from the uploading of submissions to the publishing stage. OJS can be used by journal managers, editors, authors, reviewers, copy editors, section editors, proofreaders, and layout editors. Each user can be assigned multiple roles, and each role carries a corresponding set of administrative tasks and permissions. When a user is assigned the role of journal manager, for instance, he or she has the authority to assign various roles to other users. The editor, on the other hand, is in charge of accepting or rejecting a submission.

A user with the role of author can submit articles to the journal. The submission process can be done in a number of steps. Firstly, the user has to select one of the journal sections for his/her article based on the article type (book review, etc.). Before submitting an article, the author must agree to the journal's policies for publishing in journals. In the next step the author provides personal information and information about the article, including title, abstract, and indexing terms. OJS also allows the author to upload any supplementary files to his/her article such as data set, pictures, graphs, etc. The final step is confirming submission. Additionally, OJS supports tracking the status of one’s submission. Figure 5-3 shows a number of our published article in our journal.

Figure 5-3 A number of published articles in our journal

In The next sections we will introduce the Default search function in Open Journal system and how it works with the Arabic language texts.

5.4 The Default Search of Open Journal System

The default search engine of OJS uses an inverted Index stored in a MYSQL relation database with the following tables:

article_search_keyword_list<keyword_id, keyword_text>

article_search_object_keywords<object_id, keyword_id, pos

article_search_objects<object_id, article_id, type, assoc_id>

Simply put, the algorithm starts with the tokenization of all text by using "whitespace" as a separator and by removing punctuation and stop words. It then stores the keywords in the MYSQL search tables. OJS provides full-text indexing for different file formats including HTML, PDF, PS, and MS Word. OJS uses external tools to convert PDF, PostScript, and Microsoft Word files into text files. OJS uses external tools that are available by default in most *UNIX* distributions to extract text from the files. The system uses the `pdftotext` tool to extract text from pdf files, `postotext` to extract text from PostScript files, and the HTML tokenizer to extract text from HTML files. These tools are configured in the OJS configuration file (Config.inc.php). Figure 5-4 shows the search configuration part in the Config.inc.php file. The OJS search provides several search categories, including author, abstract, discipline, index-term, subject, supplementary file, title, and type. By default, the search retrieval system in OJS retrieves all documents using the "AND" operation, but the user can choose to change this to "OR" or "NOT." In addition, "*" can be used as a wildcard query, and quotes can be used with a phrase query.

OJS is a multilingual system. Therefore, OJS features a general search function that supports multiple languages and provides a basic search function. As a general search function, the OJS search function does not support the morphological characteristics of Arabic, because it is not programmed to support stemming. Furthermore, the OJS search function does not have a means of normalizing Arabic text. The absence of these two techniques affects the performance of OJS's native search engine. As a result, some Arabic queries in OJS yield few, if any, matching documents. The OJS search function is thus unable to find all relevant documents. On the other hand, the OJS search function treats all returned results as if they were equally relevant to the user's query, because it does not weight or rank the retrieved documents by their similarity to the user's criteria.

```
; Minimum indexed word length
min_word_length = 3

; The maximum number of search results fetched per keyword. These results
; are fetched and merged to provide results for searches with several keywords.
results_per_keyword = 500
; The number of hours for which keyword search results are cached.
result_cache_hours = 1

; Paths to helper programs for indexing non-text files.
; Programs are assumed to output the converted text to stdout, and "%s" is
; replaced by the file argument.
; Note that using full paths to the binaries is recommended.
; Uncomment applicable lines to enable (at most one per file type).
; Additional "index[MIME_TYPE]" lines can be added for any mime type to be
; indexed.
; PDF
; index[application/pdf] = "/usr/bin/pstotext -enc UTF-8 -nopgbrk %s - | /usr/bin/tr '[:cn
trl:]' ' '"
; index[application/pdf] = "/usr/bin/pdftotext -enc UTF-8 -nopgbrk %s - | /usr/bin/tr '[:c
ntrl:]' ' '"

; PostScript
; index[application/postscript] = "/usr/bin/pstotext -enc UTF-8 -nopgbrk %s - | /usr/bin/t
r '[:cntrl:]' ' '"
; index[application/postscript] = "/usr/bin/ps2ascii %s | /usr/bin/tr '[:cntrl:]' ' '"
; Microsoft Word
; index[application/msword] = "/usr/bin/antiword %s"
; index[application/msword] = "/usr/bin/catdoc %s"
```

Figure (5-4) Configuration setting for OJS search in (Config.inc.php)

5.5 Experiment

Researchers over the years have tried to understand how efficient general search engines are at handling different languages. Mukdad, for instance, discusses whether or not a general search engine like Alti Vista can effectively handle a language with a complex morphology like Arabic.

The goal of our evaluation is to investigate how a multilingual search engine performs when both the data collection and the query are in Arabic. The OJS search algorithm is a basic algorithm with no ranking. The precision /recall evaluation method works perfectly with such algorithms.

Figure 5-2 shows that for some queries there is no matching result. Therefore, the precision and recall are zero, and even though there are a number of documents that are related to the user's queries in the corpus, the search function fails to locate any matching documents.

Table 5-1 precision and recall percentage for OJS default search.

Query	Recall	Precision
الشعر الحر	0%	0%
بكتريا عنقودية	20%	100%
الفطريات المرضية	0%	0%
مسجد الرسول	0%	0%
المسرح القومي	10%	100%
البيئة الصحية	10%	100%
سكان الاهوار	10%	100%
التلوث البيئي	30%	100%
علم النفس	50%	100%

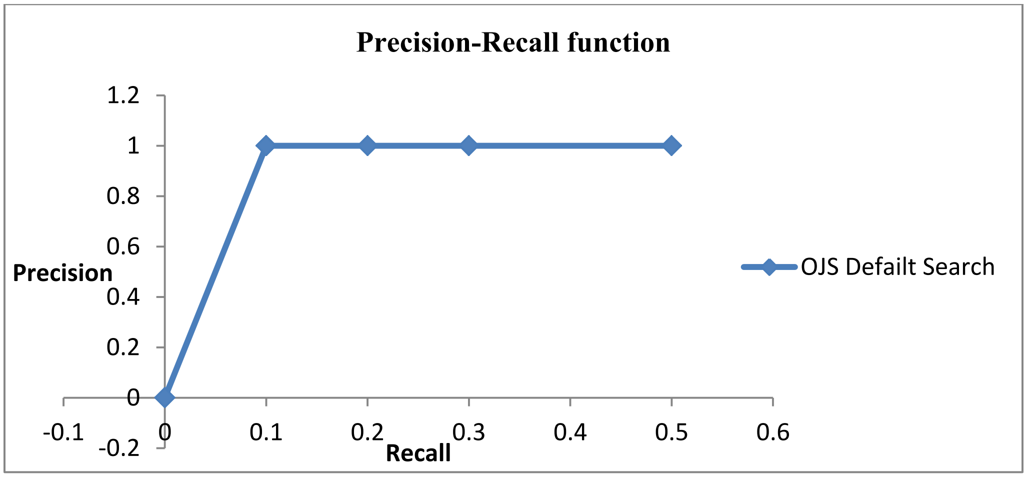


Figure 5-5 The precision/recall function for the default search of OJS

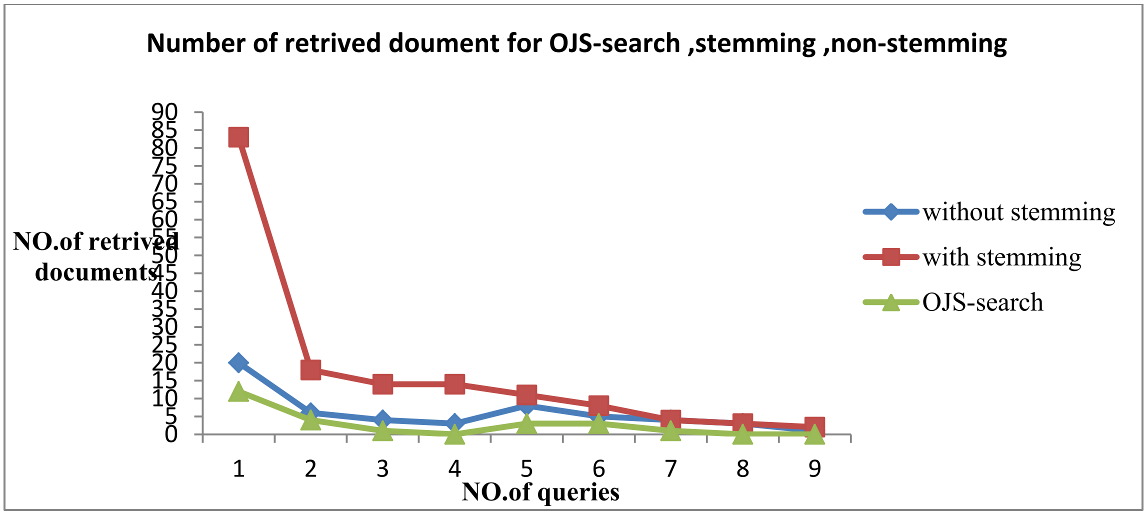


Figure 5-6 The number of retrieved documents for the same queries with stemming, without stemming, and with the OJS default search.

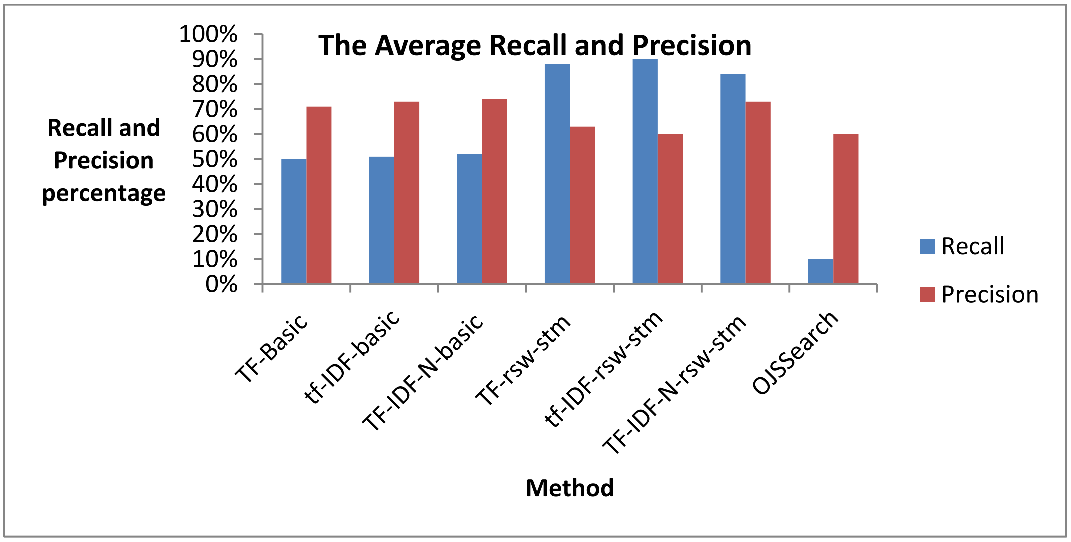


Figure 5-7 Average recall and precision for OJS- search and our three schemes

Figure 5-7 shows low recall of the default OJS search function. The average recall is only 10%. Our proposed IR system has a comparatively high recall, between 40% and 60% without stemming and between 70% and 90% with stemming.

6 Chapter Six: Conclusion and Future Works

6.1 Conclusion and Summary of the Experimental Results

- Arabic requires tools that support the encoding of its unique character sets, its writing script and the direction of the text. Using UTF_8 or another Arabic encoding does not ensure that a valid Arabic script is displayed, specifically with regard to the direction of the text (from right to left for text and left to right for numbers). If Arabic is not supported by system software and an application that can display a valid Arabic script, then the scripts will be displayed in an invalid format.
- Support of Arabic text requires such techniques as normalization, which eliminates inconsistencies in Modern Standard Arabic. By doing so, normalization maximizes the number of matching documents, thus improving recall and precision.
- The stemming technique significantly increases the number of documents that match a user's query. Stemming algorithms analyze words and remove elements like prefixes, prepositions, pronouns, and the definite article “ال”“al,” all of which are connected to other words. However, stemming produces some unexpected results. Due to complex Arabic morphological rules, many words with different denotations can be derived from the same root. As a result, the ISRI stemming algorithm is not fully accurate, because it fails to find valid roots for some Arabic words. One possible reason might be a lack of valid roots in the dictionary that the algorithm uses. In addition, the ISRI stemming algorithm tries in vain to find Arabic roots for words adopted from other languages.
- Lexical processing helps reduce the size of the inverted index. Removing stop words can reduce the size of the inverted index by 20%, and stemming reduces the number of inverted index entities by approximately 40%.
- The vector space model successfully supports free text queries by accurately identifying a user's query terms and their occurrences in a document collection. However, the vector space model does not support the sequential order of query terms, because it represents a document as a vector of terms regardless of the order in which the terms appear within a document.

- The vector space model weighting schemes have different ranking preferences. The term frequency (TF) weighting schemes always assign the highest score to a document with the highest occurrence of a queried term, but this favors long documents over short ones. TF_IDF weighting gives good results for queries that include rare terms such as scientific terms. TF_IDF is based on normalization and outperforms the other weighting schemes
- The evaluation of the default search function of OJS proves that without normalization and stemming the number of retrieved documents is minimized. OJS's default search function retrieves matching results for some queries, but for others the output is zero results, because the default search function of OJS is a general search engine that does not acclimate to Arabic character sets and scripts.

6.2 Future works

IR technology is an ever-growing area of research, especially with regard to Arabic text and the unique challenges presented by the complex characteristics of the language. We can extend this thesis to the following inquiries:

- This thesis focuses on the evaluation of IR systems using the traditional precision/recall method; however, there are several strategies that could be used to evaluate the system. In particular, multi-grade strategies could be effective at ranking IR systems. We plan to evaluate the output of our system using different strategies such as mean average precision.
- Clustering techniques can be used to reduce the ambiguity of retrieved documents by grouping similar documents into one cluster. We want to explore how clustering could improve the output of our IR system.
- In this work we rely on the inverted index technique, but there are other indexing techniques and architectures that could be used to expedite document retrieval and reduce the size of the index. For example, compression techniques use customized architectures to store the inverted index more efficiently. The distributed indexing technique can be used to store the inverted index for a large document collection,

increasing the effectiveness of the inverted index. We plan to explore techniques that could lead to faster indexing and smaller storage sizes.

- Stemming techniques are very important for morphological languages like Arabic. Several Arabic stemming algorithms have been proposed. We plan to explore how the designs of these algorithms have been adopted to build the Arabic stemmer.

The importance of IR has increased tremendously since the creation of the World Web Wide. The Internet adds totally new dimensions to the IR process that go far beyond text-based searches. It would be interesting to see how these experiments with more traditional IR systems may be adapted to different kinds of data.

References

[1] Albalooshi, Noora, Nader Mohamed, and Jameela Al-Jaroodi. "The challenges of Arabic language use on the Internet." Internet Technology and Secured Transactions (ICITST), 2011 International Conference for. IEEE, 2011.

[2] Al-Taani, Ahmad T., Ahmed S. Ghorab, and Hazem M. Al-Najjar. "AN ARABIC-ENGLISH INDEXING SYSTEM USING INVERTED INDEX ALGORITHM."

[3] Al-Maimani, Maqbool R., A. A. Naamany, and Ahmed Zaki Abu Bakar. "Arabic information retrieval: techniques, tools and challenges." GCC Conference and Exhibition (GCC), 2011 IEEE. IEEE, 2011.

[4] Atwan, Jaffar, Masnizah Mohd, and Ghassan Kanaan. "Enhanced Arabic Information Retrieval: Light Stemming and Stop Words." Soft Computing Applications and Intelligent Systems." Springer Berlin Heidelberg, 2013. 219-228.

[5] Baldi, Pierre, Paolo Frasconi, and Padhraic Smyth. "Modeling the Internet and the Web." Probabilistic methods and algorithms (2003).

[6] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.

[7] Fox,Christopher ," A Stop List for General Text". SIGIR Forum, Vol. 24, No. 1-2, 1990, pp.19-35.

[8] Chen, Aitao, and Fredric C. Gey. "Building an Arabic Stemmer for Information Retrieval." TREC. Vol. 2002. 2002.

[9] Faloutsos, Christos, and Douglas W. Oard. "A survey of information retrieval and filtering methods." (1998).

[10] D.L. Lee, H. Chuang, and K. Seamons. Document ranking and the vector space model. IEEE Transactions on Software, 14(2): 1997.

[11] Hiemstra, Djoerd. "Information retrieval models." Information Retrieval: searching in the 21st Century (2009): 1-19.

[12] El-Khair, Ibrahim Abu. "Effects of stop words elimination for Arabic information retrieval: a comparative study." International Journal of Computing & Information Sciences 4.3 (2006): 119-133.

[13] Hariguna, Taqwa, and Fandy Setyo Utomo. "Implementation of Information Retrieval Indonesian Text Documents Using the Vector Space Model." ICSIIT 2012 (2012): 145.

[14] Liddy, Elizabeth D. "Document retrieval, automatic." (2005).

[15] Larkey, Leah S., Lisa Ballesteros, and Margaret E. Connell. "Improving stemming for Arabic information retrieval: light stemming and co-occurrence analysis." Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2002.

[16] M.Greenwood 'IMPLEMENTING A VECTOR SPACE DOCUMENT RETRIEVAL SYSTEM',Dept. of Computer Science, University of Sheffield Regents Court, 211 Portobello St, Sheffield,2001 .

[17] Moukdad, Haidar, and Andrew Large. "Information retrieval from full-text Arabic databases: can search engines designed for English do the job?." Libri 51.2 (2001): 63-74.

[18] Moukdad, H., "Lost In Cyberspace: How Do Search Engines Handle Arabic Queries?" In: Access to Information: Technologies, Skills, and Socio-Political Context. University of Manitoba, Winnipeg, Manitoba. June 3 - 5, 2004.

[19] MUNTEANU, Dan. "VECTOR SPACE MODEL FOR DOCUMENT REPRESENTATION IN INFORMATION RETRIEVAL."

[20] Smecher, Alec. "OJS Technical Reference." (2008).

[21] Ramos, Juan. "Using tf-idf to determine word relevance in document queries."Proceedings of the First Instructional Conference on Machine Learning. 2003.

[22] Raghavan, Vijay V., and SK Michael Wong. "A critical analysis of vector space model for information retrieval." Journal of the American Society for information Science 37.5 (1986): 279-287.

[23] Singhal, Amit. "Modern information retrieval: A brief overview." IEEE Data Eng. Bull. 24.4 (2001)

[24] Salton, Gerard, and Michael J. McGill. "Introduction to modern information retrieval." (1983).

[25] Singh, Jitendra Nath, and Sanjay Kumar Dwivedi. "Analysis of Vector Space Model in Information Retrieval." IJCA Proceedings on National Conference on Communication Technologies & its impact on Next Generation Computing 2012. No. 2. Foundation of Computer Science (FCS), 2012.

[26] Sparck Jones, Karen, Steve Walker, and Stephen E. Robertson. "A probabilistic model of information retrieval: development and comparative experiments: Part 1." Information Processing & Management 36.6 (2000): 779-808.

[27] Taghva, Kazem, Rania Elkhoury, and Jeffrey S. Coombs. "Arabic Stemming Without A Root Dictionary." ITCC (1). 2005.

[28] Teufel, Simone. "An overview of evaluation methods in TREC ad hoc information retrieval and TREC question answering." Evaluation of text and speech systems. Springer Netherlands, 2007. 163-186.

[29] Zobel, Justin, Alistair Moffat, and Ron Sacks-Davis. "An efficient indexing technique for full-text database systems." PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES. INSTITUTE OF ELECTRICAL & ELECTRONICS ENGINEERS (IEEE), 1992.

[30] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze.Introduction to information retrieval. Vol. 1. Cambridge: Cambridge university press, 2008.

[31] Bird, Steven, Ewan Klein, and Edward Loper. Natural language processing with Python. " O'Reilly Media, Inc.", 2009.

Appendix

```
SEARCH QUERIES FOR AN INFORMATION RETRIVAL SYSTEM FOR
ARABIC LANGUAGE
#the project is divided in two parts with two files
•    Indexing Processprogram :
#indexing processing for the corpus and building the
inverted index file
#Author: Zaianb Albujasim.
#Date: 5/5/2014.
#Description:
#input: A list of Arabic document in text file format with
encoding UTF_8.
#the output is an inverted index text file.
"""
Methods: the program starts by Indexing processing steps
which  includes
1- Normalization (clean text from Arabic Vowel signs. for
example: input: زَيْـنَب  ,output: زيـنب )
2- Tokenization (split text into tokens using white space
as a separator)
3- Stop word removal (remove the common words which don’t
consider relevant to user's query
4- Stemming (using ISRIStemmer)(find roots for Arabic
words)
5- Inverted index: is a dictionary includes all distinct
terms in the corpus with their docid and a number of
occurrences)
Options:
 We have implements 4 ways of building the inverted index
('basic: without stemming/stop word removal)')
```

('-rsw: (stop word removal only)')

('-stm: without (stemming only)')

('-rsw-stm: with(stemming/stop word removal')

Limitation:

1-In the inverted index file we store only the number of term occurrence within a document with docid ,We don’t have a mean to store term positions within a document

2- The directory of document files should be in a specific location and the inverted file also """

CODE:

```
import glob
from collections import defaultdict
from functools import reduce
from nltk.stem.isri import ISRIStemmer
import re
import math
import sys
import pickle
import time
import unicodedata
posting=defaultdict(dict)
stopword=[]
Q_terms=[]
totalterms=[]
query=''
files_dict={}
start_time=time.clock()
filename=''
def normalization(text):
    """Cleaning text by removing vocalization marks"""
    Return ''.join ([word for word in 
unicodedata.normalize('NFD', text) \
        if unicodedata.category(word) != 'Mn'])

#-------------------------------------------
def stem(word):
   # returns the root of a word using ISRI stemmer.

   st = ISRIStemmer()
   stem_w=st.stem(word)
   return (stem_w)

def tokenization(text):
    cleanterms=[]
    punctuation='-_.,!#$%^&*();:\n\t\\"?{}[]<>+=|/؟.'
    text=normalization(text)
    # we have used Regular expressions module to process 
text
    text=re.sub(r'[a-zA-Z0-9]','',text) # remove letters 
and numbers
    text = re.sub(r'[^\w\s]','',text)
    text= re.sub(" \d+", "", text)
    #x=re.sub (r'[-_.,!#$%^&*();:\n\t\\\"?{}[]<>]','',term)
    Text = re.sub("[ا" ,"[إأآا]", text) # normalize "أ" 
forms
    text = re.sub("ي" ,"ى", text) # normalize "ي"  forms
    text = re.sub("ة" ,"ه", text) # normalize "ة" forms
    terms = text.split()  # split text into a lists of 
strings
    for term in terms:
        x=term.strip(punctuation)
        cleanterms.append(x)

  return cleanterms

def stopwordlist():

    """upload stop word from text file to a list"""

    global stopword

    stop_wordfile=open('stop.txt','r',encoding='utf_8')
    for line in stop_wordfile.readlines():
        for sw in line.split():
            sw=sw.strip('\n')
            stopword.append(sw)


    stop_wordfile.close()
    return stopword

#-------------------------------------
def removestopword(keywordsindex):
    global stopword
    i=1
    """ Remove stop word """
    for word in keywordsindex:
        if len(word )<=2 or word in stopword:

           #print(word)
           i=i+1
           keywordsindex.remove(word)

    #print ('i=',i)
    return keywordsindex
     #----------------------------------

def inial_indexing():
    global filename
    i=0
    print('           Indexing Process::    Create the
inverted index')
    print('\n')
    filename=input('enter a name for the inverted index ::
')

    print('select  the indexing  method:     ')
    print('basic: without stemming/stopwordremoval)')
    print('-rsw:  (stop word removal only)')
    print('-stm: without (stemming only)')
    print('-rsw-stm: with(stemming/stop word removal' )
    print('\n')
    method=input("select -rsw,-stm ,-rsw-stm or
none(without):")
    print('\n')
    print('         creating index......................')

    print('\n')
list_of_files=glob.glob('vsm-dir/*.txt')
    for file_name in list_of_files:
        file1=open(file_name ,'r',encoding='utf_8')

        i=i+1
        docID=i
        files_dict[docID]=file_name
        line =file1.read()
        index =tokenization(line)

        if method=="-rsw":
           termsw=removestopword(terms)
           #termsw=terms
           keywords=(termsw)
           index =keywords

        elif method=="-stm":
          stem_terms=[stem(x) for x in terms ]
          keywords=(stem_terms)
          index =keywords

        elif method=="-rsw-stm":
          termsw=removestopword(terms)
          stem_terms=list([stem(x) for x in termsw ])
          keywords=(stem_terms)
          index =keywords

        else:
            keywords=index
        totalterms.extend(terms)

         for term in totalterms:
           posting[term][docID]=index.count(term)


#-------------------------------------------------------

def writing_filenames():
    """store document information in a binary file"""

    file3=open('info.txt','wb')
    pickle.dump(files_dict,file3)
    file3.close()

#-------------------------------------------------------
def writing_inverted_index():
    global filename

    file2=open(filename,'w',encoding='utf_8')

    for term in posting.keys():
        post_file=[]

        for post in posting[term].items():

            docid=str(post[0])
            tf=str(post[1])
            x=docid+':'+tf
            post_file.append(x)
        xxx=(term,'#',';'.join(post_file))
        file2.writelines(xxx)
        file2.write('\n')

    file2.close()

def main():
    global filename
    stopwordlist()
    print(time.clock()- start_time,"seconds")
    inial_indexing()
    print(time.clock()- start_time,"seconds")
    writing_filenames()
    print(time.clock()- start_time,"seconds")
    print('The inverted index has been created with
name:',filename)


if __name__ == "__main__":
    main()
```

- Query and Document Retrieval (Second file)

```
#indexing processing for the corpus and building the
inverted index file
#Author: Zaianb Albujasim.
#Date: 5/5/2014.
#Description:
#input: An inverted index file and a user's query
# A list of document relevant document
"""
Methods:  firstly: the program starts by uploading the
inverted index.
Secondly: apply indexing process to a user's query
1- Tokenization (split text into tokens using white space
as a separator)
2- Stop word removal (remove the common words which don’t
consider relevant to user's query
3- Stemming (using ISRIStemmer) (find roots for Arabic
words)
4- Inverted index: is a dictionary includes all distinct
terms in the corpus with their docid and a number of
occurrences)
```

```
3- User should select one of the weighting methods to rank
the return document by their similarity from highest to
lowest one.
Options:
 We have implements 3 weighting method to rank the out put
 ("Choose one of the ranking methods to ranking the
retrieved result"
 "1:TF(term frequency)"
  2:TF_IDF"
  3:TF_IDF based on normalization"

Limitation:
1-   The location of the inverted is hardcoded.
2-   The program does not provide a way to display the text
of relevant file. It only provides a list of document that
contain the user’s query terms.
"""

files_dict={}
posting=defaultdict(dict)
index=defaultdict(dict)
stopword=[]
Q_terms=[]
query=''
ranking_method=0
def tokenization(text):
    cleanterms=[]
    punctuation='-_.,!#$%^&*();:\n\t\\"?{}[]<>+=|/؟.'
    text=normalization(text)
    # we have used Regular expressions module to process
text
    text=re.sub(r'[a-zA-Z0-9]','',text) # remove letters
and numbers
    text = re.sub(r'[^\w\s]','',text)
    text= re.sub(" \d+", "", text)
    #x=re.sub (r'[-_.,!#$%^&*();:\n\t\\\"?{}[]<>]','',term)
    Text = re.sub("[ا" ,"[إأٱآا", text) # normalize "أ"
forms
    text = re.sub("ي" ,"ى", text) # normalize "ي"  forms
    text = re.sub("ة" ,"ه", text) # normalize "ة" forms
    terms = text.split()  # split text into a lists of
strings
    for term in terms:
        x=term.strip(punctuation)
        cleanterms.append(x)
```

```
def stem(word):
   """the function function uses IRSI stemmer to stem 
word"""

   st = ISRIStemmer()
   stem_w=st.stem(word)
   return(stem_w)

#-----------------------------------------------------------
def stopwordlist():
    """this function for loading stop word list form stop 
word file and stored as a list"""

    global stopword

    stop_wordfile=open('stop.txt','r',encoding='utf_8')
    for line in stop_wordfile.readlines():
        for sw in line.split():
            sw=sw.strip('\n')
            stopword.append(sw)


    stop_wordfile.close()

    return stopword


#-------------------------------------------------------

def indexing_query(q):
     global stemming
     #split the terms of the query using defualt spliter 
(white spaces)
     q_terms=tokenization(q)
     print(q_terms)
     if(stemming =='2'):

         q_terms=list(stem(t) for t in q_terms)
     return q_termsdef readinginvertedindex():


#---------------------------------------
def reading_files_info():
```

```
# uploading document information such as docid, doc_title
    global n
    global files_dict
    file1=open('D:\Python33\info.txt','rb')
    files_dict=pickle.load(file1)
    n=len(files_dict)
    print('n=',n)
    file1.close()
#---------------------------------------------
def readinginverted_index():
   dictinteger={}
   global inverted_file
# reading the inverted index to dictionary

   file2=open('D:\Python33\ invertedindex-
basic.txt','r',encoding='utf_8')
   for line in file2:
          postlist=[]
          dictlist={}
          dictinteger={}
          line=line.rstrip()
          term, postlist = line.split('#')
#term|postinglist

          #print('term:',term,'postlist:',postlist)
          postlist=postlist.split(';')
#postings={'docId1:tf,docid2:tf}
          dictlist=dict([x.split(':')for x in postlist ])
#postings={'docId1', 'pos1,pos2'}
          #print(dictlist)
          for key, value in dictlist.items():
                key1=int(key)
                value1=int(value)
               #print(term,key,value)
                dictinteger[key1]=value1
          #print(dictinteger)
          index[term]=dictinteger


   return index
   file2.close()
#------------------------------------

def term_weight(term,docid):
    global posting
    """return the weight of a term within a document
    if the term is not found in the document return 0"""
    if docid in posting[term]
        tw=math.log10(posting[term][docid])
    else:
       tw=0
   return tw

#----------------------------------------
def term_frequency(term,docid):
    global posting
    """return the term frequency """
    if docid in posting[term]
        tf=posting[term][docid]
    else:
       tf=0
return tf
#------------------------------------------

def doc_frequency_per_term():

     """for each term in the index ,count the number. Of
documents that contain the term and store the value a
doc_frequency[term]."""
     global df
     for term in posting:
        lenght=len(posting[term])
        df[term]=lenght
def TF(query,docid):
    global norm
    global posting
    sum1=0.0
    for term in query:
        if term in posting:
           sum1+=term_weight(term,docid)

    Relevance=sum1
    return Relevance



#*******************************************method2********
*********************************
def TF_IDF(query,docid):
    global length
    global posting

        sum1=0.0
    for term in query:
        if term in posting:
         df1=df[term]
           sum1+=IDF(term)*term_weight(term,docid)

    Relevance =sum1
    return Relevance


#*************************************method3**************
***************************************
def TF_IDF_NORM(query,docid):

    global length
    global posting

    """Return the cosine similarity between query and
document id"""
    sum1=0
    for term in query:
        if term in posting:
            df1=df[term]
            sum1+=IDF(term)*term_weight(term,docid)

        Relevance =sum1
           Relevance =math.cos(Relevance/norm[docid])
            return Relevance


   #----------------------------------------

def IDF(term,df1):

    '''compute the inverse document frequency of term
    if term in posting:

        t_idf=math.log10(n/df1)
    else:
        t_idf=0
return t_idf
#----------------------------------------
def search_VSM(q_terms,postinglist):
    #global Q_terms
    global ranking_method
    global  rank_documens
    relevant={}

    """the user enter the query  in a textbox and search 
function return a list of document
    in decreasing order"""
    """find the document Id containing all query terms by 
intersect the posting list for
    each term and return a postinglist for all term in a 
query"""
    print(postinglist[term].keys() for term in q_terms)

    rel_IDs =reduce(end_[set(postinglist[term].keys()) for 
term in q_terms])
    print('The result in decreasing order::')
    print('ranking_method:',ranking_method)
    if not rel_IDs:
        print("Sorry,We couldn't find a matching document 
for your query")

    else:
        print(rel_IDs)

        if(ranking_method=='1'):
           for docid in rel_IDs:

              relevent[docid]=TF(q_terms,docid)
           rank_documens=sorted(rel.items(),key=lambda 
t:t[1],reverse=True
        elif (ranking_method=='2'):
           for docid in relevant_doc_IDs
              relevent[docid]=TF_IDF(q_terms,docid)
           rank_documens=sorted(rel.items(),key=lambda 
t:t[1],reverse=True)

        elif (ranking_method=='3'):
            for docid in relevant_doc_IDs
              relevent[docid]=TF_IDF_NORM(q_terms,docid)
            rank_documens=sorted(rel.items(),key=lambda 
t:t[1],reverse=True)
        print_result( rank_documens)
           def main():
```

```
    df= defaultdict(float)
    global posting
    global ranking_method
    print('enter the inverted index file')
    inverted_file=input('enter the inverted index file:')
    stemming=input('Select 1 or 2 for inverted index method
:1 basic  2: with indexing  : ')
    print("Choose one of the ranking methods to ranking the
retrived result")
    print("1:TF(term frequency)")
    print("2:TF_IDF")
    print("3:TF_IDF based on normalization")
    ranking_method=input("Select the number of the ranking
method:")
         # uploading the Inverted Index to Posting list
    posting=reading_inverted_index()

    #uploading the document collection information such as
the name and DOCID
    reading_files_info()

    while(1):
                text=input('what you would like to
search:')
        Q_terms=indexing_query(text)

        #print the number of documents in the collections
        print('the number of documents in the courpous:',n)
        doc_frequency_per_term()
        if(ranking_method=='3'):
            #call norm function to find the length for
documents in the courpus
            norm(files_dict,posting)
        search_VSM(Q_terms,posting)

    if __name__ == "__main__":

    main()
```

Vita

Name:	Zainab Majeed Albujasim
Place of Birth:	Iraq, Babylon
Education:	University of Babylon
	Iraq, Babylon
	B.Sc. in Computer Science (2003-2007)
	University Of Kentucky
	Lexington, Kentucky
	Master in Computer Science (2012-2014)
Professional:	University of Babylon
	Assistant Programmer (2007-2010)