ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
SCHOOL OF INFORMATION SCIENCE

DEVELOPMENT OF A STEMMER FOR AFARAF TEXT RETRIEVAL

A thesis proposed to School of Graduate Studies of Addis Ababa University in partial fulfillment of the Requirement for the Degree of Master of Science in Information Science

By
OSMAN TAHA OMER

ADDIS ABABA
November, 2015

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
SCHOOL OF INFORMATION SCIENCE

DEVELOPMENT OF A STEMMER FOR AFARAF TEXT RETRIEVAL

A THESIS SUBMITTED TO THE SCHOOL OF INFORMATION SCIENCE OF ADDIS ABABA UNIVERSITY

BY
OSMAN TAHA OMER
IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE IN INFORMATION SCIENCE

NOVEMBER 2015
ADDIS ABABA, ETHIOPIA

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
SCHOOL OF INFORMATION SCIENCE

DEVELOPMENT OF A STEMMER FOR AFARAF TEXT RETRIEVAL

BY
OSMAN TAHA OMER

Name and signature of members of the examining board

Name	Title	Signature	Date
Ato _____	Chairperson:	_____	_____
Ato Ermias Abebe	Advisor:	_____	_____
Dr. Solomon Teferra	Examiner:	_____	_____
Dr. Million Meshesha	Examiner:	_____	_____

Dedication

To my family

DECLARATION

This thesis is my original work. It has not been presented for a degree in any other university

_____

OSMAN TAHA

This is to certify that I have examined this copy of Master's thesis by

OSMAN TAHA and have found it is complete and acceptable in all respects, and has been submitted for examination with my approval as university advisor.

Name of Advisor	Signature of Advisor	Date
Ato Ermias Ababe	_____	_____

NOVEMBER 2015
ADDIS ABABA, ETHIOPIA

Abstract

This study describes the design of a stemming algorithm for Afaraf text retrieval system. Nowadays, a considerable amount of electronical information has produced in Afaraf. Information retrieval system is a mechanism that enables users to retrieve relevant unstructured information material from large collection.

The Afaraf morphology leterature reviewed in order to develop the rule-based stemmer. Each natural language has words structure in its own forms, that are different prefixes and suffixes, which need special handling of affixes with specific rules. The rule based stemmer proposed based on grammar and dictionary of Afaraf, included Numbers (singular and plural), personal pronoun, adjectives, adverbs, verbal-noun, strong and weak verb, indefinite pronoun, conditional and subjunctive mood, linkage to remove suffixes and prefixes from the word and produce stem word.

For this study text document corpus are prepared by the researcher used 300 text files of Afaraf documents, which collected from different school text books, Samara university modules, Qusebaa Maca magazines and other online and experiment is made by using eight different queries. Data pre-processing techniques of VSM involved for both document indexing and query text.

The evaluation conducted on the stemmer shows that the accuracy is 65.65 % with error rate of 4.50% for over-stemming and 29.85% for under-stemming. The information retrieval system registered effective performance of 0.785 precision and 0.233 recall.

It has been witnessed that the challenging task in developing a full-fledged Afaraf text retrieval system is handling morpholoical word variations. The performance of the system may increase if the performance of the stemming algorithm is improved and if standard test corpus is used.

Table of Contents

Table list

Algorithm list

Figure and code

List of Acronyms and Abbreviations

IR	Information Retrieval
VSM	Victor Space Model
Tf	term frequency
Idf	inverse document frequency
LSI	latent semantic index
Sim	Similarity

Chapter One

1.1 Introduction

The importance of archiving the written information was initially proposed around 3000BC [**1**], when the Sumerians designated special areas to store clay tablets with cuneiform inscriptions [**2**]. For thousands of years people have realized the importance of archiving and finding information. With the advent of computers, it became possible to store large amounts of information; and became a necessity finding useful information from such collections [**3**].

An Information Retrieval system designed to make a given stored collection of information items available to users for online interactions. At one time information consisted of stored bibliographic items, such as online catalogs of books in library or abstracts of scientific articles and information retrieval used to be an activity that only a few people engaged in. In today's world, the information more likely to be full-length documents, either stored in a single location, such as newspaper archives, or available in a widely distributed form, such as the World Wide Web, and hundreds of millions of people engage in information retrieval every day when they use a web search engine [**4**].

The Indexing is an offline process of representing text documents and organizing large document collection using indexing structure such as Inverted file, sequential files and signature file to save storage memory space and speed up searching time. Searching is the process of relating index terms to query terms and return relevant hits to users query. Both indexing and searching are interconnected and dependent on each other for enhancing effectiveness and efficiency of IR systems [**2**].

Information Retrieval System evaluation is a broad topic covering many areas including information-seeking behavior usability of the system's interface; its broader contextual use; the compute efficiency, cost, and resource needs of search engines. A strong focus of IR research has been on measuring the effectiveness of an IR system, to determining the relevance of items, retrieved by a search engine, relative to a user's information need [27].

IR systems retrieve information from unstructured text material with the aim of providing the relevant documents and sorted most relevant at the top of the list responding to a user's query. Early IR systems were primarily used by users which are experts; hence initial IR methodology was based on keywords manually assigned to documents, and on complicated Boolean queries. As automatic indexing and natural language queries gained popularity in the 1970's, the Information Retrieval Systems became increasingly more accessible to none expert users [**3**]. The typical interaction between a user and an IR system has the user submitting a query to the system then IR system can match user's queries with relevant text documents [**5**], which returns a ranked list of objects that hopefully have some degree of relevance to the user's request with the most relevant at the top of the list.

Soon after computers were invented, people realized that they could be used for storing and mechanically retrieving large amounts of information. [25] In the 1950s, this idea materialized into more concrete descriptions of how archives of text could be searched automatically. Several works emerged in the mid 1950s that elaborated upon the basic idea of searching text with a computer.

There are different challenges in implementing IR system. Information retrieval is language dependent process which needs integrating knowledge of information retrieval techniques and natural language [**6**]. Most of IR techniques are developed for English language and it is always difficult task applying it for other languages. The other thing is tradeoff between efficiency and effectiveness in terms of IR system performance. IR system should be both effective and efficient but always increasing decreases the other. So coming up with efficient effective system needs tough task [**2**]. Additionally the evaluation performance of IR system is also challenging task, because performance IR evaluated in relative to relevance of retrieved document toward users'query and efficiency. But it is really difficult to identify what is relevant from the irrelevant one, because human information need behavior is fluctuating.

The need to store text documents and retrieve written information became increasingly important over regions, especially when one language come official language of one region with inventions like paper and the printing press.

1.2 Statement of the Problem

Afaraf is one of the languages of lowland Eastern Cushitic sub-group, along with Oromo, Somali etc. and is particularly close to Saho which also belongs to this sub-group of Cushitic family. Afaraf is the native language for the Afar people in Ethiopia, Djibouti and Eritrean. It is one of the widely spoken languages in red sea coast line [**7**].

There are more than 80 languages in Ethiopia; Afaraf is one of these languages [**7**]. As the 2008 Central Statistic Agency of Ethiopia report shows there are more than 1.4 million speakers of Afaraf in Ethiopia [**8**].

The natural language has its own characteristics and features that make it quite difficult to have the same stem for natural language or follow the same stemming pattern and apply the same stemming rules for all natural languages. Each natural language has words structure in its own form of that are different prefixes and suffixes, as well as individual exceptions, need special handling and a careful formation of a frame with specific rules [**6**].

Afaraf language share much of its vocabulary and grammatical structure with Afaan Oromo & Other Cushitic languages.. Nonetheless, it has its own peculiarities by which it differs [**7**]. Afaraf is morphologically very productive; derivation, reduplication and compounding are also common. The variable position affixes in Afar occur as either prefixes or suffixes depending on the initial segment of the verb root [**9**].

Like other Cushitic family languages, Afaraf uses Latin based script and it has 26 basic characters. Afaraf is the official language of Afar regional state of Ethiopia and also academic language for primary school of the region and the academic language in primary school for Afar people in Eritrea and Djibouti. Afaraf language given as a field of study in Samara University, and also a number of journals, magazines, education books; other books are available in electronic format both on the Internet and on offline sources. There is huge amount of information being released with this language, since it is the language of education and research, language of administration and political welfares, language of ritual activities and social interaction.

Languages are naturally different; development of language is highly associated development of technology [**10**]. The reason that initiated this study is luck of electronic format documents retrieval system in afaraf languge and enabling development of Afaraf to grow with current information technology support that facilitate great communications between Afar people. In this globalization age, information is highly needed than anything else, as it is needed for the society to cohere and speed up communications. But finding this important information needs system support [**11**].

In Ethiopia there have been an attempts to develop Information Retrieval system for Amharic [**12**] [**13**], Tigrinya language [**14**] and Afaan Oromo [**10**]. The work done for Afaan Oromo language includes development of retrieval algorithms for Afaan Oromo documents which are written in Latin based script characters. Additionally there was also an attempt to develop a search engine for the same language [**10**].

Prior to this work, there are no works done for stemming and Information retrieval system development for Afaraf language. Implementation of this work helps users of Afaraf to find information of their need simply without any difficulty.
The aim of this study is to develop a prototype for Afaraf text retrieval system that organize document corpus using indexing and search relevant ones as per query entered by users based on vector space model.

To the end, this study tries to answer the following research questions.

- What are the moropholoy formulations of Afar language to develop rule based stemming?
- What are the challenges accrued in developing rule based stemmer?
- What is the accuracy registered by developed stemmer based on the sample text documents?

Derived research questins:

- What are the basics formulations of Afar language to perform text operations?
- What is the effect of stemming on the VSM of Afaraf text retrieval system?

- What are the suitable components to design victor space model based retrieval system?
- What is the performance registered by designed prototype system based on the sample text documents?

1.3 Objective of the Study

1.3.1 General Objective

The main objective of this study is to develop a stemmer for Afaraf text retrieval system.

1.3.2 Specific Objectives

In order to meet the general objective, the following specific objectives are performed.

- To review the morphology of Afaraf;
- To review related literatures on previous works related with stemming algorithm and information retrieval system;
- To build a corpus of Afaraf text documents;
- To perform text operation like tokenization, stop-words removal and stemming;
- To develop rule based stemmer to experiment with Afaraf text;
- To test and the stemmer on a prototype Afaraf Information Retrieval System that searches referent documents from unstructured corpus;
- forward recommendations for further study.

1.4 Scope and limitation

The main aim of this study is developing stemmer for Afaraf information retrieval text that effectively stemming Afaraf text. The work mainly implements using Information retrieval system from corpus of Afaraf textual documents. The developed stemmer for Afaraf in this research work on conflates the inflectional and derivational variants whose affixes occur in a regular pattern and compound words occurring in the language. Developed stemmer does not involve special characters of Afaraf languge. The test system involves both indexing and searching of text. Other data types, such as image, video, and graphics are out of the focus of the research.

The text operation process of automatic indexing, stemming and stop words are highly dependent on the text documents language. The using stemming motivation is the need in order to increase effectiveness of retrieval system since stem of a term represents a broader notion than the original term itself [**15**].

To identify content of index terms and query terms a series of text operations implemented, such as tokenization, stop word removal, normalization and stemming are applied. Index terms are organized using inverted index file and searching for documents satisfying query terms guided by vector space model.

Considering the time constraint, 9000 words in 300 text files of Afaraf text document corpus used in experiment for evaluating the performance of the Information Retrieval system developed in the study.

1.5 Significance of the study

Designing or developing stemmer for information retrieval search engine reduces storage of index files. Developing stemmer for Information Retrieval system in a local language helps the speakers of the language to access information. Generally, the study has the following significance:-

- The stemmer could improves the development of the language;
- This research can be used as one input to integrate and develop a general Cushitic language information retrieval system and to form multi-lingual information retrieval;
- This research can also help to develop some tools like checker for spell, grammar checker, thesauri, word frequency counter, document summarizers and indexers;
- This research can be used for developing cross lingual retrieval systems. For example one can feed a Afaraf text query to search engine, this entered word may be translated to other languages like English and access documents in English or any other languages;
- Enable Afaraf speakers retrieving text documents in Afaraf efficiently and effectively.

1.6 Research Methodology

This research conducted in order to figure out challenges of implementing stemmer for Afaraf information retrieval system. Accordingly, the following step by step procedures are followed to achieve the main objective of the study.

1.6.1 Literature Review

To have conceptual understanding and to identify the gap that not covered by previous studies different materials, including journal articles, conference papers, books, thesis and the Internet have been consulte. In this study the review mainly concerned works that have direct relation with the topic and the objective of the study. These include previous works done on the area of stemmer and information retrieval giving more attention to local and international works that attempt to develop information retrieval system and search engine.

1.6.2 Data Collection and preparation

A text corpus is one of the resources required in natural language processing researches. A good-sized text can reasonably show a language's morphological behavior. Selection of text is an important component in developing a simple stemmer for information retrieval system. For the purpose of this research short documents were used, the researcher used 300 text file corpus of education textbooks of Afaraf which can be representative of the language Afaraf and magazines. A sample text of different disciplines collected from different textbooks. These items covered issues such as, news, social, health, art, and education.

1.6.3 Programming Tools

A program developed using the Python programming language to implement the Information Retrieval System for stmmer. This is because Python has very rich string manipulation techniques and the researcher has some experience of writing programs using Python. It is simple, strong, involves natural expression of procedural code, modular, dynamic data types, and embeddable with in applications as a scripting interface [**16**].

1.6.4 Testing Process And Evaluation Techniques

The expermintation for evaluating effectiveness of the stemmer done by using information retrieval syste and over-stemming and under-stemming techniques used for performance measuring.

After IR system designed and implemented for stemmer, its performance and accuracy should be evaluate. Evaluation of IR system involves two things effectiveness and efficiency [24]. Even if it is important to evaluate both effectives and efficiency, as the objective of this study the researcher considered as a vital part of the documentation only effectiveness. The IR system effectiveness evaluated in various ways. The main one is precision, recall and F- measure [23].

The corpus prepared and queries are constructed then relevance judgments made for evaluating effectiveness of the work. Recall and precision techniques are used for measuring retrieval effectiveness of the IR system [**2**].

Precision is the number of relevant documents a search retrieves divided by the total number of documents retrieved. In other word it is the fraction of the documents retrieved that are relevant to the user's information need. Recall is the number of relevant documents retrieved divided by the total number of existing relevant documents that should have been retrieve. It is the fraction of the documents that are relevant to the query that are successfully retrieve.

1.7 Organization of the Thesis

The thesis organized in a simple structure in which six chapters distinguish. Chapter one is initial part, in this section basic concept about IR system, the statement of the problem, objective of the study, scope of the study, research methodologies, study‘s significance and programming tools are discussed.

The rest four chapters of the thesis organized in the following way. Chapter two include of Afaraf morphology literature review and chapter three is literature review and it involves two main topics, related works and conceptual review. Conceptual review is review on Information retrieval system and related topics to the thesis. Related work involves work done so far on the research topic locally and internationally.

Major technique and methods used in this study discussed in chapter foure, include preparation of corpus, query preparations, the section techniques use for indexing and searching, which are term weighting technique (tf*idf), retrieval model (VSM), and the similarity measurement (cosine similarity) discuss in this part of the paper.

The fifth chapter of the work is an experimentation and result analysis of the study. In this section, include perimentations, retrieval performance evaluation, result analysis, Findings and challenges discussion in detail.

Finally in chapter six major findings including faced challenges written as a conclusion and works identify as future work and needs to get attention of other researchers are list in recommendation section.

Chapter Two

2 Afaraf Literature review

An information retrieval is very wide-ranging area of study, with the main aim of searching information of relevant documents from large corpus that satisfies information needs by the users. Since the IR research involves language dependent process, in this study the researcher attempt to review the literature articles of Afaraf morphology works done, to enable us to design prototype of IR system for Afaraf language.

2.1 Afaraf Morphology

Afaraf language is speaking in the horn of Africa by ethnic group of Afar. Afaraf language is part of the Cushitic branch of the Afro-Asiatic family. It is spoken by more 2 Million peoples and most of native speakers are people living in Ethiopia, Djibouti and Eretria [11] [19].

The first time writing Afaraf in Roman or Latin script was last quarter of 19th century (in the decade extending from 1875 to 1885) by Herr leo Reinisch who in the brought to light a book which he called “Die Afar Sprache”.

Morphology in Afaraf deals with all combinations that form words or parts of words. Two main classes of morphemes, stems and affixes: Stem is the root of the word, supplying the main meaning, and Affixes add “additional” meanings in words.

e.g.

- cin-a root cin (denies)
- t-able root able (see)

2.1.1 Affixes

Affixes are well-known methods that are used to identify morphological variants which is common to all languages, words variants are formed by the usage of the affixes,

An affix is a bound morph that realized as a sequence of phonemes. Concatenative morphology (a word is composed of a number of morphemes concatenated together) uses the following types of affixes [**17**]:

Prefixes: A Prefix is an affix that attached in front of a stem.

e.g. **ta** – adigee (do you know)

Suffixes: A Suffix is an affix that attached after the stem. Suffixes used derivationally and inflectionally.

e.g. ab – **te** (she did)

2.2 Word Formation

Word is defining as a smallest thought unit vocally expressible composed of one or more sounds combined in one or more syllables. The word can form one or more morphemes. A lot of ways can combine morphemes to create words. There are three broad classes of ways to form words from morphemes [**18**] and Afaraf use the three forms in word formation, they are: **inflection**, **derivation** and **compounding** [**19**]

2.2.1 Inflection

Inflection is the combination of a word stem with a grammatical morpheme, usually resulting in a word of the same class as the original stem, and usually filling some syntactic function and is productive in Afaraf [**20**].

e.g. nak (drink milk!) nak - e (I/he drunk milk)

The word inflectional morphemes modify a word's tense, number, aspect, and so on [**21**].

2.2.2 Derivation

Derivation is the combination of the word stem with a grammatical morpheme, typically resulting in a word of a different class. In case of derivation Afaraf morphology is unproductive [**21**].

E.g: 'diglo' is a noun, and ‘iggil’ is a root of verb.

2.2.3 Compounding

Compounding is the joining of two or more words to form a new word. Afaraf use a word compounding like other languages.

e.g	ruffa-**exceh**	(I get happy)
	ruffa-**inneh**	(we get happy)
	ruffa-**inteh**	(you/she get happy)

2.3 Dialects and Varieties

Afaraf is a sociolinguistic language consisting four varieties: Aussa, Baadu (afar in Ethiopia), Kilbatay (Afar in Ethiopia and Eretria), and Laaqo (Afar in Djibouti)[1]. These four varieties depend on geographical area and there are strong similarities among these five varieties in words which make effectiveness of IR system high, but there is also difference between them according to voicing three alphabets in Latin scrip they are C, Q and X [**7**].

Afar in Ethiopia was written with the Ethiopic or Ge'ez script since around 1849, the Latin script and Arabic script has been used in other areas to transcribe the language. In the 1970s, two Afar intellectuals and nationalists, Dimis and Redo, formalized the Afar alphabet based on the Latin script and they called it Qafar Feera[2].

[1] http://afar alphabet/Afar _ Ethnologue.htm
[2] http://Afar language - Wikipedia, the free encyclopedia.htm

Starting from 1875 the Afaraf written used in different ways using Latin script by number of missionaries, researchers, interested persons etc..., but starting from the seventies of the last century, they was reached on a consensus to use Dimis-Reedo orthography and till now its wide practical usage in the Afar National Regional State of FDRE and Afar Districts in Djibouti. The principal features of Dimis-Reedo orthography is its representation of the three non-existing sounds in Latin [**7**].

2.4 Alphabets and Sounds

Afaraf is phonetic language, characters sound the same in every word in contrast to English in which the same letter may sound differently in different words [**22**] [**7**].

Afaraf uses Latin character (Roman alphabet), and the basic sound system of Afaraf has some modifications on sound of consonant and vowels. Afaraf has the 17 consonants and ten vowels Afaraf has ten vowels: [a], [i], [e], [u], [o] and their long counterparts [aa], [ii], [ee], [uu] and [oo] [**20**], Furthermore, in 20th century and in the 2nd millennium the two necessities introduced in Afaraf's written words. One has something to do with the consonants; the consonants have increased from 17 to 21 that use all the Roman alphabet and the other with the double consonants, which is two-letter and three composed sounds like **sh, ch, kh, ts and tch and** they used mostly for nouns [**7**].

Here in Table 2.1 alphabets are listed in both Upper case and Lowercase. Alphabets sound is also included with how it pronounced in English words [**9**]. Also available online[3], [4].

[3] http://en.wikipedia.org/wiki/Afar

Alphabet	Sound	Alphabet	Sound	Alphabet	Sound	Alphabet	Sound	Alphabet	Sound	Alphabet	Sound
A	[a] Abē I did.	B	[ba] bakaarē be thirsty	T	[ta] tufē spit	S	[sa] saadē drought	E	[e] efqē irrigated.	C	[Ca] Cata help
K	[ka] kallacē beg	X	[xa] gaaxē guard	I	[i] idfiqē. I paid.	D	[da] duddubē swell	Q	[qa] qeegē lean	R	[ra] ruubē send
F	[fa] fiikē sweep	G	[ga] gabbatē tempt	O	[o] okmē. I ate.	L	[la] lokē Make a dough	M	[ma] Mak turn	N	[na] nookē settle on haunches.
U	[u] uqrufē. I rested.	W	[wa] duwē herd	H	[ha] hiqē crumble off	Y	[ya] Yab took	sh	Shaami	ch	Chaayina
kh	Khaliil	Ts	Tsefaay								

Table2:1 Upper Cases, lower case 1

- **Afaraf Consonants**

Most Afaraf constants do not differ greatly from English, but Afaraf rule of alphabet include additional characters after **Z** letter, the characters combination formed by two digits and three digits, they are ch, sh, kh, gh, dh, gn, ts and tch [**23**].

The three letters in Afaraf has different voice form Latin voice according to Parker & Hayward (1986), "[x] is a voiced post-alveolar plosive (represented in the IPA by), though it may occur as a flap when it occurs intervocalically (Parker & Hayward 1986:214). [q] and [c] are both pharyngeal fricatives: [q] is voiced and [c] is voiceless. The IPA symbols for these are [ʕ] and [ħ] respectively" [**9**].

The table below listed the Afaraf consonants in the standard orthography with IPA notation in brackets.

		Labial	Alveolar	Retroflex	Palatal	Velar	Pharyngeal	Glottal
Stops	Voiceless		**t** [t]			**k** [k]		
	Voiced	**b** [b]	**d** [d]	**x** [ɖ]		**g** [g]		
Fricatives	Voiceless	**f** [f]	**s** [s]				**c** [ħ]	**h** [h]
	voiced						**q** [ʕ]	
Nasals		**m** [m]	**n** [n]					
Approximants		**w** [w]	**l** [l]		**y** [j]			
Tap			**r** [ɾ]					

Table2:2 Afaraf consonant 1

- **Vowels and stress**

Afaraf vowels are similar to that of English difference is Afaraf has additional 5 vowels called long vowels (xer yangayyi). There are five vowels **a, e, o, u** and **I** called short vowels (yuxux yangayyi). All vowels pronounced in the similar way throughout every Afaraf literature. These vowels pronounced in sharp and clear fashion, which means, each word pronounced strongly.

- *short* vowels

  - **a** [ʌ]
  - **e** [e]
  - **i** [i]
  - **o** [o]
  - **u** [u]

- *long* vowels

  - **aa** [aː]
  - **ee** [eː]
  - **ii** [iː]
  - **oo** [oː]
  - **uu** [uː]

Sentence final vowels of affirmative verbs are aspirated (and stressed) for example: abeh(aˈbeʰ) “He did” Sentence final vowels of negative verbs are not aspirated which is not stressed for example maabinna /ˈmaabinna/ “He did not do.” Sentence final vowels of interrogative verbs are lengthened (and stressed) for example baritiyyaa, abee?, macaa? (aˈbeː) “Did he do?” Otherwise, stress in word-final.

2.5 Grammar

Every language has its own rules and syntax. Afaraf also have its own grammar.

2.5.1 Syntax:

In Afaraf language the basic word order, it is subject–object–verb (SOV), a verb final language like most other Cushitic languages. There are verbs ending in a final long vowel Monosyllabic Words in Afaraf, like laa (cattle) not la only and bee (took).

Example: *Usun maaqo yakmen.*

2.5.2 Gender

Afaraf has two grammatical genders, masculine and feminine in similar way to other Afro-Asiatic language. There is no neutral gender as in other language like English[4] . All names that end with 'e', 'to' and 'o' are feminine except bayo and abuuro (poor household) and the all names that end with 'a', 'u' are masculine except angu, Sunku, dumbu and cugbu and also all names that end in a consonant are masculine except wadar, baabur, Sinam, qefer, ceber and wasif[5].

Additionally Afaraf has grammatical gender for a place, all names end with ‘lu’ are masculine and all names end with ‘le’ are feminine [**24**] by removing “lu” and “le” suffixes can find common stem of noun for both, the table in below illustrate examples.

[4] http://afaraf.free.fr/page/genre.html

[5] http://afaraf.free.fr/page/genre.html

Masculine	*Feminine*	*Stem*
Qadaylu	Qadayle	Qaday
Qittalu	Qittale	Qitta
Galaqlu	Galaqle	Galaq
Cinnalu	Cinnale	Cinna

Table 2:3 Afaraf gender 1

2.5.3 Numbers morphology:

There are singular and plural numbers in Afaraf. Nouns of Afaraf are mostly divided into ten major functions of their singular and plural forms [**24**].

I. The first one is irregular forms of singular and plurals for example:

numu → labha
barra → agabu
baxa → xaylo etc...

II. The second functions of singular and plural forms are formed through the addition of suffixes. The function of plural suffix is 'itte'; a final vowel 'a' and 'u' are dropped from singular before the suffix e.g.:

iba → ibitte
bara → baritte
gita → gititte
rasu → rasitte
wadu → waditte

III. If singulars are ends with vowel 'i', 'u' and third letter equal end vowel, then a final two letters are dropped from singular before the suffix, function of plural suffix is duplicate vowel 'i' and 'u' then adding second consonant of singular and plus the vowel 'a'. e.g.:

gulubu → guluuba

duquru → duquura

ragidi → ragiida

caagidi → caagiida

IV. If singularsare ends with vowel 'a' and third and fourth letters are 'aa' vowels, then for plural form the vowels 'aa' are replaced by vowels 'oo' .e.g.:

migaaqa → migooqa

lubaana → luboona

dabaana → daboona

kitaaba → kitooba

V. If singulars are ends with vowel 'o', and third letter is 'o' vowel, then a final two letters are dropped from singular before the suffix, function of plural suffix is duplicate vowel 'i', then adding second consonant of singular and plus the vowel 'i'. e.g.:

kimbiro → kimbiiri

qingiro → qingiiri

xinbiqo → xinbiiqi

VI. If singulars are ends with vowel 'e', 'i', and 'o', then a function of plural suffix is duplicate vowels end (e, i and 'o') then adding second consonant of singular and plus the vowel 'a'. e.g.:

qale → qaleela

buqre → buqreera

ayti → aytiita

addi → addiida

mako → makooka

amo → amooma

VII. A final two letters (ta and wa) are dropped from singular before the suffix function of plural suffix is duplicate a fourth vowel and then adding second consonant of singular and plus the vowel 'u'. e.g.:

kullumta → kulluumu

baxuwwa → baxuuwu

xagorta → xagooru

VIII. A plural function form are formed by dropping a final three letters (yta, yto and ytu) from singular. e.g.:

bocoyta → boco

cadoyta → cado

gibyaytu → gibya

kallaytu → kalla

garrayto → garra

IX. A plural function form are formed by dropping a final vowel letters (a and i) from singular. The function of plural suffix is ‘wa’; e.g.:

ala → alwa

yangula → yangilwa

qari → qarwa

gali → galwa

X. Some of nouns has not plural function form ; e.g.:

Sana

Qangaara

cundubu

XI. Some of nouns has not singular function form ; e.g.:

Weeqa

Saca

Roobu

Generally, each derivational word there are many inflectional suffixes, according to the noun, some examples below in table 3.1 show the stems for the some nouns word in afaraf:

Singular number	Plural number	Stem	Suffix
Buxa	Buxaaxi	Buxa	Axi
Buta	Butaati	Buta	Ati
Lafa	Lafoofi	Lafa	Oofi
Gaba	Gaboobi	Gaba	Oobi
Gulubu	Guluuba	Gulub	Uba
Ala	Alwa	Ala	Wa
Iba	Ibitte	Iba	Itte
Gada	Gaditte	Gad	Itte
Kullumta	Kulluumu	Kullum	ta/umu
Baduwwa	Baduuwu	Baduw	wa/uwu
Bocoyta	Boco	Boco	Yta
Cadoyta	Cado	Cado	Yta
Gibyaytu	Gibya	Gibya	Yat
Garrayto	Garra	Garra	Yto
Xinbiqo	Xinbiiqi	Xinbiq	Iqi

Table 2.4 different stem of nouns 1

2.6 Personal pronoun (Numiino kee Haysit Ciggiile)

Afaraf pronouns include personal pronouns (refer to the persons speaking, the persons spoken to, or the persons or things spoken about), indefinite pronouns, relative pronouns (connect parts of sentences) and reciprocal or reflexive pronouns (in which the object of a verb is being acted on by verb's subject)[i]. The Possessive pronouns are invariable in gender and number, the long form is used when the pronoun is placed at the end of the sentence e.g. *"Ah koofiyat yiimi*. *"This hat is mine"*. *"Yi wadar geeh immay kum my genniyo"*. *"I found my goats but I have not found yours"*. The following table illustrated Afaraf pronoun, which called **Numiino kee Haysit ciggiile.** [**24**]

English	**Subject**	**Objects**	**Reflexive**	**Possessive**	**Possessive adjectives**
I	Anu	Yoo	Innih	Yim/Yiimi	Inni=Yi

You (sg.)	Atu	Koo	Isih	Kum/Kuumu	Isi =Ku
He	Usuk	Kaa	Isih	Kayim/Kayiimi	Isi=Kay
She	Is	Tet(teeti)	Isih	Tetim/Tetiimi	Isi=Tet
We	Nanu	Nee	Ninnih	Nim/Niimi	Ninni=Ni
You (pl.)	Isin	Sin (siini)	Sinnih	Sinim/Siniimi	Sinni=Sin
They	Usun	Ken (keeni)	Sinnih	Kenim/Keniimi	Ken=sinni

Table 2.5 Afaraf personal pronouns 1

Demonstrative pronouns they agree in gender according nearest and away, but not in number (in variable plural) . “ah” for nearest female and ’toh’ for away female, and “tah” for nearest male and “woh” for later male.

Examples:
‘Ahm yiimi’ ‘ this one is mine’

‘Tah yiimi’ ‘this is mine’

2.7 Adjectives (Weelo)

Adjectives are very important in Afaraf because its structure is used in every day conversation. Afaraf Adjectives are words that describe or modify another person or thing in the sentence [**25**].

Color- Bisu	**Size- Weelo**	**Shape-Ceelo**	**Qualities**
English-Afaraf	English-Afaraf	English-Afaraf	English-Afaraf
Black -- Diti Blue -- Kuclinaana Brown -- Caawinaana Yalow -- Walqinaana Green -- Inxixi Red -- Qisi White -- Qidi **Eg**. Aqdo saga tamaate The white cow comes	big -- Naba Small -- Qunxa Lon -- Kaxxa narrow -- Ceyina short -- uxxi small -- xinnaa tall -- Xer thick -- Cabbole	Square -- Affara gona Triangular -- sidiica gona	bad -- Umah clean -- Saytu dark -- dete difficult -- Gibdeh dirty -- Wasaka dry -- Kafina easy -- Qilsamali empty -- foyya

Table 2.6 Afaraf adjective (Weelo) 1

2.8 Adverbs (Abnigurra)

Afaraf adverbs are part of speech. Generally, they are words that modify any part of language other than a noun. The following table lists some of Afaraf adverbs.

Adverbs of time			
English	Afaraf	English	Afaraf
Yesterday	Kimaala	Very	Kaxxam
Today	Asaaku	Fast	Sisikih
Tomorrow	Beera	Really	Nummah
Before yesterday	Ammaaca	Quit	Tibbo
After tomorrow	Becaa	Well	Giffa
Now	Awak	Quickly	Sisikuk
This	Ah	Hard	Gebdaane
Then	Tokek	Together	Hittaluk
Morning	Saaku	Slowly	Qilsuk
Later	Sarra	Carefully	Cubbusak
Tonight	Abara	Along	Tonnaluk
Next week	Yamaate Ayyam	Absolutely	Deggaluk
Soon		Immediatelly	Saanih
Right now	Tawak	Last night	Kimaali bara
Recently	Qusebih		

Table 2.7 Afaraf adverbs (abnigurra) 1

Some of adverbs con formed from noun by adding **haak** or **aak** at the end of nouns, for verb by adding **ak/uk** and by adding **uk** or **luk** to the adjective [**24**]. By removing ak/uk from the adverbs we can come up with stem or command form of verb.

Example:

Noun		**Adverb**	**stem**
▪ Nammay	(haak)	Nammay**haak**	Nammay
▪ Sidoc	(haak)	Sidoc**haak**	Sidoc
▪ Nahar	(aak)	Naha**raak**	Nahar

Verb		**Adverb**	**stem**
▪ Kataat	(ak)	Kataat**ak**	Kataat!
▪ Ab	(ak)	Ab**ak**	Ab!
▪ Hirig	(ak)	hirig**ak**	hirig!

▪ Dadal	(ak)	dadal**ak**	dadal!
▪ Celtaam	(ak)	celtaam**ak**	celtaam!
▪ Waris	(ak)	waris**ak**	waris!
▪ Aaxig	(uk)	Aaxag**uk**	Aaxag!
▪ Ayussuul	(uk)	Aysussuul**uk**	Aysussuul**!**
▪ Argiq	(uk)	Argiq**uk**	Argiq!

Adjective		**Adverb**	**stem**
▪ Qunx	(uk)	qunx**uk**	qunx!
▪ Sissik	(uk)	sissik**uk**	sissik!
▪ Sitta	(luk)	sitta**luk**	sitta!
▪ Siital	(luk)	Siital**luk**	Siital!
▪ Baxsa	(luk)	baxsa**luk**	baxsa!
▪ Digga	(luk)	Digga**luk**	Digga!
▪			

2.9 Afaraf Verbal-Nouns

As English has verbal nouns, also Afaraf has verbal nouns. Afaraf verbal nouns (gerunds) ending in **IYYA and ISIYYA** as the English gerunds (verbal nouns) end in....**ING.** This words cited as examples, are nouns and also verbs at the same time. They are nouns because their last letter **"A"** [**7**].

E.g.	**Afaraf**	**English**
1.	Gex**iyy**a =	Go**ing**
2.	Sug**iyya** =	Wait**ing**
3.	Safar**iyya** =	Travelli**ng**
4.	Kud**iyya** =	Runn**ing**
5.	Amaat**iyya** =	Com**ing**
6.	Xiin**isiyya** =	sleep**ing**

This words are verbs, when the termination ...**IYYA** and **ISIYYA** are removed, because that removal of ...**iyya** and …**isiyya** makes of the remaining radical or stem of the verbal-noun, a verb of a command mood [**7**]. The most stem of Afaraf is command verb.

E.g. **Afaraf** **English**

- Gex(**iyya**) = Gex! Go(ing) = Go!
- Sug(**iyya**) = sug! Wait(ing) = Wait!
- Kud(**iyya**) = Kud! Runn(ing) = Run!
- Xiin**isiyya = Xiin**! Sleep(**ing) =** Sleep!

As mentioned in above a verb/iyya can form noun, which called verbal noun, these noun, is verb + suffixes [**24**]. We can stem by removing iyya or suffixes from verbs.

Verbal noun	***Stem***	***Suffix***
Gaciyya	Gac	Iyya
Abiyya	Ab	Iyya
Ciniyya	Cin	Iyya
Gexiyya	Gex	Iyya

Table 2.8 Afaraf verbal nouns 1

The verbal nouns ending with...**iyya** are categorized into two groups. The stem that remains after the removal of the ...iyya termination is known as groups to which a particular verb in Afaraf belongs to. If it remains as a command (imperative), then it called the 1st group; and if it remains not as the command, then it called the 2nd group [**7**] for example *Amaat/iyya to Amaat.*

According to conjugation of Afaraf, the 1st group verbs have **suffix** inflections while that the 2nd group verbs have **prefix** inflection [**7**] [**24**].

Conjugation of the 1st group verbs in simple present tense

1. Anu nak-a = I drink milk
2. Atu nak-ta = You (sing) drink milk
3. Usuk nak-a = He drinks milk
4. Nanu nak-na = We drink milk
5. Is nak-ta = She drinks milk
6. Isin nak-tan = You (pl.) drink milk
7. Usun nak-an = they drink milk

Conjugation of the 2nd group verbs in simple present tense

1. Anu **a**maate = I come
2. Atu **t**amaate = you (sing.) come
3. Usuk **y**amaate = He comes
4. Nanu **n**amaate = We come
5. Is **t**amaate = She comes
6. Isin **t**amaat**en** = You(pl.come)
7. Uson **y**amaat**en** = They come

2.10 Strong and weak verb

Afaraf has two type of verbs and they are called strong verb (**Kulsa le abna)** and weak verb **(Qaku le abna).** This words cited as examples, are nouns and verbs at the same time. They are nouns because their last letter [**24**]. Strong verb is verb that can generate and produce number of verbs and ending with...**iyya**, ….**isiyya**, like verbal noun but it has other additional ending and they are …**siisiyya** and ….**itiyya.** weak verb is verb that cant not generate other possible verbs and its ending the same to strong verb except …**siisiyya** ending.

Strong verb example:

• Kud**iyya**	Kud**isiyya**	Kud**siisiyya**
• Kas**iyya**	Kass**iisiyya**	Kas**itiyya**
• Gex**iyya**	Gex**siisiyya**	Gex**itiyya**
• Xiin**iyya**	Xiin**isiyya**	Xin**siisiyya**

Strong verb example:

- Amaat**iyya**
- Soon**itiyya**
- Ard**iyya**
- Wakl**isiyya**
- Rook**itiiya**

2.11 Post-positions (Yascassi)

Afaraf has post-positions for whereabouts things & persons, rather than prepositions, the post-positions are not words but letters [**7**]. They are only four in number, there is also **Y** letter which used as conjunctions mostly added at the end of nouns [**24**] and the post-positions are:

1 1 = On, upon, over, above etc....

2 k = From

3 h = For, to, towards etc...

4 t = in, at, by, into etc...

Examples in sentences

1 Wokke**l** yan = He/it is **over**/there

2 Wokke**k** bah ! = Bring fr**om** there

3 Wokke**h** bey ! = Take it **to** there

4 Wokke**t** tan = she/it is **in** there

5 Qale**y** abaluk sugne qoonat nek bayte. = a mountain was see hided by dusts.

2.12 Indefinite pronoun (Amixxige-waa ciggile)

Afaraf has Indefinite pronoun that make Afaraf little different from other language, it used in peace talk which may not certainly specified [**24**].

Example:

1. Numuk **teeni** amaatelem nakkale. = we think one of the men may come
2. Barrak **teyna** amaatelem nakkale. = we think one of the women may come

The second form of Indefinite pronoun added at the end of noun, and it is too long comparing to other suffixes [**24**].

Example:

1. Fax**innaanim** teetik yoh baaha ixxic.
2. Ab**innaanim** neh warissaanam meqe.
3. Assokoot**innaanim** doorite liton.

2.13 Conditional and subjunctive mood (Sharti kee Niyat Gurra)

Afaraf has mood called **Conditional and subjunctive** moods, and very difficult to differentiate between them, they ending verb with …ek, …eenik, …aamal, …aanama, …aanamal, … taanama…. inniyoy, …innitoy, …innay, inninoy, …ittoonuy, …innoonuy, …innitoonuy…eemil,

and …eenimi, be added at the end of verbs in tenses. And they considered as suffixes in verb [**24**].

Examples:

1. aytiitat caxxi mudek biyak neh yantaabbe
2. baritteenih tumurruqeenik ellecabol
3. Is tamaat**eemil**, usuk gexak yen = if she coming, he going out
4. Is sug**taamal**, usuk gexeh sugak yen. = if she had stayed here, he has been left.

2.14 Linkage (Qaada yasgalli)

A conjunction which called qaada yasgalli of Afaraf has two type of conjunction, the first type of conjunction lied between example: kee, innaa, ikkal, immay, akkiiy, ...aay, ...eey, ...iiy, ...ooy, ...uuy

2.15 Afaraf tenses (wargu)

Afaraf has tenses, which called wargu in Afaraf. Wargu are three like English basic tense: present, past and future. And they grouped in two called easy and hard (sahlin kemo and gibdi kemo) [**24**], when a complete action made from one word is known as sahlin kemo .e.g. Anu can nakah = I am drinking milk and when a complete action made from two words are known as gibdi kemo, for example Anu can nakeh en = I was drinking milk. The tenses of Afaraf verb are can form by taken verb + suffix or prefix + verb or prefix + verb + suffix structures.

Example:

Verb	verb + suffix	prefix + verb	prefix + verb + suffix
Amaate! (Come!)	Anu amaate**h**	Is **t**amaate	Isin **t**amaate**enim**
Able! (See!)	Anu Able**m**	Nanu **n**able	Oson **y**able**enim**

The formulation of verbs mostly depends on personal pronouns in the tenses, some of examples three tenses of sahlin and gibdi kemo are illustrated in below table.

Sahlin Kemo				
Verb	***Ciggiile (Personal pronoun)***	***yan wargu (present tense)***	***yen wargu (past tense)***	***Yanu-waa wargu (future tense)***
Gex	Anu	Gexa (h)	Gexe (h)	Gexeyyo
	Atu	Gexxa ”	Gexxe ’’	Gexetto
	Usuk	Gexe ’’	Gexe ’’	Gexele
	Is	Gexxa ’’	Gexxe ’’	’’
	Nanu	Genna ’’	Genne ’’	Gexenno
	Isin	Gexxan(aanah)	Gexxen(eenih)	Gexetton
	Oson	Gexan ’’	Gexen ’’	Gexelon
	Anu	Gexam	Gexem	Gexeyyom
	Atu	Gexxam	Gexxem	Gexettom
	Usuk	Gexem	Gexem	Gexelem
	Is	Gexxam	Gexxem	’’
	Nanu	Gennam	Gennem	Gexennom
	Isin	Gexxaanam	Gexxenim	Gexettonum
	Oson	Gexaanam	Gexeenim	Gexelonum

Table 2.9 Afaraf tenses (wargu sahlin kemo) 1

Gibdi kemo				
Verb	***Yan wargu***	***Yen wargu***		***Yanu-waa wargu***
Gex	Gexah an	Gexeh en	Gexak en	Gexu –waa
	Gexxah tan	Gexxeh ten	’’ ten	Gexxu – wayta
	Gexah yan	Gexeh yen	’’ yen	Gexu – waa
	Gexxah tan	Gexxeh ten	’’ ten	Gexxu – wayta
	Gennah nan	Geuneh nen	’’ nen	Gennu – waynu
	Gexxaanah tannin	Gexxeenih teneu	’’ tenen	Gexxoonu – waytay
	Gexaanah yanin	Gexeenih yenen	’’ yenen	Gexoonu – waau
	Gexoh anim	Gexeh enem	Gexak enem	Gexu –waam
	’’ tanim	Gexxeh tenem	’’ tenem	Gexxu – waytam
	’’ yanim	Gexeh yenem	’’ yenem	Gexu – waam
	’’ tanim	Gexxeh tenem	’’ tenem	Gexxu – waytam
	’’ nanim	Geuneh nenem	’’ nenem	Gennu – waynam
	’’ taniinim	Gexxeenih teneenim	’’ teneenim	Gexxoon waytaanam
	’’ yaniinim	Gexeenih yeneenim	’’ yeneenim	Gexoonu – waanam

Table 2.10 Afaraf tenses (Gibdi kemo) 1

According verb, Afaraf verb makes a complete surface-level of tenses [**26**], each tenses has stem verb, the tenses are formed according to stem + suffixes or prefixes + stem + suffixes. Most of the verb contain prefixes end with suffixes. The vowels in tenses suffixes *short* vowels and can be long vowels. The prefixes in present and past tenses of verbs are mostly three consonant, they are “n, t and y”. The various formations of the verb “sug” and “abl” presented on the Table 3.2 below.

Tenses	Verb	Stem	Suffix
Present Tenses	Sugah, sugaah I wait	Sug	Ah, aah
	Sugtah, sugtaah you wait(you sigular)	sug	Tah, taah
	Sugtaanah You wait (you plural)	sug	Taanah
	Suganah They wating	sug	Anah
Past Tenses	Sugneh, sugneeh we waited	Sug	Neh, neeh

	Sugteenih you waited	Sug	Teenih
	Sugeenih they waited	Sug	Eenih
Future Tenses	Sugtu wayta You wait	Sug	Tu
	Sugoonu They wait	Sug	Oonu

Table 3.2 tenses 1

Tenses	Verb	Preffixe	Stem	Suffix
Present Tenses	tableh you see	T	abl	eh
	ableh he see	Y	abl	eh
	nableh We come	N	abl	eh
	tableenih You are seeing.	T	able	eenih
Past Tenses	Tubleh You saw	T	ubl	eh
	yubleh you saw	Y	ubl	eh
	Tubleenih You saw	T	ubl	eenih

Table 2.11 tenses 1

2.16 Afaraf Ngation

Afaraf negations are indicated by “ma” as prefix and also some time “m” on the verbs, the negation “ma” stands before all consonant.

.e.g

- masoolinna
- matakma
- manaadiga

According to vowels, negation “ma” are harmonizes with the next vowels and the assimilated vowels can reduced to one short vowel before consonant.

.e.g

- Maugutta → muugutta
- Maesserinno → meesserinno
- Mailaalisa → miilaalisa

2.17 Afaraf symbol character (Astooti)

Afaraf has characters, which called astooti in Afaraf, and they are listed in a table 4.9 [**24**].

Astooti	Qafarafa	Ingliizafa
•	Ximmó	Full stop
,	Catuffá	Comma
;	Xommo le catuffa	Semicolon
:	Namma ximmo	Colon
?	Essèr asta	Question mark
!	Cakkum asta	Exclamation mark
-	Giitoh asta	Hyphen
" "	Galumu	Quotation mark
( )	Lacawa	Bracket

Table 2.12 afaraf special symbols (xagiss 1

But afaraf has other additional character which called **Xagissa**, and these characters added at the end of words and over vowels to add meaning over the word sound [**26**].

These symbols are not in our study, they are four symbols and they are:

- kaqitta, xagissa (^) = Káqita Reytâ cana
- xukkutta xagissa (´)= xukkuta Noór Cuseeni
- Maktá xagissa (..) = máka kü abba kuwabba
- xakabta xagissa (∪) = xakaba koona∪amo koonaamo
-

2.18 Ordinal number (Caddoh ixxima)

Afaraf use ordinal number like English language, and it called caddoh ixxima. Caddoh ixxima added the two common suffixes for masculine and feminine at the end of word numbers. Caddoh ixxima for masculine is haytu suffix and for feminine is hayto suffix.

Example:

Masculine ordinal number in word	Masculine ordinal number in number	Feminine ordinal number in word	Feminine ordinal number in number
Inik**haytu**	1haytu	Inik**hayto**	1haytó
Nammay**haytu**	2haytu	Nammay**hayto**	2 haytó
Sidoc**haytu**	3haytu	Sidoc**hayto**	3 haytó
Taban**haytu**	10haytu	Taban**hayto**	10haytó
Soddom kee konooy**haytu**	35haytu	Soddom kee konooy**hayto**	35haytó
Lactam kee fereey**haytu**	64haytu	Lactam kee fereey**hayto**	64haytó
Bool**haytu**	100haytu	Bool**hayto**	100haytó
Bool kee kontom kee fereey**haytu**	154haytu	Bool kee kontom kee fereey**hayto**	154haytó

Table 2:13 Ordinal number (Caddoh i 1

Chapter Three

3 Overview of IR system

The meaning of the term information retrieval is extremely wide-ranging, however from perspective of computer science a common definition provided by different scholar's books. For instance Cambridge University book [**2**] defines as: "Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)".

IR is concerned as a domain of information searching, which searches both structured and unstructured information [**2**]. Unstructured information searching of documents is from document corpus and the World Wide Web.

IR system includes various process and techniques. The whole IR system includes two main subsystems Indexing and searching [**15**]. Indexing is the process of preparing index terms, which are either content bearing or free text extracted from documents corpus. Searching is process matching users information via query to documents in collection via index terms. These searching and indexing processes is guided by model, Information Retrieval Model. There are various IR models including VSM, Probabilistic and Boolean to list a few. Additionally it is important to measure performance of IR system. The performance evaluation techniques help us to know accuracy of the IR system. A diagram below in figure 3.1 [**27**]shows IR system component and their interrelation.

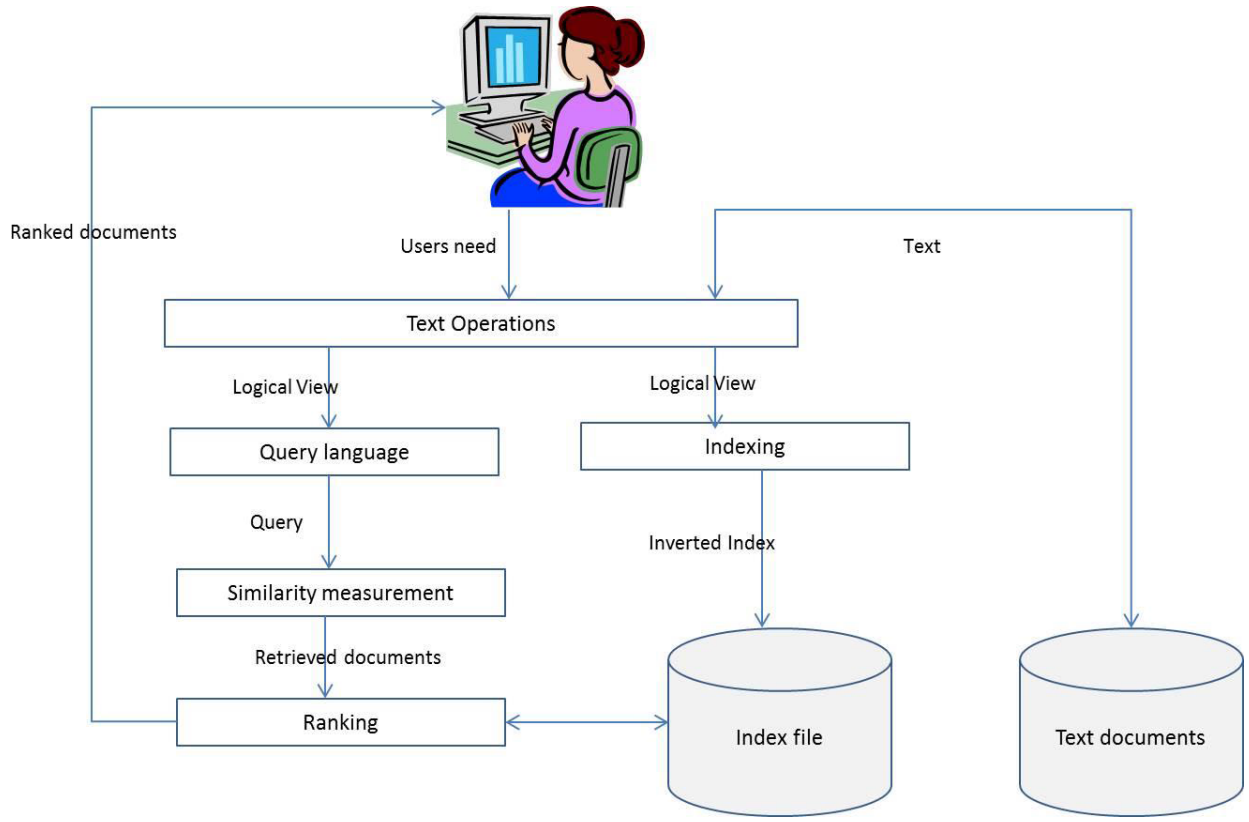


Figure 3.1 1

3.1 Indexing

Representing documents for retrieval process is called the indexing process. The process of indexing takes place in an offline process, indexing is extracting index terms from document collection and organizes them using Index structure. Indexing is an arrangement of index terms to permit fast searching and reducing memory space requirement. The main aim of indexing is to permit fast searching and reading memory space requirement used to speed up access to desired information from document collection as per users query such that it enhances efficiency in terms of time for retrieval D. Hiemstra [**27**]. An index file consists of records, called index entries. Index files are much smaller than the original file.

The indexing process consists of three basic steps: the first step defining the data source which specifying documents the operations to be performed on them, and also the specification of elements of a document can be retrieved e.g., the full text, the title, the authors. The second step is transforming document content to generate a logical view which is text operations like

tokenizing, remove stopwords and stemming and final step is building an index of the text on the logical view to allow for fast searching over large volumes of data. There three different index structures might be used, but the most popular one is the inverted file [**28**].

The inverted file stores a map from content to its locations in a database file. Inverted file is a mechanism for indexing a text collection so as to make the searching task fast. For each term the inverted file contains information related to all text location where the word occurs and frequency of occurrence of terms in a document collection.

3.1.1 Construction of inverted file

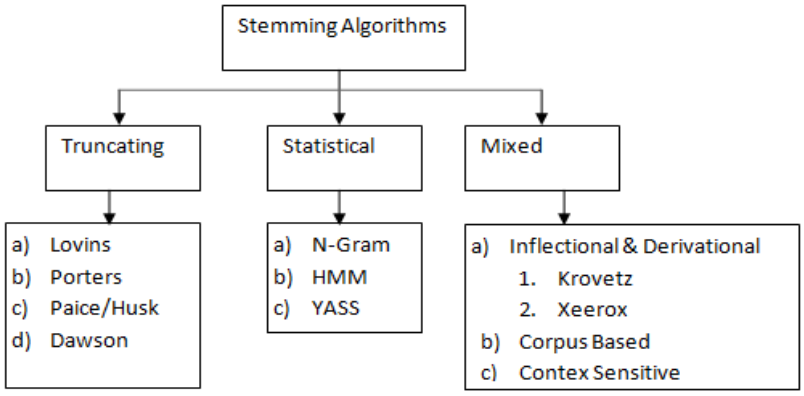
The process of construction of inverted file follows the four critical steps; firstly, collecting document to be indexed, secondly, Lexical Analysis (tokenization) of the text, turning each document in to a list of tokens, thirdly, Linguistic preprocessing, producing a list of normalized and stemmed tokens, which are the index terms, and finally index the documents that each term occurs in by creating an inverted index, consisting of two files: a directory and postings. The vocabulary file is the set of index terms in the text collection and it organized by terms. The vocabulary file stores all of the keywords that appear in any of the documents in lexicographical order and for each word a pointer to posting file [**29**].

Lexical Analysis: lexical analysis/Tokenization of the text -digits, hyphens, case folding and throwing away punctuations marks. It can define token as an instance of a sequence of characters. Each such token is a candidate for an index entry, after further processing. The remove of non-functional special characters, intuitively it should improve the performance, because most of the noise caused by punctuation such as periods and commas etc. One of challenges related to tokenization is-- These names of biomedical often contain special characters such as numerals, hyphens, slashes and brackets, and the same entity often has different lexical variants. Clearly, a simple tokenizer for general English text cannot work well in biomedical text [**30**]. The other challenge is identifying numerical values like dates, phone numbers, and IP addresses. Additionally Chinese and Japanese has no space between words, which makes difficult space and punctuation mark based tokenization. Arabic and Hebrew are

basically written right to left, but with certain items like numbers written left to right; the challenge here is it is not possible to use the same algorithm used for Latin based language for such languages too. That is why tokenization is called natural language dependent technique. The good tokenization can improve the performance by up to 80% [**30**].

Stop-words Removal: Elimination of stop words or filters out words which are not useful in the retrieval process. Stop-words are most frequent terms which are common to every document, and have no discriminating power one document from the other. So stopwords not considered in indexing process. According to *Zipf's law* [**29**] few terms occur frequently, a medium number of terms occurs with medium frequency and many terms with very low frequency. This shows that writers use limited vocabulary throughout the whole document, in which even fewer terms used more frequently than others. Further Luhn defined word significance across the document in. Luhn suggested that both extremely common and extremely uncommon terms are not very useful for indexing. So we should upper and lower cut-off points. Upper cutoff enables to remove the most frequent terms whereas, a lower cut-off controls less frequent terms which believed to be non-content bearing terms. [**29**]

Stemming: stemming is pre-process step in text mining applications, and used in most search engines and information retrieval systems. It is core natural language processing technique for efficient and effective IR system [**31**]. Broadly, stemming algorithms classified into three groups: runcating mathods, statistical methods and mixed methods, each of them has it own ways to stem the word variants, the figure 3.2 .in the below illusterats stemmer mehods.



Ffigure 3.2 1

Truncating method are removing the suffixes or prefixes (commonly remove Affixes) of the word. Number of stemmers proposed under truncationg, first popular stemmer proposed by Lovins in 1968 and it performs a lookup table and rules [**32**]. The ather one is porter-stemming algorithm, it is most popular stemming method proposed in1980. It is based on the idea of rule based that remove suffixes in English language [**33**]. There are different stemming algorithms, which developed, and as listed in figure 3.1, but Stemming is language dependent process in similar way to other natural language processing techniques. It is often removing inflectional and derivational morphology. E.g., automate automatic, automation → *automat*. Stemming has both advantage and disadvantage. The advantage is it helps us to handle problems related to inflectional and derivational morphology. That makes words with similar stem/root word to retrieve together. This increases effectiveness IR system. Stemming has disadvantage sometimes: some terms might be over stemmed, this changes meaning of the terms in the document; different terms might be reduced to the same stem, which still enforce the system as to retrieve non relevant documents [**34**]. Stem is the static part of a word and main part in any search in engine as mentioned above. Most of Stems are roots words, that means, one root word can take many derivational suffixes and change formation or meaning [**24**].

3.1.2 Document Representation and Term Weighting

This thesis focus more in term weighting of document representation and but there are different ways of document representation. Document representation helps us to give different weight for different terms with respect to given query, the term-weighting improves quality of the answer set since it displays in ranked order. It helps judge as if document is either relevant or not with respect to users query. In this work ***tf*idf*** term weighting is used for determining relevance of the terms [**35**] .

There are different mechanisms of assigning weight to terms.

I. Binary Weights
    a. Represents each document as a set of unique terms in a collection. Each of the terms are mapped to (0, 1). If the termt occurs in the document di, then the term t have value 1; if termt is absence then the term t have value 0.

II. Non-binary weigh

a. Term Frequency (tf) *Inverse Document Frequency (idf)

In this study tf *idf weighting technique be use, but there are three different ways of calculating term weight in tf*idf, firstly calculating a Term Frequency tf(i,j) = freq(i,j) / max(freq(k,j)), second calculation is Inverse Document Frequency idf(i) = log (N/n_i), finally calculation is tf *idf . The reason why tf *idf weight is selected, because it is normalize weighting technique and it is standard way of calculating weight. The term frequency*inverse document frequency also called tf*idf, and it is well know method to evaluate how important is a word in a document in Information Retrieval and text mining. Statistical method reflects how important a word/term is to a document in collection/corpus. The tf*idf is also a very interesting way to convert the textual representation of information into a Vector Space Model (VSM). The value of tf*idf increases with proportion to the number of times a word appears in the document and decreases as the term exist frequently in the document corpus. The more common terms that exist in almost all documents has lower score of tf*idf, whereas terms exist frequently in single but not in others have higher score [**35**].

The IR systems are using tf*idf weighting technique: $w_{ij} = tf(i,j) * idf(i)$. Search engines and information retrieval systems use this weighting technique often. It can be used for filtering stop-words and have application in text summarization and classification [29].

For example if the user is *"the brown cat"* it is only documents that contains terms *"the"*, *"brown"*, and *"cat"*, which should be considered as relevant. So other documents that are not having any of these three terms are excluded from retrieval. In order to distinguish importance level of document frequency of each terms with (term frequency) in each documents counted [25].

The good thing about tf-idf is that it relies on both term frequency (tf) and inverse document frequency (idf). This makes simple to reduce rank of common terms throughout the whole document corpus, but increase in rank of terms exist in fewer documents more frequently [22].

The ***term frequency*** (***tf***) is simply the number of times a given term appears in that document. This count is usually normalized to prevent a bias towards longer documents (which may have a higher term count regardless of the actual importance of that term in the document) to give a measure of importance of the term within particular document.

$Tfij = fij / max\{fij\}$ ……...…Equation 2. 1 1

The ***inverse document*** frequency ***(idf)*** is measure whether the term is common or rare across all documents. It is obtained by dividing the total number of documents by the number of documents containing the term, then taking the logarithm and quotient.

Higher idf value is obtained for rare terms whereas lower value for common terms. It is mainly used to discriminate importance of term throughout the collection.

$Idf = \log2 (N/ df i)$ ……..Equation 2. 2 1

Document frequency (*df*) — number of documents containing the given term. The more a term *t* occurs ***throughout*** all documents, the more poorly *t* discriminates between documents. The less frequently a term appears in the whole collection, the more discriminating it is.

Then the tf***idf** is ***product*** of *tf* and *idf*.

$Tf*idf = tfij * \log2 (N/ dfi)$ ……Equation 2. 3 1

3.2 IR Models

IR models identify the document representation, the query representation, and the matching function. There are two good reasons for having models of information retrieval, the first reason is that models guide research and provide the means for academic discussion; the second reason is that models is it can serves as a blueprint to implement an actual retrieval system.

An Information Retrieval models predicts and explains what a user be find relevant by given the users query. By use of the model proves correctness of evaluation and experiments. A model of information retrieval predicts and explains what a user find relevant to the given query. The correctness of the model's predictions can be tested in a controlled experiment. In order to do predictions and reach a better understanding of information retrieval, models must be firmly grounded in intuition, metaphors and some branch of mathematics. Intuitions are important because they help to get a model accepted as reasonable by the research community. Metaphors

are important because they help to explain the implications of a model to a bigger audience. Mathematical models are used in many scientific areas with the objective to understand and reason, so mathematics are essential to formalize a models, to ensure consistency, and to make sure that it can be implemented in a real system. The model of information retrieval serves as a blueprint which is used to implement an actual information retrieval system. [**27**]

Several models have been proposed for this process. The three most used models in IR research are the Boolean model, the vector space model and the probabilistic models. [**36**].

3.3 Boolean Model

Early information retrieval systems were Boolean systems. In the Boolean there are three basic logical operators AND, OR and NOT. AND is a logical product, OR is a logical sum and NOT is a logical difference. AND is used to group set of terms in to single query/statement. For example 'Information AND Technology' is two term query combined by 'AND'. In such case only document indexed with both terms be retrieved. If terms in the user query are linked by operator OR, documented with either of terms or all terms be retrieved. For example, if query is social OR political, document containing social, or social, or both be retrieved. Even though it has been shown by the research community, that the Boolean systems in retrieval process are less effective comparing with others like ranked retrieval systems [**36**].

Boolean systems have several shortcomings, there is no inherent notion of document ranking, and it is very hard for a user to form a good search request. What makes Boolean model good model is that it gives a sense of control to expert/user over the system. It is like the user who is in charge that can deciding what should or shouldn't be retrieved by the system. Query reformulation is also simple because user is in charge of deciding what should be retrieved and should not. In contrast Boolean model may not retrieve anything if there are no matching documents or, retrieves all documents if terms in query are matching the terms in the documents. As mentioned above, there is no relevance judgment and does not provide a ranking of retrieved documents [**27**].

3.4 Vector Space Model

The vector space model of information retrieval is one of the most common models used to representing documents and a query, it also widely used in document classification. In this model, each document is represented as a vector of terms. The terms are the features that best characterize the document and can be anything from strings of length, single words, phrases or any set of concepts. In the vector space model before processing the terms, stopwords, terms with little discriminatory power, are eliminated. Also it is common to use the stems of the words instead of the actual words themselves [**37**].

Vector space model is a representation of terms and query as vectors embedded in a high dimensional Euclidean space, where each terms is assigned as a separate dimension as follows.

$$Dj = (w_{1j}, w_{2j}, w_{3j}, \ldots\ldots\ldots w_{tj}) \qquad \text{..... Equation 2. 4 1}$$

$$Q = (w_{1q}, w_{2q}, w_{3q}, \ldots\ldots\ldots w_{tq})$$

A weight associated with index term ki and document ji is denoted by wi;j , while a weight associated with index term ki and query q is denoted by wi;q. According to Buckley [**35**], the weights of the index terms appearing in the documents are computed using tf*idf weighting technique(see equation 2.3).

$$\mathbf{Wi;\ q} = \left(\mathbf{0.5} + \frac{\mathbf{0.5} * Freq\ i;q}{maxlFreq\ i;q}\right) * \frac{\mathbf{logN}}{\mathbf{ni}} \qquad \text{.....equation 2.8 1}$$

Where $Freq_{i;q}$ is the frequency of the term k_i in the query q. Using these weights, a document can be defined as $d_j = w_{1;j}, w_{2;j}, \ldots, w_{t;j}$ and a query can be defined as $q = w_{1;q}, w_{2;q}, \ldots, w_{t;q}$. Although Salton and Buckley [**35**] also suggest other ways of calculating both $w_{i;j}$ and $w_{i;q}$, the above formulas provide a rather good weighting scheme. From these weight vectors the similarity between a document and a query can be computed as follows.

$$sim(dj, q) \frac{d^{-}j.\ q^{-}}{|d^{-}||q^{-}|} \qquad \text{.....equation 2.9 1}$$

$$sim(dj, q) \frac{\sum_{i=1}^{t}(wi,j)(wi,q)}{2\sqrt{\sum_{i=1}^{t}(wi,j)^2}\sqrt{\sum_{i=1}^{t}(wi,q)^2}} \qquad \text{.....equation 2.10 1}$$

The VSM has three main advantages; first it improves information retrieval performance by calculating weighing of the index terms. Second, because partial matching is allowed, also documents that approximate the query can be retrieved. Third, by using the degree of similarity, the retrieved documents can be ranked according to their degree of similarity to the query. The disadvantage of the vector model is that the index terms are assumed to be mutually independent [**38**].

3.5 Evaluation of IR Performance

Evaluation of information retrieval system is highly related to the relevance concept. Relevance is the degree of correspondence between retrieved documents and users information needed. The users often look relevance from differently sides. What is relevant to some users might be irrelevant to other users. Nature of user query and document collection affects relevance. Additionally relevance depends on individuals'personal needs, preference subject, level of knowledge, specialization, language, etc. [25].

There are two way of measuring Information Retrieval system performance, Precision and Recall [45]. Precision is a ratio of relevant items retrieved to all items retrieved, or the probability of retrieved item is relevant. Averaging the precision values from the rank positions where a relevant document was retrieved. Set precision values to be zero for the not retrieved documents. On the other way, Recall is ratio of relevant items retrieved to all relevant items in the corpus or the probability of relevant item retrieved. There is always trade-off between precision and recall. If every document in the collection is retrieved, it is obvious that all relevant documents are retrieved, so that recall is higher. In contrary when only little proportion of the retrieved document is relevant to given query, retrieving everything reduces precision (even to zero). The higher score in both recall and precision means the higher the performance of the system. [25].

Precision is the number of relevant documents a search retrieves divided by the total number of documents retrieved. In other word it is the fraction of the documents retrieved that are relevant to the user's information need.

$$\text{Precision} = \frac{\text{Relevant Retrieved}}{\text{Retrieved}} \quad \text{.....equation 4.4 1}$$

Recall is the number of relevant documents retrieved divided by the total number of existing relevant documents that should have been retrieved. It is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{Recall} = \frac{\text{Relevant Retrieved}}{\text{Relevant}} \quad \text{.....equation 4. 5 1}$$

F Measure: Harmonic mean of recall and precision. Harmonic mean emphasizes the importance of small values, whereas the arithmetic mean is affected more by outliers that are unusually large.

$$\mathbf{F} = \frac{\mathbf{2PR}}{\mathbf{P+R}} = \frac{\mathbf{2}}{\frac{\mathbf{1}}{\mathbf{P}}+\frac{\mathbf{1}}{\mathbf{R}}} \quad \text{.....equation 4. 6 1}$$

3.6 Probabilistic Model

The probabilistic retrieval models is that their framework for modeling documents and queries is based on probability theory, which states that an information retrieval system is supposed to rank the documents based on their probability of relevance to the query. The principle takes into account that there is uncertainty in the representation of the information need and the documents. There can be a variety of sources of evidence that are used by the probabilistic retrieval methods, and the most common one is the statistical distribution of the terms in both the relevant and non-relevant documents [**3**] [**2**].

3.7 Related Works

The researcher reached from reviewed literatures made sure that there is no work done on this specific area. No text retrieval system is developed for Afaraf. In the following researcher have reviewed some works has been done so far on the related area.

3.7.1 IR Systems for International Languages

With the Internet technology and the increase in the size of online text information and the globalization, information retrial (IR) has gained more importance especially in commonly used languages like English language etc. There are many search engines for searching text documents, video, audio, software, and pictures. Additionally there are special purpose search engines like shopping webs, which specifically work for online marketing of products and services. There are many works being done on the area of information retrieval in Asia and western world. But most of the work being done is for major technology languages like, English, French, Chinese and Turkish.

3.7.2 Information Retrieval on Turkish Texts

Turkish is a free constituent order language and it use latin scrips, Turkish text retrieval assigning weights to terms in both documents and queries is an important efficiency and effectiveness concern in the implementation of IR systems. The article was used the *tf.idf* model for term weighting. The term weighting has three components: term frequency component (TFC), collection frequency component (CFC), and normalization component (NC) [**39**].

Stop-word list contains frequent of words that are ineffective in distinguishing documents. Author removed stop words using three stop-word lists. First the researcher used semi-automatically generated stopword list that contains *147* words, second is stopword list and just used the top most frequent 288 words with no elimination. Finally was used short stopword list that contains the most frequent first ten words (“ve” “bir,” “bu,”, “da,” “de,” “için,” “ile,” “olarak,” “çok,”, and “daha,” their meanings in order are “and,” “a/an/one,” “this,” “too,” “too,” “for,” “with,” “time,” “very,” and “more”).

The Author was compared the effects of four different stemming options on (Turkish) IR effectiveness stemming is a major concern. They are no stemming, simple word truncation, the successor variety method adapted to Turkish, and a lemmatizer-based stemmer for Turkish [**39**].

.

The term weigh was obtained using tf*idf, then the matching function for a query Q and a document Doc is defined with the following vector product (Salton & Buckley, 1988) [**38**]. The documents were ranked according to their similarity to queries and based text retrieved

$$similarity(Q, Doc) = \sum_{k} w_{dk} \cdot w_{qk}$$

According collection of documents the Author was obtained documents with different lengths that consist three sub-collections of short (documents with maximum 100 words), medium length (documents with 101 to 300 words), and long documents (documents with more than 300 words). In a similar fashion, it was divided the relevant documents of the queries among these sub collections in the scalability experiments [**39**].

The study was provided the first thorough investigation of information retrieval on Turkish texts using a large-scale test collection. The researcher listed their findings as follows.
The stop-word list has no influence on system effectiveness; longer queries improve effectiveness and longer documents provide higher effectiveness.
The researcher recommended the future direction that Furthermore, the stemming process can be improved to handle compound words and future research within the Turkish IR, are possibilities [**39**].

3.7.3 IR Systems for Local Languages

This review is not meant to be comprehensive; however, we believe that this review provides the background necessary to understand the contribution of this study to Afaraf text retrieval system. The review has been done to find out work done for local languages in Ethiopia. But the work found by the reviewers involves Amharic and Afaan Oromo related studies.

3.7.4 Stemmer for wolaytta text

The study describes the design and developing a stemming algorithm for Wolaytta language. The researcher reviews the language phenomenon in terms of word formation, and the researcher discussed the inflectional and derivational morphologies of the language, in order to model and develop an automatic procedure for conflation of words for the language.

For research work the reasecher used a text that used by Lamberti and Sottile (1997) for studying the morphology of Wolaytta language, the text consists 4421 words, 884 words (20% of text) used in test the performance of the stemmer, and 3537 words (80% of text) were used for training the stemmer.

The experiement result of the stemmer for Wolaytta language register 90.6% accuracy on the training set and error counted were 8.6% (304 words) over stemmed and 0.8% (28 words) understemmed on the training set and respectively.
When the stemmer runs on 884 words (20% of the text), accuracy was 86.9%. The percentage of errors recorded as understemmed and overstemmed were 9% and 4.1%, a dictionary reduction of 38.92% attained on the test set. The major sources of errors reported that the language requires more context sensitive rules for more effective conflation recommendations to further improvement of the stemmer.

3.7.5 Rule Based Stemmer for Afaan Oromo Text

Study describes rule-based stemmer of Afan Oromo; the researcher signed that most of concepts of developed stemmer adopted from porter stemming algorithms. The researchers used stop word list consists of prepositions, conjunctions, articles, and particles. The stemmer based on sequential steps that each removes a certain type of affix by checking conditions of substitution rules [**41**]. Some of conditions listed by researchers are:

- Does the stem end with a vowel?
- Does the stem end with a consonant?
- Does the stem end with specific character?
- Does the 1st syllabus of the stem duplicated?

Literature review done for the studies noticed that the language morphologically very productive, derivations and word formations and the language has different linguistic features with affixation, reduplication and compounding. The researchers organized the morphological analysis of the language in to six categories. They are nouns, pronouns and determinants, case and relational concepts, functional words, verb and adverbs [**41**].

Balanced corpus collected for stemmer test from different text sources of Afaan Oromo, which involves newspapers, bulletins and magazines. The evaluation for the stemmer was counting stemming accuracy, stemming errors and reduction of dictionary size. The stemmer performed at an accuracy 94.84%, stemming error 5.16%, the compression 38%, based on the sample data of 500 words [**41**].

The researchers' recommended further study be required to increase the effectiveness of the rule-based stemmer. The described rules for developed stemmer can be a base for further research and can support extending stemming rules.

3.7.6 Amharic Retrieval Systems

Work done for Amharic retrieval is Amharic text retrieval system which is done by using latent semantic index(LSI) with singular value decomposition by Tewdros G, at Addis Ababa University School of Information Science [**13**].

The researcher, initiated to develop Amharic text retrieval using LSI technique. Amharic is morphologically rich language that has lexical variation. The researcher hypothesis is that vector space model decreases the effectiveness of the system in comparison to LSI. That is why the author preferred using LSI. The main advantage of using LSI is that it handles problems related with polysemy and synonymy. The paper used stop-word list that identified by Nega [**40**], non-content bearing terms and stop-words were removed. Additionally term weighting technique has used for measure importance of terms in the document. The term weighting technique used was tf*idf. Cosine similarity used to measure similarity and dissimilarity. The result obtained by using LSI was better. Achieved performance by using VSM is 0.6913 and by LSI 0.7157. LSI improves the VSM result by 2.4%.

The researcher recommended the future direction that includes: stemmed index terms might improve performance of the system and recommended to be used in the further work, if relevance feedback is supported performance of the system be improved, and the algorithm LSI may record better performance in cross lingual of Amharic-English, if it is used [**13**].

3.7.7 Afaan Oromo Text Retrieval Systems

Afaan Oromo is one of the languages, which arranged under Cushitic family with Afaraf [**7**]. Work done for Afaan Oromo information retrieval is called Afaan Oromo text retrieval system which is done, by using victor space model (VSM) with regular value decomposition by Gezehagn, at Addis Ababa University School of Information Science [**10**].

The researcher, initiated to develop Afaan Oromo text retrieval system to Afan Oromo text documents by applying techniques of modern information retrieval system. Afaan Oromo is morphologically rich language that has lexical variation. Vector Space Model of information retrieval system was used to guide searching for relevant document from Oromiffa text corpus. The model is selected since Vector space model that is widely used model for information retrieval system. The index file structure used in the paper is inverted index file structure.

The researcher in this study prepared text document corpus encompassing different news article and experiment made by using nine different user information need queries. Various techniques of text pre-processing including tokenization, normalization, stop word removed using stop word list and stemming are used for both document indexing and query text depending rule based stemming algorithm by Debela and Abebe [**41**].

The experiment shows that the obtained result is 0.575(57.5%) precision and 0.6264(62.64%) recall. The researcher shows that the performance of the system lowered for various reasons as it identified by that study. The challenge tasks in the study were handling synonymy and polysemy, inability of the stemmer algorithm to all word variants, and ambiguity of words in the language [**10**].

The researcher recommended the future direction that includes: The area is on beginning level and wide open place for future study that may the further study figure out best model that works for Afaan Oromo retrieval system. The performance the system can be increased if stemming algorithm is improved, standard test corpus is used, and thesaurus is used to handle polysemy and synonymy words in the language [**10**].

Chapter Four

4 Methodology

An IR system essentially includes two main subsystems, indexing and searching. Indexing is an offline process of organizing documents using keywords extracted from the collection and used to speed up access to desired information from document collection as per users query. Searching is an online process that scans document corpus to find relevant documents that matches users query. Users query represented by a set of terms just like small documents before they can be matched with documents [**42**,**12**]. Figure 4.1 depicts the basic IR system architecture.

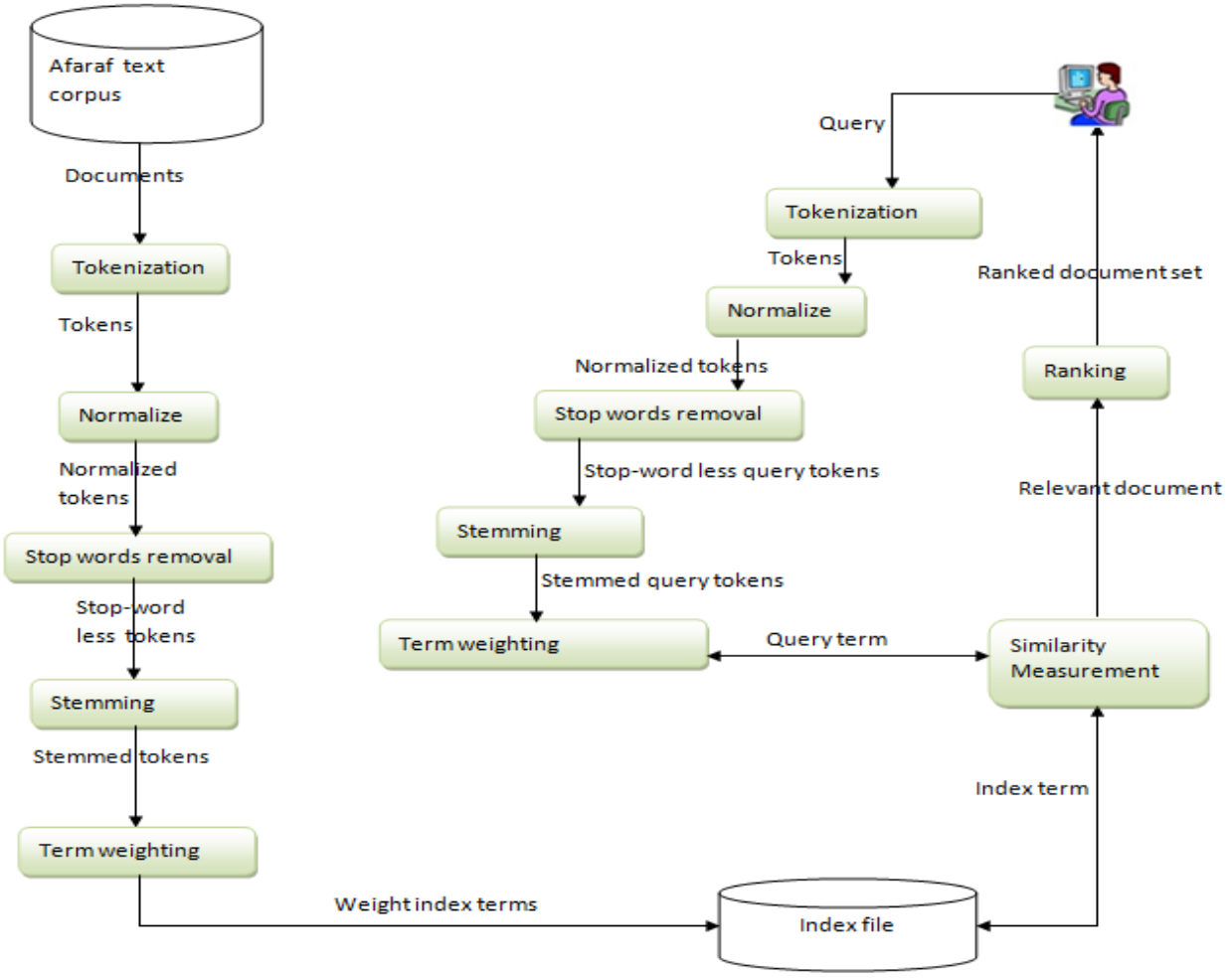


Figure 4.1 Afaraf Text Retrieval System 1

During the indexing process, given Afaraf text documents were organized using inverted index structure. Text operations were applied on the text of the documents in order to transform them

in to their logical representation of documents. The first step of indexing is tokenization of the text to identify stream of tokens, followed by normalization. Then stop-words removal applied to remove non-content bearing terms. Finally, stemming done on the terms, respective weights were calculated, and the inverted index file was constructed.

In the searching part, a similar text pre-processing (tokenization, normalization, stop-words removal, and stemming) technique is followed as in the indexing. Then similarity measurement techniques (cosine similarity) used to retrieve and rank relevant documents.

4.1 Data Preprocessing and Corpus Preparation

In Information retrieval system corpus is needed for evaluation of the system. The researcher used 300 Afaraf documents collected from different school text books, Samara university modules, Qusebaa Maca magazines and other online resources were used to compile a text corpus for the research. Each file is saved under common folder using .txt format.

As shown in table 4.1, the document corpus contains si groups, which are health, education, social, politics, news, culture, economy and art related areas.

No.	Types of Documents	Number of Documents
1	Education related	50
2	Health related	50
3	History related	30
4	Art related	30
5	Social related	50
6	News related	40
7	Culture related	30
8	Economy Related	20
	Total	300

Table 4.1: Corpus used for the developme of IR1

4.2 Query selection

Queries are identified and selected in order to make the experiment, the selected queries are eight as shown in table 4.2. These queries are marked across each document as either relevant or irrelevant to make relevance evaluation. The main importance of having identified queries is to evaluate the performance of the system. These queries are selected subjectively by the research after reviewing content each file manually.

Queries No.	**Queries terms**
Q1	baritto buxa qafarafih Bartiyya kee Barsiyyi Cogda
Q2	maaqo xagarah Qaafiyat tace
Q3	Sultaan Qalimirackee Sultaan Macammad Acawa aydaadu
Q4	Qafara rakaakayih doolatak xiinissoo kee saay biiro ummatta konferensi
Q5	Kas takke Maxcoo kee Qadar Maxcooca
Q6	Qafar qaadal Digib bux madaqa digaala aalle wayta abtoota
Q7	Baadal maddur angaaraw siyaasa maca ceelah?
Q8	Gino gadda nii dariifal maca ceelah?

Table 4. 2 query used for experiment1

4.3 Tokenization

Tokenization is the process of splitting character streams in to tokens. Tokenization in this work was also used for splitting document in to tokens and detaching certain characters such as punctuation marks [**43**]. A consecutive sequence of valid characters was recognized as a word in the tokenization process. The Algorithm 3.1 presents the tokenization procedures applied.

```
Open the file/corpus for processing
Create string container
Do
        Read the content of the file line by line and split to string by space
         Put to container for each strings
          For word in container
                If word contains punctuation marks, numbers, special characters
                Replace them with a space
           End for
While end file
```

Algorithm 4.1 Tokenization procedure 1

The above tokenizes the text documents as follows: first, the content of file is read line by line. Second, split them by space in to list of words. Third, check whether the word within the list contains punctuation marks, control characters or special characters of Afaraf; if any exist within the word replace it with space. This step continues until end of line is reached.

4.4 Normalization

Afaraf Normalization involves process of normalizing a terms in the document in to similar case format. For instance ‘Xaagu’ to ‘xaagu’ ‘Qari’ to ‘qari’ are all normalized to understandable as lower case.

4.5 Stop Word Removal

Afaraf domain independent stopwords include demonstrative, conjunctions, and Pronouns
The stop-words lists are developed mainly to remove stopwords there by reducing the size of the file. Removing the stop-words is essential because they do not contribute much to the content of the documents [**44**]. The conjunctions of Afaraf “kee”, “immay”, “innaa” and the pronouns “anu”, “atu”, and “yoo” etc… interrogative “iyyi?”, “macaa?”, “annih?”, “iyyiniimi?”, etc… possessive pronoun “yim”, “kum”, “kayim”, “tetim”, “yiimi”, “kuumu”, “kayiimi”, “tetiimi”, and “niimi”, etc…, Demonstrative “Ah”, “tah”, “woh”, “toh”, “ahim” “tahim”, “wohim” and “tohim” etc.. and indefinite pronoun “tuk ”, “tu”, “teyna”, “geerim”, etc… are examples of such highly frequent words. Removing Afaraf stopwords can reduce file size and processing time.

Stopwords could be removed using one of the two methods available. The first method is to remove high frequent terms by counting the number of occurrences of terms (frequency). The second method is using stopwords list for the language [**45**].

In this research the second method is used to apply stop word removal. Removing stop words is applied because some stop words may appear is differently if they are stemmed and can be considered as content bearing terms. In this study, stop words are complied by consulting books and dictionaries. The consulted books and dictionaries help to identify preposition, conjunction, articles and pronoun of the Afaraf language. After identifying the stop words in the Afaraf documents, the algorithm removes the stop words from the document corpus. The stop-words list is available in Appendix I.

```
Open stop word file
Read stop word list file
Open the corpus for processing
Do
    Read the content of the file line by line and assign the content to string
    If word is in stop word list
            Remove word
    Else
             Continue
    End if
    While end of file
```

Algorithm 4.2: Stop word removal 1

As shown in algorithm 3.3 above, the system reads the list of stop word from stop word list file and not indexed those words.

4.6 Stemming

Stemming is the process or normalization that reduces the morphological variants of words like inflected or derived words to a common form usually called a stem by the removal of affixes, usually performed before indexing. Stemming has significant effect in both the efficiency and the effectiveness of IR for many languages [**46**]. The complexity of stemming process varies with the morphological complexity of a natural language. In Afaraf text, there are many word variants/affixes [**47**]. Generally stemming transforms inflated words in to their most basic form or collapsing words into their morphological root. For example, the terms “nakte, naktenii,

nakteh, nakenih, nakneh, and nakenno" might be conflated to their stem, nak. The researcher developed a prefix and suffixes remover.

4.6.1 Compilation of Afaraf Affixes

The Afaraf affixes are of two different types, which are the prefix, and suffix [**48**]. Unlike English stemmers which work quiet well just by removing suffixes alone to obtain the stems [**33**]. An effective and powerful Afaraf stemmer not only must be able to remove the suffixes, but also the prefixes.

1.1.1 Compilation of Prefixes

Prefixes that were used to develop the algorithm is compiled from different sources based on their grammatical functions and from among the Afaraf words found in the document collection. For example: 'ya-agory' (is-he-hit) root is 'agor' (hit), 't-able'(you-see) root is 'able', 'm-akma'(I—not-eat), root is 'akm', 'ma-t-akma' (you-won't eat), 'aaqabe'(i-dringing), the root is 'aqab' and 'ya-amateh'(I am coming), the root is 'am'. The list is collected from Afaraf dictionary [**49**] , grammar books [**26**] and morph [**47**]. The most of prefixes are found in the verbs, same of prefixes list ranges from prefixes such as "n", "t", "y", "aa" and "ma".
Table 4.2 shows some of the prefixes collected for the development of the algorithm.

N
T
Y
Ma
Double vowels: aa

Table 4.2: Sample prefixes of Afaraf 1

4.6.2 Compilation of Suffixes

A similar approach as that used to compile the list of prefixes is used to compile the suffixes. The suffix range from single suffixes such as "h", "k", "","l", "t", "y", to combinations of suffixes

such as " taanah ", "loonuh", " aana". Steps lists shows the suffixes collected for the development of the algorithm.

The rules below remove suffixes successful or otherwise, convert given on the right. The step list examples arranged in four columns a first column is list of suffixes second replacement of suffixes, third examples of words show words ends with suffixes and the fourth column contain stemmed words. The algorithm follows steps mentioned below:

Step a:

Step a deals with a prefixes letters present and past tenses like t, n, y and ma negation. To remove preffixe and long vowel “aa” have checked. The subsequent steps are much more straightforward.

t	----->	t(aa)xige	----->	aaxige
	----->	tabl(eh)	----->	abl
	----->	tabl(eenih)	----->	abl
	----->	t(aa)xi	----->	aaxi
	----->	t(aa)bbe	----->	aabbe
	----->	taama	----->	taama
y	----->	yitbiq(eeniik)	----->	itbiq
	----->	yabl(eenih)	----->	abl
	----->	abl(eh)	----->	abl
	----->	yaqab(eenimil)	----->	aqabe
	----->	yaab(eyyo)	----->	aab
	----->	y(aa)lloonu	----->	all
	----->	y(aa)baanam	----->	aabaanam
	----->	y(aa)xigeenim	----->	aaxigeenim
	----->	y(aa)baanam	----->	aabaanam
n	----->	nabl(eh)	----->	abl
	----->	nableh	----->	ableh
	----->	nak	----->	nak
	----->	n(aa)bbek	----->	aabbek

Step 1:

Step 1 deal with four postpositions, adverbs, Subjunctive mood (Niya gurra) and conditional mood (sharti Gurra), present and past tenses. The four postpositions suffixes are “h, l, k, t”,t he adverbs suffixes are “haak ak, uk, luk”, the Subjunctive mood and conditional mood suffixe is “teek, ek”the present tense suffixe are “ah, tah nah” and past tense suffixes are “eh, teh neh”.

taah	----->	xiq**taah**	----->	xiq
tah	----->	sug**tah**	----->	sug
	----->	digir**tah**	----->	digir
naah	----->	ab**naah**	----->	ab
		xag**naah**	----->	xag
nah	----->	sug**nah**	----->	sug
aah	----->	cell**aah**	----->	cell
	----->	bartaan**aah**	----->	bartaan
ah	----->	sug**ah**	----->	sug
	----->	gex**ah**	----->	gex
teeh	----->	bic**teeh**	----->	bic
the	----->	sug**teh**	----->	sug
	----->	buulum**teh**	----->	buulum
neeh	----->	taamit**neeh**	----->	taamit
	----->	ab**neeh**	----->	ab
neh	----->	fugaa**neh**	----->	fugaa
	----->	gex**eh**	----->	gex
teek	----->	gey**teek**	----->	gey
	----->	kal**teek**	----->	kal
luk	----->	digga**luk**	----->	digga
	----->	siital**luk**	----->	siital
uuk	----->	numuu**k**	----->	num
uk	----->	qilsa**uk**	----->	qilsa
	----->	diiroon**uk**	----->	diiroon
haak	----->	nammay**haak**	----->	nammay
	----->	sidoc**haak**	----->	sidoc
	----->	fereey**haak**	----->	fereey
aak	----->	nahar**aak**	----->	nahar
	----->	salaf**aak**	----->	salaf
ak	----->	bartiyy**ak**	----->	bartiyy
	----->	ab**ak**	----->	ab
eek	----->	qammis**eek**	----->	qammis
ek	----->	gex**ek**	----->	gex
h	----->	fooca**h**	----->	fooca

k		irok		iro
l	----->	foc**al**	----->	faaca
t	----->	fooca**t**	----->	fooca
	----->	wokke**t**	----->	wokke

Step 2 :

Step 2 deal with Conditional and subjunctive mood: inniyoy, innitoy, innay, inninoy, ittoonuy, innoonuy, innitoonuy, eemil, eeni and eenimil. The subsequent steps are much more straightforward.

(v-0)eemi	----->	tamaat**eemi**(l)	----->	tamaat
	----->	tamq**eemi**(l)	----->	tamq
(v>0)eenimi	----->	yaqab**eenimi**(l)	----->	yaqab
(v>0)eeni	----->	taamit**eeni**(k)	----->	taamit
(v>0)eenii	----->	bart**eenii**(k)	----->	bart
(v>0)innay	----->	gex**innay**	----->	gex
(v>0)inniyoy	----->	gex**inniyoy**	----->	gex
(v>0)innitoy	----->	gex**innitoy**	----->	gex
(v>0)inninoy	----->	amaat**inninoy**	----->	amaat
(v>0)innoonuy	----->	gex**innoonuy**	----->	gex
(v>0)innitoonuy	----->	gex**innitoonuy**	----->	gex
	----->	amaat**innitoonuy**	----->	amaat
(v>0)ittoonuy	----->	gex**ittoonuy**	----->	gex

Step 3:

Step 3 deal with subjunctive mood and conditional mood: aamal, taanama, present: aanam, aana. The subsequent steps are much more straightforward.

(v>0)aama	----->	ab**aama**(h)	----->	ab
	----->	taama(l)	----->	taama
(v>0)taanama	----->	sugtaanama(l)	----->	sug
(v>0)aanama	----->	sug**aanama**(l)	----->	sug
	----->	gabbaaq**aanama**	----->	gabbaaq
(v>0)aanam	----->	gex**aanam**	----->	gex

Step 4:

Step 4 deals with ordinal number suffixes for masculine and femininet, haytu for masculine and hayto for feminine. The subsequent steps are much more straightforward.

(v>0) hayto	----->	ferey**hayto**(h)	----->	ferey
	----->	nammay**hayto**(h)	----->	nammay
(v>0)haytu	----->	taban**haytu**	----->	taban
	----->	nammay**haytu**	----->	nammay
(v>0) to	----->	gey**to**(h)	----->	gey
	----->	goran**to**(h)	----->	goran
(v>0)tu	----->	dadal**tu**(h)	----->	dadal
	----->	xag**tu**(h)	----->	xag

Step 5:

Step 5 deals with a singular and plural nouns. The subsequent steps are much more straightforward.

(v>1) ooti	----->	a	ast**ooti**	----->	asta
(v>1) aati	----->		but**aati**	----->	buta
(v>1) eera	----->	e	buqr**eera**	----->	buqre
(v>1) iina	----->	i	ayn**iina**	----->	ayni
(v>1) ooqa	----->		arq**ooqa**	----->	arqo
(v>1) uubu	----->	b	gul**uubu**	----->	gulub
(v>1) lu	----->		qaday**lu**	----->	qaday
(v>1) le	----->		qaday**le**	----->	qaday
(v>1) eela	----->	e	qal**eela**	----->	qale
(v>1)la			gita**la**		gita
(v>1) wa	----->		al**wa**	----->	al
(v>1) yta	----->		boco**yta**	----->	boco
(v>1) yto	----->		garra**yto**	----->	garra
(v>1) ytu	----->		kalla**ytu**	----->	kalla
(v>1) two	----->		barseeni**two**	----->	kalla
(v>1) le	----->		baxaabaxsa**le**	----->	kalla

Step 6:

Step 6 deals with a conjunction nouns. The subsequent steps are much more straightforward.

(v>1) aay	----->		camad**aay**	----->	camad
			qafaraay	----->	qafar
(v>1) eey	----->	e	gile**ey**	----->	gile
			leey	----->	leey
(v>1) iiy	----->		cuseen**iiy**	----->	cuseen
			kabeel**iiy**	----->	kabeel
(v>1) ooy	----->	o	maaqo**oy**	----->	maaqo
			cado**oy**	----->	cado
(v>1) uuy	----->		coox**uuy**	----->	coox
			cusul**uuy**	----->	cusul

Step 7:

Step 7 deals with a verbal nouns, strong and weak verb, ending with **iyya, isiyya, siisiyya** and **itiyya**. The subsequent steps are much more straightforward.

(v>0) siisiyya	----->	wag**siisiyya**	----->	wag
		cel**siisiyya**	----->	cel
(v>0) isiyya	----->	baxxaaq**isiyya**	----->	baxxaaq
		oob**isiyya**		oob
		xisiyya	----->	xisiyya
(v>0) itiyya	----->	kaq**itiyya**	----->	kaq
		meek**itiyya**		meek
(v>0) iyya	----->	bart**iyya**	----->	bart
		digr**iyya**	----->	digr
		xis**iyya**	----->	xis

Step 8:

Step 8 deals with present tenses “nam, am, an, tan, ta”. The subsequent steps are much more straightforward.

(v>0)tam	----->	nak**tam**	----->	nak
	----->	**gactam**	----->	**gac**
(v>0)tan	----->	nak**tan**	----->	nak

	----->	ma**tan**	----->	matan
(v>0)ta	----->	ciggil**ta**	----->	ciggil
	----->	bah**ta**	----->	bah
(v>0)nam	----->	nak**nam**	----->	nak
(v>0)am	----->	nak**am**	----->	nak
	----->	ab**am**	----->	ab
(v>0)an	----->	nak**an**	----->	nak

Step 9:

Step 9 deals with a future tenses. The subsequent steps are much more straightforward.

(v>0)eloonum	----->	abl**eloonum**	----->	abl
	----->	ab**eloonum**	----->	
(v>0)elem	----->	abl**elem**	----->	abl
	----->	nak**elem**	----->	nak
(v>0)elon	----->	gex**elon**	----->	gex
	----->	abl**elon**	----->	abl
(v>0)ele	----->	gex**ele**	----->	gex
(v>0)ennom	----->	abl**ennom**	----->	abl
	----->	aall**ennom**	----->	aall
	----->	sug**ennom**	----->	sug
	----->	amaat**ennom**	----->	amaat
(v>0)enno	----->	abl**enno**	----->	abl
(v>0)oonu	----->	yabl**oonu**	----->	yabl
	----->	sug**oonu**	----->	sug
(v>0)ettonum	----->	gex**ettonum**	----->	gex
(v>0)ettom	----->	gex**ettom**	----->	gex
(v>0)etton	----->	gex**etton**	----->	gex
(v>0)eyyom	----->	gex**eyyom**	----->	gex
etto	----->	gex**etto**	----->	gex
		abl**etto**		abl
eyyo		gex**eyyo**		gex
		amaat**eyyo**		amaat

Step 10:

Step 10 deal with genders, masculine suffixe *lu* and feminine suffixes are *le*. The subsequent steps are much more straightforward.

le	----->		kulsa**le**	----->	kulsa
	----->		rooci**le**(h)	----->	rooci
lu	----->		qaday**lu**	----->	qaday
	----->		qitta**lu**	----->	qittalu

Step11:

Step 11 deals with a double letter, if last letter equal precede letter then remove last letter, because most of root words form not end by double letters. The subsequent steps are much more straightforward.

aa	----->	a	seehad**aa**	----->	seehada
dd	----->		raddii(k)	----->	raddi
ee	----->	e	mood**ee**	----->	moode
kk	----->		a**kk**(ele)		ak
ii	----->	i	dafatir**ii**	----->	dafatiri
ll	----->		yaa**ll**(oonu)		yaal
oo	----->	o	ceel**oo**	----->	ceelo

Step 12:

Step12 deals with a prefixes *ma* negation. To remove preffixe the suffixe have checked. The subsequent steps are much more straightforward.

ma	----->		**ma**soolinna	----->	soolinna
	----->		**ma**naadiga	----->	naadiga
maa	----->	a	**maa**binnoonuy	----->	ab
muu	----->	u	**muu**gutta	----->	ugutta
mee	----->	e	**mee**sserinno	----->	esserinno
mii	----->	i	**mii**laalisa	----->	ilaalisa

Step 13:

Step 13 deals with double vowels of preffix, by removing one of vowels have no affect on root word, but morel produce proper stem. If words start by double vowels then remove first vowels of word. The subsequent steps are much more straightforward.

aa	----->	a	**aa**gar(uk)	----->	agar
		a	**aa**rr(iyya)		arr

4.6.3 The Rules

Trying to deal with each affix individually, the following rules are created. The rules are presented below in python-code:

4.6.4 Rules for removing prefixes

4.6.4.1 Rule-step a

```
sufcheker = ['eh','en','eenih','eeniik','eenimil','eenim','aanam']
def thestemmer(word):
l=len(word)
   if word.startswith("t") and (word.endswith not in sufcheker):
      word = word.replace('t','',1)
   elif word.startswith("y") and (word.endswith not in sufcheker):
     word = word.replace('y','',1)
   elif word.startswith("n") and (word.endswith not in sufcheker):
     word = word.replace('n','',1)
```

Figure 4.1: Python code for step a

Example:

t	----->	t(aa)xige	----->	aaxige
	----->	tabl(eh)	----->	abl
	----->	tabl(eenih)	----->	abl
	----->	t(aa)xi	----->	aaxi
	----->	t(aa)bbe	----->	aabbe
	----->	taama	----->	taama
y	----->	yitbiq(eeniik)	----->	itbiq
	----->	yabl(eenih)	----->	abl

	----->	abl(eh)	----->	abl
	----->	yaqab(eenimil)	----->	aqabe
	----->	yaab(eyyo)	----->	aab
	----->	y(aa)lloonu	----->	all
	----->	y(aa)baanam	----->	aabaanam
	----->	y(aa)xigeenim	----->	aaxigeenim
	----->	y(aa)baanam	----->	aabaanam
n	----->	nabl(eh)	----->	abl
	----->	nableh	----->	ableh
	----->	nak	----->	nak
	----->	n(aa)bbek	----->	aabbek

4.6.5 Rules for removing suffixes

4.6.5.1 Rule-step 1

```
def step1(stem):
    l=len(stem)
    if stem.endswith("taah"):
        return stem[:l-4]
    elif stem.endswith("tah"):
        return stem[:l-3]
    if stem.endswith("naah"):
        return stem[:l-4]
    elif stem.endswith("nah"):
        return stem[:l-3]
    elif stem.endswith("aah"):
        return stem[:l-3]
    elif stem.endswith("ah"):
        return stem[:l-2]
    elif stem.endswith("teeh"):
        return stem[:l-4]
    elif stem.endswith("teh"):
        return stem[:l-3]
    elif stem.endswith("neeh"):
        return stem[:l-4]
    elif stem.endswith("neh"):
        return stem[:l-3]
    elif stem.endswith("haak"):
        return stem[:l-4]
    elif stem.endswith("aak"):
        return stem[:l-3]
    elif stem.endswith("ak"):
        return stem[:l-2]
```

```
        elif stem.endswith("luk"):
            return stem[:l-3]
        elif stem.endswith("uuk"):
            return stem[:l-3]
        elif stem.endswith("uk"):
            return stem[:l-2]
        elif stem.endswith("eek"):
            return stem[:l-3]
        elif stem.endswith("ek"):
            return stem[:l-2]
        elif stem.endswith("ak"):
            return stem[:l-2]
        elif stem.endswith("h"):
            return stem[:l-1]
        elif stem.endswith("k"):
            return stem[:l-1]
        elif stem.endswith("l"):
            return stem[:l-1]
        elif stem.endswith("t"):
            return stem[:l-1]
        else:
    return stem.lower()
```

Figure 4.1: Python code for step1 1

Examples:

taah	----->	xiq**taah**	----->	xiq
	----->	cel**taah**	----->	cel
	----->	kasit**taah**	----->	kasit
	----->	ossit**taah**	----->	ossit
tah	----->	sug**tah**	----->	sug
	----->	digir**tah**	----->	digir
	----->	ciggil**tah**	----->	ciggil
	----->	xiq**tah**	----->	xiq
naah	----->	ab**naah**	----->	ab
		xag**naah**	----->	xag
	----->	sug**naah**	----->	sug
nah	----->	sug**nah**	----->	sug
	----->	xiq**aanah**	----->	xiqaa
	----->	sugaa**nah**	----->	sugaa
	----->	daffey**nah**	----->	daffy
aah	----->	cell**aah**	----->	cell
	----->	cedd**aah**	----->	cedd
	----->	qid**aah**	----->	qid

	----->	bartaan**aah**	----->	bartaan
ah	----->	sug**ah**	----->	sug
	----->	gex**ah**	----->	gex
	----->	nak**ah**	----->	nak
	----->	ciggiil**ah**	----->	ciggiil
	----->	wa**ah**	----->	wa
	----->	gex**ah**	----->	gex
teeh	----->	bic**teeh**	----->	bic
	----->	guf**teeh**	----->	guf
	----->	kal**teeh**	----->	kal
teh	----->	sug**teh**	----->	sug
	----->	door**teh**	----->	door
	----->	tatur**teh**	----->	tatur
	----->	buulum**teh**	----->	buulum
neeh	----->	taamit**neeh**	----->	taamit
	----->	ab**neeh**	----->	ab
neh	----->	fugaa**neh**	----->	fugaa
	----->	gex**eh**	----->	gex
	----->	sug**neh**	----->	sug
teek	----->	gey**teek**	----->	gey
	----->	kal**teek**	----->	kal
	----->	gac**teek**	----->	gac
	----->	xal**teek**	----->	xal
luk	----->	digga**luk**	----->	digga
	----->	siital**luk**	----->	siital
	----->	baxsa**luk**	----->	baxsa
	----->	maal**luk**	----->	maal
	----->	ittal**luk**	----->	ittal
uuk	----->	numuu**k**	----->	num
uk	----->	qilsa**uk**	----->	qilsa
	----->	numu**k**	----->	num
	----->	qunx**uk**	----->	qunx
	----->	diiroon**uk**	----->	diiroon
haak	----->	nammay**haak**	----->	nammay
	----->	sidoc**haak**	----->	sidoc
	----->	fereey**haak**	----->	fereey
aak	----->	nahar**aak**	----->	nahar
	----->	salaf**aak**	----->	salaf
	----->	geer**aak**	----->	geer
ak	----->	bartiyy**ak**	----->	bartiyy
	----->	ab**ak**	----->	ab
	----->	hirig**ak**	----->	hirig

	----->	qaar**ak**	----->	qaar
	----->	gac**ak**	----->	gac
	----->	af**ak**	----->	af
	----->	dadal**ak**	----->	dadal
eek	----->	qammis**eek**	----->	qammis
	----->	carar**eek**	----->	carar
	----->	ab**eek**	----->	ab
	----->	kal**eek**	----->	kal
ek	----->	gex**ek**	----->	gex
	----->	barac**lek**	----->	baraclek
	----->	ciggiil**ek**	----->	ciggiil
	----->	ab**ek**	----->	ab
	----->	agdaab**ek**	----->	agdaab
h	----->	fooca**h**	----->	fooca
	----->	iro**h**	----->	iro
k		irok		iro
		dimisi**k**		dimis
l	----->	foc**al**	----->	faaca
	----->	iro**l**	----->	iro
	----->	wokke**l**	----->	
t	----->	fooca**t**	----->	fooca
	----->	wokke**t**	----->	wokke

4.6.5.2 Rule-step 2

```
def step2( stem ):
    l=len(stem)
    if stem.endswith("eemi"):
        return stem[:l-4]
    elif stem.endswith("eenimi") and (countVowel(stem[:l-0]) > 1):
        return stem[:l-6]
    elif stem.endswith("eeni") and (countVowel(stem[:l-0]) > 0):
        return stem[:l-4]
    elif stem.endswith("eenii") and (countVowel(stem[:l-0]) > 0):
        return stem[:l-5]
    elif stem.endswith("innay") and (countVowel(stem[:l-0]) > 1):
        return stem[:l-5]
    elif stem.endswith("inniyoy") and (countVowel(stem[:l-0]) > 0):
        return stem[:l-7]
    elif stem.endswith("innitoy") and (countVowel(stem[:l-0]) > 0):
        return stem[:l-7]
    elif stem.endswith("inninoy") and (countVowel(stem[:l-0]) > 1):
        return stem[:l-7]
    elif stem.endswith("ittoonuy") and (countVowel(stem[:l-0]) > 0):
        return stem[:l-8]
    elif stem.endswith("innitoonuy") and (countVowel(stem[:l-0]) > 0):
        return stem[:l-10]
    else:
        return stem.lower()
```

Figure 4.1: Python code for step2 1

Example:

eemi	----->	tamaat**eemi**(l)	----->	tamaat
	----->	tamq**eemi**(l)	----->	tamq
	----->	yemeet**eemi**(l)	----->	yemeet
	----->	nanxuq**eemi**	----->	nanxuq
eenimi	----->	yaqab**eenimi**(l)	----->	yaqab
	----->	yakm**eenimi**	----->	yakm
	----->	yaaxig**eenimi**(l)	----->	yaaxig
	----->	bart**eenimi**(h)	----->	bart
	----->	ab**eenimi**(k)	----->	ab
	----->	yasmit**eenim**i	----->	yasmit
	----->	yard**eenimi**	----->	yard
	----->	geenimi(k)	----->	geenimik

eeni	----->	taamit**eeni**(k)	----->	taamit
	----->	bart**eeni**(k)	----->	bart
	----->	bart**eeni**(t)	----->	bart
	----->	gex**eeni**(h)	----->	
	----->	yasmit**eeni**(k)	----->	yasmit
eenii	----->	bart**eenii**(k)	----->	bart
	----->	kaqt**eenii**(k)	----->	kaqt
	----->	ab**eenii**(k)	----->	ab
innay	----->	gex**innay**	----->	gex
inniyoy	----->	gex**inniyoy**	----->	gex
innitoy	----->	gex**innitoy**	----->	gex
	----->	amaat**innitoy**	----->	amaat
inninoy	----->	amaat**inninoy**	----->	amaat
	----->	gex**inninoy**	----->	gex
	----->	matr**inninoy**	----->	matr
	----->	an**inninoy**	----->	an
innoonuy	----->	gex**innoonuy**	----->	gex
	----->	amaat**innoonuy**	----->	amaat
	----->	maab**innoonuy**	----->	yabl
innitoonuy	----->	gex**innitoonuy**	----->	gex
	----->	amaat**innitoonuy**	----->	amaat
ittoonuy	----->	gex**ittoonuy**	----->	gex

4.6.5.3 Rule-step 3

```
def step3 ( stem ):

    l=len(stem)
    if stem.endswith("aama") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-4]
    elif stem.endswith("taanama") and (endsWithCVC(stem) == 1) and
(countVowel(stem[:l-1]) > 0):
        return stem[:l-7]
    elif stem.endswith("aanama") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-6]
    elif stem.endswith("aanam") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("aana") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-4]
    else:
        return stem.lower()
```

Figure 4.4: Python code for step3 1

Example:

aama	----->	ab**aama**(h)	----->	ab
	----->	reed**aama**(k)	----->	reed
	----->	gir**aama**(l)	----->	gir
	----->	wagit**aama**(l)	----->	wagit
	----->	xalsit**aama**l	----->	xalsit
	----->	ceel**aama**(h)	----->	ceel
	----->	xiq**aama**	----->	xiq
	----->	taama(l)	----->	taama
taanama	----->	sugtaanama(l)	----->	sug
	----->	sugt**aana**(h)	----->	sug
	----->	baraabart**aanama**	----->	baraabar
aanama	----->	sug**aanama**(l)	----->	sug
	----->	gabbaaq**aanama**	----->	gabbaaq
	----->	gex**aanam**a	----->	gex
	----->	kaqit**aanam**	----->	kaqit
	----->	fax**aanama**(h)	----->	fax
aanam	----->	gex**aanam**	----->	gex
	----->	nak**aanam**	----->	nak
	----->	sug**aanam**	----->	sug
	----->	ab**aanam**	----->	ab

-----> gey**aanam**(ih) -----> gey

4.6.5.4 Rule-step 4

```
def step4 ( stem ):
    l=len(stem)
    if stem.endswith("hayto") and (countVowel(stem[:l-1]) > 0):
      return stem[:l-5]
    elif stem.endswith("haytu") and (countVowel(stem[:l-1]) > 0):
      return stem[:l-5]
    elif stem.endswith("to") and (countVowel(stem[:l-1]) > 0):
      return stem[:l-2]
    elif stem.endswith("tu") and (countVowel(stem[:l-1]) > 0):
      return stem[:l-2]
    else:
      return stem.lower()
```

Figure 4.4: Python code for step4

Example:

hayto	----->	ferey**hayto**(h)	----->	ferey
	----->	nammay**hayto**(h)	----->	nammay
	----->	inki**hayto**(h)	----->	inki
	----->	sedoc**hayto**(h)	----->	sedoc
haytu	----->	taban**haytu**	----->	taban
	----->	konooy**hyatu**	----->	konooy
	----->	fereey**haytu**	----->	fereey
	----->	nammay**haytu**	----->	nammay
to	----->	gey**to**(h)	----->	gey
	----->	bool**to**(h)	----->	bool
	----->	xin**to**(h)	----->	xintoh
	----->	barit**to**(h)	----->	barittoh
	----->	ab**to**(h)	----->	ab
	----->	goran**to**(h)	----->	goran
tu	----->	dadal**tu**(h)	----->	dadal
	----->	gibba**tu**(h)	----->	gibba
	----->	bic**tu**(h)	----->	bic
	----->	kibbim**tu**(h)	----->	kibbim
	----->	sol**tu**(h)	----->	sol
	----->	tatur**tu**(h)	----->	tatur
	----->	xag**tu**(h)	----->	xag

4.6.5.5 Rule-step 5

```
def step5 ( stem ):
    l=len(stem)
    if (stem.endswith("i")) and (countVowel(stem[:l-1]) > 1) and (stem[:l-2].endswith("oo") or
stem[:l-2].endswith("aa") or stem[:l-3].endswith("xx")):
        return stem[:l-4] + "a"
    elif (stem.endswith("i")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("ii"):
        return stem[:l-3] + stem[-2]
    elif (stem.endswith("a")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("ee") or
stem[:l-2].endswith("oo"):
        return stem[:l-3]
    elif (stem.endswith("a")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("ii"):
        return stem[:l-3]+ stem[-2]
    elif (stem.endswith("itte")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-4]
    elif (stem.endswith("a")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("uu"):
        return stem[:l-3] + stem[-2]
    elif (stem.endswith("u")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("uu"):
        return stem[:l-3] + stem[-2]
    elif (stem.endswith("wa")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-2]
    elif (stem.endswith("yta")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif (stem.endswith("ta")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-2]
    elif (stem.endswith("la")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    elif (stem.endswith("yto")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif (stem.endswith("to")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-2]
    elif (stem.endswith("ytu")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif (stem.endswith("two")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif stem.endswith("li"):
        return stem[:l-2]
    else:
        return stem.lower()
```

Figure 4.5: Python code for step5 1

Example:

ooti	----->	a	ast**ooti**	----->	asta
			dab**oobi**		daba
			gab**oobi**		gaba
			laf**oofi**		lafa
			mak**ooka**		maka
aati	----->		but**aati**	----->	buta
			bux**aaxi**		buxa
			koom**aamí**		kooma
			but**aati**		bita
eera	----->	e	buqr**eera**	----->	buqre
			qal**eela**		qale
iina	----->	i	ayn**iina**	----->	ayni
			ayt**iita**		ayti
			add**iida**		addiida
iiri		i+second letter	kimb**iiri**		kimbir
iida		i+second letter	rag**iida**		ragid
			qing**iiri**		qingir
			caag**iida**		caagid
			xinb**iiqi**		xinbiq
ooqa	----->		arq**ooqa**	----->	arqo
			am**ooma**		amo
uubu	----->	b	gul**uubu**	----->	gulub
uuqu		q	duq**uura**		duqur
eela	----->	e	qal**eela**	----->	qale
la			gita**la**		gita
wa	----->		al**wa**	----->	al
yta	----->		boco**yta**	----->	boco
			cado**yta**		cado
yto	----->		garra**yto**	----->	garra
			garra**yto**		garra
ytu	----->		kalla**ytu**	----->	kalla
			gibya**ytu**		gibya
two	----->		barseeni**two**	----->	kalla

4.6.5.6 Rule-step 6

```
def step6 ( stem ):
 l=len(stem)
if (stem.endswith("y")) and (countVowel(stem[:l-1]) > 0) and (stem[:l-1].endswith("aa") or stem[:l-1].endswith("ii")or stem[:l-1].endswith("uu")):
        return stem[:l-3]
elif (stem.endswith("y")) and (countVowel(stem[:l-1]) > 0) and l - 3 > 1 and (stem[:l-1].endswith("ee") or stem[:l-1].endswith("oo")):
        return stem[:l-2]
```

```
else:
        return stem.lower()
```

Figure 4.6: Python code for step6 1

Examolpe:

Camadaay, Qafaraay, Gileey, Cuseeniiy, Kabeeliiy, maaqooy , Cadooy, Cooxuuy , Cusuluuy

4.6.5.7 Rule-step 7

```
def step7( stem ):
    l=len(stem)
    if (stem.endswith("siisiyya")) and (countVowel(stem[:l-5]) > 0):
        return stem[:l-8]
    elif (stem.endswith("isiyya")) and (countVowel(stem[:l-2]) > 0 and l-7 > 0 ):
        return stem[:l-6]
    elif (stem.endswith("itiyya")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-6]
    elif (stem.endswith("siyyi")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("iyya")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("iyyi")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("iyy")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("sis")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("isiis")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("oola")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    else:
        return stem.lower()
```

Figure 4.7: Python code for step7 1

Example:

Siisiyya: bahsiisiyya, Celsiisiyya, Wagsiisiyya and Xinsiisiyya

Isiyya: koobaahisiyyal, agiirisiyya, Xiinisiyya, miraacisiyya and ciggiilisiyya

itiyya :adoobitiyyal Sirkitiyyaay, adoobitiyyal, falitiyyat and Baritiyyaa

isiyyi: Sekkaacisiyyi, barsiyyi, barsiyyih, and baxsiyyi

iyya: Kuriyya, Aatukiyya, Kaliyya, kaawiyya and adoobitiyyal

iyyi: Aalliyyi, kemsimiyyi, ayyaaqiyyi, Wadiyyi and gexiyyi

iyy: afqiyyah, dariyyuk, keexiyyal daabisiyyih and Sirkitiyyaay

sis: kataysis, bahsis, celsis and kassis

issis:madqisiisak, qedmisiisak,

oola: weeloola, xongoloola, and ceeloola

4.6.5.8 Rule-step 8

```
def step8( stem ):
    l=len(stem)
    if (stem.endswith("tam")) and (countVowel(stem[:l-7]) > 0):
        return stem[:l-7]
    elif (stem.endswith("tan")) and (countVowel(stem[:l-6]) > 0):
        return stem[:l-6]
    elif (stem.endswith("ta")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-6]
    elif (stem.endswith("nam")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("am")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-3]
    elif (stem.endswith("an")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-3]
    else:
        return stem.lower()
```

Figure 4.8: Python code for step8 1

Example:

tam	----->	nak**tam**	----->	nak
	----->	**gactam**	----->	**gac**
	----->	bey**tam**	----->	bey
	----->	kaf**tam**	----->	kaf
	----->	taamit**tam**	----->	taamit
tan	----->	nak**tan**	----->	nak
	----->	nak**tan**	----->	nak
	----->	ab**tan**	----->	ab
	----->	rub**tan**	----->	rub
	----->	sug**tan**	----->	sug
	----->	ma**tan**	----->	matan
ta	----->	ciggil**ta**	----->	ciggil
	----->	bah**ta**	----->	bah
	----->	deqsit**ta**	----->	deqsit
	----->	biyaakit**ta**	----->	biyaakit
	----->	ab**ta**	----->	ab
nam	----->	nak**nam**	----->	nak
	----->	gen**nam**	----->	gen
	----->	sugn**am**	----->	sug

	----->	gey**nam**	----->	gey
	----->	way**nam**	----->	way
am	----->	nak**am**	----->	nak
	----->	**sugam**	----->	sug
	----->	rab**am**	----->	rab
	----->	ab**am**	----->	ab
an	----->	nak**an**	----->	nak
	----->	sug**an**	----->	sug
	----->	tab**an**	----->	ab
	----->	fax**an**	----->	fax
	----->	seec**an**	----->	seec
	----->	orbiss**an**	----->	orbiss

4.6.5.9 Rule-step 9

```
def step9( stem ):
    l=len(stem)
    if (stem.endswith("eloonum")) and (countVowel(stem[:l-4]) > 0):
        return stem[:l-7]
    elif (stem.endswith("elem")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-6]
    elif (stem.endswith("elon")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("ele")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("ennom")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("enno")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("oonu")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-4]
    elif (stem.endswith("ettonum")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-7]
    elif (stem.endswith("ettom")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("etton")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("etto")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("eyyom")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("eyyo")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    else:
        return stem.lower()
```

Figure 4.9: Python code for step9 1

Example:

eloonum	----->	abl**eloonum**	----->	abl
	----->	aall**eloonum**	----->	aall
	----->	amaat**eloonum**	----->	amaat
	----->	abl**eloonum**	----->	
	----->	ab**eloonum**	----->	
elem	----->	abl**elem**	----->	abl
	----->	nak**elem**	----->	nak
	----->	gex**elem**	----->	gex
	----->	amaat**elem**	----->	amaat
	----->	aall**elem**	----->	aall
elon	----->	gex**elon**	----->	gex
	----->	dacris**elon**	----->	dacris
	----->	sug**elon**	----->	sug
	----->	amaat**elon**	----->	amaat
	----->	abl**elon**	----->	abl
ele	----->	gex**ele**	----->	gex
	----->	abl**ele**	----->	abl
	----->	gex**ele**	----->	gex
	----->	amaat**ele**	----->	amaat
	----->	yaab**ele**	----->	yaab
ennom	----->	abl**ennom**	----->	abl
	----->	aall**ennom**	----->	aall
	----->	sug**ennom**	----->	sug
	----->	amaat**ennom**	----->	amaat
enno	----->	abl**enno**	----->	abl
	----->	amaat**enno**	----->	amaat
	----->	gex**enno**	----->	gex
	----->	guf**enno**	----->	guf
	----->	wagt**enno**	----->	wagt
oonu	----->	yabl**oonu**	----->	yabl
	----->	yamaat**oonu**	----->	yamaat
	----->	yaall**oonu**	----->	yaall
	----->	taall**oonu**	----->	taall
	----->	sug**oonu**	----->	sug

ettonum	----->	gex**ettonum**	----->	gex
	----->	abl**ettonum**	----->	abl
	----->	gex**ettonum**	----->	gex
ettom	----->	gex**ettom**	----->	gex
	----->	aall**ettom**	----->	aall
	----->	sug**ettom**	----->	sug
	----->	amaat**ettom**	----->	amaat
	----->	akk**ettom**	----->	akk
etton	----->	gex**etton**	----->	gex
	----->	sug**etton**	----->	sug
	----->	amaat**etton**	----->	amaat
	----->	abl**etton**	----->	abl
	----->	aall**etton**	----->	aall
eyyom	----->	gex**eyyom**	----->	gex
	----->	**sugeyyom**	----->	sug
	----->	**amaateyyom**	----->	**amaat**
	----->	**ableyyom**	----->	**abl**
	----->	**aalleyyom**	----->	**aall**
etto	----->	gex**etto**	----->	gex
		aall**etto**		aall
		sug**etto**		sug
		amaat**etto**		amaat
		abl**etto**		abl
eyyo		gex**eyyo**		gex
		abl**eyyo**		abl
		yaab**eyyo**		yaab
		ab**eyyo**		ab
		amaat**eyyo**		amaat

4.6.5.10 Rule-step 10

```
def step10( stem ):
    l=len(stem)
    if (stem.endswith("le")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    elif (stem.endswith("lu")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    else:
        return stem.lower()
```

Figure 4.10: Python code for step10 1

Example:

le	----->	kulsa**le**	----->	kulsa
	----->	rooci**le**(h)	----->	rooci
	----->	sitallak**le**(h)	----->	sitallak
	----->	baxaabaxsa**le**(h)	----->	baxaabaxsa
	----->	xiq**le**	----->	xiq
	----->	assakoxxa**le**	----->	assakoxxa
	----->	qaday**le**	----->	qaday
	----->	qitta**le**	----->	qitta
	----->	galaq**le**	----->	galaq
	----->	cinna**le**	----->	cinna
	----->	qitta**le**	----->	qitta
lu	----->	qaday**lu**	----->	qaday
	----->	qitta**lu**	----->	qittalu
	----->	galaq**lu**	----->	galaq
	----->	cinna**lu**	----->	cinna
	----->	taddagal**lu**	----->	taddagal
	----->	kalal**lu**	----->	kalal

4.6.5.11 Rule-step 11

```
def step11( stem ):
    l=len(stem)
    if endsWithDouble(stem)==1 and (countVowel(stem[:l-1]) > 0):
        return stem[:l-1]
    else:
        return stem.lower()
```

Figure 4.11: Python code for step11 1

Example:

aa	----->	a	seehad**aa**	----->	seehada
dd	----->		raddii(k)	----->	raddi
ee	----->	e	mood**ee**	----->	moode
kk	----->		a**kk**(ele)		ak
ii	----->	i	dafatir**ii**	----->	dafatiri
ll	----->		yaa**ll**(oonu)		yaal
oo	----->	o	ceel**oo**	----->	ceelo

4.6.6 Prefixes removal

4.6.6.1 Rule-step 12

```
def step11( stem ): #remove prefix
    l=len(stem)
    if stem.startswith('ma') or stem.endswith('maa'):
        stem1=stem.replace('ma','',1)
        return stem
    elif stem.startswith('mee'):
        stem1=stem.replace('me','',1)
        return stem
    elif stem.startswith('mii'):
        stem1=stem.replace('mi','',1)
        return stem
    elif stem.startswith('muu'):
        stem1=stem.replace('mu','',1)
        return stem
    else:
        return stem.lower()
```

Figure 4.12: Python code for step12

Example:

ma	----->		**ma**soolinna	----->	soolinna
	----->		**ma**naadiga	----->	naadiga
maa	----->		**maa**binnoonuy	----->	ab
muu	----->	u	**muu**gutta	----->	ugutta
mee	----->	e	**mee**sserinno	----->	esserinno
mii	----->	i	**mii**laalisa	----->	ilaalisa

4.6.6.2 Rule-step 13

```
def step12 ( stem ):
    l=len(stem)
    if stem.startswith("aa"):
        stem=stem.replace('a','',1)
        return stem
    elif stem.startswith("t"):
        stem=stem.replace('t','',1)
        return stem
    elif stem.startswith("ee"):
        stem=stem.replace('e','',1)
        return stem
    elif stem.startswith("ii"):
```

```
            stem=stem.replace('i','',1)
            return stem
        elif stem.startswith("oo"):
            stem=stem.replace('o','',1)
            return stem
        elif stem.startswith("uu"):
            stem=stem.replace('u','',1)
            return stem
        else:
    return stem.lower()
```

Figure 4.13: Python code for step13 1

Example:

aa	----->	a	**aa**gar(uk)	----->	agar
		a	**aa**rr(iyya)		arr
		a	(y)**aa**b(eyyo)		ab
		a	(y)**aa**l(l)(oonu)		al

4.6.7 The Proposed Stemming Algorithm

1. open stop-word list file
2. open files(files in corpus)
Read a word from the file until match occurs or End of File reached
Normalize the word
Tokenize the word
3. Read the next word to be stemmed
IF word exists in the stop word list
Got to 7
4. If word matches one of the rules
Get the word and count word length
Remove the suffix and do the necessary adjustments
Go back to 4
Else
6. Return the word and record it in stem dictionary
7. IF end of file not reached
Go to 3
Else
Stop processing

Table 3.5: the proposed stemming algorit 1

Chapter five

5 Design and Experimentation

The aim of this study is to design a retrieval system that searches relevant documents from Afaraf text repositories that satisfy the information needs of users.

The study is in experimentally test stage. The performance of IR system using corpus collected from elementary school text books, grade 1 up to grade 4, Afaraf grammar, university module 2006 and Qusba-Maaca magazine and other resources sources in Afaraf available on the web.

5.1 Index construction

The prototype system of Afaraf text retrieval developed has indexing and searching components. Indexing involves tokenizing, normalizing, stop word removal, stemming, and creating inverted file structure which includes a vocabulary file and a posting file.

5.1.1 Tokenization and Normalization

The code 4.1 is fragment code that tokenizes and normalizes the terms in the document.

```
Specialcharacters: = “ . , ? / | \ @* =^&( ) +_; : “ ‘ ! # $ % [ ] { }< > - 1 2 3 4 5 6 7 8 9 0”
    def tokenize(word):
    terms = word.lower( ).split( )
    return [term.strip(Specialcharacters) for term in terms]
```

Code 4.1 Tokenization and normalization 1

The code first splits document based on space between each words, then convert every words in document in to lower case if it was not in lower case primarily. The documents in lower case are checked as if it is not punctuation mark or number. Then normalized and tokenized document were returned for next process.

5.1.2 Stop word Removal

The terms are selected for indexing process and they are not part of stop list. The list of the stop-words primarily identified and listed in text file called *stopwordlist* by the researcher. The code 4.2 is fragment code that removes stop word.

```
stopwordfile = open('stopword.txt','r')
os.chdir('corpus')
stopwordlist= stopwordfile.read( )
stopwordfile.close( )
for i in word:
        if i not in stopwordlist:# Stop word list removal
                stemmed=thestemmer(i)
```

Code 4.2 Stop word removal 1

As mentioned above Afaraf stopwords listed and saved in text file named *stopwordlist.txt*. Some of Afaraf stop words are listed in above table. First the algorithm reads the files and saves it on variable, and then the normalized token is checked to be different from the terms in the stop word.

5.1.3 Stemming

If terms are not stop word then process forwarded to the stemmer function. The code 4.3 is fragmenting code that stemming the terms in the document.

```
def thestemmer(word):
   l=len(word)
   if word.startswith("t") and (word.endswith not in sufcheker):
      word = word.replace('t','',1)
   elif word.startswith("y") and (word.endswith not in sufcheker):
     word = word.replace('y','',1)
   elif word.startswith("n") and (word.endswith not in sufcheker):
     word = word.replace('n','',1)
```

```
            return word
def step1(stem):
    l=len(stem)
    if stem.endswith("taah"):
        return stem[:l-4]
    elif stem.endswith("tah"):
        return stem[:l-3]
    if stem.endswith("naah"):
        return stem[:l-4]
    else:
         return stem.lower()
def step2( stem ):
    l=len(stem)
    if stem.endswith("eemi"):
        return stem[:l-4]
    elif stem.endswith("eenimi") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-6]
    elif stem.endswith("eeni") and (countVowel(stem[:l-0]) > 0):
        return stem[:l-4]
    elif stem.endswith("eenii") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("innay") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
else:
        return stem.lower()
```

Code 4. 3 Stemming 1

The stemming code is creates stemmed words depending rule based stemming algorithm. The algorithm checks first affixes and then measure the vowels in token, them implements rules of stripping inflections of words.

The next part is creating inverted file structure. The inverted file has two separates files vocabulary and posting file; the vocabulary file contains Terms, Doc frequency and Collection frequency and

illustrated in below table 4.1 and posting documents ID, Term frequency and trems location also illustrated in the Table 4.2. That depicts structure of inverted file (vocabulary file and posting file).

Vocabulary files		
Terms	***Doc frequency***	***Collection frequency***
xaga	2	2
sarisa	1	1
Lifiiqa	1	1

Table: 4.1 Vocabulry 1

Posting file		
DocId	***Term frequency***	***Location***
Affile1	2	26,232
Affile4	1	15
Affile5	1	89
Affile21	3	34,67,81
Affile14	6	1 32, 56,123
Affile53	1	327

Table: 4.2 Posting 1

5.2 Searching and VSM Construction

The prototype of searching sub system component uses VSM model. Information a users‘need is represented as query... Here is in this component similarity measurement and ranking is done. From the actual code used in the prototype some of the code is posted for explanation of the system.

The file of vocabulary holds each unique non-stop word terms in the documents with its document frequency and collection frequency. The document frequency and collection frequency are referenced to posting file. The Posting file holds term with their term frequency ***tf***, positions in the documents and name of each document holding the term. So the search algorithm brings together these things in order to calculate similarity and rank documents. The inverted file index has computed weight terms in the document and weight for the query is also has computed.

```
def tfidf():
    fN=filecounter()
    vf=open('vocabulary.txt','r')
    voc=vf.readlines()
    pf=open('post.txt','r')
    pos=pf.readlines()
    dic={}
    for i in voc:
        x=i.split()
        df=x[1]
        term1=x[0]
        stes=''
        for ii in pos:
            xx=ii.split()
            term2=xx[0]
            doc=xx[1]
            tf=xx[2]
            stes=term2+' '+doc
            if term1==term2:
                doc=xx[1]
                wf=wordfreq(doc)
                tf1=float(tf)
                tf1=tf1/wf
                dfi=float(df)
                df1=fN/dfi
                idf=math.log(df1,2)
```

```
            tw=tf1*idf
            stes=doc+' '+term2
            dic[stes]=tw
    return dic
```

Code 4. 8 vocabulary and posting 1

5.2.1 Query relevance judgement

Summary of Query relevance judgement for selected 8 queries are listed In Table 4.3. According to that the relevance judgement each document is identified to be as a relevant or non-relevant toward each query terms.

Queries No.	Queries terms	Relevant	Non relevant
Q1	baritto buxa qafarafih Bartiyya kee Barsiyyi Cogda	135	165
Q2	maaqo xagarah Qaafiyat tace	104	196
Q3	Sultaan Qalimirackee Sultaan Macammad Acawa aydaadu	39	261
Q4	Qafara rakaakayih doolatak xiinissoo kee saay biiro ummatta konferensi	115	185
Q5	Kas takke Maxcoo kee Qadar Maxcooca	63	257
Q6	Qafar qaadal Digib bux madaqa digaala aalle wayta abtoota	57	243
Q7	Baadal maddur angaaraw siyaasa maca ceelah?	32	268
Q8	Gino gadda nii dariifal maca ceelah?	40	260

Table 4: 3 Query relevance judgement 1

5.2.2 Experimentation

5.2.2.1 Stemer evaluation

The stemmer developed for removing prefixes and suffixes. The stemming is performed by applying the longest possible (if any) that the stem produced if contains at least one vowel. The stemmer algorithm transformed words to correct stem words, or either changing to improper stem, or leaving unchanged. Numbers of words are used 4982, numbers of unique words before stemming are 2908, and number of unique words registered after stemming are 2053. The table below summarized the results of stemmer evaluation.

Number of words	Stemmed correctly (%)	Errors	
		Over-stemming (%)	Under-stemming (%)
2908	1909 (51.34 %)	131 (4.50%)	868 (29.85%)

Stemming table 4.2 1

The stemmer follow affixes removal techniques, it registered 65.65% for correct stem, it registered 4.50% for over-stemming and 29.85 % only for under-stemming error, which shows large errors are occurred in over stemming, and it take place because of word formation in Afaraf morphology, which means one root word can generate more then fifty possible words. The second problem of error stems are Xagissa characters, the letter which hold one of the characters are outomatically remover from the word, for example: **Leê** to Le, and **Qalé** to Qal etc.

5.2.2.2 System evaluation

In system experiment, eight queries are used and the relevance judgments are prepared to construct document query matrix that shows all relevant documents for each test query prepared as shown in appendix III. The performance of the prototype system is evaluated before and after stemming of text. Each test query is measured by precision and recall. A result of each query before stemming is presented in term of precision and recall in Table 4.4, and the result of each query after stemming is presented in term of precision and recall in Table 4.5, that represents the performance registered by the system.

Queries No.	Queries terms	Precision	Recall
Q1	baritto buxa qafarafih Bartiyya kee Barsiyyi Cogda	0.833	0.111
Q2	maaqo xagarah Qaafiyat tace	0.90	0.278
Q3	Sultaan Qalimirac kee Sultaan Macammad Acawa aydaadu	1	0.277
Q4	Qafar rakaakayih doolatak xiinissoo kee saay biiro ummatta konferensi	0.8	0.0347
Q5	Kas takke Maxcoo kee Qadar Maxcooca	1	0,0476
Q6	Qafar qaadal Digib bux madaqa digaala aalle wayta abtoota	1	0.140
Q7	Baadal maddur angaaraw siyaasa maca ceelah?	1	O.210
Q8	Gino gadda nii dariifal maca ceelah?	1	0.25
Average		**0.94**	**0.168**

Table 4.4 Detailed result of performance 1

As it is shown in Table 4.4 the obtained result is 0.94(94%) precision and 0.168 (17%) recall. The performance result before stemming for precision show greate when compare with recall result, but the recall result show that set of retrieved relevant documents are less.

Queries No.	Queries terms	Precision	Recall
Q1	baritto buxa qafarafih Bartiyya kee Barsiyyi Cogda	0.53	0.170
Q2	maaqo xagarah Qaafiyat tace	0.894	0.567
Q3	Sultaan Qalimirac kee Sultaan Macammad Acawa aydaadu	0.928	0.333
Q4	Qafar rakaakayih doolatak xiinissoo kee saay biiro ummatta konferensi	0.625	0.0869
Q5	Kas takke Maxcoo kee Qadar Maxcooca	1	0.0476

Q6	Qafar qaadal Digib bux madaqa digaala aalle wayta abtoota	0.916	0.210
Q7	Baadal maddur angaaraw siyaasa maca ceelah?	0.5	0.25
Q8	Gino gadda nii dariifal maca ceelah?	0.888	0.2
Average		**0.785**	**0.2330**

Table 4.5 Detailed result of performance 2

The table 4.5 show performance result after stemming and the obtained result is 0.785(79%) for precision and it decreased by 0.155 but the recall is 0.233 (23%) and increased by 0.065. The recall result lower, because of morphological variation in word form and the way of Afaraf writing system, afaraf can use xer yangayyi or yuxux yangayyi at any place in the words.

For example: barteeenit # barteenit, baddaafa # baddafa, Xiiniyya # Xiniyya, barritto # baritto, baxqis # baxxaqis, dacariso # dacriso and ittoobiy # ityopiya

Chapter Six

Conclusion and Recommendations

6 Conclusion

Afaraf is morphologically very rich; typically, one word can drive 50 or more words from one root. For example the word “ab” = “do”: *ab, aba, abaak, abaan, abaana, abaanah, abaanak, abaanam, abaanama, abaanamfaxximta, abak, abam, aban, abanama, abe, abee, abeenah, abeenat, abeenih, abeeniik, abeh, abem, aben, abenno, abenta, abbey, abeyyo, abi, abina, abinak, abinal, abini, abinih, abinnal, abit, abiteyyo, abna, abnak, abnam, abnu, aboonay, aboonu, aboonuh, absiis, absiisaanama, abta, abtay, abteh, abto, abtok, abtoota, abut and abtuh*. The language uses affixes technique to create variant word, aften it appearce in verbs to create another variant of the verbs.

The rule based stemmer developed based on grammar and dictionary of Afaraf, included Numbers(singular and plural), Personal pronoun, Adjectives, Adverbs, Verbal-noun, Strong and weak verb, Indefinite pronoun , Conditional and subjunctive mood, Linkage and Afaraf tenses to remove suffixes and prefixes from the word and produce stem word. But the stemmer not handle efficiently word variants do to word formation in Afaraf morphology.

The verbs patterns are different in present and past tenses like “able, tamaateh” in present tense and “tuble, temeete” in past tense. In addition, some of words had different stem because of long vowels (deryangayyi) .e.g. qasiirih or qasirih are stemmed two different stems, “qasiir “and “qasir” etc. And not all words form root word by removing ah, nah e.g. massah, katassah, barseynah, gitah .And in nouns a vowel preceded postposition and it can’t be removed because it affect root word e.g. buxah → bux, ibah → ib, marah → mar, dorak →dor, and the same for adverbs abaluk → aba, aylaaluk → aylaal ,Booluk → boo, axcuk → axc.

The indexing components developed enable the arrangements of index terms to permit searching and to access to desired information from document collection as per users query. Afaraf text

retrieval prototype developed using VSM model with indexing and searching modules. The indexing part of the work involves tokenizing, normalization, stop word removal, stemming and it involves in searching components with addition of similarity measurement and ranking.

According to the experimentation, the stemmer result shows that the accuracy is 65.65 %, 4.50% for over-stemming error and 29.85% for under-stemming error and based on the 2908 sample data, large errors were occurred in over stemming, because of word formation in language. The system reflected effective performance registered 0.785 precision and 0.233 recall which is request to design Afaraf text information retrieval system that could searches within large corpus. Additionally synonymy and polysemy nature of Afaraf language greatly affect the retrieval performance.

7 Recommendations and future directions

Afaraf text retrieval system is on beginning level and it is open area for more future studies. It needs collaboration of researcher and funding organization. The researcher likes to recommends the followings to the different stakeholders.

- Rule based stemmer is conducted for the first time for Afaraf text retrieval system. So further study of stemmer is needed to improve retrieval effectiveness of Afaraf retrieval system,
- This study is conducted for the first time by using VSM. So the further study is needed to figure out best model that works for Afaraf retrieval system,
- Test and experimentation of Afaraf Information retrieval needs standard corpus. Standard corpus development is requested as future work,
- To in enhancing the performance of Afaraf IR system the researcher recommends integrating mechanisms of controlling synonyms and polysemy terms and query expansion in the VSM model to enhance precision and recall of the system.
- Cross language IR for sharing information written with other local language, like, Amharic, Tigrinya, Afan Oromo …etc

8 Reference:

[1] Fisnik Dalipi, "Analyzing Some Recent Architecture For Semantic Searching At Ir And Qa System That Use Ontologies," Department Of It, Faculty Of Math-Natural Science Telovo State University, Department Of It, Faculty Of Natural Science University Of Tirana.

[2] Prabhakar Raghavan, Hinrich Schütze Christopher D. Manning, An Introduction To Information Retrieval, Online Edition Ed.: Cambridge University Press, April 1, 2009.

[3] Amit Singhal, "Modern Information Retrieval: A Brief Overview," Bulletin Of The Ieee Computer Society Technical Committee On Data Engineering, P. 9, 2001.

[4] Gerard Salton And Harman Donna, "Information Retrieval," In Encyclopedia Encyclopedia Of Computer Science 4th. Chichester, Uk: John Wiley And Sons Ltd, 2003, Pp. 858-863.

[5] Riyad Al-Shalabi2, Ghassan Kanaan2, And Alaa Al-Nobani Mohamad Ababneh, "Building An Effective Rule-Based Light Stemmer For Arabic Language To Improve Search Effectiveness," The International Arab Journal Of Information Technology, Vol. 9, 4, July 2012.

[6] Stephen Marlett, an Introduction Phonological Analysis, Stephen A. Marlett, Ed.: Summer Institute Of Linguistics, 2001.

[7] J A Director Of Alsec Redo, Afaraf (Afar Language) & Its Dictionary Preparation, Anrs, Ed. Samera, Afar National Regional State Of Ethiopia: Alsec.

[8] Samia Zekaria Director General Central Statistical Agency, "Summary And Statistical Report Of The 2007 Population And Housing Census," Federal Democratic Republic Of Ethiopia Population Census Commission, Addis Ababa, United Nations Population Fund(Unfpa) 2008.

[9] Sandra Lee Fulmer, "Parallelism And Planes In Optimality Theory:Evidence From Afar,"

Department Of Linguistics, The University Of Arizona, Phd Thesis 1997.

[10] Gezehagn Gutema Eggi, "Afaan Oromo Text Retrieval System," Department Of Information Science Addis Ababa University, Addis Ababa, Degree Of Master Thesis 2012.

[11] Alvin Toffler, Future Shock The Third Wave, Bantam Edition Ed., Alvin Toffler., Ed. United States And Canada: Alvin Toffler, 1980.

[12] Tessema Mindaye And Solomon Atnafu, "Design And Implementation Of Amharic Search Engine," In 5th International Conference On Signal Image Technology And Internet Based System, 2009.

[13] Tewodros Hailemeskel Gebermariam, "Amharic Text Retrieval : An Experiment Using Latent Semantic Indexing (Lsi) With Singular Value Decomposition (Svd)," School Of Information Science, Addis Ababa University, Addis Ababa, Msc Thesis 2003.

[14] Atalay Luel, "A Probabilistic Information Retrieval System For Tigrinya," School Of Information Science, Addis Ababa University, Addis Ababa, Msc Thesis 2014.

[15] M. E. Corporate Santa Mon~Ca, Cahfornza Maron, "Automatic Indexing: An Experimental Inquiry," Pp. 404-417, January 1961.

[16] Online. Available. (2015, January) Python Software Foundation. [Online]. Http://Www.Python.Org/About/

[17] Andrew Spencer And Arnold Zwicky, The Handbook Of Morphology., 2001.

[18] M. Porter, "An Algorithm For Suxffix Stripping Program," Vol, Vol. Volume 14, 1980,.

[19] Loren F. Bliese, "A Generative Grammar Of Afar," The Summer Institute Of Linguistics And The University Of Texas, Arlington, Phd 1981.

[20] Sandra Lee Fulmer, "Parallelism And Planes In Optimality Theory:Evidence From Afar, A Dissertation, University Of Arizona," 1997.

[21] Ali Mohammed Ibrahim, "Development Of Qafar Morphological Analyzer," Debre Berhan University School Of Computing Department Of Information Systems, Debre Berhan, Msc 2014.

[22] Richard J. Hayward, "Qafar (East Cushitic)," In The Handbook Of Morphology, 2001.

[23] Ethiopia Standard Es 3453:2008. (2008, April) Localization Standard For Afaraf.

[24] Jamaaluddiin Q.Reedo, Qafar Afak Yabti - Rakiibo. Samara: Qafarafih Cubbbussoo Kee Gaddaaloysiyyi Fanteena, 2007.

[25] Qafa Afih Cubbusoo Kee Gaddloysiyyi Fanteena, Qafarafih Barittoh Kitaaba, 2nd Ed. Addis Ababa, Ethiopia: Culture And Art Society Of Ethiopia, 2002.

[26] Jamaal Reedo, Qafar Afak Yabti - Rakiibo. Sanera, Ethiopia: Qafarafih Cubbussoo Kee Gaddaloysiyyi Fanteena, 1999/2007.

[27] Djoerd Hiemstra, "Information Retrieval Models," Isbn-13: 978-0470027622, Pp. 1-9, November 2009.

[28] "Chapter 2 The Information Retrieval Process," S. Ceri Et Al., Web Information Retrieval, Data-Centric Systems And Applications, Berlin Heidelberg, 2013.

[29] Prabhakar Raghavan, Hinrich Schütze Christopher D. Manning, An Introduction Information Retrieval, Online Edition (C) Ed. Cambridge, England: Cambridge University Press, 2009.

[30] Chengxiang Zhai Jing Jiang, "An Empirical Study Of Tokenization Strategies For Biomedical Information Retrieval," Department Of Computer Science, University Of Illinois At Urbana-Champaign Urbana, Il 61801,.

[31] Anjali Ganesh Jivani, "A Comparative Study Of Stemming Algorithms," Int. J. Comp. Tech. Appl, Vol. 2, No. 6, 1930-1938.

[32] Julie Beth Lovins, "Development Of A Stemming Algorithm," Mechanical Translation And Computational Linguistics, Vol. Vol.11, P. 10, 1968.

[33] M. Porter, An Algorithm For Suxffix Stripping Program., 1980, Vol. Volume 14.

[34] James Allan And Giridhar Kumaran, "Details On Stemming In The Language Modeling Framework," Center For Intelligent Information Retrieval, Department Of Computer Science, University Of Massachusetts Amherst, Amherst, Workshop 2001.

[35] Cheristopher Buckle Gerard Salton, "Term-Weighting Approach In Automatic Text Retreival," Information Processing Management, Vol. 24, No. 5, Pp. 513-523, January 1988.

[36] Amit Singhal, "Modern Information Retrieval: A Brief Overview," Bulletin Of The Ieee Computer Society Technical Committee On Data Engineering, P. 9, 2011.

[37] Ahmad Khonsari, Farhad Oroumchian Mostafa Keikha, "Rich Document Representation And Classification: An Analysis," Knowledge-Based Systems, No. 5, P. 5, July 2008.

[38] Cheristopher Buckle Gerard Sattn, "Term-Weighting Approach In Automatic Text Retreival," Information Processing Management, Vol. 24, No. 5, Pp. 513-523, January 1988.

[39] Seyit Kocberber, Erman Balcik Fazli Can, "Information Retrieval On Turkish Texts," Journal Of The American Society For Information Science And Technology, Vol. 59, No. 3, Pp. 407-421, February 2008.

[40] Stemming Of Amharic Words For Information Nega Alemayehu And P. Willett, "Stemming Of Amharic Words For Information Retrieval," American Society For Information Science, Vol. 50, No. 6, Pp. 524-529, Vol. 50, No. 6, Pp. 524-529, 2002.

[41] Ermias Abebe Tesfaye Debela, "Designing A Rule Based Stemmer For Afaan Oromo Text," International Journal Of Computational Linguistics, Addis Ababa,Vol. 1, Vol. 1, No. 2, Pp. 1-11, 2010.

[42] Karl Aberer, Information Retrieval And Data Mining, Part 1 – Information Retrieval., 2006/7.

[43] C. Manning, "Introduction To Information Retrieval," Cambridge University Press, Vol. 1, 2008.

[44] Baeza-Yates Ribeiro-Neto, "Information Retrieval: Data Structure & Algorithms," University Of Waterloo, Waterloo, University Of Waterloo, 2004.

[45] N. Alemayehu And P. Willett, "The Effectiveness Of Stemming For Information Retrieval In Amharic," Program: Electronic Library And Information Systems, Vol. 37, Pp. 254-259, 2003.

[46] James Allan And Giridhar Kumaran, "Details On Stemming In The Language Modeling Framework," Department Of Computer Science University Of Massachusetts Amherst, Amherst Usa, 2002.

[47] Loren F. Bliese, A Generative Grammar Of Afar, Ph.D. Dissertation.: The Summer Institute Of Linguistics And The University Of Texas At Arlington, 1981.

[48] Francis E. Mahaffy, "An Outline Of The Phonemics And Morphology Of The Afar (Danakil) Language Of Eritrea,East Africa," 1956.

[49] Gifta Jamaal Reedo, Qafarafay Qafarafat Maqnisimeh Maysarraqa. Samera, Ethiopia, 2009.

[50] Sanderson Mark, Test Collection Based Evaluation Of Information Retrieval Systems, University Of Sheffield The Information School, Ed. Sheffield, Uk, 2010.

[51] C. R. Kothari, Research Methodology: Methods And Techniques, 2nd Ed. New Delhi: New Age International Ltd, 2004.

[52] Online Edition (C) Cambridge Up, "Boolean Retrieval," Cambridge University, Press April 2009.

[53] Riyad Al-Shalabi, Ghassan Kanaan, And Alaa Al-Nobani Mohamad Ababneh, "Building An Effective Rule-Based Light Stemmer For Arabic Language To Improve Search Effectiveness," The International Arab Journal Of Information Technology, Vol. 9, 4, July 2012.

[54] Stephen Marlett, An Introduction To Phonological Analysisphonological Analysis, Stephen A. Marlett, Ed.: Summer Institute Of Linguistics, 2001.

[56] Ed Greengrass, Information Retrieval: A Survey., 30 November 2000.

[57] Robert Krovetz., "Viewing Morphology As An Inference Process".

[58] Wakshum Mekonen, "Development Of Stemming Algorithm For Afaan Oromo Text ," M.Sc. Theses, Addis Abeba University, 2000.

[59] Lemma Lessa, "Development Of Stemming Algorithm To Wolytta Text," M.Sc Thesis, Addis Ababa University, 2000.

[60] Abiyot Bayou, "Developing Automatic Word Parser For Amharic Verbs And Their Derivation," 2000.

[61] R J Hayward And Enid M Parker, An Afar-English-French Dictionary (With Grammatical Notes In English).: School Of Oriental And African Studies Of The University Of London, 1985.

[62] Enid M. Parker, Afar-English Dictionary. Hyattsville, Md 20782: Dunwoody Press, 2009.

[63] Jamal A. Reedo, "Afaraf (Afar Language) & Its Dictionary Preparation," 2006.

[64] Andrew Spencer And Arnold Zwicky, "The Handbook Of Morphology," , 2001.

[65] Martine Vanhove, Roots And Patterns In Beja (Cushitic): The Issue Of Language Contact

With Arabic.: Morphologies In Contact, 2011.

[66] Kimmo Koskeniemmi, "Two-Level Morphology: A General Computational Model For Word-Form Recognition And Production," 1983.

[67] Yona Shlomo, "A Finite-State Based Morphological Analyzer For Hebrew," 2004.

[68] Michael Gasser, "Hornmorpho1.0: A System For Morphological Processing Of Amharic, Oromo, And Tigrinya," 2009.

[69] Tesfaye Bayu Bati, "Automatic Morphological Analyzer For Amharic," 2002.

[70] Saba Amsalu, Girma A. Demeke, "Non-Concatinative Finite State Morphotactics Of Amharic Simple Verbs," 2006.

[71] Al-Shami, Al-Sheikh Jamaladdin & His Son Al-Shami, Dr. Hashim Jamaladdin, "Almanhal Fi Tarikh Wa Akhbarul ‘Afar (Arabic)," Cairo, 1997.

[72] Kenneth R. Beesley, "Finite-State Morphological Analysis And Generation Of Arabic," 2001.

[73] Jonathan Lipps, "A Finite-State Morphological Analyzer For Swahili," 2011.

[74] Flammie Prinen, "Development Of Finnish Morphological Analyzer," 2010.

[75] R. M. Kaplan And M. Kay, "Regular Models Of Phonological Rule Systems," 1994.

[76] L. Karttunen, R. M. Kaplan, And A Zaenen, "Two-Level Morphology With Composition," In Proceedings Of The International Conference On Computational Linguistics, 1992.

[77] Kenneth Beesley And Lauri Karttunen, Finite-State Morphology., 2003.

[78] Rucart, Pierre, "Templates From Syntax To Morphology:Affix Ordering In Qafar," 2006.

[79] Rucart, Pierre, "The Vocalism Of Strong Verbs In Afar," Berkeley Lingustic Society, 2001.

[80] Parker, E. M, English-Afar Dictionary.: Dunwoody Press, 2006.

[81] Kenneth R Beesley And Lauri Karttunen, Finite-State Morphology., 2003.

[82] Beesley, Kenneth R., "Morphological Analysis And Generation:A First-Step In Natural Language Processing," , 2004.

[83] C.D. Johnson, "Formal Aspects Of Phonological Description.," 1972.

[84] Prof. Dr. Martin Volk, Applying Finite-State Techniques To A Native American Language: Quechua., 2010.

[85] Diriba Megersa, "An Automatic Sentence Parser For Oromo Language Using Supervised Learning Technique," 2002.

1 Appendix I:

Afaraf Stop Words

a	aki	anee	axcuk
abba	akkak	anih	aysa
adda	akke	aniinim	ayyunti
addal	akkele	animiya	bey
addat	akkinnaan	aninnaanih	caddol
af	akkinnaanih	anni	dagoo
afa	akkuk	anniyyi	dudda
afat	akmew	anu	duma
afih	akmewaanam	anuk	edde
ah	alle	asaaku	ekkek
ahak	ama	atu	ekkem
ahhak	amo	away	ekken
akah	amol	awayih	elle
akak	an	axce	

ene

eneenim

enem

enen

exxa

fan

fanah

fanat

gaba

gabat

geytima

gifta

gubal

gubat

haanama

hay

hay

hay

hee

heeh

hi

hinna

hinnay

hinnayi

hununu

ikkah

ikkal

ikkalah

ikkel

immay

inki

inkih

inkiimih

inkim

inkinnah

inna

innah

innam

inni

inta

is

isi

isih

isim

isin

isinni

itta

iyya

iyya

iyyaanam

iyyal

iyyan

iyyay

iyye

iyyeeh

iyyeh

iyyen

iyyi

kaa

kaa

kaadu

kaah

kaak

kaak

kaal

kaat

kah

kak

kal

kalah

kaxxa

kaxxam

kay

kee

keeh

keenik

ken

kinnaane

kinni

kinni

kinni

kinniih

kinnim

kinnim

kinninnom

kinnon

kinnuk

kinnuk

koh
kok
kol
koo
ku
kulli
kullim
lakal
lakal
le
le
leeh
leh
leh
lem
lih
lino
litoonu
liyo
loonumu
luk
luk
luk

lukuk
ma
ma
maca
maca
macaay
macal
mali
mango
manna
matan
may
may
meqe
meqem
naa
naah
nabam
naharsi

nan
nanu
nee

neek
neh
nek
nel
net
ni
nim
ninni
ohim
ohum
qafar
qiisi
qusba
saaku
sarra
siinih
siit

siita
sin
sinam
sinni

sinni
sinnim
sitt

sitta
sitta

sittin

sugte
ta
ta
taagah
taama
tah
tahaak
takkay
takke
takkeemiiy
takkeh

takkek	tekke	wak	yaanama
takkem	tekkek	waqdi	yaanamal
takku	ten	way	yakke
takkuh	tet	waynam	yalli
tama	toh	wayta	yan
tamah	tohuuy	waytam	yani
tamaha	tonnah	waytek	yanih
tan	too	week	yanim
tan	tu	woh	yanuh
tani	tuk	woh	yekke
tanih	usuk	wohih	yen
taniih	uxih	Wohuuy	yi
tanim	w.w.	wokke	yoh
tanu	w.w.	wokkel	yok
taway	w.w…	wonna	yol
teetih	waa	woo	yoo
teetik	waa	xiqnta	yoo
teetil	waam	y anu	yot
teetit	wadir	yaanam	

2 Appendix II:

Indexing code

```
import re
import os
import math
from operator import itemgetter


characters = "/=-.,!#$%^&*-();:-\n\t\\\"?!{}[]_ - <>/0123456789/"



def tokenize(string):
    """lowercase every string. it returns a list whose elements are the separate string."""
    terms = string.lower().split()
    return [term.strip(characters) for term in terms]


def vowelinstem(stem,vls):
    for x in vls:
        if x in stem:
            return 1
        else:
            return 0
def checkforv(x):
   if x=='a'or x=='e' or x== 'u' or x=='i' or x=='o':
        return 1
   else:
        return 0

longvowelcherk = ['aa']
sufcheker = ['eh','en','eenih','eeniik','eenimil','eenim','aanam']
##  The stemmer function
def thestemmer(word):
    l=len(word)
    if word.startswith("t") and (word.endswith not in sufcheker):
        word = word.replace('t','',1)
    elif word.startswith("y") and (word.endswith not in sufcheker):
       word = word.replace('y','',1)
    elif word.startswith("n") and (word.endswith not in sufcheker):
       word = word.replace('n','',1)



    word = step1(word)
    word = step2(word)
    word = step3(word)
    word = step4(word)
    word = step5(word)
```

```
    word = step6(word)
    word = step7(word)
    word = step8(word)
    word = step9(word)
    word = step10(word)
    word = step11(word)
    word = step12(word)
   # word = step13(word)
    return word

def countVowel ( stem ):
    vowel = ['a','e','i','o','u']
    chek = 0
    count = 0
    i=0
    for s in stem:
        if s in vowel and i+1 <= len(stem)-1 and stem[i+1] not in vowel:
            chek = chek + 1
        else:
            if chek == 1:
                count = count + 1
                chek = 0
            else:
                chek = 0
        i= i+1
    return count

def checkVowel ( stem ):
    vowel = ['a','e','i','o','u']
    count = 0
    for s in stem:
        if s in vowel:
            count = count + 1
    return count


def endsWithDouble ( stem ):
    l = len(stem)
    chek = 0
    if stem[l-1] == stem[l-2]:
        chek =1
    return chek



def endsWithCVC ( stem ):
```

```
    vowel = ['a','e','i','o','u']
    l = len(stem)
    chek = 0
    if stem[l-6] not in vowel and stem[l-7] not in vowel:
        chek = 1
    return chek



def step1(stem):
    l=len(stem)
    if stem.endswith("taah"):
        return stem[:l-4]
    elif stem.endswith("tah"):
        return stem[:l-3]
    if stem.endswith("naah"):
        return stem[:l-4]
    elif stem.endswith("nah"):
        return stem[:l-3]
    elif stem.endswith("aah"):
        return stem[:l-3]
    elif stem.endswith("ah"):
        return stem[:l-2]
    elif stem.endswith("teeh"):
        return stem[:l-4]
    elif stem.endswith("teh"):
        return stem[:l-3]
    elif stem.endswith("neeh"):
        return stem[:l-4]
    elif stem.endswith("neh"):
        return stem[:l-3]
    elif stem.endswith("haak"):
        return stem[:l-4]
    elif stem.endswith("aak"):
        return stem[:l-3]
    elif stem.endswith("ak"):
        return stem[:l-2]
    elif stem.endswith("luk"):
        return stem[:l-3]
    elif stem.endswith("uuk"):
        return stem[:l-3]
    elif stem.endswith("uk"):
        return stem[:l-2]
    elif stem.endswith("eek"):
        return stem[:l-3]
    elif stem.endswith("ek"):
```

```
        return stem[:l-2]
    elif stem.endswith("ak"):
        return stem[:l-2]
    elif stem.endswith("h"):
        return stem[:l-1]
    elif stem.endswith("k"):
        return stem[:l-1]
    elif stem.endswith("l"):
        return stem[:l-1]
    elif stem.endswith("t"):
        return stem[:l-1]
    else:
         return stem.lower()

def step2( stem ):
    l=len(stem)
    if stem.endswith("eemi"):
        return stem[:l-4]
    elif stem.endswith("eenimi") and (countVowel(stem[:l-1]) > 1):
        return stem[:l-6]
    elif stem.endswith("eeni") and (countVowel(stem[:l-0]) > 0):
        return stem[:l-4]
    elif stem.endswith("eenii") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("innay") and (countVowel(stem[:l-1]) > 1):
        return stem[:l-5]
    elif stem.endswith("inniyoy") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-7]
    elif stem.endswith("innitoy") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-7]
    elif stem.endswith("inninoy") and (countVowel(stem[:l-1]) > 1):
        return stem[:l-7]
    elif stem.endswith("ittoonuy") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-8]
    elif stem.endswith("innitoonuy") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-10]
    else:
        return stem.lower()


def step3 ( stem ):

    l=len(stem)
    if stem.endswith("aama") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-4]
    elif stem.endswith("taanama") and (endsWithCVC(stem) == 1) and (countVowel(stem[:l-1]) > 0):
```

```
        return stem[:l-7]
    elif stem.endswith("aanama") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-6]
    elif stem.endswith("aanam") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("aana") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-4]
    else:
        return stem.lower()


def step4 ( stem ):
    l=len(stem)
    if stem.endswith("hayto") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("haytu") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("to") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    elif stem.endswith("tu") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    else:
        return stem.lower()



def step5 ( stem ):
    l=len(stem)
    if (stem.endswith("i")) and (countVowel(stem[:l-1]) > 1) and (stem[:l-2].endswith("oo") or
stem[:l-2].endswith("aa") or stem[:l-3].endswith("xx")):
        return stem[:l-4] + "a"
    elif (stem.endswith("i")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("ii"):
        return stem[:l-3] + stem[-2]
    elif (stem.endswith("a")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("ee") or
stem[:l-2].endswith("oo"):
        return stem[:l-3]
    elif (stem.endswith("a")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("ii"):
        return stem[:l-3]+ stem[-2]
    elif (stem.endswith("itte")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-4]
    elif (stem.endswith("a")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("uu"):
        return stem[:l-3] + stem[-2]
    elif (stem.endswith("u")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("uu"):
        return stem[:l-3] + stem[-2]
    elif (stem.endswith("wa")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-2]
```

```
    elif (stem.endswith("yta")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif (stem.endswith("ta")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-2]
    elif (stem.endswith("la")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    elif (stem.endswith("yto")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif (stem.endswith("to")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-2]
    elif (stem.endswith("ytu")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif (stem.endswith("two")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif stem.endswith("li"):
        return stem[:l-2]
    else:
        return stem.lower()


def step6 ( stem ):
    l=len(stem)
    if (stem.endswith("y")) and (countVowel(stem[:l-1]) > 0) and (stem[:l-1].endswith("aa") or stem[:l-
1].endswith("ii")or stem[:l-1].endswith("uu")):
        return stem[:l-3]
    elif (stem.endswith("y")) and (countVowel(stem[:l-1]) > 0) and l - 3 > 1 and (stem[:l-
1].endswith("ee") or stem[:l-1].endswith("oo")):
        return stem[:l-2]

    else:
        return stem.lower()


def step7( stem ):
    l=len(stem)
    if (stem.endswith("siisiyya")) and (countVowel(stem[:l-5]) > 0):
        return stem[:l-8]
    elif (stem.endswith("isiyya")) and (countVowel(stem[:l-2]) > 0 and l-7 > 0 ):
        return stem[:l-6]
    elif (stem.endswith("itiyya")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-6]
    elif (stem.endswith("siyyi")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("iyya")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("iyyi")) and (countVowel(stem[:l-2]) > 0):
```

```
        return stem[:l-4]
    elif (stem.endswith("iyy")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("sis")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("isiis")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("oola")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]

    else:
        return stem.lower()


def step8( stem ):
    l=len(stem)
    if (stem.endswith("tam")) and (countVowel(stem[:l-7]) > 0):
        return stem[:l-7]
    elif (stem.endswith("tan")) and (countVowel(stem[:l-6]) > 0):
        return stem[:l-6]
    elif (stem.endswith("ta")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-6]
    elif (stem.endswith("nam")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("am")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-3]
    elif (stem.endswith("an")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-3]
    else:
        return stem.lower()



def step9( stem ):
    l=len(stem)
    if (stem.endswith("eloonum")) and (countVowel(stem[:l-4]) > 0):
        return stem[:l-7]
    elif (stem.endswith("elem")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-6]
    elif (stem.endswith("elon")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("ele")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("ennom")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("enno")) and (countVowel(stem[:l-2]) > 0):
```

```
        return stem[:l-4]
    elif (stem.endswith("oonu")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-4]
    elif (stem.endswith("ettonum")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-7]
    elif (stem.endswith("ettom")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("etton")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("etto")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("eyyom")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("eyyo")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    else:
        return stem.lower()


def step10( stem ):
    l=len(stem)
    if (stem.endswith("le")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    elif (stem.endswith("lu")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    else:
        return stem.lower()


def step11( stem ): #remove prefix
    l=len(stem)
    if stem.startswith('ma') or stem.endswith('maa'):
        stem1=stem.replace('ma','',1)
        return stem
    elif stem.startswith('mee'):
        stem1=stem.replace('me','',1)
        return stem
    elif stem.startswith('mii'):
        stem1=stem.replace('mi','',1)
        return stem
    elif stem.startswith('muu'):
        stem1=stem.replace('mu','',1)
        return stem
    else:
        return stem.lower()
```

```
def step12 ( stem ):
   l=len(stem)
   if stem.startswith("aa"):
      stem=stem.replace('a','',1)
      return stem
   elif stem.startswith("t"):
      stem=stem.replace('t','',1)
      return stem
   elif stem.startswith("y"):
      stem=stem.replace('y','',1)
      return stem
   elif stem.startswith("ee"):
      stem=stem.replace('e','',1)
      return stem
   elif stem.startswith("ii"):
      stem=stem.replace('i','',1)
      return stem
   elif stem.startswith("oo"):
      stem=stem.replace('o','',1)
      return stem
   elif stem.startswith("uu"):
      stem=stem.replace('u','',1)
      return stem
   else:
      return stem.lower()


stopword=open('stopwordlist.txt','r')
os.chdir('corpus')
string=''
stem=[]
mystr=''
stoplist=stopword.read()
stopword.close()
for file in os.listdir("."):
   if file.startswith('af'):
      txtfile=open(file,'r')
      read=txtfile.read()
      string=read
      string=string.lower()
      token=tokenize(string)
      name='stemmed_'
      for word in token:
         if word  not in stoplist:
            stemmed=thestemmer(word)
            stem.append(stemmed)
```

```
        else:
            continue


stemed1=[]
stemterm=''
for read in stem:
   stemterm=stemterm+read+' '

def writeuniqwordtofile():

   term=stemterm
   term=re.split(r'\W+',term)
   term.sort()

   uniqterm=''
   for x in term:
      if x in uniqterm:
         v=0
      else:

         uniqterm=uniqterm+x+' '
         stemed1=d
         setmingword=stemed1.split()
   f=open('vocfile.txt','w')
   f.write(d)
   f.close()


keywordfile = open('vocfile.txt','r')
keyword = keywordfile.read()
keyterms = keyword.split()
dic={}

def countf():
   c=0
   for files in os.listdir("."):
      c=c+1
   return c

dic11={}
def docfreq():
   a=countf()
   Zdic2={}
   for word in keyterms:
      df=0
```

```
        li=[]
        for file in os.listdir("."):
            if file.startswith('af'):
                f=open(file,'r')
                read=f.read()
                term=read.lower()
                if word in term:
                    df=df+1
                    dic11[word,term.count(word)]=df
    return dic11

def writevocb():
    df=docfreq()
    voc = ""
    vf = open("vocabulary.txt", 'w')
    for a,b in df.iteritems():
        if a[0] != '':
            voc = a[0] + " " + str(a[1]) + " " + str(b)
            vf.write(voc)
            vf.write("\n")


writevocb()
def wordfilefq():
    dic={}
    dic2={}
    for word in keyterms:
        df=0
        li=[]
        for file in os.listdir("."):
            if file.startswith('af'):
                f=open(file,'r')
                read=f.read()
                term=read.lower()
                if word in term:

                    li.append(file)
        dic2[word]=li
    return dic2

wordfilefq()
def post():
    fi=open('post.txt','a')
    inv=wordfilefq()
    dic={}
    for k,v in inv.iteritems():
```

```
        doc=v
        print
        li=[]
        for y in doc:
            di={}
            di2={}
            f=open(y,'r')
            read=f.read()
            read=read.lower()
            tf=read.count(k)

            di[y]=tf
            fi.write(k)
            fi.write(' ')
            fi.write(y)
            fi.write(' ')
            x=str(tf)
            fi.write(x)
            a=findLocations(read,k)
            fi.write(' ')
            fi.write(a)
            fi.write('\n')
            li.append(di)
        dic[k]=li
    return dic



def findLocations(read,word):
    read1=read.split()
    dic={}
    string=''
    for w in read1:
        if word not in dic:
            found=read.find(word)
            while found > -1:
                if word not in dic:
                    dic[word]=found
                    x=str(found)
                    string=string+x
                    found=read.find(word, found+1)

                else:
                    dic[word]=str(dic[word])+" "+str(found)+" "
                    y=str(found)
                    string=string+','+y
```

```
            found=read.find(word, found+1)

    return string

post()
```

3 Appendix III:

Searching code

```
import re
import os
import math
import operator
characters = ".,!#$%\^&*();:\n\t\\\"?!{}[]_-<>/0123456789"
def tokenize(string):
    terms = string.lower().split()
    #return terms
    return [term.strip(characters) for term in terms]
    #print terms
#print tokenize(string)
def vowelinstem(stem,vls):
    for x in vls:
        if x in stem:
            return 1
        else:
            return 0
def checkforv(x):
   if x=='a'or x=='e' or x== 'u' or x=='i' or x=='o':
        return 1
   else:
        return 0
# Function to count the number of vowel-consonant ocurances in a s (m)
def count_m(st):
    msr=1
    for i in st:
        if checkforv(i)==1:
            inxt=st.index(i)+1
            if inxt<len(st)-1 and checkforv(st[inxt])==0:
                msr=msr+1
    return msr
longvowelcherk = ['aa']
```

```
sufcheker = ['eh','en','eenih','eeniik','eenimil','eenim','aanam']
##  The stemmer function
def thestemmer(word):
   l=len(word)
   if word.startswith("t") and (word.endswith not in sufcheker):
      word = word.replace('t','',1)
   elif word.startswith("y") and (word.endswith not in sufcheker):
     word = word.replace('y','',1)
   elif word.startswith("n") and (word.endswith not in sufcheker):
     word = word.replace('n','',1)


   word = step1(word)
   word = step2(word)
   word = step3(word)
   word = step4(word)
   word = step5(word)
   word = step6(word)
   word = step7(word)
   word = step8(word)
   word = step9(word)
   word = step10(word)
   word = step11(word)
   word = step12(word)
  # word = step13(word)
   return word

def countVowel ( stem ):
   vowel = ['a','e','i','o','u']
   chek = 0
   count = 0
   i=0
   for s in stem:
      if s in vowel and i+1 <= len(stem)-1 and stem[i+1] not in vowel:
         chek = chek + 1
      else:
         if chek == 1:
            count = count + 1
            chek = 0
         else:
            chek = 0
      i= i+1
   return count

def checkVowel ( stem ):
   vowel = ['a','e','i','o','u']
```

```
    count = 0
    for s in stem:
        if s in vowel:
            count = count + 1
    return count


def endsWithDouble ( stem ):
    l = len(stem)
    chek = 0
    if stem[l-1] == stem[l-2]:
        chek =1
    return chek



def endsWithCVC ( stem ):
    vowel = ['a','e','i','o','u']
    l = len(stem)
    chek = 0
    if stem[l-6] not in vowel and stem[l-7] not in vowel:
        chek = 1
    return chek



def step1(stem):
    l=len(stem)
    if stem.endswith("taah"):
        return stem[:l-4]
    elif stem.endswith("tah"):
        return stem[:l-3]
    if stem.endswith("naah"):
        return stem[:l-4]
    elif stem.endswith("nah"):
        return stem[:l-3]
    elif stem.endswith("aah"):
        return stem[:l-3]
    elif stem.endswith("ah"):
        return stem[:l-2]
    elif stem.endswith("teeh"):
        return stem[:l-4]
    elif stem.endswith("teh"):
        return stem[:l-3]
    elif stem.endswith("neeh"):
        return stem[:l-4]
```

```
    elif stem.endswith("neh"):
        return stem[:l-3]
    elif stem.endswith("haak"):
        return stem[:l-4]
    elif stem.endswith("aak"):
        return stem[:l-3]
    elif stem.endswith("ak"):
        return stem[:l-2]
    elif stem.endswith("luk"):
        return stem[:l-3]
    elif stem.endswith("uuk"):
        return stem[:l-3]
    elif stem.endswith("uk"):
        return stem[:l-2]
    elif stem.endswith("eek"):
        return stem[:l-3]
    elif stem.endswith("ek"):
        return stem[:l-2]
    elif stem.endswith("ak"):
        return stem[:l-2]
    elif stem.endswith("h"):
        return stem[:l-1]
    elif stem.endswith("k"):
        return stem[:l-1]
    elif stem.endswith("l"):
        return stem[:l-1]
    elif stem.endswith("t"):
        return stem[:l-1]
    else:
         return stem.lower()

def step2( stem ):
    l=len(stem)
    if stem.endswith("eemi"):
        return stem[:l-4]
    elif stem.endswith("eenimi") and (countVowel(stem[:l-1]) > 1):
        return stem[:l-6]
    elif stem.endswith("eeni") and (countVowel(stem[:l-0]) > 0):
        return stem[:l-4]
    elif stem.endswith("eenii") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("innay") and (countVowel(stem[:l-1]) > 1):
        return stem[:l-5]
    elif stem.endswith("inniyoy") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-7]
    elif stem.endswith("innitoy") and (countVowel(stem[:l-1]) > 0):
```

```
        return stem[:l-7]
    elif stem.endswith("inninoy") and (countVowel(stem[:l-1]) > 1):
        return stem[:l-7]
    elif stem.endswith("ittoonuy") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-8]
    elif stem.endswith("innitoonuy") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-10]
    else:
        return stem.lower()


def step3 ( stem ):

    l=len(stem)
    if stem.endswith("aama") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-4]
    elif stem.endswith("taanama") and (endsWithCVC(stem) == 1) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-7]
    elif stem.endswith("aanama") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-6]
    elif stem.endswith("aanam") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("aana") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-4]
    else:
        return stem.lower()


def step4 ( stem ):
    l=len(stem)
    if stem.endswith("hayto") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("haytu") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-5]
    elif stem.endswith("to") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    elif stem.endswith("tu") and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    else:
        return stem.lower()



def step5 ( stem ):
    l=len(stem)
```

```
    if (stem.endswith("i")) and (countVowel(stem[:l-1]) > 1) and (stem[:l-2].endswith("oo") or stem[:l-2].endswith("aa") or stem[:l-3].endswith("xx")):
        return stem[:l-4] + "a"
    elif (stem.endswith("i")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("ii"):
        return stem[:l-3] + stem[-2]
    elif (stem.endswith("a")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("ee") or stem[:l-2].endswith("oo"):
        return stem[:l-3]
    elif (stem.endswith("a")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("ii"):
        return stem[:l-3]+ stem[-2]
    elif (stem.endswith("itte")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-4]
    elif (stem.endswith("a")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("uu"):
        return stem[:l-3] + stem[-2]
    elif (stem.endswith("u")) and (countVowel(stem[:l-1]) > 1) and stem[:l-2].endswith("uu"):
        return stem[:l-3] + stem[-2]
    elif (stem.endswith("wa")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-2]
    elif (stem.endswith("yta")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif (stem.endswith("ta")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-2]
    elif (stem.endswith("la")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    elif (stem.endswith("yto")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif (stem.endswith("to")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-2]
    elif (stem.endswith("ytu")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif (stem.endswith("two")) and (countVowel(stem[:l-2]) > 1):
        return stem[:l-3]
    elif stem.endswith("li"):
        return stem[:l-2]
    else:
        return stem.lower()


def step6 ( stem ):
    l=len(stem)
    if (stem.endswith("y")) and (countVowel(stem[:l-1]) > 0) and (stem[:l-1].endswith("aa") or stem[:l-1].endswith("ii")or stem[:l-1].endswith("uu")):
        return stem[:l-3]
    elif (stem.endswith("y")) and (countVowel(stem[:l-1]) > 0) and l - 3 > 1 and (stem[:l-1].endswith("ee") or stem[:l-1].endswith("oo")):
        return stem[:l-2]
```

```
    else:
        return stem.lower()


def step7( stem ):
    l=len(stem)
    if (stem.endswith("siisiyya")) and (countVowel(stem[:l-5]) > 0):
        return stem[:l-8]
    elif (stem.endswith("isiyya")) and (countVowel(stem[:l-2]) > 0 and l-7 > 0 ):
        return stem[:l-6]
    elif (stem.endswith("itiyya")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-6]
    elif (stem.endswith("siyyi")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("iyya")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("iyyi")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("iyy")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("sis")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("isiis")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("oola")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]

    else:
        return stem.lower()


def step8( stem ):
    l=len(stem)
    if (stem.endswith("tam")) and (countVowel(stem[:l-7]) > 0):
        return stem[:l-7]
    elif (stem.endswith("tan")) and (countVowel(stem[:l-6]) > 0):
        return stem[:l-6]
    elif (stem.endswith("ta")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-6]
    elif (stem.endswith("nam")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("am")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-3]
    elif (stem.endswith("an")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-3]
```

```
    else:
        return stem.lower()



def step9( stem ):
    l=len(stem)
    if (stem.endswith("eloonum")) and (countVowel(stem[:l-4]) > 0):
        return stem[:l-7]
    elif (stem.endswith("elem")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-6]
    elif (stem.endswith("elon")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("ele")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-3]
    elif (stem.endswith("ennom")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("enno")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("oonu")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-4]
    elif (stem.endswith("ettonum")) and (countVowel(stem[:l-3]) > 0):
        return stem[:l-7]
    elif (stem.endswith("ettom")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("etton")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("etto")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    elif (stem.endswith("eyyom")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-5]
    elif (stem.endswith("eyyo")) and (countVowel(stem[:l-2]) > 0):
        return stem[:l-4]
    else:
        return stem.lower()


def step10( stem ):
    l=len(stem)
    if (stem.endswith("le")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    elif (stem.endswith("lu")) and (countVowel(stem[:l-1]) > 0):
        return stem[:l-2]
    else:
        return stem.lower()
```

```
def step11( stem ): #remove prefix
   l=len(stem)
   if stem.startswith('ma') or stem.endswith('maa'):
      stem1=stem.replace('ma','',1)
      return stem
   elif stem.startswith('mee'):
      stem1=stem.replace('me','',1)
      return stem
   elif stem.startswith('mii'):
      stem1=stem.replace('mi','',1)
      return stem
   elif stem.startswith('muu'):
      stem1=stem.replace('mu','',1)
      return stem
   else:
      return stem.lower()

def step12 ( stem ):
   l=len(stem)
   if stem.startswith("aa"):
      stem=stem.replace('a','',1)
      return stem
   elif stem.startswith("t"):
      stem=stem.replace('t','',1)
      return stem
   elif stem.startswith("y"):
      stem=stem.replace('y','',1)
      return stem
   elif stem.startswith("ee"):
      stem=stem.replace('e','',1)
      return stem
   elif stem.startswith("ii"):
      stem=stem.replace('i','',1)
      return stem
   elif stem.startswith("oo"):
      stem=stem.replace('o','',1)
      return stem
   elif stem.startswith("uu"):
      stem=stem.replace('u','',1)
      return stem
   else:
      return stem.lower()

s=open('stopwordlist.txt','r')
stopwordlist=s.read()
```

```
os.chdir('corpus')

string=''
stems=[]
mystr=''

print ('Gurrisiyya Wara Firisi !!!')
str=raw_input()
F=open('query.txt','w')
F.write(str)
F.close()

##for file in os.listdir("."):
##    print file

for file in os.listdir("."):
   if file.startswith('query'):
      txt=open(file,'r')
      read=txt.read()
      string=read
      string=string.lower()#Normalization
      word=tokenize(string)#Tokenization
      name='stemmed_'
      for i in word:
         if i  not in stopwordlist:# Stop list removal
            stemmed=thestemmer(i)# Stemming
            stems.append(stemmed)
         else:
               continue
      #print stem
      for x in stems:
         x1=x# conver list to string
         mystr=mystr+ x1 +' '
      if file.startswith('stemmed'):
         continue
      else:
         stemmedname=name+file
         f=open(stemmedname,'w')
         f.write(mystr)
         f.close()
#
def wordfreq(file):
   dict0={}
   txt=open(file,'r')
   read=txt.read()
   string=read
```

```
    string=string.lower()#Normalization
    word=tokenize(string)#wordization
    wordList=word
    wordfreq = [wordList.count(p) for p in wordList]
    dictionary = dict(zip(wordList, wordfreq))
    count2 = 0
    for t in wordList:
        count2+=1
##    print 'total number of words',count2
    dict0[file]=count2
    return dict0[file] #returns dictionary of file name with the word count hello.txt=4, hello2.txt=5
##print wordfreq('doc1.txt')


def countf():
    c=0
    for file in os.listdir("."):
        if file.startswith('af'):
            c=c+1
    return c
countf()
def tfidf():
    N=countf()
    f=open('vocabulary.txt','r')
    y=f.readlines()
    ff=open('post.txt','r')
    yy=ff.readlines()
    dic={}
    for x in y:
        z=x.split()
        df=z[1]
        term1=z[0]
        st=''

        for xx in yy:
            zz=xx.split()
            term2=zz[0]
            doc=zz[1]
            tf=zz[2]
            st=term2+' '+doc
            if term1==term2:
                doc=zz[1]
                wf=wordfreq(doc)
                tf1=float(tf)
                tf1=tf1/wf
                dfi=float(df)
```

```
            df1=N/dfi
            p=math.log(df1,2)
            idf=tf1*p
            st=doc+' '+term2
            dic[st]=idf
    return dic

readq=open('stemmed_query.txt','r')
Query=readq.read()
Query=Query.split()
if Query==[]:
    sys.exit()
   # print 'your query is not significant term please try again'
readq = open('stemmed_query.txt','r')
Query = readq.read()
Query = Query.split()
d={}

def docfreq_query():
    NN=countf()
    print NN
    for word in Query:
        #print word
        vocf=open('vocabulary.txt','r')
        yyy=vocf.readlines()
        for x in yyy:
            z=x.split()
            term11=z[0]
            c=0
            tf=z[1]
            df=z[2]
            if word in term11:
                tf=z[1]
                df=z[2]
                if df==0:
                    print "no matching document"
                else:
                    fl=float(NN)/float(df)
                    idf=math.log(fl,2)
                    tf=0.5+0.5*float(tf)
                    w=we*idf
                    d[word]=w
    return d

docfreq_query()
weight=tfidf()
```

```
print weight
query=d
sim={}
result=[]

for key in query:
   for docword in weight:
      if key in docword:
         sim[docword]=query[key]*weight[docword]
         for i, j in sim.items():
            ii=i
            t=[ii,j]
            result.append(t)
            result.reverse()


##################
result=[]
for x, y in sim.items():
      ii=x
      # print ii
      t=[x,y]
      result.append(t)
      for i in range(len(result)):
         for j in range (len(result)-1):
            if result[j][1] < result[j+1][1]:
               t=result[j]
               result[j]=result[j+1]
               result[j+1]=t

print('\n   result')
print('Rank\tDoc. List\t\t    RW')
print("|--------------------------------------------------|")
for i in range(len(result)):

   if result[i][1]>=0.0000:
      print(i+1,    result[i][0],    round(result[i][1],5))
      print("|--------------------------------------------------|")
```

Appendix III:

Document-Query matrix used for relevance judgment**; where R= relevant, NR=non relevant**

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Affile1.txt	R	R	NR	NR	NR	NR	NR	NR
Affile2.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile3.txt	R	R	NR	NR	NR	NR	NR	NR
Affile4.txt	R	R	NR	NR	NR	NR	NR	NR
Affile5.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile6.txt	R	NR	NR	NR	R	NR	NR	NR
Affile7.txt	R	R	NR	NR	NR	NR	NR	NR
Affile8.txt	R	NR	NR	NR	R	R	NR	NR
Affile9.txt	R	NR	NR	NR	R	R	NR	NR
Affile10.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile11.txt	R	NR	NR	R	NR	R	NR	NR
Affile12.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile13.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile14.txt	R	R	NR	NR	NR	NR	NR	NR
Affile15.txt	R	R	R	NR	NR	NR	NR	NR
Affile16.txt	R	R	NR	R	R	R	NR	NR
Affile17.txt	R	R	NR	R	R	R	NR	NR
Affile18.txt	R	R	NR	NR	R	R	NR	NR
Affile19.txt	R	R	NR	NR	NR	R	NR	NR
Affile20.txt	R	R	NR	NR	NR	NR	NR	NR
Affile21.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile22.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile23.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile24.txt	R	NR	NR	R	R	R	NR	NR
Affile25.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile26.txt	R	NR	NR	R	NR	NR	NR	NR

Affile27.txt	R	R	NR	NR	NR	NR	NR	R
Affile28.txt	R	R	NR	NR	NR	NR	NR	NR
Affile29.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile30.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile31.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile32.txt	R	NR	NR	NR	R	NR	NR	NR
Affile33.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile34.txt	R	R	NR	NR	NR	NR	NR	NR
Affile35.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile36.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile37.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile38.txt	R	R	NR	R	NR	NR	NR	R
Affile39.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile40.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile41.txt	R	R	NR	R	R	R	NR	NR
Affile42.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile43.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile44.txt	R	NR	NR	R	NR	NR	NR	NR
Affile45.txt	R	R	NR	R	R	R	NR	NR
Affile46.txt	R	R	NR	NR	NR	NR	NR	NR
Affile47.txt	R	NR	R	R	R	R	NR	NR
Affile48.txt	R	NR	NR	R	NR	R	NR	NR
Affile49.txt	R	NR	NR	NR	NR	R	NR	NR
Affile50.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile51.txt	NR	R	NR	R	NR	NR	NR	NR
Affile52.txt	R	R	NR	NR	NR	NR	NR	NR
Affile53.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile54.txt	R	R	NR	NR	NR	NR	NR	NR
Affile55.txt	R	R	NR	NR	NR	NR	NR	NR

Affile56.txt	NR	R	NR	NR	NR	NR	NR	R
Affile57.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile58.txt	R	R	NR	NR	NR	NR	NR	R
Affile59.txt	R	R	NR	NR	NR	NR	NR	NR
Affile60.txt	R	R	NR	NR	NR	NR	NR	NR
Affile61.txt	R	R	NR	NR	NR	R	NR	NR
Affile62.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile63.txt	R	R	NR	NR	NR	NR	NR	NR
Affile64.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile65.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile66.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile67.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile68.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile69.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile70.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile71.txt	R	R	NR	NR	NR	NR	NR	NR
Affile72.txt	R	R	NR	NR	NR	NR	NR	NR
Affile73.txt	R	R	NR	NR	NR	NR	NR	NR
Affile74.txt	NR	R	NR	R	NR	NR	NR	NR
Affile75.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile76.txt	R	R	NR	NR	NR	R	NR	R
Affile77.txt	R	R	NR	NR	NR	R	NR	NR
Affile78.txt	R	R	NR	NR	NR	NR	NR	NR
Affile79.txt	R	R	NR	NR	NR	NR	NR	NR
Affile80.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile81.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile82.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile83.txt	R	R	NR	NR	NR	NR	NR	NR
Affile84.txt	NR	R	NR	NR	NR	NR	NR	R

Affile85.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile86.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile87.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile88.txt	R	R	NR	NR	NR	NR	NR	NR
Affile89.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile90.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile91.txt	R	R	NR	NR	NR	NR	NR	NR
Affile92.txt	R	R	NR	NR	NR	NR	NR	NR
Affile93.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile94.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile95.txt	NR	R	NR	R	NR	NR	NR	NR
Affile96.txt	NR	R	NR	R	NR	NR	NR	NR
Affile97.txt	NR	R	NR	R	NR	NR	NR	NR
Affile98.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile99.txt	R	R	NR	NR	NR	NR	NR	NR
Affile100.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile101.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile102.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile103.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile104.txt	R	NR	R	R	NR	NR	NR	NR
Affile105.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile106.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile107.txt	R	NR	R	NR	NR	NR	NR	NR
Affile108.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile109.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile110.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile111.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile112.txt	NR	R	R	NR	NR	NR	NR	NR
Affile113.txt	NR	NR	R	NR	NR	NR	NR	NR

Affile114.txt	NR	NR	R	R	NR	NR	NR	NR
Affile115.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile116.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile117.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile118.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile119.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile120.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile121.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile122.txt	NR	NR	R	R	NR	R	NR	NR
Affile123.txt	R	NR	NR	NR	R	NR	NR	NR
Affile124.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile125.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile126.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile127.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile128.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile129.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile130.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile131.txt	NR	NR	NR	NR	R	R	NR	NR
Affile132.txt	NR	NR	NR	NR	R	NR	NR	R
Affile133.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile134.txt	R	NR	NR	NR	R	NR	NR	NR
Affile135.txt	R	R	NR	NR	R	NR	NR	NR
Affile136.txt	R	NR	NR	NR	R	NR	NR	NR
Affile137.txt	R	NR	NR	NR	R	NR	NR	NR
Affile138.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile139.txt	R	NR	NR	NR	R	NR	NR	NR
Affile140.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile141.txt	R	NR	NR	NR	NR	R	NR	NR
Affile142.txt	NR	NR	NR	NR	R	R	NR	NR

Affile143.txt	NR	NR	NR	NR	R	NR	NR	NR
Affile144.txt	R	NR	NR	NR	R	NR	NR	NR
Affile145.txt	R	R	NR	NR	NR	NR	NR	NR
Affile146.txt	R	NR	NR	NR	R	NR	NR	NR
Affile147.txt	R	NR	NR	NR	R	NR	NR	NR
Affile148.txt	NR	NR	NR	NR	R	R	NR	NR
Affile149.txt	R	R	NR	NR	NR	NR	NR	NR
Affile150.txt	R	NR	NR	NR	R	NR	NR	NR
Affile151.txt	NR	NR	NR	R	R	NR	NR	NR
Affile152.txt	R	NR	NR	R	R	R	NR	NR
Affile153.txt	R	NR	NR	NR	R	NR	NR	NR
Affile154.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile155.txt	NR	NR	NR	NR	R	NR	NR	NR
Affile156.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile157.txt	R	NR	NR	NR	R	R	NR	R
Affile158.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile159.txt	R	NR	NR	R	NR	R	NR	NR
Affile160.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile161.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile162.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile163.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile164.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile165.txt	R	NR	NR	NR	NR	NR	NR	R
Affile166.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile167.txt	R	NR	NR	R	NR	NR	NR	NR
Affile168.txt	NR	R	NR	NR	NR	NR	NR	R
Affile169.txt	NR	NR	NR	NR	NR	NR	NR	R
Affile170.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile171.txt	NR	R	NR	NR	NR	NR	NR	NR

Affile172.txt	NR	R	R	R	NR	NR	NR	R
Affile173.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile174.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile175.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile176.txt	NR	R	NR	NR	NR	NR	NR	R
Affile177.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile178.txt	NR	R	NR	NR	NR	NR	NR	R
Affile179.txt	NR	NR	NR	NR	NR	NR	NR	R
Affile180.txt	NR	R	NR	NR	NR	NR	NR	R
Affile181.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile182.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile183.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile184.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile185.txt	R	NR	NR	NR	NR	NR	NR	R
Affile186.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile187.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile188.txt	NR	NR	NR	NR	NR	NR	NR	R
Affile189.txt	NR	NR	NR	NR	NR	NR	NR	R
Affile190.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile191.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile192.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile193.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile194.txt	NR	R	NR	NR	NR	NR	NR	R
Affile195.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile196.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile197.txt	R	R	NR	NR	NR	NR	NR	NR
Affile198.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile199.txt	NR	R	NR	R	NR	R	NR	R
Affile200.txt	NR	NR	NR	NR	NR	NR	NR	NR

Affile201.txt	NR	NR	NR	NR	NR	NR	NR	R
Affile202.txt	R	NR	NR	NR	NR	R	NR	R
Affile203.txt	R	R	NR	NR	NR	NR	NR	NR
Affile204.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile205.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile206.txt	R	NR	NR	NR	NR	NR	NR	R
Affile207.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile208.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile209.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile210.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile211.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile212.txt	NR	NR	NR	R	R	R	R	NR
Affile213.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile214.txt	NR	R	NR	R	NR	NR	NR	R
Affile215.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile216.txt	NR	NR	R	NR	NR	NR	R	NR
Affile217.txt	NR	R	NR	R	NR	R	NR	NR
Affile218.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile219.txt	R	NR	R	R	NR	NR	NR	R
Affile220.txt	R	NR	NR	NR	NR	NR	R	NR
Affile221.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile222.txt	NR	NR	NR	R	R	R	R	NR
Affile223.txt	NR	R	NR	R	NR	NR	NR	NR
Affile224.txt	R	NR	NR	R	NR	NR	NR	NR
Affile225.txt	NR	NR	NR	NR	R	NR	NR	NR
Affile226.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile227.txt	R	R	NR	NR	NR	NR	NR	NR
Affile228.txt	R	NR	NR	NR	R	NR	NR	NR
Affile229.txt	NR	NR	NR	NR	NR	R	NR	NR

Affile230.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile231.txt	NR	NR	NR	R	R	NR	NR	NR
Affile232.txt	NR	R	NR	NR	R	NR	NR	NR
Affile233.txt	NR	NR	NR	NR	R	R	NR	NR
Affile234.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile235.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile236.txt	NR	R	NR	R	NR	R	R	NR
Affile237.txt	NR	R	NR	NR	NR	NR	NR	NR
Affile238.txt	NR	NR	R	R	NR	NR	NR	NR
Affile239.txt	R	NR	R	R	NR	NR	NR	R
Affile240.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile241.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile242.txt	R	NR	R	NR	NR	NR	NR	NR
Affile243.txt	NR	NR	NR	R	NR	NR	NR	R
Affile244.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile245.txt	NR	NR	NR	R	NR	NR	NR	NR
Affile246.txt	R	NR	NR	R	R	NR	NR	NR
Affile247.txt	R	NR	NR	R	NR	NR	NR	NR
Affile248.txt	R	NR	NR	R	NR	NR	NR	NR
Affile249.txt	R	R	NR	NR	NR	NR	R	NR
Affile250.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile251.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile252.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile253.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile254.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile255.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile256.txt	NR	NR	NR	NR	R	R	NR	NR
Affile257.txt	R	NR	NR	NR	NR	R	NR	NR
Affile258.txt	NR	NR	NR	NR	NR	NR	NR	NR

Affile259.txt	R	NR	NR	NR	NR	R	NR	NR
Affile260.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile261.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile262.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile263.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile264.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile265.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile266.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile267.txt	NR	NR	NR	NR	R	R	NR	R
Affile268.txt	NR	NR	NR	NR	NR	R	R	NR
Affile269.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile270.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile271.txt	R	R	NR	NR	NR	R	NR	NR
Affile272.txt	NR	NR	R	NR	NR	NR	NR	NR
Affile273.txt	NR	NR	R	NR	NR	R	NR	NR
Affile274.txt	R	R	NR	NR	NR	R	NR	NR
Affile275.txt	R	R	NR	NR	NR	R	NR	NR
Affile276.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile277.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile278.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile279.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile280.txt	NR	NR	NR	NR	NR	R	NR	NR
Affile281.txt	NR	NR	NR	NR	R	R	R	NR
Affile282.txt	NR	NR	NR	NR	R	R	R	R
Affile283.txt	NR	NR	NR	R	NR	NR	R	NR
Affile284.txt	R	NR	NR	NR	R	NR	NR	NR
Affile285.txt	NR	NR	NR	NR	NR	NR	R	NR
Affile286.txt	R	NR	NR	NR	R	NR	NR	NR
Affile287.txt	R	NR	NR	NR	NR	NR	R	NR

Affile288.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile289.txt	NR	NR	NR	NR	NR	NR	R	R
Affile290.txt	R	NR	NR	NR	NR	NR	R	NR
Affile291.txt	R	NR	NR	NR	NR	NR	NR	NR
Affile292.txt	NR	NR	NR	NR	NR	NR	NR	NR
Affile293.txt	NR	NR	NR	NR	NR	NR	R	NR
Affile294.txt	NR	NR	NR	NR	NR	R	R	NR
Affile295.txt	NR	NR	NR	NR	NR	NR	R	NR
Affile296.txt	NR	NR	R	NR	NR	NR	R	NR
Affile297.txt	R	NR	NR	R	NR	NR	R	NR
Affile298.txt	NR	NR	NR	NR	NR	NR	R	NR
Affile299.txt	NR	NR	NR	NR	NR	NR	R	R
Affile300.txt	NR	NR	NR	NR	NR	NR	R	NR