ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

SCHOOL OF INFORMATION SCIENCE

SEMANTIC INDEXING AND DOCUMENT CLUSTERING FOR AMHARIC INFORMATION RETRIEVAL

BY

MULUALEM WORDOFA

ADDIS ABABA UNIVERSITY

JUNE, 2013

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

SCHOOL OF INFORMATION SCIENCE

SEMANTIC INDEXING AND DOCUMENT CLUSTERING FOR AMHARIC INFORMATION RETRIEVAL

A Thesis Submitted to the School of Graduate Studies of Addis Ababa University in Partial Fulfillment of the Requirements for the Degree of Master of Science in Information Science

BY

MULUALEM WORDOFA

JUNE, 2013

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

SCHOOL OF INFORMATION SCIENCE

SEMANTIC INDEXING AND DOCUMENT CLUSTERING FOR AMHARIC INFORMATION RETRIEVAL

MULUALEM WORDOFA

Name and signature of Members of the Examining Board

Name	Title	Signature	Date
_____	Chairman	_____	_____
Ermias Abebe	**Advisor**	_____	_____
Workshet Lamnew	**Examiner**	_____	_____

DECLARATION

This thesis is my original work, and has not been presented/submitted as a partial requirement for a degree in any university and that all sources of material used in the thesis have been duly acknowledged.

_____

Mulualem Wordofa Regassa

June, 2013

This thesis has been submitted for examination with my approval as University advisor.

_____

Ermias Abebe

June, 2013

DEDICATED TO:

MY LOVELY SON, YEDIDIA MULUALEM

TABLE OF CONTENTS

LIST OF TABLES

TABLE OF FIGURE

LIST OF ALGORITHMS

LIST OF ACRONYMS

IR **:-** Information Retrieval

LSI :- Latent Semantic Indexing

SVD :- Singular Value Decomposition

FDRE :- Federal Democratic Republic of Ethiopian

ENA :- Ethiopian News Agency

NLP :- Natural Language Processing

TF :- Term Frequency

IDF :- Inverse Document Frequency

DFT :- Document Frequency term

TFIDF :- Term frequency Inverse Document Frequency

ATR :- Automatic Term Recognition

VSM :- Vector Space Model

MAP :- Mean Average Precision

ICS :- Inter Cluster Similarity

ECS :- Intra Cluster Similarity

Acknowledgment

Before all, I praise the almighty God and his mother St. Mary for making everything the way it is. I owe a great debt of gratitude to my advisor Ato Ermias Abebe. This thesis would not have been possible without his guidance, inspiration and enlightening ideas, comments and suggestions. I am glad to work with him and to share his rich, broad and deep knowledge. Thank you very much.

I would like to express my gratitude to my beloved family for their encouragement, patience, and financial support they have given me during the course of this thesis. Special thanks go to my wife, Saba Ayimro and my mother, Abebech Bekele

I am also indebted to Hawassa University for granting me to attend my post-graduate study at Addis Ababa University.

Abstract

Introduction:

We have experienced an exponential increase in the electronic Amharic text information inside and outside of the organization. This accumulation of information is challenging for archival and searching of information. Due to that, an information retrieval (IR) system for Amharic language become indispensible and it allows the user to retrieve relevant documents that satisfies information need of users. Some Amharic IR systems were developed in the last couples of decade, however, the performance measure of the systems were not adorable. It happened because of different reasons but the major one was not properly address semantic natures of the language. Moreover, there was no any attempt to make retrieval effective through document clustering. In order to solve these issues, integrating of semantic indexing of documents and document clustering techniques with generic IR system will improve the retrieval performance.

Methods:

In this research semantic indexing and document clustering of Amharic IR system is developed. It comprises three basic components indexing, clustering and searching. The system comprises all processes exist in generic IR plus to that C-value technique multi word term extraction, k-means algorithms document clustering, cluster base searching strategy used.

Conclusion:

The system tested using tagged Amharic news documents size of 650Kb and it registered F-measure of 66% accuracy. It is by far good compared with the latest work (Amanuel[31] work). Nevertheless, the performance of the system is greatly affected by synonyms and polysemous, incorrect clustering, cluster representative problems, Amharic knowledge base.

Keywords: Information Retrieval, Semantic indexing, Document Clustering, Amharic

Chapter One

INTRODUCTION

1.1. Background

In the last two decades, we have experienced an exponential increase in the electronic information. Internet is one huge repository of information. Nowadays, it is common, specially looking for information on Internet in order to satisfy information needs. But not all retrieved information satisfy the information needs of individuals, the user has to decide if information are interesting or not. Having words in the information by itself does not meet the users information needs, semantic and contextual understanding of the user information need and information in the collection have to take into consideration to return effective result to the user [2]. Like others, considerable amounts of information have been produced in Ethiopia, especially in Amharic language. This information explosion have become a challenge for document management and searching for information from this huge amount of information collection produced in Amharic language [3].

Research in document IR is aimed at designing and evaluating systems that try to fulfill a user's information need. A document IR system can react in a variety of ways. For instance, it can present a list of documents presumably containing the information the user is looking for (ad-hoc retrieval), or it can directly answer a user's question by generating natural language sentences or extracting sentences from the corpus (Question-Answering) [4].

The corpus itself is simply a collection of natural language documents. From an abstract point of view, a document ***d*** is relevant to a query ***q*** if ***d*** 'is about' ***q***. Defining 'is about' is a non-trivial task and several approaches have been proposed [1].

Usually, information retrieval systems do not work on the documents themselves, but on representations or abstraction. Therefore, deciding whether a document is relevant to a query depends on the kind of representation that is being used for the document and the query. Almost

all existing IR systems simply represent documents and queries as a 'bag-of-words' with its weight. When users give some query to the system the system tried to match those documents which have terms in the query, and the documents will be automatically retrieved but not otherwise [4].

There are several problems tied to the simple 'bag-of-words' representation of documents. Two of them are word-sense ambiguity (polysemy) and synonymy [4]. If a query contains terms which are lexically ambiguous, some irrelevant documents which contain terms of a query are retrieved. However, the retrieved documents may not have the intended meaning of users' information need. On the other hand, some documents are not retrieved because they don't share terms with the query, although they are relevant to user's information need. These are very major problems in IR world that are not well addressed [4].

There are various methodologies and techniques tried so as to make effective and efficient way of retrieving documents from very large and unstructured corpus. Some of the scientific approaches to resolve the problems are ontology based IR, latent semantic indexing, query expansion and reformulation, statistical method for semantic indexing, and others.

Ontology based retrieval is an approach that uses the knowledge of the language for document retrieval purpose. Ontology based information retrieval system works by associating of concepts from knowledge base to documents and queries, e.g. synonymous of the terms can be found using knowledge base. For a given query first concepts are extracted from the query itself. Then the set of concepts associated with each document is extracted from corpus. Next, these two sets are compared using simple metric, which expresses the similarity between a document $\boldsymbol{D_i}$ and given query $\boldsymbol{Q}$. Based on the similarity measure relevant documents are retrieved. The ontology based retrieval system needs intensive work in linguistic, in the domains, and NLP. Therefore it is much more language and domain dependent [4].

Latent Semantic Indexing (LSI) method helps to overcome the problems of lexical matching. LSI assumes that there is latent structure in word usage that words are partially obscured by variability in words choice. A truncated Singular Value Decomposition (SVD) is used to estimate the structure in word usage across documents. Retrieval is then performed using singular values vectors obtained from the truncated SVD. The performance measure shows that

these statistically derived vectors are more robust indicators of meaning than individual terms. Latent semantic indexing (LSI) approach is based on singular value decomposition of ***tf-idf*** matrix. In order to determine similarity between a query and documents, SVD vectors of each document ***Di*** is used.

The disadvantage of this approach is as the documents size increased, the computational cost drastically increased [1].

Query expansion is another approach to enhance the performance of an IR system. In query expansion approach the terms in the query will be expressed into several ways (synonyms) and the system will add some more terms which make the idea in the query expressed well. Query expansion technique however is language and domain dependent [1].

Statistical base semantic information extraction is also another approach to enhance the performance of IR system. It applies statistical co-occurance analysis to get semantic relationship of documents. Information gain and mutual information are the main concept used in this approach to identify the most salient information. And the document can be conveyed in very important key terms and phrases. Using this approach text compression is also possible because redundancy of terms in text and relatively unevenly spread of important (salient) information in documents.[1][4]

The Ethiopian census conducted in 2008[38] indicated that Amharic is spoken by over 40 million people as a first or second language, making it the second most spoken Semitic language in the world (after Arabic), probably the first largely spoken language in Ethiopia. According to the census [38] over 55% of the population speak Amharic. Amharic has its own scripting style. Like any other language, documents written in Amharic are increasing drastically. In spite of these facts, Amharic language has very few computational linguistic resources. Because of this most researches start their work from scratch. For example Alemayehu, Willett and Fissaha works have been severely hampered by the lack of large-scale linguistic resources of the language [2][4]. With regard to Amharic information retrieval system, some prototype Amharic IR systems were developed. However; Google search engine can give Amharic document retrieval. But there are very less research works to make the Amharic search engine to forward in line with contemporary Information retrieval concepts.

1.1. Statements of the problem

According to Ethiopian census conducted in 2008 [38], the population of Ethiopia is estimated at over 73 million. There are more than 80 languages in Ethiopia and Amharic is the mother tongue language for more than 25 million people [38]. According to population census commission, it is the first language for more than 25 million people and second language for over 15 million people. This means over 55% of the populations speak Amharic [38].

As discussed in the background section the Amharic language has large collection of documents, in the form of books, magazines, newspapers, novels, official and legal documents, etc. This means that effective and efficient Amharic information retrieval system is essential. Previously, there were different efforts have been put to fill or mitigate the gap between the individual information needs and provision of information.

There were several research works done in Amharic information retrieval to enhance the effectiveness of the system. Bethlehem [19] proposed and developed N-gram-based automatic indexing for Amharic text retrieval to overcome the problem of having standard stemming procedure and stop word list for Amharic language. The researcher reported F-measure of 0.09.

Tewodros [23] developed latent semantic indexing (LSI) system to solve the problem of exact term matching retrieval system. It helps to retrieve documents which express the same concept though they don't have terms matched. He registered an F-measure of 0.60.

Tessema and Solomon [32] designed and implemented Amharic search engine. The researchers implemented a crawler, indexer and search engine components for Amharic IR. Vector space model has been used to compute similarity score documents and queries. The IR system developed by them registered F-measure of 0.68.

Abiy [34] studied on semantic query expansion to Amharic retrieval system to solve the problem of polysemy and synonymy exists in Amharic text and registered F-measure of 0.63.

Amanuel [31] tried to develop probabilistic information retrieval system to tackle the uncertainty problem in query expression observed in previous works using vector space model of Amharic IR system. His system has registered F-measure of 0.60.

Though different researches have been conducted to enhance the performance of the Amharic information retrieval system, they didn't properly address the semantics of the documents except Tewodros [23]. Tewodros tried to build semantic indexing using statistical approach. Statistical approach is unable to solve polysemy and synonymy accurately. So it needs to have combination of statistical and knowledge base approach to appropriately address the problem.

Effective information retrieval systems do not work on the documents themselves, but on (semantic) representations [3][4]. As discussed in the background section, one of the problems which make IR system to register less performance is not extract the semantics of the documents and queries [1]. Hence, in this study due attention have been given in semantic indexing and document clustering for Amharic IR. The techniques have been implemented so as to address the semantic problems exhibited in the Amharic IR system. Therefore we have the following research questions that need to be answered at the end of this research work.

- Could a statistical approach combined with a knowledge base approach yield better result in semantic index construction?
- Whether word level, phrase level or combination of both is preferable for semantic information abstraction of Amharic language text?
- Could semantic indexing improve the Amharic IR performance?
- Could document clustering make difference in the performance of IR system?

1.3. Significance of the Study

The output of this study can be applied in Amharic information retrieval system so as to improve the effectiveness. When the semantic indexing and document clustering being implemented in Amharic retrieval system, it helps information seekers to retrieve information which meet their needs. The system retrieves as per the information needs of the user by retrieving documents which have similar meaning to the query formulated by the user rather than exact terms match. In addition to that the computational cost of the system would be reduced. During the system looking for documents, the system not goes through the whole documents of the corpus instead it searches only from specific cluster. So that it reduced the computational cost by $\frac{1}{K}$ factor, where ***K*** is the number of clusters

Abstracting the semantics of the documents can also be applied to documents clustering. While documents are clustered, the main issue is proper extraction of semantics of the documents. If the semantics of the documents not extract well, it misleads the system to cluster the documents into incorrect cluster. Therefore, it has influential role in document clustering.

The other application area of the study is context aware sentiment/opinion mining. In context aware sentiment/opinion mining, semantics and the context of the sentiments have to be thoroughly distinguished. This study helps to identify the semantics of sentiments through identifying what are possible related concepts of the sentiments and what are not.

In addition to that document summarization is one of main research area where it needs this research output. As we all know summarization is the process of narrating the large text into smaller one. While doing that it has to have as much as possible semantics of the text accurately. If not, the summary will not able to abstract the right concept of the text and not perform well. Therefore our study will give important advantage to this study area to improve their performances.

1.4. Objectives

The general objective of this study is to investigate whether semantic indexing and document clustering techniques improve the performance of Amharic information retrieval.

Specific objective of the research include the following:

- To review related literature and prepare dataset.
- To design the architecture of the semantic indexing and document clustering based information retrieval system.
- To extract semantic of Amharic texts to index and document clustering.
- To explore and identify better document clustering algorithm.
- To setup experiment by organizing Amharic documents and queries.
- To evaluate the performance of the system using IR effectiveness measures such as recall, precision and F-measure.

1.5 Scope and limitations of the study

This study focused on Amharic language and aimed to improve the performance measure of the Amharic information retrieval system taking into consideration the semantics of the documents and document clustering. The IR system has been developed and tested using news articles found from the Ethiopian News Agency (ENA). The articles were identified and selected based on topic of the articles, articles talk about business, sport, politics, and social affairs were selected. Using these documents semantic indexing and document clustering based Amharic retrieval system was tested. The study basically encompasses the following main tasks, semantic indexing, documents clustering and search component.

Lack of large size tagged Amharic documents is the limitation of the study. And lack of full fledges Amharic knowledge base was also the limitations of study.

1.6 Methodology of the study

1.6.1. Study design

The researcher used an experimental research. The experimental approach involves identifying potential methods, and implementing and testing iteratively.

1.6.2. Literature review

In this study, related literatures that deal with information retrieval, document clustering and Amharic language were reviewed. In addition to that research works related with document clustering base information retrieval, Amharic information retrieval, and semantic text extraction were reviewed. In this regard books, journal articles, proceeding papers, internet were used as source of information.

1.6.3. Data source

Documents were collected for testing the system. Since very limited Amharic tagged documents were available, the researcher preferred to use the dataset used by previous research works. The data sources for this research work were tagged Amharic news collected from Ethiopian News Agency, which used by the previous research works. Most of the times news articles are

classified based on their topics. Articles for this study were selected on the base of their topics, which enable the researchers to cluster articles by topic.

1.6.4. Development tools

To prototype the proposed system the Python programming language was used. Because Python is an open-source object-oriented programming language that offers two to tenfold programmer productivity increases over languages like C++, Java, and C#. Python is also much more suitable to text operation and information retrieval. [35]

1.6.5. Test procedure

To test the system test queries were formulated and relevancy judgments were prepared. And documents query matrix constructed to find relevant documents for each test queries. To evaluate the effectiveness of the proposed system (i.e., the quality of its search results) the most common and basic statistical measures; recall, precision and F-measure were used. Precision is used to measures the number of relevant documents out of the retrieved documents. Recall measures the number of relevant documents retrieved out of the collection. F-measure is harmonic measure of precision and recall [1]. To visualize the performance of the system precision recall curve was plotted, which reflect precision at different standard recall levels.

1.7. Organization of the Research

This study is organized in to five chapters. The first chapter discussed about introduction of the study. It comprised background information, the problem statement, the research objective, research question, scope, significant of the study, and methodology. The second chapter is literatures review part. It covered review of philosophical grounds of IR, methods and techniques of IR system, essential of Amharic language, and research work related with cluster based IR system and Amharic IR system.

The third chapter presents the technique implemented, the architecture of the system and the algorithm used in this study. The experimental setup, test results interpretations and the findings of the experiment were presented in chapter four. Finally, chapter five presented conclusion drawn from the findings of the study and recommendations that should be considered in future researches for designing an applicable Amharic IR system.

CHAPTER TWO

LITERATURE REVIEW

2.1. INTRODUCTION

This chapter is broadly divided into three sections. The first section discusses about concepts related to IR, indexing, the series of activities involved in document and query indexing process, components of IR system and document clustering. Concepts like term extraction, semantics of the term and term weighting are briefly introduced. Evaluation of information retrieval system is also dealt with. The two conventional and most common measures of effectiveness of IR systems, recall and precision, are discussed in detail.

In the second section the features of Amharic writing system related to information retrieval is discussed. The Amharic alphabets, numbers and punctuation marks, morpheme are also introduced. In addition to that the parts of speech of Amharic language are discussed.

The third section discusses about research works related to cluster based IR, Amharic indexing, information retrieval.

2.2. INFORMATION RETRIEVAL, AUTOMATIC INDEXING AND DOCUMENT CLUSTERING

2.2.1 INFORMATION RETRIEVAL AND AUTOMATIC INDEXING

The need to store and retrieve written information became increasingly important over centuries, especially inventions like scientific paper. Soon after computers were invented, people realized that they could use the machine for storing and mechanically retrieving large amounts of information. In 1945 Vinegar Bush published a ground breaking article titled “As We May Think” that gave birth to the idea of automatic access to large amounts of stored knowledge. In the 1950s, this idea materialized into more concrete descriptions of how to archives the text could be automatically searched [2][1]. Several key developments in the field happened in the

1960s. Most notable was the works of Gerard Salton and his students; they were laid the foundation for retrieval system. They developed Retrieval system and formulated a technique to evaluate the IR system, which is using still today [2].

With the enormous increase of electronic information inside and outside of the industry and available on-line would consequent need for better techniques to access this information [3]. Information retrieval (IR) is process of looking for material (usually documents) of an unstructured nature that satisfies an information need of the user from within large collections (usually stored on computers) [1]. IR can also cover other kinds of data and information beyond textual document; it could be image, multimedia or other data. The main concept in IR is finding relevant resources on the net that satisfies user information needs. Computerized system which can perform information retrieval system is called information retrieval system [1].

2.2.1.1 Information Retrieval Process

Information retrieval has two basic processes; indexing and searching. The first process is before retrieval process can even be initiated, it is necessary to define document's representational terms set. These terms set comprise the logical view of the documents or abstraction of the whole document. With these term set indexing constructed. Indexing is a critical data structure; it allows fast searching over large volume of data. Different indexing structures might be used but the most popular and common one is inverted file. Once the indexing constructed, it is amortized by querying the retrieval system proceeds [4].

The second process of information retrieval is retrieval process, the user first specify information need which then parsed and transformed by applying text operation on it[4][9]. The query is then processed to obtain the retrieved documents. Fast query processing is made possible by with the aid of index structure [4].

2.2.1.2 Information Retrieval Model

The users of IR systems expect from IR systems to provide ranked list of documents. IR systems rank documents based on their usefulness to the user query. Most IR systems assign a numeric score to every document and rank documents by this score. Several models have been proposed

for this process. The three most commonly known models are the vector space model, the probabilistic models, and the inference network model [4].

Vector Space Model

In this model text is represented by a vector of terms from the documents and query. The term is either word or phrase. Words in the vocabulary and their respective document wills construct two dimensional matrixes, called a vector space [4]. Any text can then be represented by a vector in this two dimensional space. To assign a numeric score to a document retrieved by a query, the model measures the similarity between the query and vector created by vocabulary in the index versus documents in the corpus. The similarity between two vectors is once again not inherent in the model. Typically, the angle between two vectors is used as a measure of divergence between the vectors. The similarity can be measured using cosine similarity or dot product or others [4][2].

Probabilistic Models

This model is based on the general principle that documents in a collection should be ranked by decreasing probability of their relevance to a query or user information needs [4][3]. Probabilistic IR models estimate the probability of relevance of documents for a user query . The estimation is the essential part of the model, and this is a point where probabilistic models differ from one others [4][3].

The probabilistic models works on the following basic probability notation, the probability of relevance for document D is log

$$\overrightarrow{P(R/D)} \Big/ \vec{P}\left(\frac{R}{D}\right) \qquad 2.1$$

where (R/D) is the probability that the document is non-relevant [4]. This, by simple bayes transform, becomes

$$\log \overline{P}(R/D)P(R) \Big/ \overline{P}(R/D)P(R) \qquad 2.2$$

Inference Network Model

In this model, document retrieval is modeled as an inference process in an inference network. Most techniques used by IR systems can be implemented under this model [4]. In the simplest implementation of this model, a document instantiates a term with certain strength, and the credit from multiple terms is accumulated given a query to compute the equivalent of a numeric score for the document [4]. From an operational perspective, the strength of instantiation of a term for a document can be considered as the weight of the term in the document, and document ranking based on term strength of the documents [4].

2.2.1.3 Text preprocessing

During the preprocessing phase, someimportant text operations can be performed. It is performed to control the size of the vocabulary. Some of the major operations are[1]:

• Lexical analysis

• Elimination of stop words

• Stemming

• Thesaurus construction.

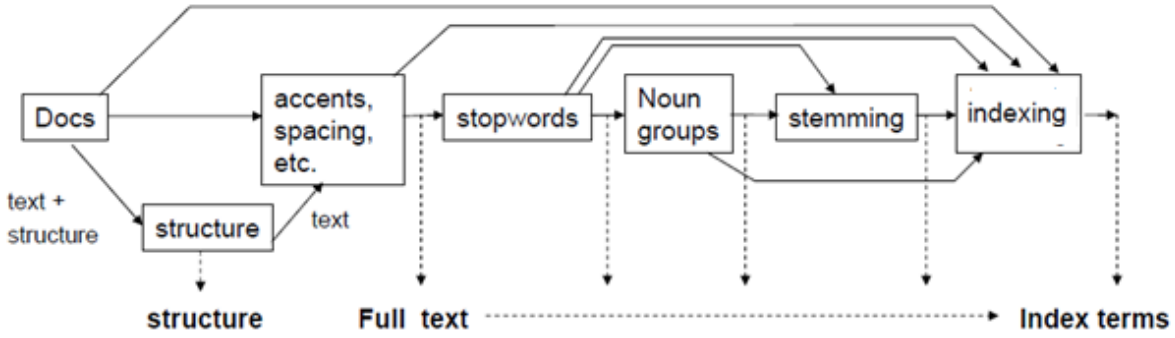

Fig 2.1: Generic IR system [1].

Lexical analysis is a process of collecting tokenized terms which needs some other operation on it and it may be used for indexing[1][2]. The tokens are taken from the document which may or may be word.

Elimination of stop words is the process of avoiding a word which has fewer relevancies to determine the content of the document. Sometimes, some extremely common words which would appear t o be of little value in helping select documents matching a user nee d are

excluded from the vocabulary entirely. In other word it is non content bearing words. In English language, these words are most of the time articles, prepositions, sometime conjunctions and others. Therefore elimination of stopping word is very critical task to control the size of the vocabulary [1][2].

Stemming is the process of finding the root word of the words or conflations. Conflation is occurrence of same word in little modification in its morpheme. For example a word nation can be written as nationalized nationality, national, and others. So this different form of the same word comes to its root term [1][2].

Thesaurus is knowledge base which describes the meaning of a word and finds the synonymy or polysemy of the word. In addition to that it can tell us the context to which the words used. Therefore constructing thesaurus has very important impact on the information retrieval of the effectiveness [1].

2.2.1.4 Automatic indexing

Automatic indexing is the ability for a computer to scan large volumes of documents against a controlled vocabulary, taxonomy, thesaurus or ontology and use those controlled terms to quickly and effectively index large document depositories. As the number of documents exponentially increases with the proliferation of the Internet, automatic indexing will become essential to maintaining the ability to find relevant information in a sea of irrelevant information.

There are two major approaches for the automatic indexing of text documents: statistical approaches that rely on various word counting techniques, vector space model used this approach. The aim of statistical indexing is to capture content bearing words which have a good discriminating ability and a good characterizing ability for the content of a document. Discrimination ability means that the words are able to distinguish documents from one another. And those terms are going to be counted and will have some weight [6].

The other approach is linguistic approach that involves syntactical analysis [5]. It is based on knowledge base of the language it is working on [7]. Therefore, it needs ontology of the language to determine the terms which describe a document. It can use the extraction of the

semantics the document using content bearing words extracted from document that going to be indexed [7].

Semantic indexing

To improve the performance of a traditional keyword-based search, a web documents should be represented with their concept rather than 'bag-of-words'. However; most of the previous works on indexing and information retrieval depend on lexical analysis and statistical methods. Using these techniques, it is difficult to abstract the semantics of the documents [20]. Typically, information is retrieved by literally matching terms in documents and in query. Since there are usually many ways to express a given concept (synonymy), the literal terms in a user's query may not match those of a relevant document. In addition, most words have multiple meanings (polysemy), so terms in a user's query will literally match terms in irrelevant documents. A better approach would allow users to retrieve information on the basis of a conceptual topic or meaning of a document [21].

Latent Semantic Indexing (LSI) tries to overcome the problems of lexical matching by using statistically derived conceptual indices instead of individual words for retrieval. LSI assumes that there is some underlying or latent structure in word usage that is partially obscured by variability in word choice. A truncated singular value decomposition (SVD) is used to estimate the structure in word usage across documents [21].

The other method to solve lexical matching problem is Lexical chains technique for the extraction of concepts from the document and represents by concept vectors. And then text vectors, semantic indexes and their semantic importance degree are computed. this indexing method has an advantage in being independent of document length because we regarded over all text information as a value 1 and represented each index weight by the semantic information ratio of overall text information[20].

2.2.1.5 Ontology

Ontology is a collection of concepts and their interrelationships which can collectively provide an abstract view of an application domain [7]. The use of ontology to overcome the limitations of keyword-based search has been put forward as one of the motivations of the Semantic Web since its emergence in the late 90's. While there have been contributions in this direction in the last

few years, most achievements so far either make partial use of the full expressive power of an ontology-based knowledge representation, or are based on Boolean retrieval models, and therefore lack an appropriate ranking model needed for scaling up to massive information sources[8].

The ontology (WordNet) lexical database is now quite large and offers broad coverage of general lexical relations in the natural language. It describes the relationship between the words in different situations. WordNet has been employed as a resource for many applications in natural language processing (NLP) and information retrieval (IR). Word relationships in the database are useful for NLP and IR applications, are not necessarily appropriate for a general, some time the WordNet can be domain-independent lexical database[9][2].

2.2.1.6 Semantic Index term extraction

Terms that are extracted from textual document are known to be an important and fundamental linguistic descriptor of documents [10]. Extracted terms are used for representing contents of specific. The terms extracted from the document can be used for indexing, it helps to distinguish a document from others [11]. In many cases, the terms that best describe the contents of a document are at the same time terminological units of the text's domain [11]. Extraction of key terms from the textual document for indexing purpose using computer is said to be automatic index text extraction. It is a basic requirement for many text related applications such as text clustering, indexing and others [10].

There are two main kinds of approaches to automatic index text extraction, statistical method and linguistic method [10]. Statistical method basically relies on word frequencies: words that are repeated frequently within a document are likely to be good descriptors of its content. On the other hand, terms that occur in several documents (like "the", "about" or "believe") do not have capability to distinguish one document from another [11]. It can be computed for a given term by multiplying its frequency in the current document (TF = term frequency) with its inverse document frequency (IDF) [11].

Linguistic method relies on syntactic criteria and do not use any morphological processes. Most researchers seem to agree that terms are mainly noun phrases to represent semantics of the text

[10]. Mostly noun compounds, including adjectives verbs and sometimes with very small proportions [10].

Furthermore, C-Value/NC-value method is a newly proposed approach other than the aforementioned approaches, which combines linguistic information and statistics. It is divided into two parts: 1) the C-value, that aims to improve the extraction of nested multi-word terms; and 2) the NC-value that incorporates context information to C-value method, aiming to improve multi-word terms extraction in general[10][3].

2.2.1.7 Term frequency and weighting

A document that mentions a query term more often has more to do with that query and therefore should receive a higher score. Thereof, assign to each term in a document a weight for that term that depends on the number of occurrences of the term in the document[1]. This concept can be applied through assigning the weight; it is equal to the number of occurrences of term t in document d. This weighting scheme is referred to as term frequency and is denoted ***tft, d***, with the subscripts denoting the term and the document in its order [1]. The exact ordering of the terms in a document is ignored but the number of occurrences of each term is material (in contrast to Boolean retrieval). We only tried to capture information concerning to the number of occurrences of each term. [1].

Inverse document frequency

It will create a problem when we make all terms are equally important; the impact would be explicitly viewed when it comes to assessing relevancy against a query [1]. In fact certain terms have little or no discriminating power in determining relevance. For instance, a collection of documents on the auto industry is likely to have the term auto in almost every document. To this end, we introduce a mechanism for attenuating the effect of terms that occur too often in the collection to be meaningful for relevance determination. An immediate idea is to scale down the term weights of terms with high collection frequency, defined to be the total number of occurrences of a term in the collection. The idea would be to reduce the tf weight of a term by factor that grows with its collection frequency [1].

document frequency of the term, ***dft***, defined to be the number of documents in the collection that contain a term t. its aim is to obtain document level statistics, which deals with the number of documents containing a term [1]. Using ***dft*** is difficult to measure the discrimination power of the term among the documents. To come over this problem, it better to use inverse document frequency, which is logarithmic function of total number of the document N and over dft[1]. It is defines as

$$\text{idf} = \log N/dft \qquad 1.3$$

Tf-idf weighting

Combining the definitions of term frequency and inverse document frequency helps to produce a composite weight for each term in each document. The ***tfidf*** weighting scheme assigns to term ***t*** in document ***d*** given by

$$tfidf_{t,d} = tf_{t,d} \times idf_t \qquad 1.4$$

$tfidf_{t,d}$ assigns to term ***t*** is a weight in document ***d*** that is

1. Highest when t occurs many times within a small number of documents (thus lending high discriminating power to those documents);
2. Lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
3. lowest when the term occurs in virtually all documents.

2.2.1.8 Query matching model

Comparison of query and document representations is very important task of IR system, it is performs in order to retrieve documents that are more similar to the specific query. The three classic models of Information retrieval: the Boolean model, the Vector space model and the probabilistic model are often used to accomplish this particular task [4]. The Boolean, vector space model and probabilistic models are briefly discussed below.

The Boolean Model

The Boolean model of Information retrieval is the oldest of the three classic retrieval models and it relies on the use of Boolean operators in combination with set theory [4]. A query is usually a Boolean expression of words input directly by the user. The query can be converted into a Boolean expression taking from a user's query free sentence. The terms in a query are linked together with AND, OR and NOT [12]. This method is often used in search engines on the Internet (e.g Google) because it is fast and can therefore be used online [12].

Unfortunately, the Boolean model has got its own drawbacks. It requires the users to have some knowledge of the search topic for the search to be effective [4]. A wrong word in a query could rank a relevant document non-relevant. In addition to that, all retrieved documents are considered to be equally important.

Another problem of the Boolean model is that, most users find it difficult to translate their information need into a Boolean expression [4]. This means most users need an intermediary to work with the Boolean retrieval model, which in turn brings its own problems, such as misunderstanding and possibly misrepresentation of the information needs of the user.

The Vector space model

In vector space model, a document, as well as a query, is represented as a vector of weights. A vector space is determined by all the index words selected from the entire document collection [12][7]. A value in a document (query) vector denotes the importance of the corresponding word in that document (query). In other words, given a vector space as follows [12][4]:

Vector space: $< t_1, \quad t_2, \ ..., \quad t_n >$

A document and a query may be represented as the following vectors of weights:

$$d \rightarrow <w_{d1}, \quad w_{d2}, ..., w_{dn}>$$

$$q \rightarrow <w_{q1}, \quad w_{q2}, ..., w_{qn}>$$

Where w_{di} and w_{qi} are the weights of t_i in document d and query q. Query matching involves measuring the degree of similarity sim(d, q) between the query vector q and each document vector d. Equation 2.5 is a formula of cosine similarity[12][8]:

$$\text{sim}(\boldsymbol{d}, \boldsymbol{q}) = \frac{\sum_{i=1,n} w_{di} * wqi}{[\sum_{i=1,n} w_{di}^{2} * \sum_{i=1,n} w_{qi}^{2}]^{1/2}} \quad 2.5$$

Again, the documents with the highest degrees of similarity are the answers to the query.

The Probabilistic Model

In the probabilistic model, the retrieval is based on the Probability Ranking Principle. The probability ranking principle asserts that the best retrieval effectiveness will be achieved when documents are ranked in decreasing order according to their probability of relevance [1].

Given a user query, the document collection can be divided in to two sets; a set which contain exactly the relevant documents and a set which contains the non relevant documents [1]. Hence, the querying process can be seen as specifying the properties of the relevant document set using index terms. However, since the properties of the relevant document set is not known in advance, there is always the need to guess at the beginning the descriptions of this set. The user then takes a look at the retrieved documents for the first query and decides which ones are relevant and which ones are not. By repeating this process iteratively the probabilistic description of the relevant document set can be improved [1].

The characteristic of the probabilistic model to rank documents in decreasing order of their probability of relevance to the user query is taken as the most important feature of the model[1][30]. However, lack of initial information on the relevant and non-relevant documents in the collection with respect to specific query; its ignorance to the frequency with which an index term occurs in a document and the assumption of index terms independence are taken as its major drawbacks[1][30].

2.2.2 DOCUMENT CLUSTRING

Nowadays the information on the internet is exploding exponentially through time, and much more information stored in the form of text. So text mining particular document clustering is

active research area and which is a major topic in the Information Retrieval community [13]. The search engines often returns more of pages in response to query, making it difficult for users to browse or to identify relevant information, though it lists in relevance order [13].

Document clustering methods can be used to automatically group the documents into different categories. Document clustering has been investigated for use in a number of different areas of text mining and information retrieval. Initially, document clustering was investigated for improving the precision and recall of information retrieval systems as well improves efficient retrieval system [14]. More recently, clustering has been proposed for use in browsing a collection of documents or in organizing the results returned by a search engine in response to a user's query [14]. Google is known to use clustering methods to match certain websites with a query, since a website can be viewed as a collection of topics, and a query itself is a topic or a combination of several topics [13]. Document clustering in IR system can be studied so as to optimize the retrieval process.

2.2.2.1 Document clustering techniques

The two well known and most common document clustering techniques are Agglomerative hierarchical clustering and K-means[14]. Agglomerative hierarchical clustering is often portrayed as "better" than K-means, although slower [14]. A document browsing system based on clustering uses a hybrid approach involving both K-means and agglomerative hierarchical clustering. K-means is used because of its efficiency and agglomerative hierarchical clustering is used because of its quality

Hierarchical techniques produce a nested sequence of partitions, with a single, all inclusive cluster at the top and singleton clusters of individual points at the bottom. Each intermediate level can be viewed as combining two clusters from the next lower level (or splitting a cluster from the next higher level). The result of a hierarchical clustering algorithm can be graphically displayed as tree, called a dendogram. This tree graphically displays the merging process and the intermediate clusters. The dendogram at the right shows how four points can be merged into a single cluster. For document clustering, this dendogram provides a taxonomy, or hierarchical index [14].

There are two basic approaches to generate a hierarchical clustering [14]:

a) Agglomerative: Start with the points as individual clusters and, at each step, merge the most similar or closest pair of clusters. This requires a definition of cluster similarity or distance.

b) Divisive: Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain. In this case, we need to decide, at each step, which cluster to split and how to perform the split.

Agglomerative techniques are more common, and these are the techniques that we will compare to K-means and its variants. We summarize the traditional agglomerative hierarchical clustering procedure as follows:

Simple Agglomerative Clustering Algorithm

1. Compute the similarity between all pairs of clusters, i.e., calculate a similarity matrix whose ijth entry gives the similarity between the ith and jth clusters.
2. Merge the most similar (closest) two clusters.
3. Update the similarity matrix to reflect the pairwise similarity between the new cluster and the original clusters.
4. Repeat steps 2 and 3 until only a single cluster remains.

In contrast to hierarchical techniques, partition clustering techniques create a one-level (un-nested) partitioning of the data points. If K is the desired number of clusters, then partition approaches typically find all K clusters at once. Contrast this with traditional hierarchical schemes, which bisect a cluster to get two clusters or merge two clusters to get one. Of course, a hierarchical approach can be used to generate a flat partition of K clusters, and likewise, the repeated application of a partition scheme can provide a hierarchical clustering. The bisecting K-means algorithm that we present later is such an approach [14].

There are a number of partition techniques, but we shall only describe the K-means algorithm which is widely used in document clustering. K-means is based on the idea that a center point can represent a cluster. In particular, for K-means we use the notion of a centroid, which is the mean or median point of a group of points. Note that a centroid almost never corresponds to an actual data point [14].

The basic K-means clustering technique is presented hereafter [14].

Basic K-means Algorithm for finding K clusters.

1. Select K points as the initial centroids.
2. Assign all points to the closest centroid.
3. Recomputed the centroid of each cluster.
4. Repeat steps 2 and 3 until the centroids don't change.

2.3 AMHARIC: ESSETIAL INTRODUCTION

2.3.1 Amharic language

According to Ethiopian census commission [38], Amharic is mainly spoken language in Ethiopia; which has an estimated of over 40 million speakers, it puts in second position as the most spoken Semitic language in the world (after Arabic) [38]. Amharic is related to Hebrew, Arabic and Syrian languages [16]. Amharic is the language for country-wide communication and was also for a long period the principal literal language and medium of instruction in primary and secondary schools of the country, while higher education is carried out in English [17]. The number of speakers of the language increased through time. The number of speakers of the language is on the rise for two reasons. First, it is the working language of the Federal Democratic Republic of Ethiopia, a country with more than 85 million people [CIA, 2008]. Second, unlike most other African languages, Amharic is a written language with its own alphabet and written materials, actively being used every day in newspapers and other media outlets [15]. It has its own script that is borrowed from Ge'ez, another Ethiopian Semitic language [16].

However, even under aforementioned favorable conditions, Amharic has been one of the under-resourced languages both in terms of electronic resources and processing tools. Availability computational linguistic resources for Amharic are very limited; despite Amharic language has wide speaker population, [17]. Recently, however, there have been independent attempts to make some progress in resource in the language [15]. It is observed that resources prepared in various industries in electronic form and are being made accessible [16].

2.3.2 Amharic morphology

Like other Semitic languages such as Arabic, Amharic exhibits a root-pattern morphological phenomenon. A root is a set of consonants (called radicals) which has a basic 'lexical' meaning [16]. Amharic has a rich verb morphology which is based on triconsonantal roots with vowel variants describing modifications to, or supplementary detail and variants of the root form [17]. In Semitic languages, words, especially verbs, are best viewed as consisting of discontinuous morphemes that are combined in a non concatenate manner. Put differently, verbs are commonly analyzed as consisting of root consonants, template patterns, and vowel patterns. With the exception of very few verb forms (such as the imperative), all derived verb forms take affixes in order to appear as independent words. In addition to this non-concatenative morphological feature, Amharic uses different affixes to create inflectional and derivational word forms [16].

Most function words in Amharic, such as conjunction, preposition, article, relative marker, pronominal affixes, negation markers, are bound morphemes, which are attached to content words, resulting in complex Amharic words composed of several morphemes [17]. Amharic adjectives share some morphological properties with nouns, such as definiteness, case, and number. As compared to nouns and verbs, there are fewer primary adjectives. Most adjectives are derived from nouns or verbs. Amharic has very few lexical adverbs. Adverbial meaning is usually expressed morphologically on the verb or through prepositional phrases. While prepositions are mostly bound morphemes, postpositions are typically independent words[17][48].

Amharic nouns can be inflected for gender, number, definiteness, and case, although gender is usually Neutral[17][106]. Nouns are derived from other basic nouns, adjectives, stems, roots, and the infinitive form of a verb by affixation and intercalation. For example, from the noun "ልጅ' 'child' another noun "ልጅነት" 'childhood'; from the adjective "ደግ" 'generous' the noun "ደግነት" 'generosity'; from the stem "ሰነፍ", the noun "ስንፍና" 'laziness'; from root "ከልድ", the noun "ቀላጅ" 'joke'; from infinitive verb "መስበር" 'to break' the noun "መስበሪያ" 'an instrument used for breaking' can be derived. Case, number, definiteness, and gender marker affixes inflect nouns [17][428].

Verbs are inflected for person, gender, number, aspect, tense and mood [20]. Other elements like negative markers also inflect verbs in Amharic [17]. Significantly large part of the vocabulary consists of verbs, which exhibit different morph syntactic properties based on the arrangement of the consonant-vowel patterns [17].

Atlachew et.al. [17] Constructed 65 rules for Amharic based on the entire Amharic morphology. The rules vary from simple affixation rules to each word category to allowed combinations of prefixes and suffixes for each word category and set of affixes [17].

2.3.3 Amharic Writing Style

2.3.3.1 Writing style

Written Amharic uses a unique script which has originated from the Ge'ez alphabet (the liturgical language of the Ethiopian Orthodox Church). Written Ge'ez can be traced back to at least the 4th century A.D. The first versions of the language included consonants only, while the characters in later versions represent consonant-vowel (CV) phoneme pairs[17]. In the modern Ethiopic script each syllable pattern comes in seven different forms (called orders), reflecting the seven vowel sounds. The first order is the basic form; the other orders are derived from it by more or less regular modifications indicating the different vowels. There are 33 basic forms, giving 7*33 syllable patterns (syllographs), or Fidels. Two of the base forms represent vowels in isolation, but the rest are for consonants (or semi-vowels classed as consonants) and thus correspond to CV pairs, with the first order being [17]. The writing system also includes four (incomplete, five character) orders of labialized velars and 24 additional labialized consonants. In total, there are 275 Fidels, but not all the letters of the Amharic script are strictly necessary for the pronunciation patterns of the spoken language; some were simply inherited from Ge'ez without having any semantic or phonetic distinction in modern Amharic [17].

There are many cases where numerous symbols are used to denote a single phoneme, as well as words that have extremely different orthographic form and slightly distinct phonetics, but with the same meaning. So are, for example, most labialized consonants basically redundant, and there are actually only 39 context independent phonemes (monophones): of the 275 symbols of the script, only about 233 remain if the redundant ones are removed. The script also has a unique

set of punctuation marks and digits. Unlike Arabic or Hebrew, the language is written from left to right[17].

2.3.3.2 Amharic Punctuations

There are about 17 Amharic marks, some of them which are commonly used and have representations in Amharic computing tools are listed as follow according to [Beletu (1982)] as it is cited in [19]: two square dots arranged like a colon (Y hulet netib) are word delimiters. The equivalent of a full stop is four dots arranged in a square pattern (YY arat netib). Some others include equivalents for the comma (. netela serez) and the semi-colon (. dirib serez), as well as a number of the borrowed symbols (? , ! , “ , ”, ‘, /, \, etc.). The word delimiter (two dots) is mostly used in handwritten text. It is more and more becoming the practice to exclude the word separator punctuation (two dots) from computer written text. The space is being used as word separator instead.

2.3.3.3 Numbering

Numbers in Amharic consist of single characters for one to ten, for multiples of ten (twenty to ninety), hundred, and thousand. (see Appendix 1 for the list).These characters are derived from Greek letters, and some were modified to look like Amharic Fidel[19]. Each of the symbols has a horizontal stroke above and below. There is no symbol for zero in the Amharic script. Thus, arithmetical computations using the symbols are very difficult, if ever done. As a result, people generally use the Hindu-Arabic numerals. Ethiopic numbers are used mostly in writing dates and page numbers in text.

2.4 RELATED RESEARCH WORKS

2.4.1 Chinese language research work

Jian Z. et.al[22] had observed in IR system for each individual query, different IR systems usually retrieve different sets of documents. The researchers proposed fusion and document clustering for IR system combining the result of different searching tools. Fusion is a technique that combines results retrieved by different systems to form a single list of documents. Different ranked lists usually have a high overlap of relevant documents and [22]. Clustering can be

performed on each ranked list obtained, and then identify clusters that contain more relevant documents. Documents that are relevant to the same query can be clustered together since they tend to be more similar [22].

In this novel approach, the researcher used three basic methods, combMAX, CombSUM, and CombMNZ to cluster Chinese language documents collected from Xinhua News Agency. The methods consist of three steps. First, they cluster each ranked list. Then they identified the reliable clusters and adjust the relevance value of each document according to the reliability of the cluster. Each cluster will have 1000 documents of similar size.

To identify reliable clusters, we assign each cluster a reliability score. According to the Fusion Hypothesis, we use the overlap between clusters to compute the reliability of a cluster[22]. Each original ranked list has been adjusted by clustering and re-ranking. We now combine these improved ranked lists together, it is called fusion [22]. Combined relevance of document d is the sum of each adjusted relevance values that have been computed some formula stated in the paper[22].

The experimentation carried out in this research has shown some result. The system which built using fusion and clustering of document could return, first, they studied three methods, namely, CombMAX, CombSUM, and CombMNZ. Their fusion results are calculated on the same data set and the average precision of the seven retrieve results is 0.3086. The cluster size of the document has not impact on the performance of the system. The researchers conclude that combining multiple retrieval results is certainly a practical technique to improve the overall performance of an information retrieval System. The results showed that this approach brings some improvements and has favorably resulted. They also investigated the impact of cluster size. They found that their approach is rather stable over the change size of cluster.

2.4.2 English language research work

Xiaoyong L. [37] Previous research on cluster-based retrieval has been inconclusive as to whether it does bring improved retrieval effectiveness over document-based retrieval. Recent developments in the language modeling approach to IR have motivated them to re-examine this

problem within this new retrieval framework. They propose two new models for cluster-based retrieval and evaluate them.

The study used cluster based document clustering model, it viewed as a mixture model of three sources: the document, the cluster/topic the document belongs to, and the collection. A relevant document assumed being generated by this mixture model. Both partitioning and hierarchical agglomerative clustering algorithms have been studied in the context of IR. They used a three-pass K-means algorithm as an example of partitioning methods in their static clustering experiments, primarily motivated by its efficiency.

To experiment their work they used over six data sets taken from TREC. Two sets of experiments are performed in this study. The first set of experiments investigates whether a simple language model of clusters can be used to rank clusters. And the second set of experiments examines the effectiveness of cluster-based retrieval using our CBDM model in the context of query likelihood retrieval and the relevance model (RM), for both static clustering and query-specific clustering. Both experiments have given promising result to clustering based retrieval is more effective than the traditional research engine.

2.4.2 Local Research Work

There were several research works done in Ahmaric information retrieval to enhance the retrieval performance of the system. Some of research work conducted regarding with Amharic information retrieval and indexing are titled with N-Gram based automatic indexing for Amharic text, Amharic text retrieval using latent semantic indexing with singular value decomposition, design and implementation of Amharic search engine, semantic based query expansion technique for Amharic IR, and probabilistic information retrieval system for Amharic language, they are discussed hereafter based on their chronological order.

Bethlehem [20] conducted a research which entitled with N-Gram based automatic indexing for Amharic text. She aimed to make Amharic indexing more efficient than the previous works. She tried to apply N-Gram based indexing method to index the Amharic texts. Since N-Gram based indexing is a language-independent method for automatic indexing that it did not make use of tools like; stemmers, stop-word lists, thesauri. The method used in this research could contribute

to the simplification Amharic information retrieval efforts and it improved the effectiveness of the retrieval.

She design and develop a system that implement the concept of N-gram to index Amharic document for a purpose of information retrieval. Basically she tried to use word's bi-gram and try-gram tokens for indexing. The weighting scheme she employed was ***tfidf*** and the similarity measure between queries and the documents could be computed using cosine similarity method.

She performed experiment to check N-gram based indexing for Amharic corpus. For the purpose of her research, she prepared test data set consisting of 100 short news articles (obtained from Walta Information Center and used by previous research) and 24 queries. The average document length was 179 words. The average query length, on the other hand was 5 words. To evaluate the performance of the system, she used the most widely and commonly used evaluation parameters, recall and precision. Of the three indexing methods, the word-based retrieval is shown to be more effective than the bi-gram and the tri-gram-based retrieval. This means word level indexing has relatively better recall and precision in the same query than bigram and trigram indexing. However, the bi-gram and tri-gram indexes still have comparable performance than word level indexes. This happen due to the larger number of documents retrieved using bi-grams and tri-grams than in the case of words.

Bethelehem[20] concluded that Although the N-gram method has incurred storage requirement and processing time, it still offers one method of indexing that is divorced from language dependent and domain specific lists (e.g. stop-word lists, thesauri, etc.) and rules, which makes the method attractive. And she recommends try to use very large corpus and look what happen. The other recommendation is she used bigram or trigram it would be better when we use higher gram. In addition, the type of N-grams used for this experiment is overlapping N-grams. None overlapping N-grams may be tested in further research.

Tewdoros [23] tried to observe and overcome the major problems exhibited in Saba and Bethelems works. As he stated Amharic language lexical variation is very common as Beder[1976] as it was cited [23]. As study indicated Amharic speaking regions have different types of lexical variations. Some words are found to be used with different meaning in different regions . For instance, the word 'zämän' is used to mean 'year' in Gojjam, where as in Addis

Ababa and its environs it is used to mean ‘long period of time’. Some genuine dialectal difference between pairs of synonyms is also observed, in which one of the synonym is used to the exclusion of the other in a given area.

These characteristics of the language causes inconvenience for exact term match retrieval systems, such as standard vector space and Boolean models, as different people with different background, different needs or different linguistic habit use different words to express the same concept. There are problems of polysemy and synonymy in the language and it caused to register less performance in retrieval process.

Tewodros [23] used Latent Semantic Indexing (LSI) to overcome some of the deficiencies of exact term matching retrieval systems and provides a way of dealing with synonymy and polysemy automatically without the need for manually constructed thesaurus, a grammar, or ontology according to [Berry et.al] as it was cited by [23] .

The researcher pursued three successive phases to develop a latent semantic indexing model to make the mode suited with Amharic documents. In in the first phase preprocessing (extracting terms, term document matrix generation and calculating term weighting) and indexing were included. The next phase could be performing K-dimensional Singular Value Decomposition (SVD). Finally, Query Projection, Matching and Ranking of Relevant documents

The system evaluated by the two commonly known parameters, recall and precision. The average precision for the LSI approach (0.7157) and the average precision of the vector space approach (0.6913) with 206 news articles and 9256 indexing terms. This represents a 2.4% improvement over the standard vector space method by Bethlehem [20].

The researcher recommends that if the size of the document increased, it would have a chance to improve the performance of the IR system. Furthermore the index terms used in the system were not stemmed, if they could be stemmed it will have possibility of enhancing the IR system performance.

Abiy [24] tried to articulate the problem that was clearly seen in the Amharic IR system, specially the impact of polysemy and synonymy of terms in the documents. Amharic has a rich verb morphology which is based on triconsonantal roots with vowel variants describing

modifications to, or supplementary detail and variants of the root form [17]. As a result of that different users may describe their information needs in different way, for that matter their queries may differ lexically. Therefore to overcome this problem the researcher proposed semantic based query expansion.

Abiy [24] has used text preprocessing to make avail the terms in the document to indexed in efficient manner. Vector space model was devised to the IR system developed by him. The expansion of the query given to the system based on the knowledge base of Amharic language or thesaurus.

In his study the performance was evaluated using the usual IR system measurement criteria, recall and precision. The performance of the system measured compared with standard vector space model.

The performance of the system registered recall of 0.92 and precision of 0.68. These shown us query expansion in Amharic IR system registered better recall but the precision gone down.

This research focuses on semantic indexing and documents clustering for Amharic documents to improve the performance of Amharic IR system based on the recommendation of previous researchers and research works in other languages.

CHAPTER THREE

SYSTEM DESIGN

3.1. INTRODUCTION

This chapter discusses the design and development of semantic indexing and document clustering system for Amharic text retrieval. The peculiar characteristics of Amharic language and writing system in relation to text retrieval as well as basic concepts in the C-value method and K- means document clustering are discussed in depth in this chapter.

Various researchers had conducted researches to tackle problems exhibited in Amharic text retrieval and to enhance the performance of the retrieval system as discussed in section 2.4.2. The researches tried to overcome the problems of indexing using N-gram[20], and semantic indexing using statistical method(LSI) [23], searching components problems using query expansion[24] and using probabilistic methods of Amharic IR [31]. Though these and others research were conducted, still there are several problems with regard to Amharic IR and the best result of achieved was F-measure of 60%. This is because problem related with extract semantics of the documents. Plus to that inefficient way of search engine strategy makes the system to have less performance [2][2]. In this research the problems stated above are handled and tried to solve.

3.2. SYSTEM ARCHITECTURE

The development of an IR system involves various techniques and methods. Basically the generic information retrieval system incorporates two major components: indexing and searching [4]. However, semantic indexing and document clustering for Amharic text retrieval system has three major components; semantic indexing, document clustering, and search sub components. Under each components there are number of constituent elements so as to make it works well. These components collaborate and integrate together to provide better effectiveness for the

whole retrieval process. The components are discussed in detail afterward. The architecture are depicted in Fig 3.1.

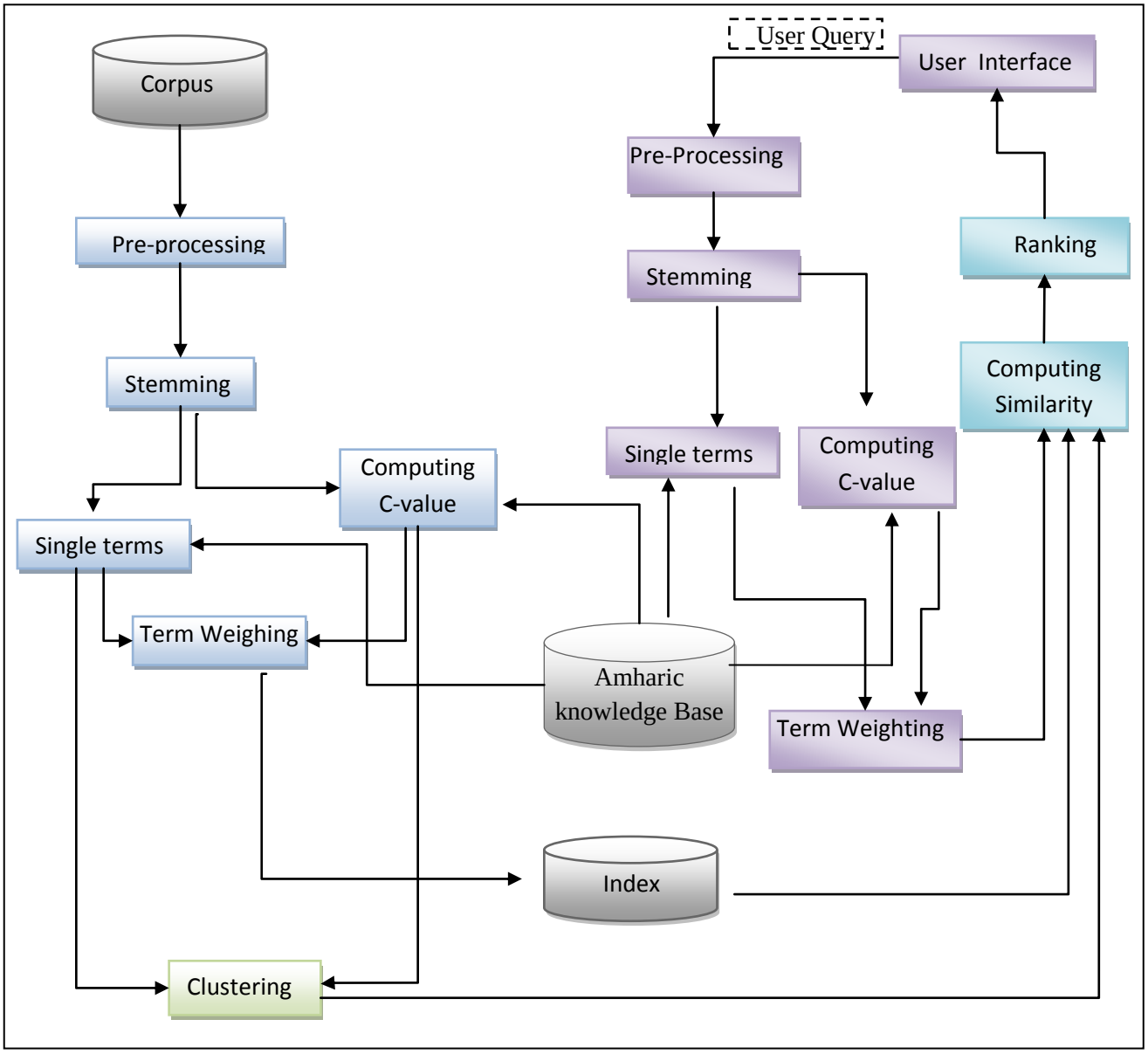

Fig 3.1: Architecture of the system

3.3 CORPUS PREPARATION AND DATA PREPROCESSING

In Information retrieval system corpus can be used for evaluation of the system. Since very limited Amharic tagged documents were available, the researcher preferred to use the dataset used by previous research works. The data sources for this research work were tagged Amharic

news collected from Ethiopian News Agency, which used by the previous research works. Most of the times news articles are classified based on their topics. Articles for this study were selected on the base of their topics. The dataset used by this research contains articles talk about politics, social affairs, sport and business which enable the researchers to cluster articles in accordance with their topics. These topics were selected because there are more Amharic news related with these topics than any other.

Several preprocessing computation were applied on the collected documents to make them readily available to be being indexed and searched. After the document identified and collected the data preprocessing tasks are followed. The major preprocessing tasks are discussed hereafter.

3.3.1 Tokenization

Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called tokens, perhaps at the same time throwing away certain characters, such as punctuation or non content bearing words [2]. These tokens are often loosely referred to as terms, but it is sometimes important to make a token distinction. A token is an instance of a sequence of characters in some particular document that are grouped together as a useful unit for processing the document [2]. The tokens obtain from chopping the texts up are labeled with the grammatical tag of the language since the texts used by the system are tagged documents.

Like English Amharic tokens are strings separated by punctuation or space but in Amharic there are different punctuations which separate the tokens. Punctuation in Amharic language not same as English as discussed in section 2.3.3.2 there are more punctuations not included in English like ፡,፣,፨ they have different meaning in sentence.

3.3.2 Identification of Semantics Terms

The tokens set generated from tokenization process is labeled with grammatical tag. The documents used by the system are tagged documents which have eight tag groups. Hence each term has fallen down into one of the tag group. All terms in the token set are not employed by the prototype system instead the terms which are semantic descriptor of the document are identified and extracted. Nouns and verb of the documents have the capability to describe the semantic of the documents [2]. Therefore nouns and verbs of the token set are distinguished and

extracted identify and the multi word terms are constructed using them. The terms both single and multi word are indexed.

3.3.3. Normalization

Having broken up our documents (and also our query) into tokens, we get lists of tokens having lexical difference. However, there are many cases when two character sequences are not quite the same but you would like a match to occur. Normalization is the process of making token to come to uniform writing format so that matches occur despite superficial, hyphen, abbreviation, and equivalent characters differences in the character sequences of the tokens.

In Amharic language there are different characters having the same sound/meaning and different persons use different characters to write same concept or words, for example to say 'stolen' it can be written as 'ፅ[k' or 'W[k', to say 'Thursday' it can be written as 'GS<e' or 'NS<e'. These terms have lexical difference but they have similar meaning and sound. The lexical differences of the same terms are resolved in this step. In addition abbreviations are going to be translated into extended writing form.

3.3.4 Stemming

For grammatical reasons, documents are going to use different forms of a word. There are families of derivational words with similar meanings, for instance words have same root 'democrat' but have different morphologically variant, 'democracy', 'democratic', and 'democratization' [2]. The goal stemming is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Morphological variant words have similar semantic interpretations. In IR system, there are words considered as equivalent words [26]. As a result words have to be reduced to their root using stemming. Beside to that, stemming also used to reduce the dictionary size (i.e. the number of distinct terms used in representing a set of documents). The smaller the dictionary size results the smaller storage space and processing time required.

Stemming techniques are language dependent. Therefore, every language is using its own specific stemming technique. In Amharic text, there are many word variants/affixes [25]. To conflate them into stem word, stemming technique/ algorithm developed by Nega were used. He

developed the stemmer that involves the removal of both prefixes and suffixes and also consider letter inconsistency and reiterative verb forms [25].

According to Nega [26], a stemmer that stems words without consideration of remaining stem, which removes prefix and suffix list, a simple removal of affixes from terms is said to be context-free stemmer. For instance, English word “regular” and “metal” will be “gular” and “met‟ respectively if context-free stemmer removes “re” and “al” . Similarly, weak result will appear if such technique is applied for Amharic language. For example, for Amharic word “¡M¡M” which means forbid, “ŸKŸK<›†¨<” which means forbid them and “ŸKŸK<›ƒ” which means forbid her, if we remove string ħ “ke” which is found in prefix list, the result will be “KŸK<›ƒ” which does not have any relation with the correct stem. Therefore, context-sensitive approach should be used with affix removal being controlled by two action codes and five conditions [25]. But context aware not consider in this research rather we mainly focused on prefix and suffix removal.

The five conditions are:

C1: Characters following the assumed affix (i.e. if the assumed affix is followed by:). This is used to cater for words where vowel elision occurs at morpheme boundaries in the case of prefixing, and to reduce the removal of terminal consonants in the case of suffixing.

C2: Characters preceding the assumed affix (i.e. if the assumed affix is preceded by :). This is used to avoid the removal of terminal consonants that are not genuine suffixes.

C3: If the assumed affix is part of an identified consonants structure this is used to avoid the removal of non-genuine affixes from specific words and their variants.

C4: A minimum stem-length condition for the specified affix. The stem length includes both vowels and consonants and the condition is used to reduce over stemming.

C5: A combination of the above

The following table shows some of Amharic language affixes adopted from Naga

Prefix lists					Suffix lists				
የ	ስለ	የሚ	እየ	ስለሚ	ች	ኝ	ችን	ቸው	ዊት
እያ	እንደ	አል	አለ	በ	ና	ዎች	ኛ	ዎቻቸው	ው-ም
ለ	ከ	ይ	እንዳ	ሲ	ው	ዎች	ው-ያን	ዎቹ	ናቸው
እንዲ	እስክ	ከነ	እን	እነ	ባቸው	ዊያን	ነት	ያዊ	ን
					ት	ሉ	ችው	ዊ	ዊቲ
					ቹ ን	ዬ	ዎ	ህ	ሽ
					ዋ	ሁ	ለት	ላት	ላቸው
					ላችሁ	በት	ባት	ባቸው	ባችሁ
					ቱ	ሽ	ይቱ	የው	ኞች

Table 3.1 : Prefix and Suffix list

Step 1: Get WORD and count the number of radicals (or consonants)

Step 2: IF number of radicals < 3 THEN stop and return WORD

Step 3:IF length (Word) < -(MAXPL + 2) THEN assign the first length (WORD) -2 Characters to a temporary prefix, TEMPP

ELSE assign the first MAXPL characters to TEMPP (where MAXPL is the Length of the longest prefix in the list)

Step 4: Find TEMPP in the prefix list

Step 5: IF found THEN

IF context sensitive THEN check the context-sensitive rules cl-c5

Apply al or a2 and GOTO step 7

ELSE GOTO step 6

ELSE

IF length (TEMPP) >1 THEN assign length (TEMPP) -1 Characters to TEMPP and GOTO step 4

ELSE return WORD

Step 6: Remove prefix

Step 7: IF number of radicals of WORD >3 or length (WORD) > 4 THEN GOTO step 1 ELSE GOTO step 2

Algorithm 3.1: Naga **[26]** Prefix remover algorithm

Step 1: Get WORD and count the number of radicals (or consonants)

Step 2: IF number of radicals < 3 THEN stop and return WORD

Step 3: IF length (Word) < -(MAXPL + 2) THEN assign the last length (WORD) -2 Characters to a temporary prefix, TEMPP

ELSE assign the first MAXPL characters to TEMPP (where MAXPL is the Length of the longest prefix in the list)

Step 4: Find TEMPP in the prefix list

Step 5: IF found THEN

IF the ending required recoding THEN recode and GOTO step 1

ELSE

IF context sensitive THEN check the context-sensitive rules C1- c5 apply a1 or a2 and GOTO step 7

ELSE GOTO step 6

ELSE

IF length (TEMPP) > 1 THEN assign length (TEMPP) – 1 characters to TEMPP and GOTO step 4

ELSE return WORD

Step 6: Remove prefix

Step 7: IF number of radicals of WORD >2 THEN GOTO step 1

Algorithm 3.2: Nega [26] Suffix remover algorithm

Based on the prefix and suffix list defined in the Table 3.1 words taken from the corpus and user query are going to be conflated to their stem using the Algorithm 3.1 and Algorithm 3.2. The order of removal of the suffix and prefix do have some leverage on the stem word that in this research prefix removal process precede the suffix.

3.3.5 Removal of Stop Word

Some extremely common words which would appear to be of little value or relevance in discrimination documents are removed from token set. These words are called stop words. The general strategy for determining a stop list is to sort the terms by collection frequency (the total number of times each term appears in the document collection), and then to take the most

frequent term to remove. In addition the other method is using stop word list for the language itself. Some of the design of modern IR systems has focused precisely on how we can exploit the statistics of language so as to be able to cope with common words in better ways[2][24].

In English language we can predetermine some of the stop words like articles and preposition can be categorized under stopping words [2], whereas in Amharic language stopping words are very large in number and difficult to predetermine them [26]. Stop words more of dependent on the domain that the document deals with. Stop words in Amharic that are categorized as news specific are those terms which are frequently used by Journalists and reporters while reporting some news to the public. For example, 'inform', 'notify' or 'reported' (ኅÓxªM, ÑMëªM, ›e}¨<qªM, ›dexªM, ›w^\, ›e`Æ , ›eSeÓxªM and etc.). Similarly, terms that are frequently used in the text and have no specific relation with the content discussed in the document are categorized as common/stop words [2].

Some of Amharic language stop words presents in the table below.

Stop Word Lists									
ነው	እኔ	እኛ	እነሱ	እሱ	እሷ	አንተ	እናንተ	እና	ወደ
ነይ	ወይ	ከ	ናቸው·	ትናት	ጥቂት	በርካታ	ብቻ	ሁሉም	ሌላ
ለሉኞች	ሁሉ	እያንዳንዱ	እያንዳንዳቸው	ስለ	እንዲሁም	አንጂ	ደግሞ	መካከለከ	ሰሞኑን
ከሰሞኑ	በሰሞኑ	የሰሞኑ	ትናንት	ትናንትና	ጋራ	የጋራ	ከጋራ	ተሰያዩ	ተሰያየ
ድረስ	እስከ	በጣም	ግን	ሲሆን	ሲል	ውስጥ	ላይ	ናት	ነበሩ
ነበረች	ያ	ወይዘሮ	ወይዘሪት	ነገሮች	ከፊት	ከላይ	ታች	ከታች	በታች
የታች	በውስጥ	ከውስጥ	ጋር	ናቸው	ይህ	በላይ	ወደ	ወዘተ	እና
ወይም	እንደ	አሉ	ፊት	ወደፊት	ነገር	በፊት	በብላ	በኩል	

Table 3.2 : some of Amharic stop words

3.4 INDEXING

3.4.1 Inverted File

The major concept in information retrieval is how the documents/corpus are going to be represented in information retrieval system [25]. This logical representation of documents using its content bearing words is inverted file. An Inverted index always maps back from terms to the parts of a document where they occur. Inverted index, or sometimes inverted file, has become the standard term in information retrieval [25]. The basic idea of an inverted index is a dictionary of terms (sometimes also referred to as a vocabulary or lexicon). Each item in the list – which records that a term appeared in a document is conventionally called a posting. The list is then called a postings list (or inverted list) [25].

So as to have fast retrieval time we need to build the index in well organized and structured manner [25]. The advance index structure or inverted file list that we have will have substantial impact on the performance of the whole retrieval system. The major steps in creating posting files/inverted file are the following [25].

i. Corpus preparation
ii. Data preprocessing
iii. Multi word terms extraction
iv. Term weighting and constructing posting file

The first two, corpus preparation and data preprocessing tasks were discussed in section 3.3. The third step of indexing process is multiple words term extraction. Extraction of the multiple words terms basically used the output of the tokenization process. At the end of text tokenization process, it yields tokens of each document. There by the Amharic words which describe concept of the document have been obtained through this process. But there are words created by combination of multiple words. When we see the atomic words separately it gives different meaning whereas when combine the two or more words in together it will give some other meaning. For example '**ሰማይ ጠቀስ**' when we look at the words independently the word '**ሰማይ**' is sky and the word "**ጠቀስ**" is citing but when we combine the word together "**ሰማይ ጠቀስ**" it is long. The other is "**የአካል ብቃት እንቅስቃሴ**", "**የአካል**" is body , "**ብቃት**" is fitness and "**እንቅስቃሴ**" is exercise, but when we look at the three words in together it is sport.

C-value technique is used to identify and extract multi words terms. The technique is discussed in section 3.4.2. Therefore, this research mainly focused on how Amharic multiple words terms obtained and to be indexed.

The last task performed during indexing is term weighting and creation of inverted file. The weighting scheme used for in this research work is ***tfidf***, it is discussed in section 2.2.1.7. To index the documents we needs to have semantically extracted terms, either single or multi words terms, obtained from each document and the weight of each terms. Inverted file consists of a dictionary or postings as main building elements. Each postings has a unique serial number, and each postings has document identifier and their weight (docID) [25]. The input to indexing is a list of semantically extracted tokens for each document. And the tokens/terms are going to be sorted based on their alphabetical order. The list has to get hold of the frequency of the terms appears in the document or term weighting as well. Generally the list is split into two lists, vocabulary and post file [25].

3.4.2 Multi words terms extraction

Terms, the linguistic representation of concepts, are important elements for textual information [29][1]. For a rapid changes in many specialized knowledge domains (particularly in areas like computer science, engineering, medicine etc.), means that new terms particularly multi word concepts are being created all the time[29].

There can be various approaches to extract multi words from the corpus; some are statistical method like the likelihood ratio for terms consisting of multi words, based on mutual information. The other technique is based on linguistic patterns analysis. And there is the combination of statistical methods and linguistic method that is C-value method.

The C-value method is used in this research work to extract multiple words terms from the corpus. The C-value method extracts multi word terms from the corpora combining linguistic and statistical information filtering together. The C-value aims to improve the extraction of nested multi-word terms. The C-value method takes Amharic tagged corpus as input and produces a list of candidate multi word terms generated by combining the nouns and verb [29].

The C-value approach combines linguistic and statistical information, emphasis given on the statistical part. The linguistic information consists of the part-of-speech tagging of the text. using

part-of-speech tag information linguistic filter performed. The statistical part combines statistical features of the candidate string, in a form of measure frequency. According to [29] C-value method describes and justifies the linguistic part and the statistical part of the method.

3.4.2.1 The linguistic part

The linguistic part consists of the following:

1. Part-of-speech information from tagging the corpus.
2. The linguistic filter applied to the tagged corpus to exclude those strings not required for extraction.
3. The stop-list

Tagging.

The corpus prepared to be used by the prototype system is tagged documents. It is discussed in the section 3.3.

The linguistic filter

The tagged set of the terms are noun phrases, adjectival phrases, verbal phrases, etc. Taking this all to extract multi word terms without linguistic filter would have problem. The reason is that the statistical information without any linguistic filtering is not enough to produce content bearing terms set results. Without any linguistic information, undesirable strings such as stopping words and other undesirable part of the document would also be extracted. Since most terms consist of nouns and verb, linguistic filter is used to generate valid and content bearing term [29].

The choice of the linguistic filter affects the precision and recall of the output list. A number of different filters have been used, opened filter and closed filter. A `closed' filter which is strict about the strings it permits, will have a positive effect on precision but a negative effect on recall [29]. This filter only permits sequences of nouns, as a result produces high precision since noun sequences in the corpus are the most likely to be content bearing terms. At the same time, it negatively affects recall, since there are many noun compound terms that consist of adjectives and nouns, which are excluded by this filter. An `open' filter, permits more types of strings, has the opposite effect: high recall but low precision. In this research work the closed filtering is used because I need to get the semantic of the document that words other than nouns or verb would be discarded.

Stop word

The stop words removal process is discussed in section 3.3.5. Dropping out the stop words from the multi word terms will increase effectiveness in retrieval process.

3.4.2.2 The statistical part

The C-value statistical measure assigns a termhood to a candidate string, ranking it in the output list of candidate terms. Termhood is a likelihood of the multi word terms being valid term in vocabulary of the language. The measure is built using statistical characteristics of the candidate string. The parameters used to compute the statistical characteristic of the termhood are[25][3]:

1. The total frequency of occurrence of the candidate string in the corpus.
2. The frequency of the candidate string as part of other longer candidate terms.
3. The number of these longer candidate terms.
4. The length of the candidate string (in number of words).

We will look at each of the parameters turn by turn. The frequency of occurrence of the candidate string in the corpus is the number of times the candidate string appears in the corpus. In this case, the termhood of a string equals its frequency of occurrence in the corpus. It will be stated in equation 3.1

$$termhood(a) = f(a) \qquad 3.1$$

a is the candidate string,
f(a) its frequency of occurrence in the corpus

The frequency produces good results since terms tend to occur with relatively high frequencies [25][3]. For example if we have some document which talks about pregnant, it can be multiple word term in Amharic "**ነብሰ ጡር**" which has 1000 terms, among them if "**ነብሰ ጡር**" appears 130 times, it could be a good descriptor of the document and the strings are multiple words term.

In candidate string, there could be some other candidate string inside of the candidate string. It is called nested candidate string [25]. For example "የአካል ብቃት እንቅስቃሴ" is multiple words term, in this candidate string there could be some other candidate string "የአካል ብቃት".

Therefore the nested candidate string or the substring is "የአካል ብቃት". Its substrings, "የአካል ብቃት" , would be also extracted since they would have frequencies at least as high as "የአካል ብቃት እንቅስቃሴ" .

The solution to this problem is to extract only a substring of a candidate term if it appears a sufficient number of times by itself in the corpus (i.e. not only as a substring).. Though the substring appears in the document they could be multiple words if they can be appears in the vocabulary of the language. In our case not all substring for that matter, longest substrings are not be right multiple words term if they didn't defined in knowledge base of the Amharic language. In order to calculate the termhood of a string, we should subtract from its total frequency its frequency as a substring of longer candidate terms. The formula is written in the equation 3.2

$$\textbf{termhood(a)} = \textbf{f(a)} - \sum_{b \epsilon T} \boldsymbol{f}(\boldsymbol{b}) \qquad 3.2$$

a is the candidate string,
f(a) is its total frequency of occurrence in the corpus,
Ta is the set of candidate terms that contain a,
b is such a candidate term, ***f(b)*** is the frequency of the candidate term ***b*** that contains ***a***

The second parameter is the frequency of the candidate string as part of other longer candidate terms. Nested terms appear within other longer terms, and may or may not appear by themselves in the corpus. The terms that are substrings do not have to appear by themselves in a text. As a result, a measure like equation(equ) 3.2 would exclude terms if these have been only found as nested, or if they are not nested but present a very low frequency and at the same time extract those substrings that are terms

The third parameter is the number of these longer candidate terms it the number of occurrence of the substring terms in different longest candidate strings. One substring may appear in different candidate string as sub string.

The last parameter in the C-value measure is the length of the candidate string in terms of number of words. Since it is less probable that a longer string will appear f times in a corpus than a shorter string1, the fact that a longer string appears f times is more important than that of a shorter string appearing f times. For this reason, we incorporate into the measure the length of the candidate string.

Since the maximum length terms cannot be nested in longer terms, and some strings are never found as nested anyway, we distinguish two cases [25][4].

1. If ***a*** is a string of maximum length or has not been found as nested, then its termhood will be the result of its total frequency in the corpus and its length.
2. If ***a*** is a string of any other shorter length, then we must consider if it is part of any longer candidate terms. If it appears as part of longer candidate terms, then its termhood will also consider its frequency as a nested string, as well as the number of these longer candidate terms. Though the fact that it appears as part of longer candidate terms affects its termhood negatively, the bigger the number of these candidate terms, the higher would be its independence from these. This latter number moderates the negative effect of the candidate string being nested in longer candidate terms

The measure of termhood, called C-value is given as

$$\textbf{C-value}(a)=\begin{cases} \log_2|a|.f(a), & if\ a\ is\ not\ nested \\ \log_2|a|.f(a)-\frac{1}{p(Ta)}\sum_{b\epsilon Ta}f(b), & otherwise \end{cases} \quad (3.3)$$

Where,

a is the candidate string,

f(.) is its frequency of occurrence in the corpus,

Ta is the set of extracted candidate terms that contain ***a***,

P(Ta) is the number of these candidate terms.

3.4.2.3 C-value algorithms

In this subsection all steps in the C-value method to extract list of valid multiple words terms from a corpus are discussed as follow.

Step 1 – part of speech tagging

Step 2 – extract candidate string

This stage extracts those strings that satisfy the linguistic filter and frequency threshold. The multi words terms that going to be indexed will be extracted from among these strings. The process of finding this maximum length is as follows: We attempt to extract strings of a specific length. If we do not find any strings of this length, we decrease the number by 1 and make a new attempt. We continue in this way until we find a length for which strings exist. At the end, extraction of the candidate strings can take place.

Step 3 - assign C-value for candidate strings

This is the stage where the C-value for each of the candidate strings is evaluated. C-value is calculated in order of the size of the strings, starting with the longest ones and finishing with the bigrams. The C-value for the longest terms is given the top branch by equation Equ.3. 3.

We set a C-value threshold, so that only those strings with C-value above the threshold set are added onto the list of candidate terms. For the evaluation of C-value for any of the shorter strings, we need two more parameters (their frequency as part of longer candidate terms, and the number of these longer candidate terms).

$$\text{Threshold} = \sqrt{2\boldsymbol{Maxtf}} \qquad 3.4$$

Where ***Maxtf*** = maximum term frequency in the document

To obtain these two parameters, we perform the following:

For every string ***a***, that it is extracted as a candidate term, we create for each of its substrings ***b, a*** triple (***f(b); t(b); c(b)***).

f(b) is the total frequency of b in the corpus,

t(b) is the frequency of b as a nested string of candidate terms,

c(b) is the number of these longer candidate terms

In order to calculate C-value for a string ***a*** which is shorter by one word, we either already have for it a triple (***f(a)***; ***t(a)***; ***c(a)***) or we do not. If we do not, we calculate the C-value from the top branch of formula 3. If we do, we use the bottom branch of Equ 3.3. In that case, ***P(Ta)*** = ***c(a)*** and $\sum_{b\ e\ T}$ = ***t(a).***

After the calculation of C-value for strings of length k finishes we move to the calculation of C-value for strings of length k-1. This way it is evident whether the string to be processed has been found nested in longer candidate terms. At the end of this step, a list of candidate terms has been built. The strings of the list are ranked by their C-value.

for all strings ***a*** of maximum length
    calculate C-value(a) = $\log_2|a|$. ***f(a)***;
    if C-value(***a***) >=Threshold
        add ***a*** to output list;
        for all substrings ***b***
            revise ***t(b)***;
            revise **c(b)**;
for all smaller strings ***a*** in descending order
    if ***a*** appears for the first time
        C-value(a) = $\log_2|a|$. ***f(a)***;
    else
        C-value(a) = $\log_2|a|$. ***f(a)*** - $\frac{1}{c(a)}$t(a)
        if C-value(a) >=Threshold
            add ***a*** to output list;
            for all substrings b
                revise ***t(b)***;
                revise ***c(b)***;

Algorithms 3.3: the C-value algorithm [29]

3.5 DOCUMENT CLUSTRING

3.5.1 Document Clustering for Information Retrieval

Data mining is a technique helps find the pattern from hidden information. This technique is to find and describe structural patterns in data collection as a tool for helping to explain that data and make predictions from it. Generally, data mining tasks are divided into two major categories: predictive tasks which aim to predict the value of a particular attribute based on the values of other attributes and another one is descriptive tasks which aim to derive patterns. Clustering is a method to organize automatically a large data collection by partition a set data, so the objects in the same cluster are more similar to one another than to objects in other clusters. Document clustering is a fundamental task in text mining that is concerned with grouping documents into clusters according to their similarity [30].

The cluster hypothesis states, documents in the same cluster behave similarly with respect to relevance to information needs. The hypothesis states that if there is a document from a cluster that is relevant to a search request, then it is likely that other documents from the same cluster are also relevant. This is because clustering puts together documents that share many terms [25].

Search service suffer from two major problems, the first problem is low precision, which is due to the irrelevance of many of the search results. Retrieving more none relevant information made the information seeker difficult to find the relevant information. The second problem is low recall, which is due to the inability to index all the information available on the Web [30]. Document clustering has always been used as a tool to improve the performance of retrieval

Most of document clustering algorithms use the vector space model (VSM) for document similarity measurement. VSM represents documents as vectors in the space of terms and uses the cosine similarity between document vectors to estimate their similarity.

As mentioned before the corpus is the tagged Ethiopian news agency news articles. The articles are clustered into four clustered, politics, socio economy, sport, business. Therefore any article taken from the corpus is going to be fallen under any of the cluster with its nearest neighbor.

There are various techniques and algorithms in the document clustering; among them the researcher use k-means techniques. The K- means is used because linear space and time complexity and more popular in document clustering.

3.5.2 K-means Algorithms

The K-means partitioning clustering algorithm based on the concept of nearest neighbors. This is done by considering the term frequencies and document frequencies and preparing a document summary for each cluster containing the distinct terms whose frequencies are high after preprocessing of the documents. This is done to measure the document relevant score.

The steps involved in document clustering are:

Step 1 : Document preprocessing

Step 2: Extracting content bearing terms

Document summary is prepared with the terms with highest frequency for each document. The summary of the document is collection of important terms which capable to describe the document.

Step 3: apply semantic synonyms

Semantic synonyms of extracted terms in the document summary are carried out using Amharic knowledge base. The Amharic knowledge base prototype which contains thirty to fifty terms developed by the researchers themselves and used by this research work.

Step 4: implement K-means

Implement K-means clustering algorithm by term-document vector as representative of the document collection and using Cosine similarity as replacement of Euclidean distance.

$\overline{X}_1 \ldots\ldots \overline{X}_N$ corpus documents set

K is the number of cluster

K-means($\{\overrightarrow{x_1}, \ldots, \overrightarrow{x_N}\}$, K)

1 $(\vec{s}_1, \vec{s}_2, \ldots \vec{s}_k) \leftarrow$ SelectRandomSeeds ($\{\overrightarrow{x_1}, \ldots, \overrightarrow{x_N}\}$, K)

2 ***for*** K $\leftarrow$ 1 to K

3 ***do*** $\vec{\mu}_k \leftarrow \vec{s}_k$

4 While stopping criterion has not been met

5 ***do*** for k $\leftarrow$ 1 to *K*

6 ***do*** $w_k \leftarrow \{\}$

7 ***for*** *n* $\leftarrow$ 1 to *N*

8 ***do*** *j* $\leftarrow$ arg $min_{j'} |\vec{\mu}_k - \vec{x}_k|$

9 *Wj* $\leftarrow$ *wj* $\cup \{ \vec{x}_n \}$ *(reassignment of vectors)*

10 ***for*** k $\leftarrow$ 1 to *N*

11 ***do*** *k* $\leftarrow \frac{1}{w_k} \sum_{\vec{x} \in w_k} \vec{x}$ (recomputation of centroid)

12 ***return*** $\{\vec{\mu}_1, \ldots, \vec{\mu}_K\}$

Algorithms 3. 4: k-means algorithm [3]

3.6 SEARCH STRATEGY

After the corpus is indexed and the documents are clustered into four clusters, the similarity between query and the documents has to compute. The user information needs will be formulated using combinations of terms. This type query formulation given to the retrieval system is said to be free text query [25]. In this research the query given to the search system is in the form of free text query.

3.6.1 Vector Space Model

In this concept we developed the notion of a document vector that captures the relative importance of the terms in a document. The representation of a set of documents as vectors in a common vector space is known as the vector space model. It is fundamental to find relevant documents from the collection by calculating the similarity between query and documents in the collection.

V (d) the vector derived from document ***d***, with one entry in the vector for each dictionary term. Unless otherwise specified, the reader may assume that the entry holds the value of ***tfidf*** weighting. The set of documents in a collection then may be viewed as a set of vectors in a vector space, in which there is one axis for each term [25].

To compute the similarity between two documents in this vector space, the magnitudes of the vectors differ from the two document vectors will be consider. The standard way of computing the similarity between two documents ***d1*** and ***d2*** is cosine similarity.

$$\text{Sim (d1,d2)} = \frac{\overline{V}(d1).\overline{V}\,(d2)}{\left|\overline{V}\,(d1)\right|\left|\overline{V}\,(d2)\right|} \qquad 3.5$$

Where, the numerator represents the *dot product* (also known as the *inner product*) of the vectors ***V (d1)*** and ***V (d2)***. The denominator is the product of their *Euclidean lengths*. The dot product vectors $\boldsymbol{x \cdot y}$ of two vectors is defined as$\sum_{i=1}^{m} \boldsymbol{xiyi}$. Let ***V (d)*** denote the document vector for ***d,*** with ***M*** entries ***V1(d) . . VM(d)***. The Euclidean length of ***d*** is defined as $\sqrt{\sum_{i=1}^{m} Vi^2}$ (***d***).

3.6.2 Cluster Based Searching

The similarity between query and documents can be computed using cosine similarity measure. In vector space model the documents and query are represented as vector that similarity measure can be done in between documents and query vectors to find relevant document. Whereas, in this proposed system, it first computes the similarity measure of the query with cluster representative document. It helps to indentify to which cluster the query closer or which cluster contains more relevant documents to the query. The representative of the document is the centriod of the cluster that has proximity to more documents in the cluster are higher.

To find to which cluster the query is more similar, the cosine similarity of the query vector with vectors of the representative documents can be computed. And one which has high similarity measure will be the cluster in which relevant documents are resided in. And then it finds the ranked list of relevant documents from specified cluster.

To retrieve the ranked relevant documents from the cluster, we need to have prepared vector space of documents which grouped in the specified cluster. Then the vector space of each and every document which come under the identified cluster constructed and the other documents

should be rejected since they are irrelevant document to the query. Then cosine similarity will be calculated between the documents vectors against the query. And the document which has highest cosine score will be the first relevant document out of the corpus and based on the score the documents listed down.

3.7 IR SYSTEM EVALUATION

Retrieving relevant document from the collection that satisfies users information need is the heart of IR system evaluation. Two strategies are identified in measuring the effectiveness of IR systems. The first is the user-centered strategy, which uses relevant judgment has been made by the user so as to evaluate the performance of the system and the second is system centered strategies which work based on reference judgment available prior to testing process [25]. Based on the concept of relevance (i.e. given query or information need), there are several techniques of measures of IR performance available, such as, precision and recall, F-measure, E-measure, MAP (Mean average precision), R-measure, etc [25]. In this study, the three widely used techniques precision, recall, and F-measure are used to measure the effectiveness of the IR system designed

Precision and recall are the two most frequent and basic statistical measures. Recall is the percentage of relevant documents retrieved from the corpus in response to users query and precision is percentage of retrieved documents that are relevant to the query [1]. To show these metrics, assume the document collection be ***D***. Let ***Rt*** is all retrieve documents from the collection ***D*** and ***Rl*** a number of relevant documents in ***D***. The joint of ***Rt*** and ***Rl*** is a set of documents retrieved and relevant, ***RA***. Therefore, the recall and precision can be calculated using equation 3.4. and 3.5 respectively.

$$\text{Recall}=\frac{|\{\boldsymbol{relevant\ documents} \cap \{\boldsymbol{retrieved\ documents}\}|\}}{|\{\boldsymbol{relevant\ documents}|} \qquad 3.6$$

$$\text{Precision}=\frac{|\{\boldsymbol{relevant\ documents}\} \cap \{\boldsymbol{retrieved\ documents}\}|}{|\{\ \boldsymbol{retrieved\ documents}|} \qquad 3.7$$

A measure that combines precision and recall is the harmonic mean of precision and recall, the traditional F-measure or balanced F-score [1]:

$$\text{F-measure} = \frac{2(recall * precision)}{recall + precision} \qquad 3.8$$

The formula show above for precision and recall assume that, all documents retrieved ($\boldsymbol{R_t}$) is examined by user. Thus, the retrieved document cannot be presented for the user at once. Rather, the retrieved documents are presented according to their degree of relevance as per the user query. Then, the user examines the ranked documents starting from the top. This kind of examination of documents by the user leads the recall and precision measures to vary. Therefore, for appropriate evaluation of recall and precision, plotting a precision versus recall curve is necessary [25]. To draw precision-recall curve, for example let documents retrieved by the system is $\boldsymbol{D_r}$ Where $\boldsymbol{D_r}$ = {$\boldsymbol{d_3, d_{33}, d_9, d_1, d_{10}, d_{11}, d_{14}, d_7, d_{44}, d_{49}, d_{50}, d_{53}, d_{55}, d_{77}, d_{133}}$} in ranked order. Assume $\boldsymbol{R_q}$ contain a set of relevant documents for the query. Where, $\boldsymbol{R_q}$ = {$\boldsymbol{d_1, d_3, d_7, d_{10}, d_{14}, d_{33}, d_{44}, d_{49}, d_{55}, d_{133}}$}. Based on the given information recall- precession curve is constructed. However, when constricting the curve based on the original recall and precession may result in saw tooth curve, there is a need to smooth the curve using interpolation technique.

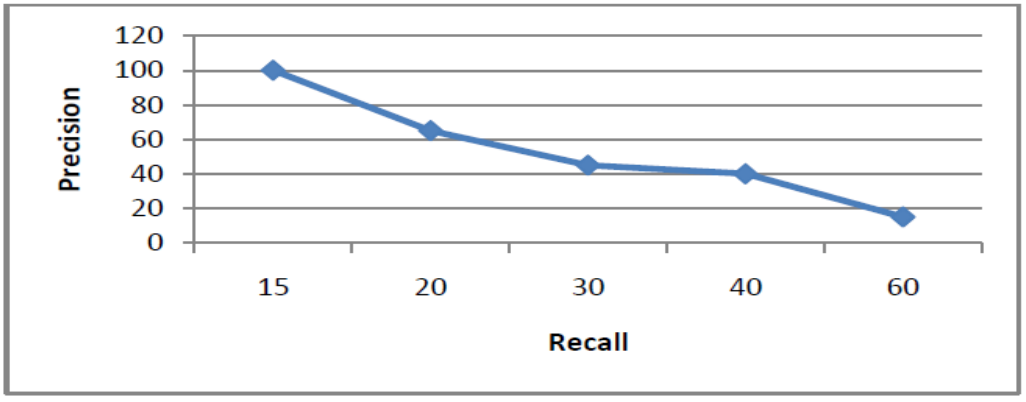

Fig 3.2 recall versus precision curve

Recall and precision is the evaluation technique of the proposed information retrieval system. The clustering methods used by the system to group similar documents also be measure through effectiveness of the retrieval system.

The performance of clustering algorithm being used in this research going to be measured using well known measures; inter cluster measure and intra cluster measure. Good cluster algorithm has small inter cluster score or the documents in the cluster are closest each other. However, the intra cluster score should be large; it means similarity score of one cluster representative to other should be large or representative documents are far away each other.

To calculate the inter cluster measure of the cluster the following formula used.

$$ICS(\pi(D)) = \sum_{i=1}^{|\pi(D} 3.8 \quad {}_{d,d' \in ci} similarrity(d, d') \qquad 3.8$$

To calculate the intra cluster measure of the clusters the following formula will be used

$$ECS(\pi(D)) = \sum_{i=1}^{|\pi(D)|} \sum_{j=i+1}^{|\pi(D)|} \frac{1}{|C_i||C_j|} \sum_{d \in C_i, d' \in C} \qquad 3.9$$

CHAPTER FOUR

EXPERIMENTATION

4.1. EXPERIMENTAL SETUP

In this research attempts have been made to design and implement a semantic indexing and document clustering system for Amharic information retrieval. As discussed in previous chapters, the system aims to overcome the semantic problems exhibited in indexing and similarity measurement. As a result, it will improve the performance of the Amharic IR.

The system has three basic components; indexing, clustering, and searching sub parts. Inverted file indexing structure is used to organize documents and index them so as to make the documents suitable to search. K means clustering is the clustering algorithm being used in the system. Each components of the system is evaluated.

To test the performance of the prototype system, 85 tagged Amharic news articles collected from Ethiopian news agency were used as a document corpus, which was used by previous research work. The size of the documents is 650 KB. The articles cover politics, social affairs, business and sports. Table 4.1 presents the distribution of the documents by topic category.

No	Type of news	Number of document
1	Politics	21
2	Social	23
3	Business	21
4	Sport	20
	Total	85

Table 4.1: Type and numbers of news articles used

The news articles are all in .txt format, which is supported by most programming language. 16 test queries were systematically selected by the researchers to test the performance of the system. The selection was made based on equivalent distribution of queries over the clusters. Subjective

relevance judgment is also done for identifying which document is relevant for a given test query.

4.1. DESCRIPTION OF THE PROTOTYPE SYSTEM

The prototype system development begins with constructing inverted file or vocabulary file, which is usually called indexing. Once the inverted file is constructed, the query will be supplied to the system. The system process the query by performing preprocessing tasks, identifying multi word terms, finding synonymous terms of each term, and calculating term weighting. Preprocessing tasks were filtering nouns and verbs in the query string, stemming, stop word removal, and normalization. Multi word terms identification it will filter out a valid multiple words terms from the query. After content bearing terms, either single or multi words terms, are identified the synonymies are extracted later. In addition, each content bearing terms and synonymies terms need to incorporate weight, ***tfidf*** term weighting has been employed. The posting file structure is shown as follow:

[term1], [[document_name,weight],[document_name,weight]…….]
[term2], [[document_name,weight],[document_name,weight]…….]
[term3], [[document_name,weight],[document_name,weight]…….]

Table 4.2: posting file structure

The system has been built using python version IDLE 3.3.1. Figure 4.1, presents a screen shot which shows the first list of retrieved document using a given query.

```
እባኮን  የሚፈልጉትን  ፋይል  ለማግኘት  መጠይቆን  ያስገቡ!:-የኢሃዴግን ደረጅታዊ ጉባኤ

=========================  የፍለጋው ውጤት እደሚከተለው ነው  ================================


cosine simmilarity with  p10.txt is  0.9720260707418382
cosine simmilarity with  p14.txt is  0.9273861281028285
cosine simmilarity with  p18.txt is  0.8656445530511221
cosine simmilarity with  p5.txt is  0.8883723910625765
cosine simmilarity with  p6.txt is  0.9720260707418382
cosine simmilarity with  p7.txt is  0.8656445530511221
cosine simmilarity with  p8.txt is  0.9788564609699942
cosine simmilarity with  poletic1.txt is  0.9457445528338374
cosine simmilarity with  poletics2.txt is  0.8128942795459233
cosine simmilarity with  poletics3.txt is  0.920970069027512
1 p8.txt
2 p10.txt
3 p6.txt
4 poletic1.txt
5 p14.txt
6 poletics3.txt
7 p5.txt
8 p18.txt
9 p7.txt
10 poletics2.txt
>>>
```

Fig 4.1 : A Screen shot of retrieved document for a given query

One of preprocessing task is tokenization. The tokenization of tagged documents in the corpus and query string is implemented in the python 3.3.2. The python source code of tokenization hereafter.

```
word= params
 try :
      stopw = open("stopword.txt", "r",encoding='utf8')
      i=0
      for line  in stopw.read().split():
            if word==line:
                  i=i+1
                  return 0
                  break
      if i==0 :
            return word
finally:
      stopw.close()
```

Fig 4.2: python code for tokenization

After the tagged documents tokenized, the system extracts the nouns and verbs of the document from the token set to keep the semantics of the document. On the semantics set of the token set three major activities carried out; stemming, stop word removal, and normalization. Stemming can be done using Nega's Amharic stemmer algorithm but the algorithm used by this research

removes only prefixes and suffixes but not infix and it was not context aware. The affixes list prepared by Nega[26] was list down in Table 3.1**.** The algorithms described in Algorithm 3.1 and Algorithm 3.2 implemented in python as follow.

```
def stemmer(parms):
   stem = []
   stem= parms
   #PREFIX RMOVAL
    if len(stem)>=3:
        try:
            prefix = open("prefix.txt", "r",encoding='utf8')
            for line  in prefix.read().split():
                  prefixsize= len(line)
                  tempPrefix=stem[:prefixsize]
                  if tempPrefix==line:
                              stem= stem[prefixsize:]
         finally:
                 prefix.close()
         # SUFIX REMOVAL
        try:
               suffix = open("suffix.txt", "r",encoding='utf8')
                for line  in suffix.read().split():
                        suffixsize= len(line)
                        tempsuffix=stem[-suffixsize:]
                        if tempsuffix==line:
                                #print (line + "suffix stemmed")
                                 stem= stem[:-suffixsize]

        finally:
                prefix.close()

   stem=normalization(stem)
   return stem
```

Fig 4.2: Python code of Amharic stemmer

The stop word removal could be done based on Nega's stop word list stated in Table 3.2. It removes the stop words from the sub set of the token set, as a result token set would have only content bearing terms. Finally normalization performed so that it made terms to be appearing or written in uniform format. Amharic language character variant which has same sound was described in section 3.3.3 being implemented in prototype system.

Prototype system would able to identify valid multi words terms out of the text using C-value technique. Identification of multi word terms had two major filtering mechanisms, statistical and linguistic filtering, as discussed in chapter two. At the end the process extract a valid multi word terms with a length of two and three words. Multi word terms identification performed with the help of a Amharic knowledge base, the a prototype knowledge base establish for testing of the system which has thirty to fifty vocabulary terms. The python source code of multi word terms identification and extraction attached in Appendix IV.

Terms that were in the documents or in the query needed to get hold of weight with respect to their documents. The weight can be calculated using ***tfidf*** term weighting methods. The term weighting was done by finding of the frequency of terms and its synonyms in the documents and product of its inverse document frequency. The most important thing was once weight of a term was calculated its synonymy terms weight never been calculated because its frequency already add up to fore coming synonymy term. The python source code is hereafter.

```
for terml2 in validMterm[n:]:
        ss1=" ".join(terml1[0])
        ss1.strip()
        dword=[]
        #print(ss1)
        dic=open("dictionary.txt", "r",encoding='utf8')
        for dwords  in dic.read().split("."):
            xx=[]
            for wordl in dwords.split(","):
                world="".join(wordl)
                world=world.strip()
                #print(world)
                if world not in xx:
                    xx.append(world)
            if ss1 in xx :
                #print(ss1,terml2[2])
                dword.extend(xx)
                dword.remove(ss1)
        #print(dword)
```

```
        for t in dword:
            if t not in dicwords:
                dicwords.append(t)
        dic.close()
        if terml2[0]==terml1[0] :
            #print(dword)
            doc= doc+ 1
    k= terml1[1]
se = open("stemmedfile/"+ terml1[2], "r",encoding='utf8')
#print(terml2[2])
for t2 in se.read().split():
    if t2 in dword :
            k=k+1
#print(k,dword)
se.close()
 n=n+1
tfidf=k*math.log(i/k,2)
doclist.append(terml1[2])
doclist.append(str(tfidf))
```

Fig 4.3 : python code for ***tfidf*** term weighting

Document clustering was one of the sub components of the prototype system. K-means clustering algorithm was used for document clustering. As discussed in the section 3.5 the documents are expected to cluster into four clusters; business, politics, social, and sport clusters. The prototype system performs clustering by taking some inputs from the user. The inputs were strategically selected document which represent the cluster better than the other. The representative documents were four, one for each cluster. Thereby those documents which have higher similarity score with the representative document should come under that representative document cluster. These clusters were required by searching sub component of the prototype system. Each cluster would be text file containing document names of cluster element. The cluster text file will be saved with its representative document file name. In each saved cluster name there are document names belongs to that cluster. Python source code attached in appendix V.

A search sub component is used for retrieving relevant document out of document corpus with a given query. After preprocessing applied on the query, the system finds the synonymies of the query terms from knowledge base and assigns equal term weight to the synonymy terms. As shown in the architecture of the system in Fig 3.1, the relevant documents could be retrieved with two steps. In the first step, similarity measure in between the query and the representative documents of the document clusters could be calculated. The higher similarity score with cluster representative imply the document under that cluster was more relevant than the others, thereby it could be selected for further processing. Second step was calculating the similarity score of each document in the selected cluster to the query document and sort the documents with descending order of their score. After all relevant documents would be displayed according to their rank as shown in Fig 4.1.

4.2 PERFORMANCE EVALUATION

As discussed in section 3.7, Precision, Recall and F-measure are the most frequent and basic statistical measures which widely used to measure the effectiveness of IR systems (i.e., the quality of the search results). These three parameters are used in this research so as to measure the effectiveness of the designed IR system.

Before the whole system is tested we need to evaluate the performance of the cluster algorithm used in the system and preprocessing module components as well stemmer. As it discussed in section 3.5.2 the K-means algorithm has been employed in order to group documents into four categories. How correctly the documents in each cluster grouped have to be evaluated. As described in the section 3.7 the evaluation methods are inter cluster and intra cluster measures.

Cluster name	Number correctly clustered documents	Number of incorrectly clustered documents	Expected number of correct documents	Inter cluster measure
Social	15	7	23	0.69
Politics	16	4	21	0.81
Sport	18	3	20	0.88
Business	17	6	21	0.71

Table 4.3 statistics of clustering algorithm

The average inter cluster measure (ICS) is 0.77
Intra cluster of the algorithm (ECS) is 0.82

Some documents are clustered incorrectly because even though the document talks about other issue the terms in the document are more close to incorrect cluster representative. So that it fall under incorrect cluster.

Under preprocessing module there are tokenizer, stemmer, and multi word extraction are evaluated. The documents are tokenized based on the space in between terms in the texts. The expected output of the tokenizer is tokens separated by space. From the documents that have been tokenized 34581 tokens were expected but the system yields 34569. The system has above 99.4 % accuracy. The difference comes due to some words in the documents are not separated by space, especially, the tagging text in some part of the document not have space from the nearby words.

Stemming can be done after semantic filtering was performed. There are 10083 semantically extracted tokens in the corpus. 413 terms were under stemmed and 375 were over stemmed. There were 788 terms that had stemming problems, 82.3% accuracy achieved. The basic problems with the stemmer were the fact that Amharic is a complex language and the stemmer. As well as the affix list was not also exhaustive.

Multi word terms extraction module expected to yield collection of multi word terms from the corpus. The system could give the right multi word terms if and only if they satisfied the two filtering criteria, statistical and linguistic filtering. Therefore there is no any short come in the system side but the main problem is in knowledge base used in the system. Since we are using prototype knowledge base having very limited words, perhaps, a valid multi word terms might not exist in the knowledge base. This might be the cause not to extract the valid multi word terms from the corpus. With the prototype knowledge base the system extracts 10 multi word terms out of 13 terms. This means it performs with 76% accuracy.

Before testing the system, sixteen Amharic test queries were systematically selected. The queries are selected based on their frequency in the corpus. Most frequent and less frequent bigram and trigram terms in each cluster are identified.

The most and less frequent terms of trigram and bigram were selected after semantic extraction and stemming was performed. And the frequency can be measured with in a cluster. Actually, there were more terms which have maximum or minimum frequency but researcher selected first term. Table 4.5 shows less and high frequency of trigram and bigram.

Sport cluster					
Grams	No of grams	Frequency		Terms	
		less frequent	most frequent	less frequent	most frequent
Trigram	3256	1	5	እግር ኳስ ቡድን	አካል ብቃት እንቅስቃሴ
bigram	3279	1	5	ሜዶ ወድድሮ	ሀይሌ ገብረስላሴ
Social cluster					
trigram	2351	1	3	ደረጃ ትምህርት ቤቶ	ህክም ገልገያ ሳሪያ
bigram	2376	1	6	ሆስፒታል ዶክተር	ሁለተ ትምህርት
Politics cluster					
Trigram	2244	1	3	አሠዴግ ድርጅታ ጉባኤ	ፖለቲካ ድርጅ ሂደት
Bigram	2263	1	4	ፖለቲካ ህይወት	ኤርትራ ህዝብ
Business cluster					
Trigram	2054	1	4	ኢትዮጵያ ንግድ ባንክ	ክሬዲት ካርድ አገልግሎት
bigram	2076	1	8	ኢንቨስትምንት ንግድ	ኢትዮጲያ ንግድ

Table 4.4: bigram and trigram query

The subjective relevance judgment is prepared to construct document query matrix that shows all relevant documents for each test queries. Using these queries the performance of the whole system were measured by precision, recall and F-measure. Highly frequent and less frequent query strings were tested independently. Highly frequent trigram and bigram query strings were tested and then the average of each test queries was represented the performance registered by the system using frequent query strings. In similar manner less frequent query strings tested and average of each test queries was represented the performance registered by the system using less frequent query strings. The average of high frequent and less frequent query test gave the performance of whole system.

Computing average performance over a set of queries each with a different numbers of relevant documents, precision and recall curves which reflect precision at different standard recall levels from 0 to 1 inclusive in step of 0.1 is plotted to draw the curve

Multi word term was considered as single term in the sentence even though the term contains two or more words , for example, when we query the system using a multiple term "ነብሰ ጡፍ" the system look for the document containing this term as it was, and the synonymous terms; either multiple or single word terms. The following table shows retrieved when we query "ነብሰ ጡፍ እና እናቶች " and its synonyms "እርጉዝ እና እናቶች"

Query	Retrieved documents	Relevant documents
ነብሰ ጡፍ እና እናቶች	doc2,sp5,doc2	doc1,doc2
እርጉዝ እና እናቶች	doc2,sp5,doc2	doc1,doc2

Table 4.5: Multiple words term query and retrieved documents

The above table shows that the multi word terms were indexed as they were, they couldn't be broken up. As indicated in the Table 4.4 its synonymy term was retrieved same documents. When we query the multi word terms, they are going to be identified by the system and it would be consider as single term. In the above table if the "ነብሰ ጡፍ" broken into two words, the system was unable to find the relevant documents. When the query string comes in this order "ነብሰ እናቶች ጡፍ " the sytem retrieved ***sp5*** document. This was because ***doc1*** and ***doc2*** indexed in the multi word terms "ነብሰ ጡፍ" but not for each words of the term. Therefore, the system not working as classical search engine works it incorporated the concept of multi word terms concept.

Based on the information given in Table 4.5, evaluation was done by measuring the recall, precision and F-measure of each highly frequent test queries and the average of each queries result is performance of the system for frequent test query. Table 4.5 presents relevant documents retrieved from the corpus for each frequent test query.

Query number	Lists of query	List of retrieved relevant documents for queries	Relevant document retrieved
1	አካል ብቃት እንቅስቃሴ	try2,try1,sp5,sp16,s18,Sport1,sp12,sport3, sp15, sp20, sp6	sp6,sp15,sp16,try1,try2
2	ሀይሌ ገብረስላሴ	Sport1,sp7,sp15,sp9,sport2,sp8,sp5,sp14,sp4	sportt2,sport1,sp18,sp15,sp9, sp7,sp4,sp14,sp5,sp8
3	ህክም ገልገያ ሳሪያ	s20,s22,s23,s21,s8, social3	s20,s21,s22,s23
4	ሁለተ ትምህርት	s21,s12,s4,Social	s4,s12
5	ፖለቲካ ድርጅ ሂደት	p10,p21,p18,p8,poletics2,poletic1,p5,b13,poletics3	P5,p6,p8,p10,p21,poletics1,poletics3
6	ኤርትራ ህዝብ	p12,p7,p5,p14,poletics,p13,p15,poletics2,p19,sp11	P7,p12,p13,p15
7	ክሬዲት ካርድ አገልግሎት	b14,b19,s9,economy3,s8,social3, b17,b21, Social,s15, s7	b14,b16,b17,b19,b21
8	ኢትዮጲያ ንግድ	economy,b5,b18,b19,b16,b21,b15	b5,b18,b21

Table 4.6: most frequent string query, relevant documents and retrieved document by prototype system

Query number	Lists of query	List of retrieved relevant documents for queries	Relevant document retrieved
1	እግር ኳስ ቡድን	sp11,s2, Sp11,sp12,sp16,sp19	Sp11,sp12,sp16,sp19,sp20,sport3
2	ሜዶ ወድድር	Sport1,sport2,sp10,sp9,sp13,sp15,sp12, sp14,sport3,sp4,sp7,sp11,sp20,sp5,sp8, sp19,sp17	Sp9,sp10,sp18,sport2
3	ደረጃ ትምህርት ቤቶ	s4,s21,s22,s7,s9,s11 s14,Social	S4,s12,s14
4	ሆስፒታል ዶክተር	s20, s21	S23,s22,s21,s20
5	አሠዴግ ድርጅታ ጉባኤ	p6,p8,p14,poletics3,p10,p5,p18,p19,poletic1,poletics2	Poletics2,poletics3,p21,p10,p8,p6,p5
6	ፖለቲካ ህይወት	economy,b5,p18,b9,b16,b13,b15	P18,p13,p9
7	ኢትዮጵያ ንግድ ባንክ	b18,b19,b15,b16,b17,b14,b20,b21,s12, b5,economy,s10,b10,b6,b11	b18,b14,b15,b16,b20,b12,b5,b14
8	ኢንቨስትምንት ንግድ	economy,b5,b18,b19,b16,b17,b15	economy,b18,b17

Table 4.7: less frequent string query, relevant documents and retrieved document by prototype system

Based on the information given in Table 4.5 and Table 4.6 the recall, precision and F measure wer calculated for each query, they were calculated using equation 3.6, 3.7 and 3.8 respectively. Table 4.7 and Table 4.8 shown the effectiveness of semantic indexing and document clustering for Amharic IR system based the above selected queries for the experimenting.

Query	Relevant	Retrived	Relevant retrived	R	P	F
አካል ብቃት እንቅስቃሴ	5	11	5	1.00	0.45	0.86
ሀይሌ ገብረስላሴ	10	9	7	0.7	0.78	0.89
ህክም ገልገያ ሳሪያ	4	6	4	1.00	0.67	0.86
ሁለተ ትምህርት	2	4	1	0.5	0.25	0.33
ፖለቲካ ድርጅ ሂደት	7	9	6	0.86	0.67	0.80
ኤርትራ ህዝብ	4	10	4	1.00	0.40	0.59
ክሬዲት ካርድ አገልግሎት	5	11	4	0.80	0.37	0.50
ኢትዮጲያ ንግድ	3	7	3	1.00	0.43	0.68
			Average	0.88	0.51	0.72

Table 4.8: The performance of the prototype system using frequent string query.

Query	Relevant	Retrived	Relevant retrived	R	P	F
እግር ኳስ ቡድን	6	6	5	0.83	0.83	0.83
ሜዶ ወድድር	4	17	3	0.75	0.17	0.48
ደረጃ ትምህርት ቤቶ	3	6	2	0.67	0.33	0.44
ሆስፒታል ዶክተር	4	2	2	0.50	1.00	0.67
አሠደግ ድርጅታ ጉባኤ	7	10	5	0.74	0.50	0.60
ፖለቲካ ህይወት	3	7	3	1.00	0.16	0.52
ኢትዮጵያ ንግድ ባንክ	8	15	7	0.88	0.50	0.64
ኢንቨስትምንት ንግድ	3	7	3	1.00	0.42	0.59
			Average	0.66	0.42	0.60

Table 4.9: The performance of the prototype system over less frequent query.

The above tables shown that there were irrelevant documents retrieved from the corpus and some of the relevant document from the corpus couldn't be retrieved. For instance, when we tried to search relevant document from the corpus using "ደረጃ ትምህርት ቤቶ" query. There were 3 relevant documents in the corpus, the system retrieved 6 documents out of that 2 were relevant documents; the rest 2 relevant documents were not retrieved. Similarly, in the case of "ክሬዲት ካርድ አገልግሎት " query there were 5 relevant documents in the corpus. The system retrieved 11

documents but out of them 4 are relevant documents. In general, the system registered average recall, precision and F measures of 0.88, 0.51, and 0.72 respectively for frequent queries string. And average recall, precision and F measures of 0.60, 0.42, and 0.60 respectively for less frequent string queries. When we calculate the average performance over frequent and less frequent query strings, we found 0.74, 0.46, and 0.66 average recall, precision and F measures respectively.

The values of precision at standard recall levels for each available 16 queries were important so as to show the performance of the system. Thus, interpolation of precision/recall curve is done. Fig 4.4 shown, the interpolated recall-precision curve to the performance of the designed prototype system.

As the curve shown when the recall increase the precision decrease and vise versa. It happens because one improves with the cost of the other.

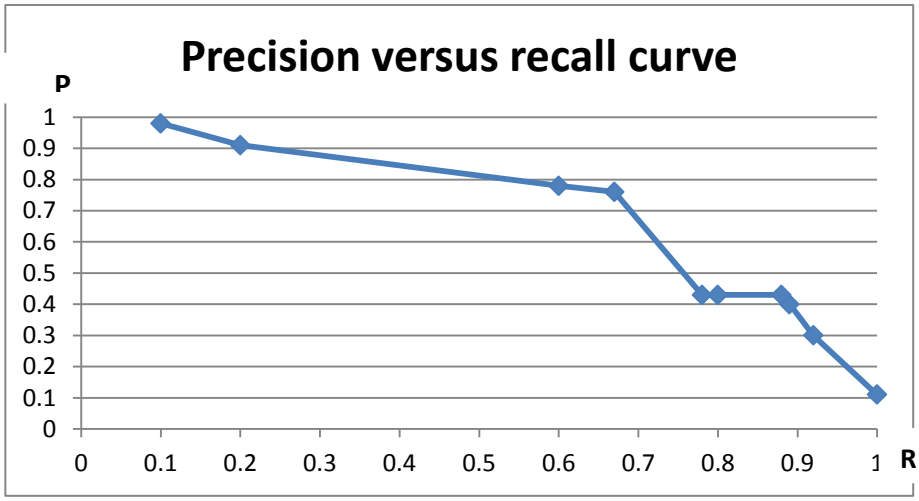

Fig 4.3: system's recall precision curve

4.3 Findings and Challenges

The result obtained in the experiments indicates IR system build based on semantic indexing and document clustering registered better result than the recent Amharic IR system developed by Amanuel [31]. The IR system build in this research has recorded 6% F-measure improvement compared to the better achievement in previous works that was Emanuel work [31]. Note: He tested using 980Kb 300 documents through probabilistic retrieval approach.

However, there were various challenges that the researcher encountered during the process of implementing the concepts. These challenges hampered the system not to register the better result.

The challenge started from the complexity of Amharic language itself. Amharic language is not well defined and the linguistic computational level of the language was not matured [24]. To get the semantic meaning of the Amharic document was bit difficult. This was because polysemy problem. If this problem not addressed well, it was difficult to get correct semantics of the Amharic documents. The researcher tried to keep the semantics of the documents by identifying and extracting noun and verbs and rejected other terms of the document. Using this technique it was difficult to keep the semantics of documents due to polysemy problem. For example when we query "ሞክል", might express one of : about drawing, coughing, or making knife sharp. The system couldn't aware the context of the words or phrases in the document. This problem has great impact on the performance of the system.

The second challenge was problem related to clustering algorithm. When the documents were not clustered properly, the system was unable to look for the relevant documents. Incorrect clustering of documents occurred because a document might have more shared terms with one cluster but it was talking about the other cluster. For instance, if the document dealt with business and financial issues in sports market has much terms of sport whereas the document has to cluster in business because it was talking more about business. The other cause for incorrect clustering was polysemy problems of the terms in the documents. If we couldn't get the accurate semantics of the document, the document might clustered based on ***tfidf*** measure of the document to clusters' representative document. As a result some documents were clustered incorrectly.

The third challenge was the representative of the cluster by itself. Representative document might have less or couldn't have similar terms; in this case the system might not search for the documents. In fact if the document had less or no similar terms, it had to be clustered into the other clusters however what would happen if it had same thing with all clusters. If this happen, the system randomly clustered under one of the cluster. But it made the system not search the incorrectly clustered documents when it was needed. Because it is difficult to find which cluster the document is found that the system responses relevant document not found in the corpus.

CHAPTER FIVE

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

5.1 Summary

Information retrieval is the method to look for relevant documents from unstructured document collection that satisfies information need of the users [1]. The goal of information retrieval system is to retrieve relevant information and rank them in accordance with level of relevancy to information needs of the user. Various models have been developed with regard to information retrieval system. Different researches were tried to devise different approaches and models to develop Amharic information retrieval system as discussed in chapter two. But results obtained in each attempt were not adorable even the recent and the better result achieved by Amanuel [31] was registered F-measure of 0.60.

Therefore, in this research an attempt has been made to design and develop an information retrieval system for Amharic language based on semantic indexing and documents clustering.

Using the advantages of semantic indexing and documents clustering the performance of Amharic retrieval system would be enhanced.

A system proposed by this study has three main components: semantic indexing, document clustering and searching. In the process of semantic indexing preprocessing, multi word terms extraction, weighting could be performed turn by tern. Multi word term extracted because in Amharic language there are multi word terms which different meaning from each word. Hence it would be an important concept to keep the semantics of the documents. Then documents were clustered into four groups to facilitate searching. Then the cluster based searching proceeds by supplying query string to the system. The system finds the relevant information passing through two steps: find the cluster the relevant information were found and find the relevant documents to the query in specified cluster

System evaluation has been done by examining each component of the system. The tokenizer of performs with 99.4% accuracy. And the stemmer of the system performs with 82.3% accuracy. The multiple word extraction performs with the aid of knowledge base. In the system the knowledge base is a prototype, developed by the researcher. With that knowledge base the system extracted multi word terms with 76% accuracy. Since the preprocessing tasks are not performing perfectly that it would have direct impacts on the whole system performance.

Document clustering is the most important part of this study and it had vital role in determining the documents being searched from the corpus. The document clustering algorithm employed in the system was K-means algorithm. The documents provided to the system clustered into four clusters; politics, social, business and sport. The clusters had average inter cluster similarity of 77% and average intra cluster similarity of 82%. This means some documents were clustered in wrong cluster. As a result the system faced difficulty or unable to look for some documents. In addition there was a problem related to representative of a cluster, not equally represent all elements of the cluster.

The General system evaluation has been done to investigate the extent to which the designed system enhances the performance of Amharic IR system considering the average F-measure. Though the above problems existed, as the experimental result shown, semantic indexing and document clustering based Amharic IR system register F-measure of 6% improvement. This

means the performance of the Amharic IR system increases from F-measure of 60% (Amanuel) to F-measure of 66% accuracy. This is a promising result to design an applicable IR system if polysemous nature of Amharic language is controlled with the help of thesaurus and improve the document clustering performance of the system.

5.2 Conclusion

The main objective of this research is to address the semantics problems related with Amharic IR and enhance the IR system through document clustering. Previous research works in Amharic IR

system attempted to address different problems like problem of stemmer and stop word, problem of synonymy and polysemy, problem of uncertainty and others. But not all appropriately addressed semantics problem in Amharic information retrieval and search strategy problem. As a result, the performance of the system was less; the highest F-measure of the system was 60%. Therefore, this study tried to address the problems using semantic indexing and document clustering and aimed to enhance retrieval system performance.

One element of this study was semantic indexing. It could be implemented through extracting salience terms of the documents. As literatures say noun and verb terms of the document can describe semantics of the documents. In this study, noun and verb terms were extracted to semantically represent the documents. In Amharic language there are number of multi word terms having different meaning from individual. Hence these terms were identified and indexed to keep the semantics of the documents.

As the experiments have shown semantic indexing has improved the performance of Amharic information retrieval system from 60% to 66% F-measure. Moreover, semantic indexing and document clustering reduced computational cost of the system by $\frac{1}{K}$ * TermsReductionRatio factor , where K is the number of the cluster. This is because the searching not gone through all elements rather in specified cluster only and the indexed terms were only nouns and verbs the rest have been discarded that the system computational cost was reduced.

Document clustering is a sub component of the system which aimed to cluster the related documents into one category to facilitate searching. As the experiment has shown document clustering along with semantic indexing has improved the performance of Amharic IR by F-measure of 6% compared to the best previous research works.

Even though, the semantic indexing and document clustering system enhanced Amharic IR by F-measure of 6%, polysemy problem was one of the main problems. As well the performance of the stemmer was the challenge. Plus to that because of limited number of words in knowledge base, it hindered the performance of the system.

As the experiment has shown, the system's effectiveness is highly affected by incorrect clustering and cluster representative. The K-means document clustering algorithm has been used

for this research. The average inters cluster accuracy of 77% and the average intra cluster accuracy of 82% has been registered. It clustered some documents in wrong cluster that the performance of the system has been affected.

Cluster representative was another challenge of the study. The mode based cluster representative selection has been used in this research but it has problem of not represent the whole cluster elements.

5.3 Recommendation

The designed system which is semantic indexing and document clustering based Amharic IR system is just an attempt to enhance the performance of IR system by considering the semantics of documents in the corpus and through documents clustering. Therefore, to obtain the optimum performance expected form the model, this work can be further pursued in several future directions.

- One of the problems in acquiring the right semantics of the document is the existence of polysemous terms in Amharic text. It has an influential role on the performance of IR system. Hence, the researcher recommends future research to implement context aware IR system using a well developed thesaurus to aware the context to which the terms in the text appears to enhance precision and recall of the system. In addition using well tagged corpus of bigger size could help to discriminate the context in which terms are presented.
- The multi word term extraction and stemming algorithms performed with 74% and 82% accuracy respectively. Since these have influential role on the performance of the whole system, the researcher recommends pursuing further research to improve the performance of algorithms.
- The other problem observed was clustering of some documents in incorrect clusters. If the document clustered in a wrong cluster, the retrieval performance would obviously be poor. Therefore, the researcher recommends that researches should be conducted to improve the performance of the document clustering system by using partitioning clustering algorithms so as to reduce the number of incorrect clustering of documents.
- The other critical problem was finding cluster representative documents. Representative document is one which contains some of terms appears in each documents. Because similarity score measured based on the existed terms and their weights. Basically there are two approach in cluster representative selection; mean and mode. In this research mode has been used; that is one document from the cluster is selected. Mean is create new document which is average of the whole cluster elements. The researcher recommends trying the mean cluster representative selection method.

Reference

[1] Christopher D., Prabhakar R. and Hinrich S.. (2009), Introduction to information retrieval, Online edition, Cambridge University Press, Cambridge, England. accessed February 2013,http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

[2] Amit Singhal.(2008) Modern Information Retrieval: A Brief Overview, Google, Inc,accessed February 2013, http://ilps.science.uva.nl/ Teaching/0405/AR/part2/ir overview. pdf

[3] W. Bruce C.. (1995), 'What Do People Want from Information Retrieval?', accessed 11 February 2013, http://www.dlib.org/dlib/november95/11croft.html

[4] Recardo B.. (1999), Modern Information Retrieval,ACM press, New York

[5] Aurélie N., James G., and Alan R.n. Automatic Indexing of Specialized Documents:Using Generic vs. Domain-Specific Document Representations, National Library of Medicine,8600 Rockville Pike,Bethesda, MD 20894,USA

[6] Linda A., Performance of Two Statistical Indexing Methods, with and without Compound-word Analysis,

[7] Latifur K., Dennis M., and Enduard H. . Retrieval Effectiveness of an Ontology-Based Model for Information Selection. Accessed February 2013,

http://www.isi.edu/natural-language/people/hovy/papers/03VLDB-ontology-based-IR.pdf

[8] David V., Miriam F., and Pablo C.. An Ontology-Based Information Retrieval Model, Universidad Autónoma de Madrid, Campus de Cantoblanco, c/ Tomás y Valiente 11, 28049 Madrid

[9] Marti A. Automated Discovery of WordNet Relations, Christiane Fellbaum (Ed.), MIT Press

[10] Bin H. Automatic Term Extraction in Large Text Corpora, Faculty of Computer Science, Dalhousie University, Halifax, Canada B3H 1W5

[11] Hans F.. Terminology Extraction and Automatic Indexing: Comparison and Qualitative Evaluation of Methods

http://wortschatz.uni-leipzig.de/~fwitschel/papers/TKEIndexing.pdf

[12] Jian-Yun N.. A General Logical Approach to Inferential Information Retrieval, Département d'Informatique et Recherche opérationnelle, Université de Montréal, Canada

[13] Yu X.. (2010) A Survey of Document Clustering Techniques: Comparison of LDA and moVMF., Accesed February 2013,

http://cs229.stanford.edu/proj2010/Xiao%20Survey%20of%20Document%20Clustering%20Techniques%20&%20Comparison%20of%20LDA%20and%20moVMF.pdf

[14] Michael S., George K. Vipin K.. A Comparison of Document Clustering Techniques, Technical Report #00-034, Department of Computer Science and Egineering.

[15] Binyam G.. Part of speech tagging for Amharic, Centre Tesnière, Research Institute in Information and Language Processing, Université de Franche-Comté, France

[16] Martha Y. and Wolfgang M.. Amharic part of speech tagger for factored language model, Department of Informatics, University of Hamburg

[17] Atelach A. and Lars A.. An Amharic Stemmer : Reducing Words to their Citation Forms, Department of Computer and Systems Sciences, Stockholm University/KTH, Sweden

[18] Sisay F. Part of Speech tagging for Amharic using Conditional Random Fields, Informatics Institute, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

[19] Bethelem M. (2002), N-gram-Based Automatic Indexing for Amharic Text,Msc thesis,school of information science for Africa, Addis Ababa university,Addis Ababa

[20] Bo-Yeong K, A Novel Approach to Semantic Indexing Based on Concept, Department of Computer Engineering, Kyungpook National University

[21] Barbara R. Latent Semantic Indexing: An overview, INFOSYS 240 Spring 2000,Final Paper

[22] Jian Z., Jianfeng G., Ming Z., Jiaxing W. Improving the Effectiveness of Information Retrieval with Clustering and Fusion, Microsoft Research China, Department of Computer Science of Tsinghua, University, China

[23] Tewdros H. (2003), Amharic text retrieval: an experiment using Latent semantic indexing (LSI) with singular value decomposing (SVD),Msc thesis, School of Graduate Studies of Addis Ababa University. Addis Ababa Univesity, Addis Ababa.

[24] Alemayehu M. (2010), query expansion technique for Amharic IR, Msc thesis, School of Graduate Studies of Addis Ababa University. Addis Ababa Univesity, Addis Ababa

[25] Christopher D., an introduction to information retrieval, online edition, Cambridge University Press, Cambridge, England

[26] Nega A. and Peter W. stemming of Amharic words for information retrieval, university Sheffield, Sheffield, England.

[27] Björn G and Lars A.. Experiences with Developing Language Processing Tools and Corpora for Amharic, Swedish Institute of Computer Science,Sweden

[28] Fissaha, Sisay. 2005. Part of speech tagging for Amharic using conditional random fields. In 43rd Annual Meeting of the Association for Computational Linguistics: Workshop on Computational Approaches to Semitic Languages , pp. 47–54, Ann Arbor, Michigan.

[29] Katerina F., Sophia A., Hideki M.. Automatic recognition of multi word terms: the C-value/NC –value methods .

[30] G.bharathi and D.venkatesan. , improving information retrieval using document clustering and semantic synonyms extraction, School of Computing, SASTRA University, Tamil Nadu, India

[31] Amanuel H. (2012), probabilistic information retrieval system for Amharic language, school of information science, school of graduate studies, Addis Ababa University, Addis Ababa

[32] Tesema M. and Solomon A. (2009) , Design and Implementation of Amharic Search Engine, fifth international conference on signal image technology and internet based systems, 3(1):318-325.

[33] Alemayehu t. (2002),"Application of Query Expansion for Amharic information retrieval System", MSc Thesis, Addis Ababa University, School of Information Science, Ethiopia.

[34] Abey B., Semantic Based Query Expansion Technique for Amharic IR, MSc Thesis, School of Information Science, Addis Ababa University, Ethiopia, 2011

[35] M. Lutuz, Learning Python , OReilly publish, USA ,4th Edition, 2009

[36] K. D.Lee. (2011), Python Programming Fundamentals , Springer-verlog press, USA, 1st Edition.

[37] Xiaoyong L. "Cluster-Based Retrieval Using Language Models" Center for Intelligent Information Retrieval,Department of Computer ScienceUniversity of Massachusetts,

[38] Commission, F.P.C. 2008. Summary and Statistic Report of the 2007 Housing Census: Population size by age and sex , Addis Ababa, Ethiopia

Appendix I: Amharic character set

	ā/ä [a]	u [u]	ī/i [i]	a [a]	ē/e [e/ɛ]	(i)/(ə) [ə]	0 [o/ɔ]
h [h]	ሀ ha	ሁ hu	ሂ hi	ሃ ha	ሄ he	ህ h(ə)	ሆ [h]
l [l]	ለ le	ሉ lu	ሊ li	ላ la	ሌ le	ል l(ə)	ሎ lo
h/ḥ [h]	ሐ ha	ሑ hu	ሒ hi	ሓ ha	ሔ he	ሕ h(ə)	ሖ ho
m [m]	መ me	ሙ mu	ሚ mi	ማ ma	ሜ me	ም m(ə)	ሞ mo
s/ś [s]	ሠ se	ሡ su	ሢ si	ሣ sa	ሤ se	ሥ s(ə)	ሦ so
r [r]	ረ re	ሩ ru	ሪ ri	ራ ra	ሬ re	ር r(ə)	ሮ ro
s [s]	ሰ se	ሱ su	ሲ si	ሳ sa	ሴ se	ስ s(ə)	ሶ so
sh/š [ʃ]	ሸ ʃe	ሹ ʃu	ሺ ʃi	ሻ ʃa	ሼ ʃe	ሽ(ə)	ሾ ʃo
kʷ/q [kʷ]	ቀ kʷe	ቁ kʷu	ቂ kʷi	ቃ kʷa	ቄ kʷe	ቅ kʷ(ə)	ቆ kʷo
b [b]	በ be	ቡ bu	ቢ bi	ባ ba	ቤ be	ብ b(ə)	ቦ bo
t [t]	ተ te	ቱ tu	ቲ ti	ታ ta	ቴ te	ት t(ə)	ቶ to
ch/č [ʧ]	ቸ ʧe	ቹ ʧu	ቺ ʧi	ቻ ʧa	ቼ ʧe	ች ʧ(ə)	ቾ ʧo
h/ẖ [h]	ኀ ha	ኁ hu	ኂ hi	ኃ ha	ኄ he	ኅ h(ə)	ኆ ho
n [n]	ነ ne	ኑ nu	ኒ ni	ና na	ኔ ne	ን n(ə)	ኖ no
ny/ñ [ɲ]	ኘ ɲe	ኙ ɲu	ኚ ɲi	ኛ ɲa	ኜ ɲe	ኝ ɲ(ə)	ኞ ɲo
ʷ/ʾ [ʔ]	አ (ʔ)a	ኡ (ʔ)u	ኢ (ʔ)i	ኣ (ʔ)a	ኤ (ʔ)e	እ (ʔ)(ə)	ኦ (ʔ)o
k [k]	ከ ke	ኩ ku	ኪ ki	ካ ka	ኬ ke	ክ k(ə)	ኮ ko
h/ḵ [h]	ኸ he	ኹ hu	ኺ hi	ኻ ha	ኼ he	ኽ h(ə)	ኾ ho
w [w]	ወ we	ዉ wu	ዊ wi	ዋ wa	ዌ we	ው w(ə)	ዎ wo
„/ʿ [ʔ]	ዐ ʔa	ዑ ʔu	ዒ ʔi	ዓ ʔa	ዔ ʔe	ዕ ʔ(ə)	ዖ ʔo
z [z]	ዘ ze	ዙ zu	ዚ zi	ዛ za	ዜ ze	ዝ z(ə)	ዞ zo
zh/ž [ʒ]	ዠ ʒe	ዡ ʒu	ዢ ʒi	ዣ ʒa	ዤ ʒe	ዥ ʒ(ə)	ዦ ʒo
y [j]	የ je	ዩ ju	ዪ ji	ያ ja	ዬ je	ይ j(ə)	ዮ jo

d [d]	ዯ de	ዱ du	ዲ di	ዳ da	ዴ de	ድ d(ə)	ዶ do
j/ǧ [ʤ]	ጀ ʤe	ጁ ʤu	ጂ ʤi	ጃ ʤa	ጄ ʤe	ጅ ʤ(ə)	ጆ ʤo
g [g]	ገ ge	ጉ gu	ጊ gi	ጋ ga	ጌ ge	ግ g(ə)	ጎ go
tʼ/ṭ [tʼ]	ጠ tʼe	ጡ tʼu	ጢ tʼi	ጣ tʼa	ጤ tʼe	ጥ tʼ(ə)	ጦ tʼo
chʼ/č̣ [ʧʼ]	ጨ ʧʼe	ጩ ʧʼu	ጪ ʧʼi	ጫ ʧʼa	ጬ ʧʼe	ጭ ʧʼ(ə)	ጮ ʧʼo
pʼ/p [pʼ]	ጰ pʼe	ጱ pʼu	ጲ pʼi	ጳ pʼa	ጴ pʼe	ጵ pʼ(ə)	ጶ pʼo
tsʼ/ṣ [ts]	ጸ tsʼe	ጹ tsʼu	ጺ tsʼi	ጻ tsʼa	ጼ tsʼe	ጽ tsʼ(ə)	ጾ tsʼo
tsʼ/ṣ́ [ts]	ፀ tsʼe	ፁ tsʼu	ፂ tsʼi	ፃ tsʼa	ፄ tsʼe	ፅ tsʼ(ə)	ፆ tsʼo
f [f]	ፈ fe	ፉ fu	ፊ fi	ፋ fa	ፌ fe	ፍ f(ə)	ፎ fo
p [p]	ፐ pe	ፑ pu	ፒ pi	ፓ pa	ፔ pe	ፕ p(ə)	ፖ po
v [v]	ቨ ve	ቩ vu	ቪ vi	ቫ va	ቬ ve	ቭ v(ə)	ቮ vo

Appendix II: Amharic punctuation mark set

፡ ፣ ፤ ፥ ፦ ፧

Full stop comma colon semi colon preference colon question mark (no longer used)

Appendix III: Amharic numbering set

፩ ፪ ፫ ፬ ፭ ፮ ፯ ፰ ፱ ፲

1 2 3 4 5 6 7 8 9 10

፳ ፴ ፵ ፶ ፷ ፸ ፹ ፺ ፻ ፼

20 30 40 50 60 70 80 90 100 10000

Appendix IV **:** Indexing python source code

```
import math
import os
validMterm=[]      # list for valide three word terms
validMterm2 = []    # list for valid two word term
dword=[]
mwpost=[]
wlistp=[]
st=""
def indexing ():
   #MULTIPLE WORDS INDEXING
   singlewlist1=[]
   singlewlist=[]
   dword=[]
   dicwords=[]
   terml1=[]
   terml2=[]
   n=0
   m=1
   k=0
   i=0
   mwlist=[]
   path="stemmedfile"  # insert the path to the directory of interest
   dirList=os.listdir(path)
   for fname in dirList:
      i=i+1
   #CHECK DICTIONARY
   termh=[]
   xx=[]
   for terml1 in validMterm:
      tt=[]
      doc=0
      mwlist=[]
      #dword=[]
    if terml1[0] not in mwlist:
         doclist =[]
         mwlist.append(terml1[0])
         for terml2 in validMterm[n:]:
            ss1=" ".join(terml1[0])
            ss1.strip()
            dword=[]
            #print(ss1)
            dic=open("dictionary.txt", "r",encoding='utf8')
            for dwords  in dic.read().split("."):
               xx=[]

               for wordl in dwords.split(","):
                  world="".join(wordl)
                  world=world.strip()
                  if world not in xx:
                     xx.append(world)
               if ss1 in xx :
                  #print(ss1,terml2[2])
                  dword.extend(xx)
```

```
            dword.remove(ss1)
        #print(dword)
        for t in dword:
            if t not in dicwords:
                dicwords.append(t)
        dic.close()
        if terml2[0]==terml1[0] :
            #print(dword)
            doc= doc+ 1
    k= terml1[1]
    se = open("stemmedfile/"+ terml1[2], "r",encoding='utf8')
    #print(terml2[2])
    for t2 in se.read().split():
        if t2 in dword :
            k=k+1
    #print(k,dword)
    se.close()
    n=n+1
    tfidf=k*math.log(i/k,2)
    doclist.append(terml1[2])
    doclist.append(str(tfidf))
    tt.append(terml1[0])
    tt.append(doclist)
    wlistp.append(tt)
terml1=[]
terml2=[]
n=0
m=1
k=0
mwlist=[]
for terml1 in validMterm2:
    tt=[]
    doc=0



    if terml1[0] not in mwlist:
        doclist =[]
        mwlist.append(terml1[0])
        for terml2 in validMterm2[n:]:
            if terml2[0]==terml1[0]:
                doc= doc+ 1

     for terml2 in validMterm2[n:]:
        docl=[]
        ss1=" ".join(terml2[0])
        ss1.strip()
        dword=[]
        #print(ss1)
        dic=open("dictionary.txt", "r",encoding='utf8')
        for dwords  in dic.read().split("."):
            xx=[]
            for wordl in dwords.split(","):
```

```
                world="".join(wordl)
                world=world.strip()
                if world not in xx:
                   xx.append(world)
            if ss1 in xx :
               #print(ss1,terml2[2])
               dword.extend(xx)
            dword.remove(ss1)
         for t in dword:
            if t not in dicwords:
              dicwords.append(t)
         dic.close()
         if terml2[0]==terml1[0]:
            se = open("stemmedfile/"+ terml1[2], "r",encoding='utf8')
            k=terml2[1]
            #print(terml2[0],dword,k)
            for t2 in se.read().split():
                  if t2 in dword:
                  #print(t2,terml1[2])
                  k=k+1
            se.close()
            #print(k)
            tfidf=k*math.log(i/k,2)
             docl.append(terml1[2])
            docl.append(str(tfidf))
            doclist.append(docl)
      n=n+1
      tt.append(terml1[0])


      tt.append(doclist)
      wlistp.append(tt)
#print(dicwords)
tit=[]
t=[]
st1=""
file2 = open("postingfile.txt", "w",encoding='utf8')
file = open("postingf.txt", "w",encoding='utf8')
try:
      #print(wlistp)
      final1=[]
      for yy in wlistp:
         bb=[]
         temp=[]
         s1=0
         ee=[]
         n=0
         for zz in wlistp[n:]:
            if yy[0]==zz[0] :
               #print(zz[0])
               if zz[1] not in temp:
                  #print(zz[0])
                  temp.append(zz[1])
```

```
            s1=1
            ee.append(yy[0])
            ee.append(zz[1])
    if s1==1 and  temp:
        #print(temp)
         bb.append(yy[0])
        bb.append(temp)
        final1.append(bb)
    n=n+1
    #print(final1)
tempo=[]
for iterm in final1:
    #for item in iterm:
        if iterm[0] not in tempo:
            #file2.writelines(str(" ".join(iterm[0])))
            file2.writelines(str(iterm[0]))
            file2.writelines(",")
            #zx=" ".join(iterm[1])
            file2.writelines(str(iterm[1]))
            tempo.append(iterm[0])
            file2.writelines("#\n")



#   SINGLE WORD INDEXING
 templist1=[]
#print(validMterm)
validMterm.extend(validMterm2)
#print(validMterm)
path="stemmedfile"  # insert the path to the directory of interest
dirList=os.listdir(path)
for fname in dirList:
    semf = open("stemmedfile/"+fname, "r",encoding='utf8')
    #print(validMterm)
    templist1=[]
    templist=[]
    termlist=[]
     terma=[]
    check=0
    mm=[]
    for term1 in semf.read().split():
        check=0
        s1=0
        if term1  in templist1:
            #templist1.append(term1)
             s1=1
       if s1==0 :
            #print(term1,fname)
            templist1.append(term1)
            if term1 not in dicwords:
                termcount=0
                semanticfil = open("stemmedfile/"+fname, "r",encoding='utf8')
```

```
            for term2 in semanticfil.read().split():
               if term1==term2:
                  termcount=termcount+1
            termlist.append(term1)
            semanticfil.close()
            tt=[]
            singlewlist1=[]
            for ll in validMterm:
                  for kk in ll[0] :
                     if term1==kk and ll[2]==fname :
                        #print(kk,fname,termcount,ll[1])
                        tt.append(term1)
                        termcount=termcount-ll[1]
                        if termcount > 0:
                           tfidf=termcount*math.log(i/termcount,2)
                           #print(term1,fname)
                           bb=[]
                           bb.append(fname)
                           bb.append(tfidf)
                           singlewlist1.append(term1)
                           singlewlist1.append(bb)
                           singlewlist.append(singlewlist1)
                           st11=term1 +"," + str(tfidf) + "," +fname
                           #templist.append(term1)
                           #templist.append(cc)
                           file.writelines("\n")
                        check=1
            if check == 0:
                  tfidf=termcount*math.log(i/termcount,2)
                  #print(term1)
                  bb=[]
                  bb.append(fname)
                  bb.append(tfidf)
                  singlewlist1.append(term1)
                  singlewlist1.append(bb)
                  singlewlist.append(singlewlist1)
                  st11=term1 +"," + fname + "," +str(tfidf)
                  cc=[]
                  cc.append(fname)
                  cc.append(tfidf)

yy=[]
zz=[]
aa=[]
cc=[]
temp=[]
final=[]
n=0
dd=[]
dd1=[]
for yy in singlewlist:
   bb=[]
   temp=[]
```

```
            s=0
            ee=[]
            gg=[]
            s1=0
            for zz in singlewlist[n:]:
               if yy[0]==zz[0] :
                  if zz[1] not in temp:

                     temp.append(zz[1])
                     s1=1
                     ee.append(yy[0])
                     ee.append(zz[1])
                     dd.append(ee)
            if s1==1 and  temp:
               #print(temp)
               cc.append(yy[0])
               bb.append(yy[0])
               bb.append(temp)
               final.append(bb)
            n=n+1
            capt=[]
         for iterm in final:
            if iterm[0] not in capt:
               for item in iterm:
                  file2.writelines("["+str(item)+"]")
                  file2.writelines(",")
               file2.writelines("#\n")
            capt.append(iterm[0])
         file2.close()
   finally:
          semf.close()
         file.close()
def multiplewordExtraction():
   wordlist=[]
   finaleMWords=[] # list for three word multiple term extraction
   mwtemplist=[]  # list for three word multiple term
   tmwtemplist=[] # list for two  word multiple term
   finalMWords=[] #  list for three word multiple term extraction
   mwords=[]      # multiple words variable
   maxw=[]        # list for maximum frequency term
   term=[]        # list help to find maximum frequency
   temp=[]        # list help to find maximum frequency
   templist=[]    # list help to find maximum frequency
   maxfrequency=0 # variable help to find maximum frequency
   threshold=0      # variable help to find threshold value
   mtermabovethreshold=[] # list to find multiple word term above threshold value
   tfin=[]              # list to find two word multiple term frequency
   tmwtemplist=[]         #list to extract two word multiple word
   rwmpmem=[]
   path="stemmedfile"  # insert the path to the directory of interest
   dirList=os.listdir(path)
   for fname in dirList:
    try:
```

```
 i=0
 maxword=0
 semanticfile = open("stemmedfile/"+fname, "r",encoding='utf8')
 for maxw in semanticfile.read().split():
     temp.append(maxw)
 semanticfile.close()
 #templist.append("")
 for maxw in temp :
     #CHECK THE WORD IN THE LIST
     tempmax=0
     j=0
     if maxw in templist :
         j=0
     else :
         j=1
     #IF THE FREQUENCY IN THE NOT IN THE LIST
     if j==1 :
         templist.append(maxw)
         for term in temp:
             if maxw==term:
                 tempt=maxw
                 tempmax=tempmax + 1
         #print (maxw + str(tempmax))
         if tempmax > maxword:
             maxword=tempmax
    #FIND THE TRHESOLD VALUE
 threshold=math.sqrt(2*maxword)
 threshold= math.floor(threshold)
 #THREE WORD MULTIPLE WORD EXTRACTION
 x=0
 for item in wordlist:
    x=x+1
 del wordlist[0:x]
 word=[]
 semanticfile = open("stemmedfile/" + fname, "r",encoding='utf8')
 for words  in semanticfile.read().split():
          wordlist.append(words)
semanticfile.close()
 x2=0
 for i in finaleMWords:
    x2=x2+1
 del finaleMWords[0:x2]
 for n in range (len(wordlist)-2):
       finaleMWords.append(wordlist[n:3+n])
 for fin in finaleMWords:
     lis=[]
     j=0
     frequency=0
     if [fin,fname] in mwtemplist :
         j=0
     else :
         j=1
     if j==1 :
```

```
        lis.append(fin)
        #print(fname)
        lis.append(fname)
        mwtemplist.append(lis)
        for term in finaleMWords:
          if fin==term:
               tempt=fin
               frequency=frequency + 1
        #print(fin,frequency)
        c_value=math.log (3,2) *frequency
        cvalue=[]
        if c_value>=threshold:
            cvalue.append(tempt)
            cvalue.append(frequency)
            cvalue.append(fname)
            wordl=[]
            tt=" ".join(tempt)
            #print(tt)
            dic=open("dictionary.txt", "r",encoding='utf8')
            for dwords  in dic.read().split("."):
               for wordl in dwords.split(","):
                  world="".join(wordl)
                  world=world.strip()
                  if world==tt:
                     #print (world)
                     validMterm.append(cvalue)
             dic.close()
             #print(validMterm)
            #print(c_value)
#TWO WORD MULTIPLE TERM EXTRACTION COMPUTE C_VALUE
 tempmem3=[]
 #print (validMterm)
 x1=0
 for item in finalMWords:
    x1=x1+1
 del finalMWords[0:x1]
 for n in range (len(wordlist)-1):
      finalMWords.append(wordlist[n:2+n])
 for tfin in finalMWords:

    lis=[]
    j=0
    frequency=0
    if [tfin,fname] in tmwtemplist :
        j=0
    else :
        j=1
    if j==1 :
        lis.append(tfin)
        lis.append(fname)
        tmwtemplist.append(lis)
        for term in finalMWords:
             if tfin==term:
```

```
                    tempt=tfin
                    frequency=frequency + 1
                #print (tempt)
                #print (frequency)
                tempmem1=[]
                tempmem2=[]
                tempmem4=[]
                k=0
                l=0
                if tempt not in tempmem3:
                    for tempmem1 in validMterm:
                        if tempt in tempmem3:
                            for tempmem2 in tempmem1[::2]:
                                for n in range (len(tempmem2)-1):
                                    if tempmem2[n:2+n] == tempt:
                                        k= k + int(tempmem1[1])
                                        l=l+1
                        else:
                            for tempmem2 in tempmem1[::2]:
                                for n in range (len(tempmem2)-1):
                                    if tempmem2[n:2+n] == tempt:
                                        tempmem3.append(tempmem2[n:2+n])
                                        k= int(tempmem1[1])
                                        l=l+1
                    if k>0:
                        c_value=math.log(2,2) *(frequency-((1/l)*k))
                        frequency=frequency - k
                    else:
                        c_value =math.log(2,2)* frequency
                    cvalue=[]
                    if c_value>=threshold:
                        cvalue.append(tempt)
                        cvalue.append(frequency)
                        cvalue.append(fname)
                        wordl=[]
                        tt=" ".join(tempt)
                        dic=open("dictionary.txt", "r",encoding='utf8')
                        for dwords  in dic.read().split("."):
                            for wordl in dwords.split(","):
                                world="".join(wordl)
                                world=world.strip()
                                if world==tt:
                                    #print (world)
                                    validMterm2.append(cvalue)
                        dic.close()
                frequency=0
            prevfname=fname
            #print (validMterm)
            #print (validMterm2)
    except:
            print("error")
def stopwordremoval(params):
    word= params
```

```
    try :
        stopw = open("stopword.txt", "r",encoding='utf8')
        i=0
        for line  in stopw.read().split():
            if word==line:
                i=i+1
                return 0
                break
        if i==0 :
            return word
    finally:
        stopw.close()
def normalization (parms):
    normal = []
    normal= parms
    #type(normal)
    term=[]
    term=list (normal)
    Nlist=[]
    #norm = open("normalization.txt", "r",encoding='utf8')
    normList= u"ሐሀሐሁሓሂሓሃሓሄሕህሖሆሠሰሡሢሣሤሌሥስሦሶዐአዑኡዒኣዓኤዕእዖኦጹፂጺፃጼፅጽጾፆሃኣአ"
    type(normList)
    Nlist=normList
    i=0
    for character in normal:

        index=0
        for cha in Nlist[::2]:
           if character==cha:
                term.insert(i,Nlist[index+1])
                l=term.pop(i+1)
                break
           index =index+2
        i=i+1
    fterm="".join(term)
    word= stopwordremoval(fterm)
    word =str (word)
    return word
def stemmer(parms):
   stem = []
   stem= parms
    #PREFIX RMOVAL
   if len(stem)>=3:
      try:
         prefix = open("prefix.txt", "r",encoding='utf8')
         for line  in prefix.read().split():
           prefixsize= len(line)
           tempPrefix=stem[:prefixsize]
           if tempPrefix==line:
              #print (line + "prefix stemmed")
              stem= stem[prefixsize:]
              #print(stem)
      finally:
```

```
        prefix.close()
    # SUFIX REMOVAL
    try:
        suffix = open("suffix.txt", "r",encoding='utf8')
        for line  in suffix.read().split():
          suffixsize= len(line)
          tempsuffix=stem[-suffixsize:]
          if tempsuffix==line:
             #print (line + "suffix stemmed")
             stem= stem[:-suffixsize]
             # print(stem)
    finally:
       prefix.close()
    #print(stem)
  stem=normalization(stem)
  #print(stem)
  return stem

#START POINT
n=0
path="file"
dirList=os.listdir(path)
for fname in dirList:
  try:
    f = open("file/"+fname, "r",encoding='utf8')
    stemmedTerm=""
    kk=""
    for line  in f.read().split():
      if line=="<N>" or line=="<V>" or line=="<NP>" :
         #print (kk + line)
         tt= stemmer(kk)
         if tt != None and tt!=str(0):
              stemmedTerm= stemmedTerm + tt +"\n"
    kk=line
    #print (stemmedTerm)
  finally:
    f.close()
  n=n+1
  file = open("stemmedfile/" + fname , "w",encoding='utf8')
  try:
       file.writelines(stemmedTerm) # Write a sequence of strings to a file
  finally:
        file.close()
multiplewordExtraction()
#print (validMterm)
#print(validMterm2)
indexing()
```

Appendix V : document clustering source code.

```
import os
import sys
import math
import codecs
import string
print ("...........................................................................")
lis=[]
dd=[]
clustern=[]
clusterlist=[]
clusterR=[]
for n in range(4):
    dname1=input("enter document name for cluster formation:- ")
    dname1=dname1+".txt"
    clustern.append(dname1)
#dname1="economy.txt"
i=0
for cname in clustern:
    path="stemmedfile"  # insert the path to the directory of interest
    dirList=os.listdir(path)
    clusterg=[]
    clusterR.append(cname)
    for fname in dirList:
        ss=[]
        clusterunit=[]
        dname1=cname
        dname2=fname
        '''for n in range(2):
            UserQuery=input("እባኮን የሚፈልጉትን ፋይል !:- ")
            lis.append (UserQuery)
        for n in lis:'''
        voca=[]
        voca1=[]
        terms=[]
        word=""
        dic=open("postingfile.txt", "r",encoding='utf8')
        if dname1!=dname2:
            for dwords  in dic.read().split("#"):
                for word in dwords.split("]") :
                    newstr1 = word.replace("[", "")
                    newstr2 = newstr1.replace("'", "")
                    newstr3 = newstr2.replace(",", "")
                    '''for xy in word:
                        if xy != ']':
                            dd= dd + xy'''
                    newstr4 = newstr3.replace(",", "")
                    if newstr4:
                        voca1.append(newstr4.strip())
                    #print(newstr4.strip())
                    break
                #if voca:
                #    terms.append(voca)
                #print(voca1)
```

```
            dic.close()
            dic=codecs.open("postingfile.txt", "r",encoding='utf8')
            for dwords  in dic.read().split("#"):
                  voca=[]
                  for word in dwords.split("]") :
                        rr=[]
                        newstr1 = word.replace("[", "")
                        newstr2 = newstr1.replace("'", "")
                        newstr3 = newstr2.replace(",", "")
                        '''for xy in word:
                            if xy != ']':
                                  dd= dd + xy'''
                        newstr4 = newstr3.replace(",", "")
                        for cc in newstr4.split(" "):
                              if cc:
                                    rr.append(cc.strip())
                        if rr:
                            voca.append(rr)
                  if voca:
                              voca.pop(0)
                  if voca:
                        terms.append(voca)
            dic.close()
            #print(terms)
            i=0
            nomu=0
            denomu1=0
            denomu2=0
            for mm in terms:
                  ter=mm[0]
                   w1=0
                  w2=0
                  for ll in mm:
                      if ll[0]==dname1:
                            w1=ll[1]
                      if ll[0]==dname2:
                            w2=ll[1]
                   nw=float(w1)*float(w2)
                  dw1=float(w1)*float(w1)
                  dw2=float(w2)*float(w2)
                   nomu=nomu+nw
                  denomu1=denomu1+dw1
                  denomu2=denomu2+dw2
                  i=i+1
            #print("==============================")
            c_sim= nomu/ (1 + (math.sqrt(denomu1)* math.sqrt(denomu2)))
            clusterunit.append(fname)
            clusterunit.append(c_sim)
            clusterg.append(clusterunit)
            #print("cosine simmilarity with ",fname,"is ",c_sim)
            #print("\n\n")
#print("OTHER CLUSTER",cname,"******************************")
ss.append(cname)
```

```
        ss.append(1.0)
        clusterg.append(ss)
        clusterlist.append(clusterg)
'''j=0
for s in clusterlist:
        print(clusterR[j])
        print(s)
        j=j+1'''
fclus1=[]
fclus2=[]
fclus3=[]
fclus4=[]
for l in clusterlist[0]:
        for m in clusterlist[1]:
                for n in clusterlist[2]:
                        for o in clusterlist[3]:
                            if l[0]==m[0] and l[0]==n[0] and l[0]==o[0]:
                                if l[1]>=m[1]:
                                        if l[1]>=n[1]:
                                                if l[1]>=o[1]:
                                                        fclus1.append(l[0])
                                                else:
                                                        fclus4.append(l[0])
                                        else:
                                                if n[1]>=o[1]:
                                                        fclus3.append(l[0])
                                                else:
                                                        fclus4.append(l[0])
                                else:
                                        if m[1]>=n[1]:
                                                if m[1]>=o[1]:
                                                        fclus2.append(l[0])
                                                else :
                                                        fclus4.append(l[0])
                                        else:
                                                if n[1]>=o[1]:
                                                        fclus3.append(l[0])
                                                else:
                                                        fclus4.append(l[0])
clusterfiles=[]
print(fclus1)
print(fclus2)
print(fclus3)
print(fclus4)
clusterfiles.append(fclus1)
clusterfiles.append(fclus2)
clusterfiles.append(fclus3)
clusterfiles.append(fclus4)


try:
        file = open("clusterR.txt", "w",encoding='utf8')
        namestr=""
```

```
        for nam in clustern:
            namestr=namestr+nam +","
        file.writelines(namestr) # Write a sequence of strings to a file
finally:
        file.close()
i=0
for name in clustern:
    try:
            file = open(name, "w",encoding='utf8')
            namestr=""
            for nam in clusterfiles[i] :
                namestr=namestr+nam +","
            file.writelines(namestr) # Write a sequence of strings to a file
    finally:
            file.close()
    i=i+1
```

Appendix VI : Searching source code

```
import os
import sys
import math
import codecs
import string
stri=""
validMterm=[]
validMterm2=[]
def stopwordremoval(params):
    word= params
    try :
        stopw = open("stopword.txt", "r",encoding='utf8')
        i=0
        for line  in stopw.read().split():
            if word==line:
                i=i+1
                return 0
                break
        if i==0 :
            return word
    finally:
        stopw.close()
def normalization (parms):
    normal = []
    normal= parms
    #type(normal)
    term=[]
    term=list (normal)
    Nlist=[]
    #norm = open("normalization.txt", "r",encoding='utf8')
    normList= u"ሐሀሑሁሒሂሓሃሔሄሕህሖሆሠሰሡሱሢሲሣሳሤሴሥስሦሶዐአዑኡዒኢዓኣዔኤዕእዖኦጹፂጺጻፃፄጼፅጽፆሃሀኣኣ"
```

```
    type(normList)
    Nlist=normList
    i=0
    for character in normal:
        index=0
        for cha in Nlist[::2]:
          if character==cha:
              term.insert(i,Nlist[index+1])
               l=term.pop(i+1)
               break
          index =index+2
        i=i+1
    fterm="".join(term)
    word= stopwordremoval(fterm)
    word =str (word)
    return word
def stemmer(parms):
  stem = []
  stem= parms
  #PREFIX RMOVAL
  if len(stem)>=3:
    try:
      prefix = open("prefix.txt", "r",encoding='utf8')
      for line  in prefix.read().split():
        prefixsize= len(line)
        tempPrefix=stem[:prefixsize]
        if tempPrefix==line:
          #print (line + "prefix stemmed")
          stem= stem[prefixsize:]
          #print(stem)
    finally:
      prefix.close()
    # SUFIX REMOVAL
    try:
      suffix = open("suffix.txt", "r",encoding='utf8')
      for line  in suffix.read().split():
        suffixsize= len(line)
        tempsuffix=stem[-suffixsize:]
        if tempsuffix==line:
          #print (line + "suffix stemmed")
          stem= stem[:-suffixsize]
          # print(stem)
    finally:
      prefix.close()
    #print(stem)
 stem=normalization(stem)
  #print(stem)
  return stem
def multiplewordExtraction():
  wordlist=[]
  finaleMWords=[] # list for three word multiple term extraction
  mwtemplist=[]  # list for three word multiple term
  tmwtemplist=[] # list for two  word multiple term
```

```
finalMWords=[] #  list for three word multiple term extraction
mwords=[]      # multiple words variable
maxw=[]        # list for maximum frequency term
term=[]        # list help to find maximum frequency
temp=[]        # list help to find maximum frequency
templist=[]    # list help to find maximum frequency
maxfrequency=0 # variable help to find maximum frequency
threshold=0      # variable help to find threshold value
mtermabovethreshold=[] # list to find multiple word term above threshold value
tfin=[]              # list to find two word multiple term frequency
tmwtemplist=[]         #list to extract two word multiple word
rwmpmem=[]
try:
        i=0
        maxword=0
        semanticfile = stri
       for maxw in semanticfile.split():
            temp.append(maxw)
        #templist.append("")
        for maxw in temp :
            #CHECK THE WORD IN THE LIST
            tempmax=0
            j=0
            if maxw in templist :
                j=0
           else :
                j=1
           #IF THE FREQUENCY IN THE NOT IN THE LIST
            if j==1 :
                templist.append(maxw)
                for term in temp:
                    if maxw==term:
                        tempt=maxw
                        tempmax=tempmax + 1
                #print (maxw + str(tempmax))
                if tempmax > maxword:
                    maxword=tempmax
          #FIND THE TRHESOLD VALUE
        threshold=math.sqrt(2*maxword)
        threshold= math.floor(threshold)
         #THREE WORD MULTIPLE WORD EXTRACTION
        x=0
        for item in wordlist:
           x=x+1
        del wordlist[0:x]
         word=[]
        semanticfile =stri
        for words  in semanticfile.split():
                 wordlist.append(words)


        x2=0
        for i in finaleMWords:
```

```
   x2=x2+1
del finaleMWords[0:x2]
for n in range (len(wordlist)-2):
     finaleMWords.append(wordlist[n:3+n])
for fin in finaleMWords:
    lis=[]
    j=1
    frequency=0
    if j==1 :
        lis.append(fin)
        #print(fname)
        #lis.append(fname)
        mwtemplist.append(lis)
        for term in finaleMWords:
           if fin==term:
               tempt=fin
               frequency=frequency + 1
        #print(fin,frequency)
        c_value=math.log (3,2) *frequency
        cvalue=[]
        if c_value>=threshold:
            cvalue.append(tempt)
            cvalue.append(frequency)
            #cvalue.append(fname)
            wordl=[]
            tt=" ".join(tempt)
            #print(tt)
            dic=open("dictionary.txt", "r",encoding='utf8')
            for dwords  in dic.read().split("."):
               for wordl in dwords.split(","):
                  world="".join(wordl)
                  world=world.strip()
                  if world==tt:
                     #print (world)
                     validMterm.append(cvalue)
            dic.close()
            #print(validMterm)
            #print(c_value)
#TWO WORD MULTIPLE TERM EXTRACTION COMPUTE C_VALUE
tempmem3=[]
#print (validMterm)
x1=0
for item in finalMWords:
   x1=x1+1
del finalMWords[0:x1]
for n in range (len(wordlist)-1):
     finalMWords.append(wordlist[n:2+n])
for tfin in finalMWords:
    lis=[]
    j=1
    frequency=0

    if j==1 :
```

```
        lis.append(tfin)
        #lis.append(fname)
        tmwtemplist.append(lis)
        for term in finalMWords:
            if tfin==term:
                tempt=tfin
                frequency=frequency + 1
        #print (tempt)
        #print (frequency)
        tempmem1=[]
        tempmem2=[]
        tempmem4=[]
        k=0
        l=0
        if tempt not in tempmem3:
            for tempmem1 in validMterm:
                if tempt in tempmem3:
                    for tempmem2 in tempmem1[::2]:
                        for n in range (len(tempmem2)-1):
                            if tempmem2[n:2+n] == tempt:
                                k= k + int(tempmem1[1])
                                l=l+1
                else:
                    for tempmem2 in tempmem1[::2]:
                        for n in range (len(tempmem2)-1):
                            if tempmem2[n:2+n] == tempt:
                                tempmem3.append(tempmem2[n:2+n])
                                k= int(tempmem1[1])
                                l=l+1
            if k>0:
                c_value=math.log(2,2) *(frequency-((1/l)*k))
                frequency=frequency - k
            else:
                c_value =math.log(2,2)* frequency
            cvalue=[]
            if c_value>=threshold:
                cvalue.append(tempt)
                cvalue.append(frequency)
                #cvalue.append(fname)
                wordl=[]
                tt=" ".join(tempt)
                dic=open("dictionary.txt", "r",encoding='utf8')
                for dwords  in dic.read().split("."):
                    for wordl in dwords.split(","):
                        world="".join(wordl)
                        world=world.strip()
                        if world==tt:
                            #print (world)
                            validMterm2.append(cvalue)

                dic.close()
    frequency=0
#prevfname=fname
```

```
            if validMterm2:
                validMterm.extend(validMterm2)
                #print (validMterm)
        except e:
            print(str(e))
clustersim=[]
docsim1=[]
docw=[]
queryterm=[]
qterm=[]
UserQuery=input("እባኮን የሚፈልጉትን ፋይል ለማግኘት መጠይቆን ያስገቡ!:-")
for st in UserQuery.split():
    j=0
    for st2 in UserQuery.split():
        if st==st2:
            j=j+1
    te=[]
    tok=stemmer(st)
    te.append(tok)
    te.append(j)
    queryterm.append(te)
    dic=open("dictionary.txt", "r",encoding='utf8')
    for dwords  in dic.read().split("."):
                xx=[]
                for wordl in dwords.split(","):
                    world="".join(wordl)
                    world=world.strip()
                    ll=[]
                    ll.append(world)
                    ll.append(j)
                    xx.append(ll)
                    for bb in world.split():
                        kl=[]
                        kl.append(bb)
                        kl.append(te[1])
                        xx.append(kl)
                if [tok,j]  in xx:
                        xx.remove([tok,j])
                        queryterm.extend(xx)
                        #print(queryterm)
    #print(tok)
    dic.close()
    stri=stri+tok+" "

multiplewordExtraction()
#print(validMterm)
for m in validMterm:
    kk=[]
    s=" ".join(m[0])
    kk.append(s)
    kk.append(m[1])
    queryterm.append(kk)
    dic=open("dictionary.txt", "r",encoding='utf8')
```

```
    for dwords  in dic.read().split("."):
                xx=[]
                for wordl in dwords.split(","):
                    world="".join(wordl)
    world=world.strip()
                    ll=[]
                    ll.append(world)
                    ll.append(m[1])
                    xx.append(ll)
                if [s,m[1]]  in xx:
                        xx.remove([s,m[1]])
                        queryterm.extend(xx)
    dic.close()
#print(queryterm)
for jj in queryterm:
     qterm.append(jj[0])
#SIMMILARITY MEASUR TO CLUSTER REPRESENTATIVE
clusterg=[]
cluR=open("clusterR.txt", "r",encoding='utf8')
for dwords  in cluR.read().split(","):
        if dwords:
                #print(dwords)
                 ss=[]
                clusterunit=[]
                dname1=dwords
                voca=[]
                voca1=[]
                terms=[]
                word=""
                dic=open("postingfile.txt", "r",encoding='utf8')
                #if dname1!=dname2:
                for dwords  in dic.read().split("#"):
                       for word in dwords.split("]") :
                            newstr1 = word.replace("[", "")
                            newstr2 = newstr1.replace("'", "")
                            newstr3 = newstr2.replace(",", "")
                            '''for xy in word:
                                if xy != ']':
                                      dd= dd + xy'''
                            newstr4 = newstr3.replace(",", "")
                            if newstr4:
                                  voca1.append(newstr4.strip())
                            #print(newstr4.strip())
                            break
                       #if voca:
                       #       terms.append(voca)
                #print(voca1)
                dic.close()
                dic=codecs.open("postingfile.txt", "r",encoding='utf8')
                for dwords  in dic.read().split("#"):
                       voca=[]
                       for word in dwords.split("]") :
                            rr=[]
```

```
            newstr1 = word.replace("[", "")
            newstr2 = newstr1.replace("'", "")
            newstr3 = newstr2.replace(",", "")
            '''for xy in word:
                if xy != ']':
                    dd= dd + xy'''
            newstr4 = newstr3.replace(",", "")
            for cc in newstr4.split(" "):
                if cc:
                    rr.append(cc.strip())
            if rr:
                voca.append(rr)
        if voca:
                voca.pop(0)
        if voca:
            terms.append(voca)
 dic.close()
#print(terms)
i=0
nomu=0
denomu1=0
denomu2=0
ch=1
z=0
for mm in terms:
        for ll in mm:
            if ll[0]==dname1 and voca1[i] in qterm :
                z=1
                #print(voca1[i])
        for qt in queryterm:
            if voca1[i]== qt[0] and z==1:
                ch=1
                #print(qt[0])
                break
            else:
                ch=0
        if ch==1 :
            ter=mm[0]
            w1=0
            w2=0
            for ll in mm:
                if ll[0]==dname1:
                    w1=ll[1]
                    #print("docu",voca1[i],w1)
                for qt in queryterm:
                    if voca1[i]== qt[0]:
                        w2=qt[1]
                        break
                        #print(qt[0],w2)


            nw=float(w1)*float(w2)
            dw1=float(w1)*float(w1)
```

```
                    dw2=float(w2)*float(w2)
                    nomu=nomu+nw
                    denomu1=denomu1+dw1
                    denomu2=denomu2+dw2
                i=i+1
            #print("==============================")
            c_sim= nomu/ (1 + (math.sqrt(denomu1)* math.sqrt(denomu2)))
            clusterunit.append(dwords)
            clusterunit.append(c_sim)
            clusterg.append(clusterunit)
            ss1=[]
            ss1.append(dname1)
            ss1.append(c_sim)
            clustersim.append(ss1)
             #print("cosine simmilarity with ",dname1,"is ",c_sim)
            #print("\n\n")
  #print("OTHER CLUSTER",cname,"******************************")
cluR.close()
#TO FIND THE REPRESENTATIVE OF CLUSTER WHICH CLOSE TO THE QUERY
print("\n\n========================= የ ፍለጋው ወጤት እደሚከተለው ነ ው
================================\n\n")
if clustersim[0][1]==0 and clustersim[1][1]==0 and clustersim[2][1]==0 and clustersim[3][1]==0:
    print("ባስገቡት መጠይቅ መሰረት ጠቃሚ መረጃ በመረጃ ቋአጥ ውስጥ የለም.....")
else:
    clu=""
    if clustersim[0][1]>=clustersim[1][1]:
            if clustersim[0][1]>=clustersim[2][1]:
                if clustersim[0][1]>=clustersim[3][1]:
                    clu=clustersim[0][0]
                else:
                    clu=clustersim[3][0]
            else:
                if clustersim[2][1]>=clustersim[3][1]:
                        clu=clustersim[2][0]
                else:
                        clu=clustersim[3][0]
    else:
        if clustersim[1][1]>=clustersim[2][1]:
                if clustersim[1][1]>=clustersim[3][1]:
                        clu=clustersim[1][0]
                else :
                    clu=clustersim[3][0]
        else:
                if clustersim[2][1]>=clustersim[3][1]:
                    clu=clustersim[2][0]
                else:
                     clu=clustersim[3][0]
    #print(clustersim[0][1])
    #print(clu)
```

```
# TO FIND MORE RELEVANT DOCUMENT IN THE CLUSTER WITH RANK
clusterg=[]
cluR=open(clu, "r",encoding='utf8')
for dwords  in cluR.read().split(","):
        if dwords:
                #print(dwords)
                ss=[]
                clusterunit=[]
                dname1=dwords
                voca=[]
                voca1=[]
                terms=[]
                word=""
                dic=open("postingfile.txt", "r",encoding='utf8')
                #if dname1!=dname2:
                for dwords  in dic.read().split("#"):
                        for word in dwords.split("]") :
                                newstr1 = word.replace("[", "")
                                newstr2 = newstr1.replace("'", "")
                                newstr3 = newstr2.replace(",", "")
                                '''for xy in word:
                                    if xy != ']':
                                        dd= dd + xy'''
                                newstr4 = newstr3.replace(",", "")
                                if newstr4:
                                        voca1.append(newstr4.strip())
                                #print(newstr4.strip())
                                break
                        #if voca:
                        #       terms.append(voca)
                #print(voca1)
                dic.close()
                dic=codecs.open("postingfile.txt", "r",encoding='utf8')
                for dwords  in dic.read().split("#"):
                        voca=[]
                        for word in dwords.split("]") :
                                rr=[]
                                newstr1 = word.replace("[", "")
                                newstr2 = newstr1.replace("'", "")
                                newstr3 = newstr2.replace(",", "")
                                '''for xy in word:
                                    if xy != ']':
                                        dd= dd + xy'''
                                newstr4 = newstr3.replace(",", "")
                                for cc in newstr4.split(" "):
                                        if cc:
                                            rr.append(cc.strip())
                                if rr:
                                    voca.append(rr)
                        if voca:
                                    voca.pop(0)
                        if voca:
                                terms.append(voca)
```

```
dic.close()
#print(terms)
i=0
nomu=0
denomu1=0
denomu2=0
ch=1
z=0
for mm in terms:
      for ll in mm:
           if ll[0]==dname1 and voca1[i] in qterm :
                z=1
                #print(voca1[i])
      for qt in queryterm:
          if voca1[i]== qt[0] and z==1:
               ch=1
               #print(qt[0])
               break
          else:
               ch=0
       if ch==1 :
            ter=mm[0]
            w1=0
           w2=0
           for ll in mm:
               if ll[0]==dname1:
                    w1=ll[1]
                     #print("docu",voca1[i],w1)
               for qt in queryterm:
                   if voca1[i]== qt[0]:
                        w2=qt[1]
                        break
                        #print(qt[0],w2)
            nw=float(w1)*float(w2)
           dw1=float(w1)*float(w1)
           dw2=float(w2)*float(w2)
           nomu=nomu+nw
           denomu1=denomu1+dw1
           denomu2=denomu2+dw2
      i=i+1
#print("============================")
c_sim= nomu/ (1 + (math.sqrt(denomu1)* math.sqrt(denomu2)))
clusterunit.append(dwords)
clusterunit.append(c_sim)
clusterg.append(clusterunit)
ss1=[]
ss1.append(dname1)
ss1.append(c_sim)
if c_sim >0:
     docw.append(c_sim)
     docsim1.append(ss1)
```

```
                    print("cosine simmilarity with ",dname1,"is ",c_sim)
                #print("\n\n")
    #print("OTHER CLUSTER",cname,"*******************************")
 #RANKING THE DOCUMENT
docw.sort()
docw.reverse()
#print(queryterm)
i=1
#print (dwords)
for l in docw:
    for y in docsim1:
        if y[1]== l:
            print(i,y[0])
            docsim1.remove([y[0],y[1]])
            i=i+1
```