ADDIS ABABA UNIVERSITY

COLLEGE OF NATURAL SCIENCES

SCHOOL OF INFORMATION SCIENCE

ONTOLOGY BASED QUERY EXPANSION FOR ENHANCING THE PERFORMANCE OF AMHARIC INFORMATION RETRIVAL:

The Case of Tourism Sector

WELDE JANFA

OCTOBER, 2014

ADDIS ABABA UNIVERSITY

COLLEGE OF NATURAL SCIENCES

SCHOOL OF INFORMATION SCIENCE

ONTOLOGY BASED QUERY EXPANSION FOR ENHANCING THE PERFORMANCE OF AMHARIC INFORMATION RETRIVAL:

The Case of Tourism Sector

A Thesis Submitted to the College of Natural Sciences of Addis Ababa University in Partial Fulfillment of the Requirements for the Degree of Master of Science in Information Science

By

WELDE JANFA

OCTOBER, 2014

ADDIS ABABA UNIVERSITY

COLLEGE OF NATURAL SCIENCES

SCHOOL OF INFORMATION SCIENCE

ONTOLOGY BASED QUERY EXPANSION FOR ENHANCING THE PERFORMANCE OF AMHARIC INFORMATION RETRIVAL:

The Case of Tourism Sector

By

WELDE JANFA

Name and signature of members of the examining Board

Name	**Title**	**Signature**	**Date**
_____	Chairperson	_____	_____
Dr. Dereje Teferi (Ph.D)	Advisor	_____	_____
Dr. Million Meshesha (Ph.D)	Examiner	_____	_____

Dedication

To all my family

Acknowledgment

Before all, I praise the almighty **GOD** and his mother **ST. MARY** for making everything the way it is. Next, I am so glad to express my warm gratitude to my advisor **Dr. Dereje Teferi** for being with me in all the ups and downs of this work. Thank you so much for showing and guiding me on my research work.

I would like to thanks to all my family members specially Janfa G/Senbet, Abebech Sima, Eshetu Janfa and His Wife Eskedar Kelemu and Werke Janfa. You have shaped me by giving all the necessary support to reach this day.

Also, thanks to Biniyam Asinake, for providing all the resources, precious advices and supporting ideas towards my work.

Special people (Kasahun H/Mariam, Knife H/Mariam, Damenech Tebeka, Kidist Taye, Gebiyanesh Gesese and Desalegn Wehega) who encourage, love and support me with all you have, I have nothing except THANK YOU.

My sincere thanks go to Getnet Admasu and Feleke Afewerik, for all your support and encouragement. I will not forget our time in the Billing Department.

Lastly, My heartfelt thanks to gose Mekonnen Tesifay Metaferia (MTM) and Meaza G/Abizgi, what can I say about you, You have put unqualified effort and push to take this work to an end.

Abstract

The availability of a plenty of information and knowledge sources has demanded a large amount of effort in the development and enhancement of Information Retrieval techniques. Query expansion has been one of the important techniques in an information retrieval area in order to increase the efficiency and effectiveness of the search engine. Query expansion, improves the performance of the IR system by augmenting additional word to the user query from the given knowledge source. Adding additional power full words to the user query helps to generate more useful information to the user.

This research focuses on automatic query expansion mechanism to increase the searching power of the user query. Many types of query expansion have been applied and tested to facilitate the information retrieval process. In this research work, an attempt is made to integrate an ontology based query expansion technique to enhance the effectiveness of the system in retrieval process. Ontologies are usually huge repositories of concepts and relations between concepts in a certain domain. Using the ontologies in information retrieval improves the retrieval goal. The aim of this research is to explore the impact of the use of ontology for query expansion to improve retrieval results. Small size domain specific corpus dependent ontology is constructed and used in the proposed query expansion model.

Experiments are carried out on the system before and after integrating the proposed techniques. The results show that ontology based query expansion has resulted in a higher number of relevant documents being retrieved compared to other query expansion process. Overall, ontology based query expansion improves recall by 42% and system performance by 7%. The major challenge is constructing domain specific ontology and corpus that is an important element in this research which needs more work in the future.

Table of Contents

List of figures

List of Tables

Chapter One
INTRODUCTION

1.1 Background

The growth of the World Wide Web and Internet both in content and users played an important role in the way people look and share information. This advancement also helps to radically improve the search engine technology in the way that knowledge and information is collected and shared as easily as possible [1].

IR systems typically seek to find documents in a given collection that are "about" a given topic or that satisfy certain information need [2]. Therefore, the major goal of information retrieval systems is to retrieve all the documents which are relevant to a user query while retrieving as few non-relevant documents as possible [3]. However, there are still a significant number of cases where the results obtained through an IR system, retrieved a high number of irrelevant results. There are many reasons for this problem. The main problem is that users formulate a short query for their information need or users have a problem of formulation of good query for their information need. To reduce this problem, IR systems use reformulation of a query that is submitted by users [3].

There are many ways of improving the initial query formulation through query expansion and term weighting [3]. The first approach is based on feedback information from the user. In this method, the user is presented with a list of the retrieved documents and, after examining them, marks those which are relevant. Based on this relevance mark new query terms are added to the initial query terms. The second method of improving the original query is based on information derived from the set of documents initially retrieved. This method is similar to the first one except that there is no user involvement rather, the system itself tries to identify the expansion terms automatically. The third approach is based on global information derived from the document collection. Unlike the previous methods which look only the local document, this method looks at global information to find words that can improve the original query terms [3]. This approach can be implemented using similarity thesaurus [4], statistical co-occurrence

analysis [5], WordNet [6], and using ontology based query expansion [1]. The aforementioned techniques are a form of global analysis, but we are going to use ontology in our work.

More recently, researches have been done on using ontologies in query expansion. Ontology is a collective body of knowledge which is usually created and shared by users who are experts in that domain. An ontology that is a collective view of the domain is more likely to be an accurate and comprehensive representation of that domain. Domain ontology models a specific domain. It represents the particular meanings of terms as they apply to that domain. The ontology can be collection dependent or collection independent. The advantage of using collection dependent ontologies is that relevant terms in the collection documents are found in the ontology and those terms be used for the porpous. Ontologies can also be used to infer context for ambiguous queries. The concepts in the ontology can be used for word sense disambiguation and subsequent query expansion. An ontological model can effectively disambiguate meanings of words from free text sentences [7].

In this research, the focus is on how ontology based query expansion method improves the performance of Amharic information retrieval system.

1.2 Statement of the Problem and Justification

At this time, creation, codification and storing of electronic documents has tremendously increased. In addition to collecting and storing of these documents, there is a need to share and distribute it to others. To accomplish the sharing as need, one needs an effective Information Retrieval (IR) system.

Tewodros [8] has done an experiment on Amharic text retrieval using semantic indexing (LSI) with singular value decomposition. Amanuel [9] has also developed Amharic information retrieval system using probabilistic method.

An effective information retrieval system needs a proper formulation of the query so as to retrieve relevant documents to the user [10]. However, most users have a problem of constructing a good query for their information need. A good IR system should assist the users by improving th user query representation. Various approaches have been proposed to improve IR systems. Among them, query expansion is arguably one of the most effective approaches to achieve better effectiveness of an IR system [11].

Query Expansion is the process of empowering the user's query with additional terms in order to improve the search results. The additional terms must have the same meaning with the original query. If the meaning of the new term is far from the original meaning we might end up with a query drift problem [1]. Query drift happens when the augmented user query results a different meaning from the intended one. Selecting the right term to expand a user query is the main challenge in query expansion methods in IR system.

Researches have been also done on Amharic query operation to expand a user query. One work is by Alemayehu [4]. He attempted to apply query expansion to control synonymous words using a thesaurus. Due to a tradeoff between recall and precision and because of the existence of polysemous terms; his proposed method decreased the overall precision of the IR system. Another work was by Abey [5], who used statistical co-occurrence analysis, bi-gram analysis and bi-gram thesaurus methods, to differentiate the meanings of a polysemous query term using other query terms given by a user. However, the expansion terms used are found to be polysemous themselves. That is because; his research doesn't consider the extent to which a term can be an expansion term for each query term. The latest research by Iman [6] applies Word sense disambiguation to choose the right term that can be used to expand the query. The work handled the problem caused by polysemous word in the previous research.

In this research, an attempt is made to explore ontology based query expansion for the improvement of Amharic IR system.

Towards this end, this study explores and answers the following research questions.

- How it is possible to develop ontology based query expansion model?
- Does the proposed query expansion method improve the effectiveness of the Amharic IR system?

1.3 Objectives of the Study

This research aims to attain the following general and specific objectives.

1.3.1 General objective

The general objective of this research is to design an ontology based query expansion method and integrate it with an Amharic information retrieval system so as to improve the retrieval effectivenes of the system.

1.3.2 Specific objectives

The specific objectives of this research are:

- To review different literatures that can give detailed knowledge and understanding of the problem area and to study the existing work regarding to the problem as well as the challenges of the different approaches to query expansion.
- To prepare Amharic document corpus and test queries that can be used to measure the performance of the integrated system.
- To develop ontology based query expansion technique that can improve the searching result of the expanded user query with additional query terms.
- To integrate the ontology based query expansion system with existing Amharic document retrieval system.
- To carry out an experiment on the integrated system using test query.
- To evaluate the performance of the Amharic information retrieval system that integrates the new model using a test dataset.

1.4 Scope and Limitation of the Study

Considering the previous researches on query expansion for the Amharic IR system [4] [5] [6], this research scope is limited to developing a new query expansion techniques using ontology as a source knowledge. The proposed ontology based query expansion model is integrated with an Amharic IR system designed by Amanuel [9] and testing is carried out on the integrated system. Regarding the data corpus and test query, we used domain specific corpus dependent ontology. However, because our corpus size is very limited and does not represente all the area of our domain, we collected additional concepts (words) from external source. Since this research work

focus on a domain specific ontology, new test query is identified to undertake the evaluation of the performance of the system.

The limitation of this study is that, we have used a simple ontology which does not include rich semantic information and it is also relatively very small in size. In addition to this, due to the absence of standard corpus for evaluation the coverage of the experimentation is very limited. However, despite these limitations, we think it is a good starting point for future research in this area.

1.5 Methodology of the Study

Methodology is a mechanism of conducting a research including, the sort of data needed to answer the research questions and the details of how this is to be achieved in practice [12]. Hence, in order to achieve the specific and general objectives of the study and answer the research questions, the following approaches and methods are used.

1.5.1 Literature Review

Journal articles, books, conference papers, material from the internet, and related publication are reviewed to understand information retrieval system, query expansion, ontology, and related concepts, theories, principles, approaches, algorithms, techniques and tools on the area. Moreover, all previously conducted researches related to the topic are carefully reviewed.

1.5.2 Dataset Collection

The envisage method is tested with domain specific corpus dependent dataset and selected test query. For that reason, domain specific documents are collected from WWW and the internet. The dataset consists of 195 documents which are almost specific to the tourism domain. The documents include news articles, which talks about tourism domain, and address of the places where tourists want to go. Ten test queries are identified and extracted from the ontology concepts for the purpose of testing the integrated system.

1.5.3 Implementation Tool

To develop the prototype, Python programming language is used. The major reasons for choosing a Python programming language are: first, all previous attempts that developed Amharic query expansion system used Python. Second, it is an object oriented programing

language which is suitable and flexible for text processing activity. Thirdly, the researcher has a good programming knowledge and skill in the selected programming language, Python. As a result, query expansion model for the Amharic information retrieval system is developed using Python.

1.5.4 Testing Procedure

Before beginning any final implementation of the proposed solution, the solution should be tested and evaluated to check its performance using standard ways and available tools in the area. Amharic document and test query terms are used to evaluate the performance of the query expansion model. With the given query as input, the performance of the Amharic information retrieval system is measured using the three most widely used retrieval performance evaluation measuring techniques, namely: recall, precision and f-measure [3].

Recall is the fraction of the relevant documents which has been retrieved while Precision is the fraction of the retrieved documents which are relevant. Recall and precision have been used extensively to evaluate the retrieval performance of search algorithms. This is because recall and precision are related measures which capture different aspects of the set of retrieved documents. In many situations, the use of a single measure which combines recall and precision could be more appropriate. One such measure is the harmonic mean (F-measure) of recall and precision [3].

1.6 Significance of the Study

Information retrieval system is a vital tool for distributing the accumulated information resource to those who need that information. The IR system should be effective and efficient to let the user use it for their information need. This research work attempt to integrate ontology based query expansion model, which can identify the best query terms that can be used to expand the user query. This study proved to what extent the ontology based query expansion improve the effectiveness of the Amharic Information retrieval system. And also it is a starting point for future research works on query expansion and ontology based query expansion works.

1.7 Organization of the Study

This thesis is organized into five chapters. The first chapter discusses the background of the study and a statement of the problem. It also presents general and specific objectives of the study, the methodology of the study, scope and limitation of the research , process methodology followed n this work , and the significant of the work.

In chapter two, literature review on information retrieval, ontology, query expansion, ontology based query expansion is presented. Moreover, a brief review of the history, development and characteristics of the Amharic language and global and local related works on query expansion are discussed.

Chapter three describes the proposed ontology based query expansion techniques and algorithms. The basic steps of using ontologies, selecting the terms and weighting of selected terms are briefly discussed with their algorithms.

Chapter four emphasizes the integration of ontology based query expansion with IR system and experimental results that are used to confirm the usability of the proposed ontology based query expansion techniques to retrieve relevant information from Amharic text documents.

Chapter five gives conclusions drawn from the results of the experiment in the study. It also includes recommendations that should be followed in future researches in the area of ontology based query expansion.

Chapter Two
LITERATURE REVIEW

2.1 Overview

The advancement of Information technology makes it easy to have huge amounts of electronic format information. The limitation of storage device becomes no more a problem in the field of information Science. In addition to this, the World Wide Web (WWW) allows to easily avail and makes accessible electronic format information to the information seeker. Inspite of having the huge amount of information and easy access to it, providing the right information at the right moment in time to the right person is a challenging problem or users might still have to read through a lot of content before they find exactly what they are looking for [13].

For a long period, researchers have been trying many things to bring a solution for this issue. Vannevar Bush in 1945, brought the idea of automatic accessing machine for large amounts of stored information (i.e. He raised the idea in his article titled "As We May Think" [14]). Still, attempts are underway to minimize the above problem. Around 1957 H.P. Luhn proposed using words as indexing units as a means which enhance the activity of document retrieval and providing of the right information to the information seeker [15]. In 1971 the system called SMART retrieval system was developed by Gerard Salton to retrieve documents automatically and become the beginning of modern and organized information retrieval systems [16]. Since then information retrieval become a dominant way of finding information from the large amount of information sources.

Information retrieval (IR) is a mechanisms invented by mankind that facilitates and changes the nature of information access and sharing among people worldwide. Using Information Retrieval (IR) principle, we can design an information search engine that enables users to find out the relevant information from large document collection [16]. To achieve this goal an IR system uses different tools and techniques such as IR models, Query formulation, etc.

2.2 Information Retrieval (IR)

Information retrieval (IR) refers to the process of representation, storage, organization of, and access to information items [3]. The representation and organization of the information items

allow users easy access to the information that they want [3]. According to [17] Information retrieval is defined as:

> *"Information retrieval (IR) is the process of finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)".*

In the definition above, there are three basic elements in IR: Figure 2.1, below depicts these three elements within the standard Information Retrieval model. Unstructured document, information need and the process of finding the required information.

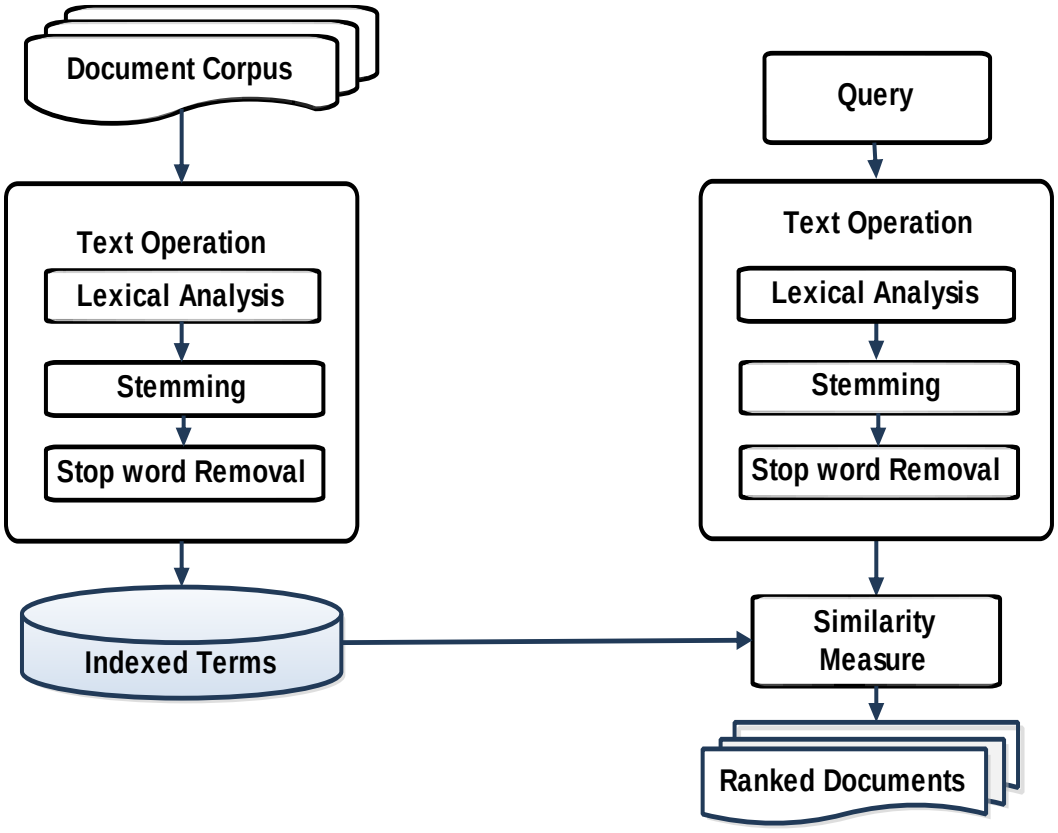

Figure 2.1: The Standard format of Information Retrieval model [5].

To retrieve some information using IR system the information should be stored inside a computer. Since the documents are unstructured in nature, the first step in any IR system is to translate this unstructured document text to a suitable representation format [20]. The logical view or representation of the documents implemented by the IR system has a direct effect on the effectiveness of the IR system in searching the relevant information [4]. Most IR systems use a

set of index term to represent the document collection. The index terms are mostly extracted directly from the text document and the process is called indexing [17, 20]. The indexing might be a full term index or keyword representation. If the indexation is a keyword based, the terms are selected based on their importance to identify the given document and it passes through a series of activities such as stop word removal and stemming. Figure 2.2 illustrates how the index terms are extracted from the document collection.

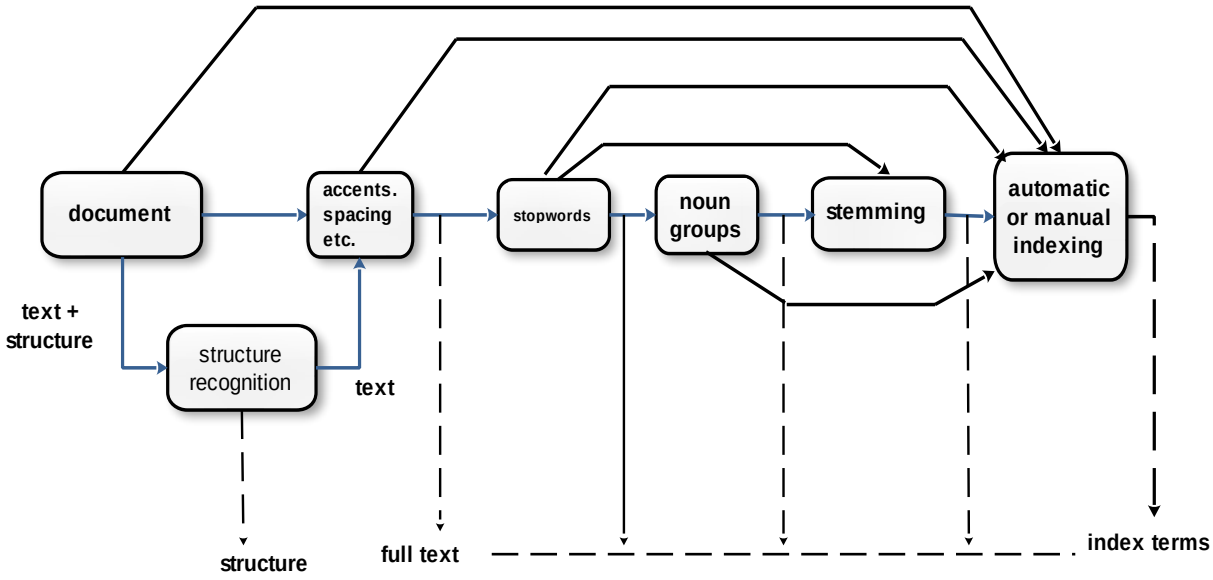

Figure 2.2 Logical view of the document using its index terms [3]

The next element in IR system is information need. Information seeker should express their information need into the appropriate query in the language supported by the IR system. This user activity also has an effect on the effectiveness of the IR system. After the IR system receive a user query it represents the query like the way it represent the document in the collection. This identical representation of documents and query helps the system in matching of the document and the query given by the user [3, 20].

The Final steps in the IR system is identifying the right document which satisfies the query (i.e. Information need) given by the user. Different IR systems use different IR models to identify documents which are more relevant to the user information need. Models such as Boolean model, Probabilistic model and Vector space model [3, 18,19] are used in IR systems. These models implement different methods to isolate the relevant document from the non-relevant one and rank them according to their relevance.

Most of the time users transform their information need into a query that is not detailed enough to retrieve the right information [21]. This challenge can be dealt with at least in two ways: one is to refine the user queries, and the other is to optimize the search service in hopes that the search service returns reasonable results whatever the user input is. Query refinement is a middle process between the indexation and searching of the document [22]. Our target in this research is to refine the user query by using domain specific ontology.

In general, the main goal of information retrieval is providing relevant information for information seeker. To achieve this aim, IR systems carry out the following three step process: Indexing, searching and query expansion [22].

1. The indexing process aims at representing resources (often documents) and queries with sets of (weighted) terms (or concepts) that best summarize their information content.
2. The search is the core process of an IR system. It contains the system strategy for retrieving documents that match a query.
3. The query expansion is an intermediate process that reformulates the user query, based on internal system information, to improve the quality of the result.

2.3 Ontology

In this study we use ontology to select good expansion terms which can expand the user's original query. To this end, this section provides basic issues related to the definition of ontology, development and construction of ontology, ontology languages and its applicability in our research.

2.3.1 Definition of Ontology

The notion of ontology is old and attached to philosophy. It is defined as the branch of philosophy which studies the existence of all kinds of entities – abstract and concrete – that make up the world [43]. There is still a debate on what is actually meant by the term ontology [37]. Because of those debates, there are many definitions available for the term ontology in different literature. Eva [37] summarized some of the definitions found in the literature as follows:

1. *"Ontology is an explicit specification of a conceptualization."*

2. *"Ontology: (sense 1) a logical theory which gives an explicit, partial account of a conceptualization; (sense 2) synonym of conceptualization."*
3. *"Ontology is a formal, explicit specification of a shared conceptualization."*
4. *"Ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e., its ontological commitment to a particular conceptualization of the world. The intended models of a logical*

However, from the perspective of information science and this research ontology is defined as:

> *"a formal representation knowledge as a set of concepts within a domain and the relationship between those concepts [44]".*

According to this definition ontology commonly consist of concepts, relations, axioms, and other constraints in the domain. It provides a shared vocabulary for describing and reasoning the concrete instance of the domain. Ontologies usually contain a hierarchical ordering of the concepts, denoted taxonomy or subsumption hierarchy [37].

Ontology is becoming increasingly important in many areas such as knowledge management, information integration, information retrieval and electronic commerce [45]. Based on the area and intended use of it, an ontology may belong to a specific or a general domain area. In this research, we constructed domain specific and corpus dependent ontology which we use to identify the best terms to expand the user query.

2.3.2 Types of ontology

Corpus Independent Ontology vs. Corpus Dependent Ontology

Ontology can be crafted regardless of the underlying corpus that the queries are searched over. These kinds of ontologies are called corpora independent ontology. Naturally, these corpus independent ontologies are unaware of the corpus content. WordNet serves as a domain-specific and corpus independent ontology. It has been widely used in query expansion tasks, for example, [6] expands a query with words that are lexically relevant to some keyword via WordNet. On the other hand ontology can be constructed based on concepts found from the given corpus that the queries are searched over. These kind of ontologies are called corpus dependent ontology and

are constructed semi-automatically [37] or fully automatically 36]. Automatic ontology construction is still a challenging task [37, 36, and 46].

Upper-Level Ontologies versus Domain Ontologies

A further distinction can be drawn between upper-level and domain ontologies. Domain ontologies are usually defined to serve specific tasks in a certain application domain, whereas upper-level ontologies serve more universal goals. They aim to define the world in terms of basic classes and relations so that other domain ontologies can build on them. To describe the basic elements of the world, the *Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE)* upper level ontology, for example makes a top-level distinction between endurant and perdurant entities, qualities and abstract entities. Unfortunately, none of the existing upper-level ontologies has found wide acceptance yet, possibly due to their complex structures whose application requires a high learning effort. This explains why ontology engineers tend to define domain ontologies rather than specifying world-embracing upper-level ontologies [46].

Wordnet

Query expansion could be achieved by adding synonymous words or semantically related words from a lexical database called WordNet [40]. The WordNet is organized as sets of synonyms (words) called Synset, with the same meaning. These synsets have different relations between them. The relation of hypernymy/ hyponymy (is-a relation) is the principal relation and creates a hierarchic structure and the relation of metonymy/holonymy (part-of relation) [39]. Wordnet is constructed manually and it is a time consuming task. The first English WordNet [38] was constructed by Princeton University. The project took more than two decades to complete [6].

During the expansion, for each term found in the user's query, the system tries to expand this query term by using its respective synonym set (synsets) fetched from the WordNet. The selected terms have relation with the user query [40]. However, using pure WordNet for the expansion do not exhibit to much performance improvement in the IR system. Additional techniques such as word sense disambiguation [6, 41] and association rules [40] are introduced to improve the performance of the Wordnet query expansion method.

2.3.3 Component of ontology

Ontology consists of a number of different components. Depending on the ontology language used or the literature we read, those components have different names. However, the main components of ontologies are concepts, relations, instance and axioms [37].

Concepts: Concepts, also known as classes, are representing an existence of certain things. They are a description of physical objects such as man and woman or non-physical object such as task, function, action and strategy, reasoning process, etc. They are considered as a backbone of ontology and organized as a node in the taxonomic tree.

Relation: Relation in an ontology that designates the way in which concepts are related with each other. Relations are usually considered as directed arrows in a taxonomic tree. An important type of relation in an ontology construction is the subsumption (i.e. Is-a or subclass of) relation. This relation specifies which concepts are grouped under which other concepts.

Instances: Instances are a specific element of a certain conceptual domain. For example the person called 'ALMAZ' is an instance of a concept called 'Woman'

Axioms: An explicit rule to constrain in the use of concepts in the ontology is called Axioms.

2.3.4 Ontology Language

As we have seen the definition of ontology above, it is an abstract model of some aspect of the world and represented by concepts and relationship between concepts [47]. The purpose of ontology is to establish the same terminology about a concept between users. To have the same understanding about a concept and avoid ambiguity, there should be a common language. Users require an ontology language that allows users to write explicit and formal conceptualization of domain knowledge. Such a language should have the following property to carry out the activity well [47].

- A well-defined syntax
- A well-defined semantics
- Efficient reasoning support
- Sufficient expressive power, and
- Convenience of expression.

A number of researches have been done across Europe and America to build a common and effective ontology language [47], which facilitates the creation of a common framework that allows data to be shared more efficiently between applications. Using ontology language unstructured domain knowledge could now be described in terms of structured data. Examples of these languages are RDF - RDF (S), OIL, DAML+OIL, the most recent and common one is OWL. OWL is the web ontology language, developed by the W3CWeb Ontology (WebOnt) Working Group. OWL is mainly based on OIL and DAML+OIL, and therefore the main features of OWL are very similar to those of OIL.OWL was proposed as a W3C recommendation in February 2004 [47].

OWL will be preferred language for representing ontology because it is assumed to be better than the others for the following reasons [47]. OWL incorporates expressions for versioning. OWL has a rich, expressive power and possesses a layered architecture for scalability. OWL supports several natural languages compared to the other considered languages. These days OWL is very well positioned in the community mobilizing lots of efforts to make it become the Semantic Web language of the future. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary. OWL can be used to represent the meaning of terms and the relationships between those terms. It is more expressive than XML, RDF and RDF-S, making it easier to represent machine interpretable content on the web. OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users [47, 48]:

- **OWL Lite**: OWL Lite was originally intended to support those users primarily needing a classification hierarchy and simple constraints. OWL Lite excludes enumerated classes, disjointness statements and arbitrary cardinality. The advantage of this species is that, it is both easier to grasp (for users) and easier to implement (for tool builders). The disadvantage is of course a restricted expressivity.
- **OWL DL**: OWL DL was designed to provide the maximum expressiveness possible\ while retaining computational. OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, number restrictions may not be placed upon properties which are declared to be transitive). OWL DL is so named due to

its correspondence with description logic, a field of research that has studied the logics that form the formal foundation of OWL.

- **OWL Full**: The entire language is called OWL Full, and uses all the OWL language primitives. It also allows combining these primitives in arbitrary ways with RDF and RDF Schema. OWL Full is based on a different semantics from OWL Lite or OWL DL, and was designed to preserve some compatibility with RDF Schema. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right; this is not permitted in OWL DL. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary.

2.3.5 Ontology Learning

In this section we discuss the detail tasks involved in semi-automatic ontology construction or ontology learning. There are many already stored knowledge that are available. Such existing knowledge sources can be documents, databases, taxonomies, web sites, and other similar structures. The main question here is how we can extract the knowledge encoded in these sources automatically or semi-automatically and formulate it in ontology. To formulate ontology about certain knowledge, ontology learning passes through a series of steps. According to Chmiano [49] the process of ontology learning can be divided into a layer stack consisting of methods and algorithms (See Figure 2.3).

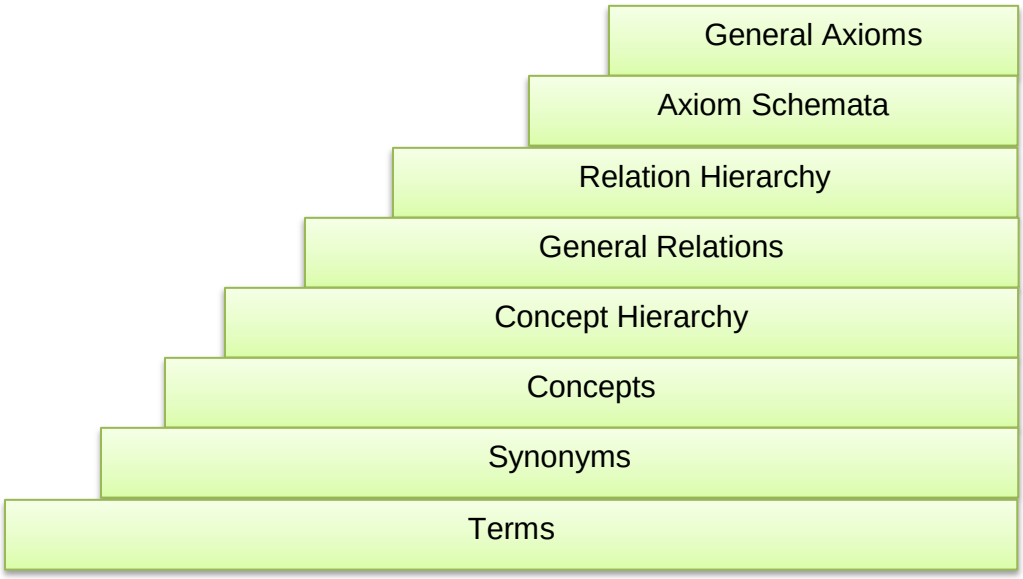

Figure 2.3: The Ontology Learning layers [37].

Term selection and Concept Extraction

This section describes how candidate concepts can be drawn from the text corpus. To extract the concept, we need to do document pre-processing, term extraction and concept formation. Each of the tasks is discussed below.

Documents Pre-processing: Ontology is constructed from the unstructured text corpus. Before term extraction begins the first activity should be text operation in the unstructured text corpus. The pre-processing of the document assists the term extraction step by supplying clean input text. To increase the quality of the text, we need to carry out a series of text operation activities. Document pre-processing includes sentence segmentation, tokenization, part-of-Speech tagging, stop word removal and Lemmatization. It is illustrated in Figure 2.4 below.

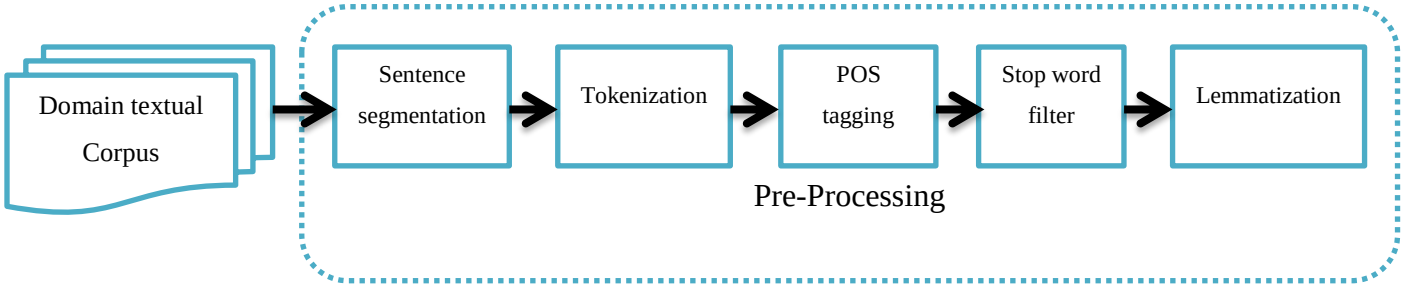

Figure 2.4: The flow chart of Pre-processing [36]

1. **Sentence Segmentation**: In this study, we simply segment document into sentences by Amharic punctuations.
2. **Tokenization**: The sentences are broken into word in this step.
3. **Part-of-Speech (POS) Tagging**: Generally, most concepts of ontology are nouns or partial nouns. In pre-processing stage the words in the corpus need to be tagged to choose the right terminology.
4. **Stop word Filter**: Words that are not important or week to identify the documents are called stop words. These words should be removed from the corpus in the pre-processing stage
5. **Lemmatization/ Morphological Analysis**: Lemmatization is a process of converting morphological variant words into their corresponding base/ Lema/ root form.

Term selection

After the documents pre-processing activities are completed, then the term selection task will follow. The aim of this step is to select important terms from the documents. These terms are single word or multiple-word terms. To extract this important term most research suggests using NLP, data mining, and text mining tools.

Single word term extraction

In most information retrieval research, TF/IDF (Term Frequency/Inverse Document Frequency) is used to extract a single word term from the corpus [50]. TF/IDF weights the frequency of a term in a document with a factor that discounts its importance when it appears in almost all documents. Therefore terms that appear too rarely in a document or too frequently in a corpus are ranked low [3]. The value of TF/IDF result helps to identify the important terms. The threshold value of the TF/IDF varies according to domain areas.

TF/IDF is a combination of two concepts that are term frequency and inverse document frequency. The first computes the normalized Term Frequency (TF), the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

- **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more time in longer documents than shorter ones. Thus, the term frequency is often divided by the document length (the total number of terms in the document) as a way of normalizing:

  TF (t,d) = (Number of times term t appears in a document) / (by the maximum raw frequency of any term in the document).

  TF (t, d) = $\frac{f(t,d)}{max\{(f(w,D):w\in D\}}$ **…… (2.4)**

- **IDF: Inverse Document Frequency**, measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as stop words, may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

  IDF (t,d) = log (Total number of documents / Number of documents which contain term t).

  $$\textbf{IDF(t, d)} = \log \frac{N}{df(t)} \quad \textbf{..... (2.5)}$$

  Where:

  - N is the total number of the document in a corpus
  - df (t) is the number of documents that term t appears in corpus. N is the

TF/IDF: **Term Frequency/Inverse Document Frequency**, which is calculated by combining of tf (t, d) and IDF (t, d) as:

$$\textbf{TF/IDF (t, d}_i\textbf{)} = tf(t, d) * log \frac{N}{df(t)} \quad \textbf{.....(2.6)}$$

A high weight in TF/IDF is achieved by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents.

Multi-word term selection

In the previous section we discussed about a statistical tool **tfidf** that is used to select content bearing single word terms from the corpus. In this section we present techniques which can be used to identify an important multi-word terms from the corpus.

.

- **Term Pair's Co-occurrence Weight (TPCW) [36]:** This method use co-occurrence frequency of the connected terms to recognize a multi-word term. If $\mathbf{T_{ij}}$, $\mathbf{T_{ik}}$ denotes two connected terms in document i , The TPCW of term pair ($\mathbf{T_{ij}}$, $\mathbf{T_{ik}}$) is calculated as :

$$\text{TPCW}(T_{ij}, T_{ik}) = \frac{2tf(T_{ij} \cap T_{ik})}{tf(T_{ij}) + tf(T_{ik})} \quad \text{..... (2.7)}$$

Where tf ($\mathbf{T_{ij}}$) denotes the number of T_j occurring in document i and tf ($\mathbf{T_{ij}}$ $\mathbf{T_{ik}}$) denotes the number of T_j, T_k consecutive co-occurring in document i. Based on the threshold value we can extract multi-words from the corpus.

- **C/NC- Value:** another alternative method used in multi-word term selection is C/NC values proposed by Frantzi et al. [51]. The method C-value and NC-value combines linguistic and statistical information. C-value, that aims to improve the extraction of nested multi-word terms and NC-value that incorporates context information to the C-value method, aiming to improve multi-word term extraction in general [51, 48].

  C-value is a domain-independent method for multi-word recognition method which aims to improve the extraction of nested terms. After the linguistic (i.e. POS, linguistic filter applied to the tagged corpus to exclude those strings not required for extraction, and stop word removal) activities are over, the C-value statistical measure assigns a termhood to a candidate string and ranking it in the output list of candidate terms [51]. The measure of termhood, called C-value is given as:

$$C - value(a) = \begin{cases} \log_2|a| \,.\, f(a), \ if\ a\ is\ not\ nested; \\ \log_2|a|\left(f(a) - \frac{1}{P(Ta)} \sum_{b \in Ta} f(b) \right), \& \ otherwise \end{cases} \quad \textbf{(2.8)}$$

  Where:

  a is the candidate string,

  F (:) is its frequency of occurrence in the corpus,

  T_a is the set of extracted candidate terms that contain a,

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

Synonym Detection

Concepts, which have similar meaning, have to be merged together. To do so, we may use some external source to identify the similarities of the concept. After the terms are identified and the

synonyms concepts are merged together, we reach a point that the concepts which represent the corpus are identified. Using these identified concepts, we discuss the idea of finding Taxonomic relations between concepts in the next section.

Learning Taxonomic Relations

Concept relation learning is a process of extracting of taxonomy for the ontology, i.e. extracting the subsumption relations between the formed concepts. There are many methods suggested in the literature to construct the subsumption relations between concepts.

Andreia Dal. et al. [52] used Latent Semantic Analysis, Singular Value Decomposition (SVD) and Hierarchical clustering algorithm to find out the relation between concepts. For details, see the reference [52]. Xin li.et al. [36] used parsing and rules of parsing to drown relation between concepts in their work in titled "Automated Chinese domain ontology construction from text corpus". In their approach they define some rules that match a sentence according to the pattern. Then if the sentence satisfies one of these rules, it should be kept. Rules defined to learn the relation, rules and concepts should match the structure of sentences. When concepts and rules match with the sentence of the documents, the relation between concepts from this sentence is extracted.

2.4 Query Expansion

Query expansion is an intermediary step between indexing and matching process in IR systems. Mostly, users of the IR system are more responsible in translating their information need into the appropriate language before they give it to the search engine. But due to Lack of detail knowledge about the information source and the document retrieval process, users formulate a week or wrong query for their information need [16, 21]. Amanda Spink et al. [21] reported that almost 60% of query given by user for searching information are less than three words (that leads to week query). Because of this week or wrong query, most of the time users don't get satisfactory retrieved results.

When users formulate the query, according to [23], there will be two fundamental problems in their query. The first one is word mismatch, meaning the word used in the user query might have different concept or meaning than in the one in the document. Another one is user submits a

short query, which are not detailed enough to generate a more relevant document set. To assist users in resolving the above problems most IR systems use query formulation (also called query rewriting or query transformation) [24]. Various approaches have been proposed for query rewriting. Among them, Query Expansion (QE) is one of the most effective and widely used methods that can effectively minimize the problem by adding additional terms which have related meaning with the original query [23]. In literature, there are many ways of doing query expansion. In the following sections we present some of them.

Different scholars classify query expansion method in different methodologies. According to [23], query expansion methods are roughly classified into three methods, namely:

- Interactive QE,
- Semantic dictionary QE and
- Document set based QE

Interactive QE: In this method the user has an involvement in choosing the right additional term to expand the query. These additional term selection is assisted by the system in such a way that the system suggests the user by showing a list of terms to choose while the user inserts his/her own query. Through this Human-Machine interaction the right terms are selected to expand the query.

The system can get good result, but it depends on the professional domain knowledge of the user. Having such a professional knowledge is out of normal users' capacity [23].

Semantic dictionary QE: In this method, the IR systems use semantic dictionary such as Wordnet and thesaurus. These additional terms, which will be used for the expansion, are fetched from those external sources. This method does not exhibit the expected improvement in the IR result. This is because such sources are too general; they introduce some noise words in the process and affect the retrieval result [23].

QE method based on documents set: In this method the query expansion approach gets the expansion terms from the document set itself. It may use the whole or top-n document to identify the most appropriate terms for the expansion [23]. Relevance feedback, local analysis and global analysis query expansion methods are examples of this category [3, 18]

Another scholars Wollersheim et al. [25] classified query expansion method into two broad methods, namely: corpus based query expansion and relationship based query expansion. Figure 2.5 shows corpus and relationship based query reformulation methods.

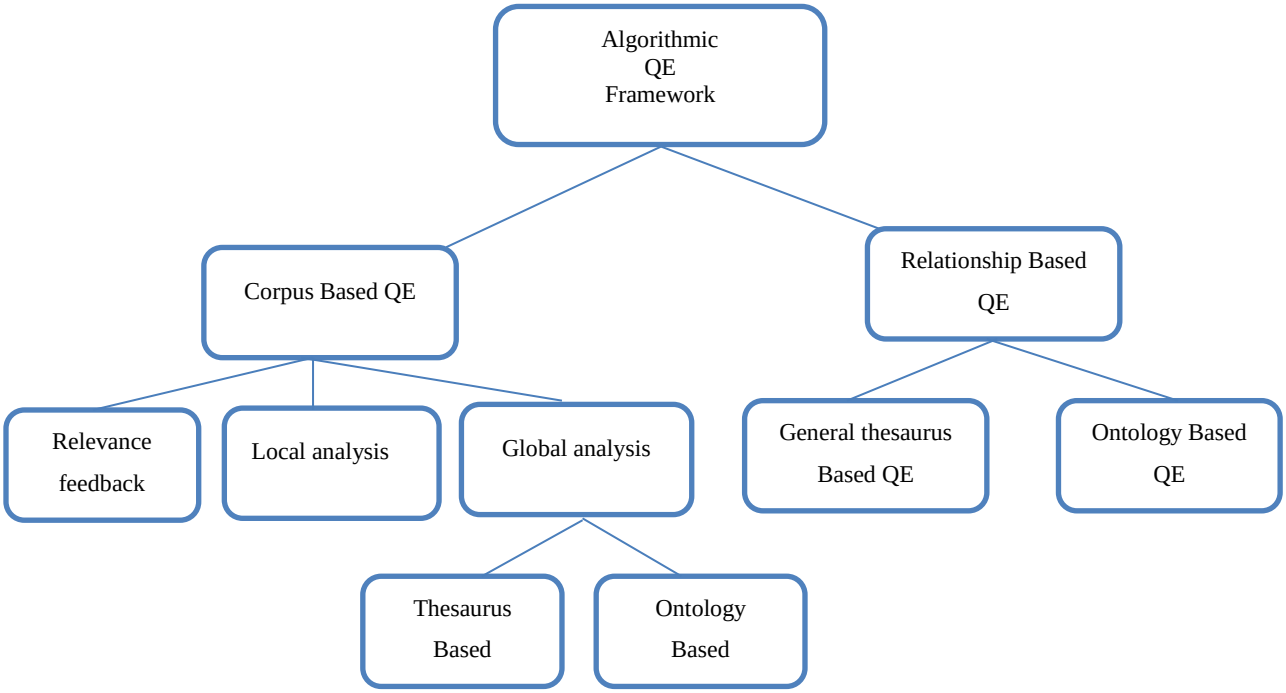

Figure 2.5: Types of Query Expansion approaches [25].

In the subsequent discussion section we adapted and follow the classification as shown in figure 2.5 to discuss the various types of query expansion in details.

2.4.1 Corpus based query expansion

Corpus based query expansion methods use the document set to identify the expansion terms that will be used in the expansion process. This method includes local analysis and global analysis methods [26].

2.4.1.1 Local Analysis

In local analysis query expansion, initial search results of a given user query are analyzed and used to identify the expanding term to the subsequent search operation [3, 27, 29, 31]. It uses top n ranked documents from the initial query result as a source of information to build the new

query. The user or the system itself judges the relevance or non-relevance of the retrieved documents from the first searching process. It is called local analysis because of the fact that it limits its domain to the top ranked documents. Relevance feedback [26, 28, 29, 30] and Local context analysis methods are grouped under local analysis.

2.4.1.2 Global Analysis

In local analysis, from the first search attempt top-ranked documents are considered relevant and used for selecting the right terms of the query expansion. Another alternative approach to expand user query is automatic global analysis. Global analysis uses the entire corpus to extracts expansion terms. Global analysis usually automatically builds a thesaurus [32] or ontology to assist the process of reformulating of users' queries [23]. A thesaurus can be established by analyzing relationships among documents and statistics of term co-occurrences in the documents. On the other hand ontology can be established by identifying the concepts and relationship between concepts in the domain [25].

2.4.1.3 Relevance Feedback

Most of the time the original query formulated by users is not well defined and don't generate the expected result. Since, users have little knowledge about the collection make-up and of the retrieved environment; they find it difficult to formulate queries. This problem suggests the idea of using relevance feedback. The idea here is the first searching attempt should be conducted with an assumption of very few results. An initial query formulation by users should be designed to retrieve a few useful items from the given collection. These initially retrieved items will then be examined for relevance and new improved query formulation can be constructed in the hope of additional useful items retrieved during subsequent search operation [27].

Relevance Feedback query expansion approach is one of the most accepted and successful way in query expansion [17, 19]. It uses feedback information in the retrieval process so as to improve the final result set. The users (in case of user relevance feedback method) or the system (in case of pseudo relevance feedback method) first judge the documents as relevant or not for the information need and give it to the system and then the system use the given feedback information as input for the re-retrieval processes [17]. Based on the feedback information, the IR system retrieval process, retrieve more relevant document. Relevance feedback technique can

be implemented either interactively (i.e. User relevance feedback method) or automatically (i.e. Pseudo relevance method).

User Relevance Feedback

User relevance feedback query expansion method is done by taking the initial set of documents from the first user query result and with additional information from users regarding the relevance of the retrieved document. Users participate in the process by giving their judgment on the set of retrieved documents. They put a remark either relevant or non-relevant document from the list presented by the initial query result [17]. This user input helps to reweight the query term and to reformulate the old query [3]. According to [17], the user relevance feedback system implementation is summarized as:

- The user issues a (short, simple) query.
- The system returns an initial set of retrieval results.
- The user marks some returned documents as relevant.
- The system computes a better representation of the information need based on the user feedback.
- The system displays a revised set of retrieval results.

User relevance feedback query expansion can be applied in different information retrieval models [3]. In the following section we discuss some of them.

User relevance feedback in Vector space model

The user *relevance feedback* in the vector model works with the assumption that the term weight vectors of the documents identified as relevant (to a given query) are similar among themselves (i.e., relevant documents resemble each other) [3]. Further, it is assumed that non-relevant documents have term-weight vectors which are dissimilar from the ones for the relevant documents. The basic idea is to reformulate the query such that it gets closer to the term-weight vector space of the relevant documents [3].

In order to calculate the degree of similarity between a document dj and a query q, different classic techniques are studied in [16] literature.

$$sim\,(dj, q) = \frac{dj.q}{|dj|x|q|} \quad \text{.....(2.1)}$$

And the optimal query vector for differentiating the relevant documents from the non-relevant documents are given by the formula:

$$\vec{q}_{otp} = \frac{1}{|C_r|}\sum_{\forall \vec{d}_j \in C_r} \vec{d}_j - \frac{1}{N-|C_r|}\sum_{\forall \vec{d}_j \notin C_r} \vec{d}_j \text{..... (2.2)}$$

Where,

$\vec{q}_{otp}$: the optimal query vector

C_r : set of relevant documents among all documents in the collection

$|C_r|$: number of documents in the set C_r.

However, the problem with the above formula is that the relevant documents in the collections (C_r) are not known beforehand [1]. To avoid this problem, formulating an initial query and incrementally changing the initial query vector method is suggested. This incremental change is allowed by restricting the calculation to the documents that are known to be relevant based on the user's judgment [16].

Rocchio [28] conducted successful experiments in query modification that combined term reweighting and query expansion based on the vector space model. The algorithm proposes using the modified query $\vec{q}_m$:

$$\vec{q}_m = \propto \vec{q}_o + \beta \frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|}\sum_{\vec{d}_j \notin D_{nr}} \vec{d}_j \text{..... (2.3)}$$

Where:

$q0$: is the original query vector

D_r and D_{nr} are the set of known relevant and non-relevant documents respectively, and *a*, *b*, and *g* are weights attached to each term. These control the balance between trusting the judged document set versus the query: if we have a lot of judges documents, we would like a higher *b* and *g*. Starting from $q0$, the new query moves you some distance toward the centroid of the relevant documents and some distance away from the centroid of the non-relevant documents. This new query can be used for retrieval in the standard vector space model.

We can easily leave the positive quadrant of the vector space by subtracting off a non-relevant document's vector. In the Rocchio algorithm, negative term weights are ignored. That is, the term weight is set to 0. Figure 2.6 shows the effect of applying relevance feedback.

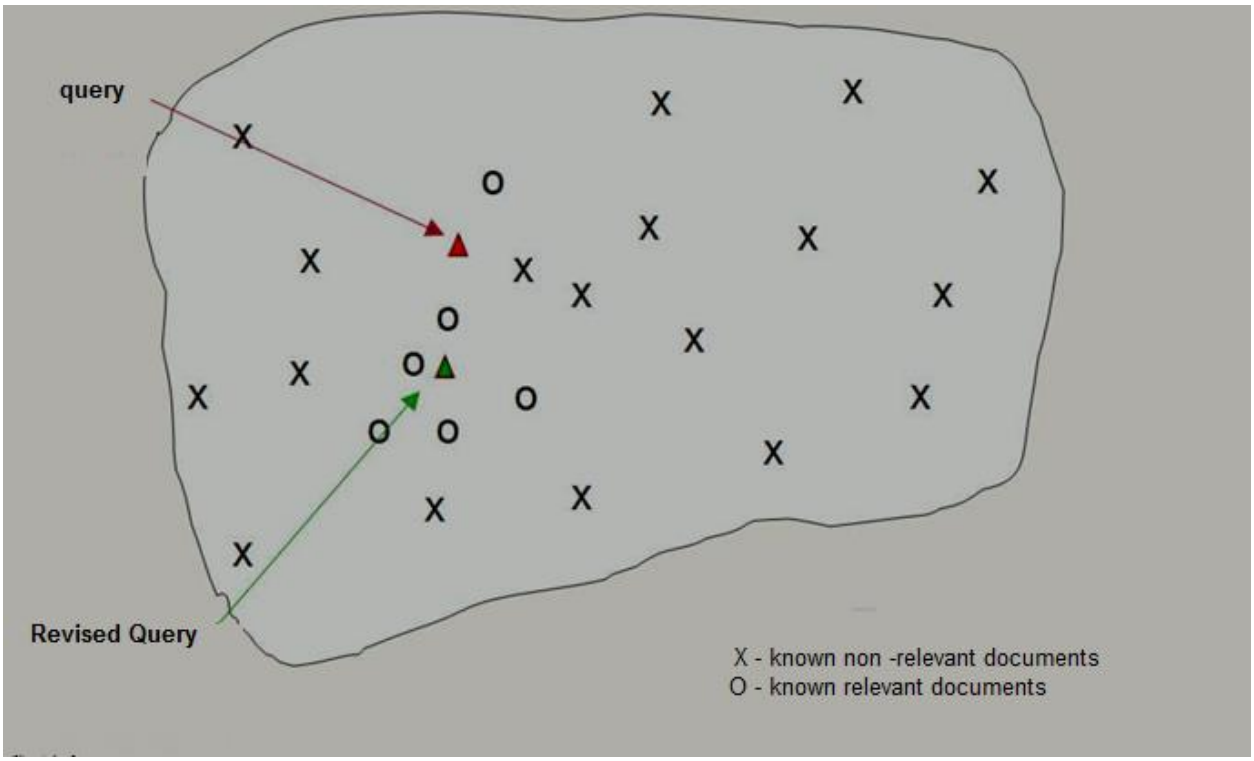

Figure 2.6: An application of Rocchio's algorithm. Some documents have been labeled as relevant and non-relevant and the initial query vector is moved in response to this feedback [3].

Pseudo Relevance Feedback

In pseudo relevance feedback technique the system automatically chooses the top n documents retrieved from the initial query result and assumes that those documents are relevant to reformulate the query [17, 30]. Related terms to the original query are selected and used to expand the query from those small top–ranked relevant documents set [23]. There is no user involvement in this technique rather the system judge the n top-ranked documents are good enough to identify terms for the expansion. Some literatures call it "Blind relevance feedback" and grouped this method under automatic local analysis query expansion approaches.

Relevance feedback approach exhibits successful improvement in IR system by reformulating the original query using additional terms. However, Manning C.C et al. [17] discuss that relevance feedback cannot be effective in some problem areas. These are [17]:

- Misspellings. If the user spells a term in a different way to the way it is spelled in any document in the collection, then relevance feedback is unlikely to be effective.
- Cross-language IR. Documents in another language are not nearby in a vector space that is based on term distribution. Rather, documents in the same language cluster more closely together.
- Mismatch of searcher's vocabulary versus collection vocabulary. If the user uses different vocabulary to that of document representation, then the query will fail, and relevance feedback is again most likely ineffective.

2.4.2 Relationship based query expansion

Relationship based query expansion method, unlike corpus based query expansion method, uses the external relationship from outside sources to choose the expansion terms [25]. General purpose thesaurus or Wordnet [38, 6] and corpus-independent ontology [22] are examples of external sources which can be used to select expanding term.

2.4.2.1 Query expansion based on thesaurus

Query can be expanded by terms which have related meaning to the initial query. The related terms can be selected from the thesaurus. A thesaurus is a precompiled list of important words in a given domain of knowledge and for each word in the list, set of related words. Related words are, in its most common variation, derived from a synonymity relationship [32].

The thesaurus can be constructed in global or local approach [33]. In global thesaurus [34] classes are constructed based on word co-occurrence and their relationship in the corpus as a whole and these classes are used to index both documents and queries, whereas the local thesaurus [28] uses information obtained from the top rank documents retrieved in response to a particular query. Thus a global thesaurus is constructed prior to the indexing process, whereas a local thesaurus is constructed dynamically during query processing and uses the information retrieved in response to a specific query to modify only that query. The global analysis

techniques are relatively robust, but, consumes a considerable amount of computing resources to perform corpus-wide statistical analysis. Moreover, since it focuses only on the document side and does not take into account the query side, global analysis only provides a partial solution to the word mismatch problem [33].

A thesaurus can be classified into four groups based on their construction approaches and usage. These are: General purpose Hand crafted thesaurus (Manual thesaurus), Co-occurrence based automatically constructed thesaurus, and Similarity based automatically constructed thesaurus, and Head Modifier based automatically constructed thesaurus [32].

Construction of manual thesaurus is very hard task because it is labor intensive work and it lacks domain specific terms as it generally cover general terms.

In Co-occurrence based automatically constructed thesaurus, similarity between terms are first calculated based on association hypothesis and then used to classify terms by selecting a similarity threshold value. In this way the set of index term is divided into classes of similar term. The query is then expanded by adding the terms of classes that contain query term. Such strategies are based on local clustering of terms [32].

A similarity based automatically constructed thesaurus is built considering the term to term relationship rather than simple co-occurrence data. Terms for expansion are selected based on similarity to the whole query rather than their similarity to individual term.

In Head –modifier based automatically constructed thesaurus term relations are gathered on the basis of linguistic relations [32].

2.4.2.2 Query Expansion based on ontology

Corpus dependent Ontology is used to extract the expansion term in the query expansion process [35]. Corpus dependent ontology can be constructed from the corpus itself either manually, automatically [36] or semi-automatically [37]. To construct an ontology all important concepts found in the corpus and their relationships are first identified. And using those identified concepts and relationships the ontology is constructed.

Once we construct the ontology from the given corpus the user query is expanded by using concepts from the ontology. During query expansion the term or concepts in the user query are searched in the domain ontology. If the concept is found in the ontology class, its subclass and

instance concepts are used to expand the query. In this thesis attempt is made to construct Amharic domain specific ontology for use on expansion of the users query.

2.5 Performance Evaluation

The main goal of this research is to improve the efficiency and effectiveness of Amharic IR system retrieval by integrating ontology based query expansion. When the user inserts the searching query to the IR system, logically the total documents in a corpus are divided into two categories (see figure 2.7). The first category consists of documents which are relevant to the user. The second category consists of documents which are non-relevant to the user information need. From these two groups, the relevant items only give valuable information to a user to his information need. The non-relevant groups of items do not give any directly useful information to the user.

Putting this in mind, an IR system becomes effective when it retrieves all the relevant documents to the user while avoiding junk (non-retrieval) documents from the output result [3, 53]

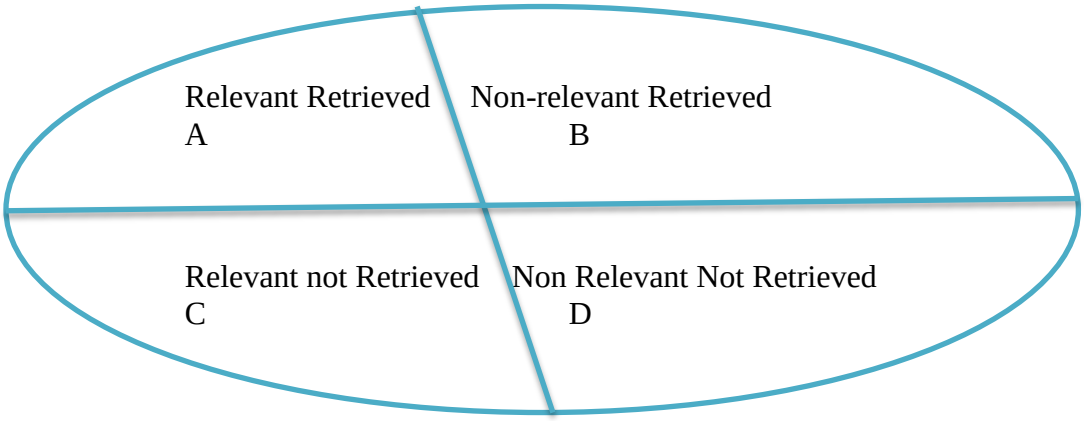

Figure 2.7: The logical category of the documents in response to user query [3].

In order to measure the effectiveness of the system, we have used a commonly used IR performance tool called Recall and Precision, which measure the ability of the system to retrieve relevant documents.

2.5.1 Recall and Precision

Recall: Recall is the fraction of relevant documents that are retrieved

Recall(R) = (number of relevant items retrieved)/ (total number relevant items)

$$R = \frac{A}{(AUC)} \text{ ..... (2.4)}$$

Precision: Precision is the fraction of retrieved documents that are relevant

Precision (P) = (number of relevant items retrieved)/ (total number retrieved items)

$$P = \frac{A}{(AUB)} \text{ ..... (2.5)}$$

Recall vs. Precision

There is a tradeoff between recall and precision. That is Recall and precision are inversely related. The following recall-precision graph shows the inverse property of these two measuring tools.

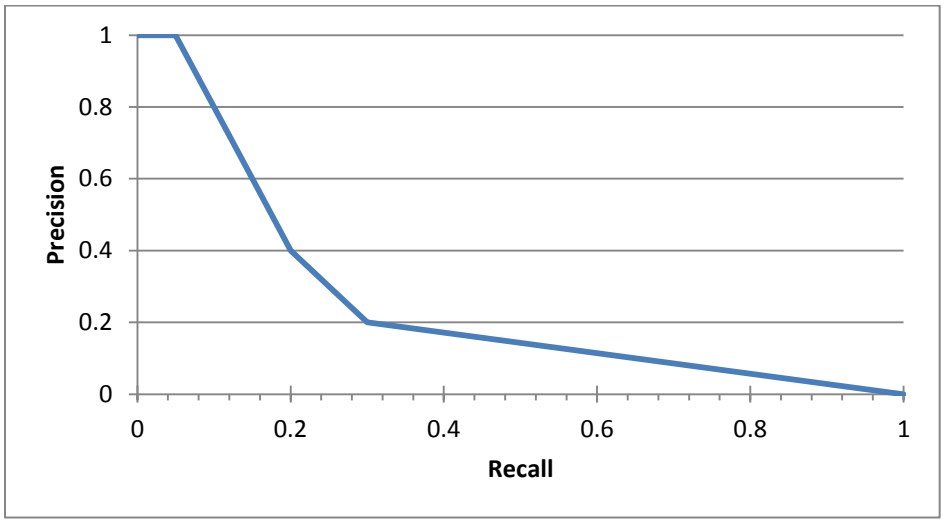

Figure 2.8: Recall-precision graph

Recall and precision are often conflicting goals; the increase in recall greatly affects precision and vise-versa. Therefore, to tackle this trade off another effectiveness evaluation metric, F-measure, is forwarded .

F-measure

The weighted harmonic mean of precision and recall, the traditional F-measure or balanced F-score is:

$$F = \frac{2.precision\ .recall}{(precision + recall)}$$ [53]…….. (2.6)

2.6 Amharic Language

Amharic language belongs to the Semitic family of language such as Arabic, Hebrew, and Syrian. It is the second most spoken Semitic families in the world next to Arabic. It is a national and official working language of Ethiopia. It is also the working language of several of the regional states within the federal system of Ethiopia. It has been the working language of government and non-governmental organizations, as well as most private institutions in Ethiopia. Amharic is spoken by roughly 30% of the population as a first language, and an additional 20% as a second language, totaling about half of the population. Outside Ethiopia, Amharic is the language of some 2.7 million people living notably in Eritrea, Egypt, Israel and Sweden. In general, there are more than 34 million speakers of Amharic [54]

2.6.1 Amharic Script

According to Thomas [55], there is little information about the precise timing and location of the emerging Amharic writing system. However, it is believed that it has its origins in the same ancestral writing system as those European and Middle East alphabets. Amharic language has its own script called "Fidel" – "**ፊደል**". The script consists of 33 core characters which are used in its writing system. Unlike Arabic, Hebrew or Syrian, Amharic is written from left to right [56]. These thirty three basic characters arranged in seven orders having consonant and vowels in each combination to represent the language syllable. There are also other characters which are derived from the core characters, numbers, punctuation marks, and labialization in the script of Amharic. Having this in mind the total number of characters in the script reaches 349. Table 2.1 shows a summary of the number of characters in each category [54, 49].

Type of Amharic Character	Total Characters
Core Characters	231
Labialized Characters	89
Punctuation marks	9
Numbers	20
Total	**349**

Table 2.1: Total number of characters in Amharic alphabet - **ፊደል** (Fidel)

For the above script characters, there are a number of computer fonts available for text document production. Among them: 'Power Geez', 'Visual Geez' and 'Nyala" are examples of Amharic fonts.

2.6.2 Amharic word Morphology

Amharic is one of a morphologically rich language with the ability of more productive, derivative and high word formation. This characteristic of the language lead to various difficulties for document matching and retrieval process [54, 57].

Like other Semitic languages, Amharic exhibits a root-pattern morphological phenomenon. A root is a set of consonants (also called radicals) which has a basic lexical meaning. A pattern consists of a set of vowels which are inserted among the consonants of a root to form a stem. In addition to this non-concatenative morphological feature, Amharic uses different affixes to create inflectional and derivational word forms [57].

Some adverbs can be derived from adjectives. Nouns are derived from other basic nouns, adjectives, stems, roots, and the infinitive form of a verb by affixation and intercalation. Case, number, definiteness, and gender marker affixes inflect nouns. Adjectives are derived from nouns, stems or verbal roots by adding a prefix or a suffix [57].

Moreover, adjectives can also be formed through compounding. Like nouns, adjectives are inflected for gender, number, and case. Amharic verbs are derived from the roots. The conversion of a root to a basic verb stem requires both intercalation and affixation. Other verb forms are also derived from roots in a similar fashion. Verbs are inflected for person, gender, number, aspect, tense and mood. Other elements like negative markers also inflect verbs in Amharic [57].

2.6.3 Challenges of Amharic Language, Script and Documents

The nature of Amharic language exhibits many challenges in the text processing activity and retrieval. Milion [54] identified and reports those challenges in his work and Biniam [56] summarizes those challenges as follows.

a) The existence of the large number of Amharic vocabulary in the writing system is a great challenge in the development of retrieval system for the language.

b) The Amharic writing system uses multitudes of ways to denote compound words and there is no agreed upon spelling standard for compounds. These words can be written as two separate words or as a single word using ‘hulet netib’ (:) in between. For instance, the word “library” can be written as “ቤተመጻሕፍት” or “ቤተ፡መጻሕፍት” or “ቤተ መጻሕፍት”.

c) The rule to Amharic language users generally follows in their writing system is that if the alphabet in a word sounds right when read aloud, then it is written. There are different ways of writing a single word due to various reasons such as regional dialects and various ways of writing loan words. Regional dialects have their own impact in word formation in the basic level where the words are more likely to be written by following their spoken form. For example, “ስራ” vs. “ሥራ”, “ዓፄ” vs. “ዓጤ”, “እነዚህ” vs. “እነኚህ”, etc. The absence of restricted rules leads to a number of problems to develop efficient Amharic information retrieval system.

d) Amharic has a very rich morphology, which forwards a challenge by itself especially during stemming.

e) In Amharic orthography, there are additional letters taken from Ge’ez that would take on the phonemic value of its nearest neighbor. The result being two syllabic series for ‘se’ (‘ሰ’ and ‘ሠ’), two series for ‘tse’ (‘ጸ’ and ‘ፀ’), two for ‘a’ (‘አ’ and ‘ዐ’) and four for ‘ha’ (‘ሀ’, ‘ሐ’, ‘ኀ’ and ‘ኸ’). These redundancies in Amharic become a source of confusion and the letters are treated as interchangeable by a person. Common Amharic spelling, then becomes highly flexible and correctness is not a matter of precision but one of acceptable proximity. For example, the word “sun” can be written as “ፀሀይ”, “ፀሃይ”, “ፀሐይ”, “ፀኀይ”, “ጸሀይ”, “ጸሃይ”, “ጸሐይ”, “ጸኀይ”, etc. all mean the same, although they are written differently.

f) It is common to write some words in shorter form by using ‘/’ (forward slash) or ‘.’ (dot). The short form of words which are linked by either of these symbols can be expanded as a single or a combination of words. For instance, “አ.አ.” is expanded as a combination of two words አዲስ አበባ (means “Addis Ababa”). “መ/ር” is a short form of the single word መምህር (means teacher) or another word መዝሙር (means song).

2.7 Related Works

Query expansion is an old research area in the field of information retrieval. Many researches have been done to explore approaches and mechanisms that can improve the query expansion process that has a direct effect on the performance of the IR system. In section 2.4 we have seen different methods, tools and techniques that can be integrated in query expansion. Current research direction in IR is the use of ontology [43]. In this section we reviewed few query expansion, specifically ontology based query expansion, researches done locally and globally.

2.7.1 Global Works

Fang et al. [23] proposed a query expansion method based on ontologies and occurrence of frequency. The proposed model has two steps to expand the user query. First, it identifies the terms that can be used to expand the original user query from the existing ontology. To identify the term first the user query will be checked if it is a concept in the ontology. If it is, its subclasses, equivalent classes and sub instances are listed as expansion words. If it is not a concept in the ontology, it will be checked if it is an instance under a concept in the ontology. If it is, its equivalent instances are taken as expansion words. After the term identification stage completed, the second stage filter out unwanted terms based on their frequency level.

In order to test their model they collected 82 articles from the internet. They carried out the experiment in three scenarios: Full text IR, IR with simple query expansion model and IR with the model they proposed. The result of the experiment showed that their model increases the recall without decrease the precision.

FU et al. [31] focused on a specific application area of ontology based query expansion, namely spatial queries. They showed how, domain and geographical ontologies can serve to expand queries that comprise fuzzy spatial expressions. The scholar's expansion mechanism was based on the interpretation of geographic expressions such as *near*. The benefits of such spatial query expansion mechanisms were illustrated by the example of four spatial queries which were issued to the SPIRIT best-match retrieval system for which the spatial expansion induced increases in the precision.

Sack [42] also showed how a domain-specific ontology can augment a traditional information retrieval system. He supported the search in a bibliographic database with a Semantic Web

standards based ontology for the domain of NP-complete problems. This ontology was used in the query formulation stage for both disambiguation and query expansion purposes. In an interactive expansion mode, semantically related expansion term suggestions like synonyms, broader and narrower terms were presented to the users. The author points out the advantages of the ontology augmented search by means of two usage scenarios. These exemplarily illustrated the benefits of providing users with ontology based context knowledge from the domain ontology.

2.7.2 Local works

Locally, there is a single research found that used external ontology (i.e. wordnet) to select expansion terms [6] and other two local researches by Alemayehu [4] and Abye [5] which used Relevance feedback and Co-occurrence analysis methods respectively.

Alemayehu [4] used a relevance feedback method in his model so as to improve the power of the query in representing the user information need. His research, improved recall from 29% to 73%, but reduce the precision from 91% to 57%.

The other local query expansion research has been done by Abey [5]. He attempted a semantic based query expansion to minimize the problem of a polysemous word. In his model he tested three different methods for selecting the terms which are relevant to the expansion.

The statistical co-occurrence technique, which selects expansion terms from the pseudo relevance feedback by analyzing terms co-occurrence information with query terms. The co-occurrence of the terms only analyzed from the top n relevant documents returned from the first searching attempt. The experiment result with this method exhibited that it increases the precision by 14% with a decrease in recall by 10%.

The bi-gram technique which also uses the pseudo relevance feedback for expansion, selects those terms which appear to the right or left of a query term from the top n relevant document returned from the first search attempt. The experiment result with this method exhibited that it increases the precision by 18% with decrease in recall by 16%.

Finally, he used the bi-gram thesaurus technique. Unlike the other two this method uses a pre-built thesaurus to select expansion terms. This method used the whole corpus when the bi-gram

thesaurus is constructed unlike the other method. The experiment result with this method showed that it increases the precision by 18% and decreasing of a recall by 16%.

Among the three methods tested the statistical co-occurrence analysis is the better one. However, because some expansion terms selected to expand the original query become polysemous themselves and affect the performance of the expansion model and the efficiency of the Amharic IR system.

The third local research in query expansion is done by Iman [6]. She followed the recommendation of Abey [5] that is query expansion using ontology. External information source wordnet is used to expand the query. And she used the word sense disambiguation technique to overcome the polysemous problem experienced in Abye [5] research.

Wordnet is used to find out additional word for the expansion process. Once the additional terms are identified from the wordnet, before the terms are used to reformulate the original query, their sense is checked using word sense disambiguation method. Using this technique in her proposed model, terms which have a correct sense are used to expand the original query.

The experiment result showed that the system overall performance improved by 31% and promisingly avoid polysemous word problems due to the ability of the system in identifying the correct sense using the word sense disambiguation technique.

Using wordnet as a source of expansion terms may improve the result. However, wordnet is too broad; it may include useless terms for the given query and can bring noises in the retrieved result [23]. To avoid this problem we use corpus dependent domain specific ontology to expand the original query. This work attempted to improve the efficiency of the Amharic IR system by augmenting the user query with terms that are mined from ontology. Domain specific and corpus dependent ontology, that is constructed manually, is used in this research.

Chapter Three
DESIGN OF AN AMHARIC QUERY EPANSION MODEL

Information retrieval (IR) system is designed to provide documents that can satisfy an information need of a user, from the available huge collection of documents. The selection of documents from the large document source is performed on the bases of a similarity measure between a document and a query. Documents with high similarity are presented to the user as the result of retrieving. With this in mind most IR systems implement different algorithms, techniques and tools to achieve the above desired goal. For example, our research integrated an ontology based query expansion module to improve the efficiency of the Amharic information retrieval system designed by Amanuel [9].

3.1 Amharic IR system

Designing an Information Retrieval (IR) system passes through a series of steps. The standard architectural view of the IR system depicted in Figure 3.1 shows that an IR system consists of two major parts, i.e., Indexing and searching.

Indexing (see appendix B.1) is an offline process that is dedicated to organize documents using list of words, which are extracted from the documents themselves. In other words, Indexing is a process of converting the collection of documents into a list that is suitable for easy search and retrieval [54]. It involves document preprocessing, term extraction, and construction of index file.

Document pre-processing has to takes place prior to the main indexing of the collection of the documents. Document pre-processing consists of 3 stages: Lexical analysis, stop-word elimination and stemming. Lexical analysis is performed to convert stream of characters into a stream of words by eliminating hyphens and punctuation marks. Next, stop-words are eliminated from the word list. This is because of the fact that, they have a low discrimination power for retrieval purposes and the elimination process helps to reduce the size of the vocabularies in the index file. After stop word removal, the remaining words are stemmed with the objective of removing affixes and allowing retrieval of documents containing syntactic variations of query

terms. Next to document pre-processing, term extraction follows to find out terms that have a potential to represent a given document. To extract terms, we use information retrieval measure TF/IDF (Term frequency/ inverse document frequency) method discussed earlier in chapter 2.

Lastly, relevant terms are indexed and stored in the index file. The indexed information is stored in two separate files. The first is vocabulary file which contain the list of terms and their frequency. The other file is posting file which contains the position of each selected term in the documents and pointers to the document links.

The second part of the IR system is an online process that is searching of relevant documents (see Appendix B.2). It consists of query processing, matching, and document ranking. Query processing is the same kinds of operation with that of document preprocessing on the given query. It consists of lexical analysis, stemming and stop word removal activities explained above. Next to query processing matching or similarity measure is performed between preprocessed query and the indexed terms. If a similarity exists between documents and the query, then those documents are identified as relevant document for the information need of the user. Finally, the identified documents are ranked based on similarity value and displayed to users. The above process is not enough to retrieve the optimal output result. To improve the searching result different methods have been introduced and integrated, the proposed system incorporates query expansion by using ontology as a support knowledge source.

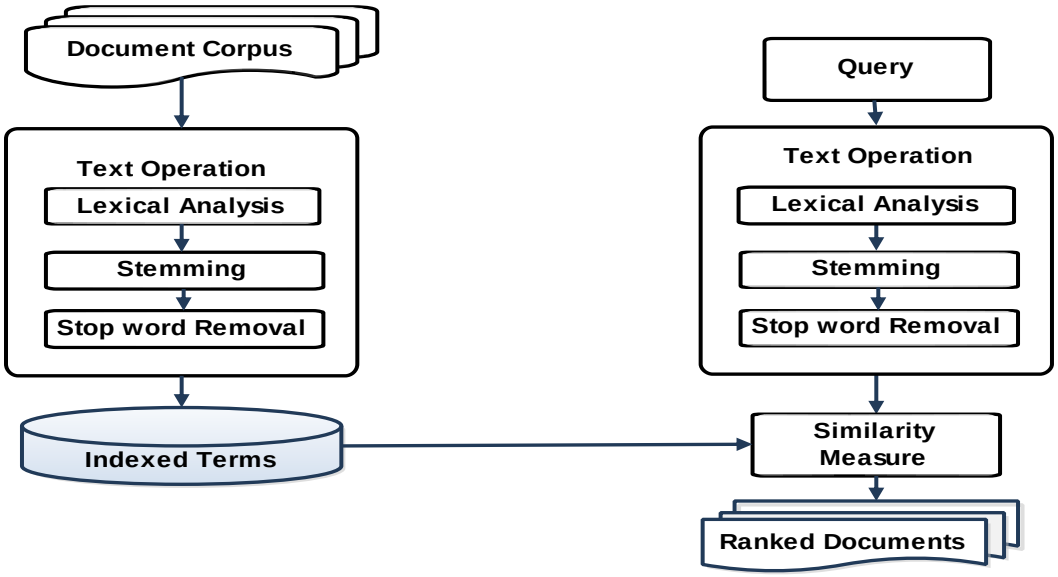

Figure 3.1 Architectural view of the original IR system [5]

3.2 Probabilistic Retrieval Model

The Amharic IR system designed by Amanuel [9] based on the probabilistic retrieval model is used to test the ontology based query expansion model in this research. Probabilistic model is based on the assumption that the terms are distributed differently in relevant and non-relevant documents. Matching documents are ranked according to their relevance to a given search query using a "best match" ranking function. The original IR system works as depicted in figure 3.1 above. However, to make the old model suitable for this research work some components are changed and some areas drastically improved. For example Alemayehu probabilistic retrieval model uses binary weighting. Binary weighting isn't appropriate to rank and display the output result with their importance order. To display the output result in ranked order, this research implemented probability retrieval model with non-binary weighting technique. And also, the relevance feedback module in the previous model is changed to automatic query expansion method so as to improve the performance of the Amharic IR system and to reduce the user load in choosing relevant document found in a feedback approach. In addition to this the model is changed from console application mode to user interface mode (see Figure 3.2). The architecture of the proposed system is discussed in the next section.

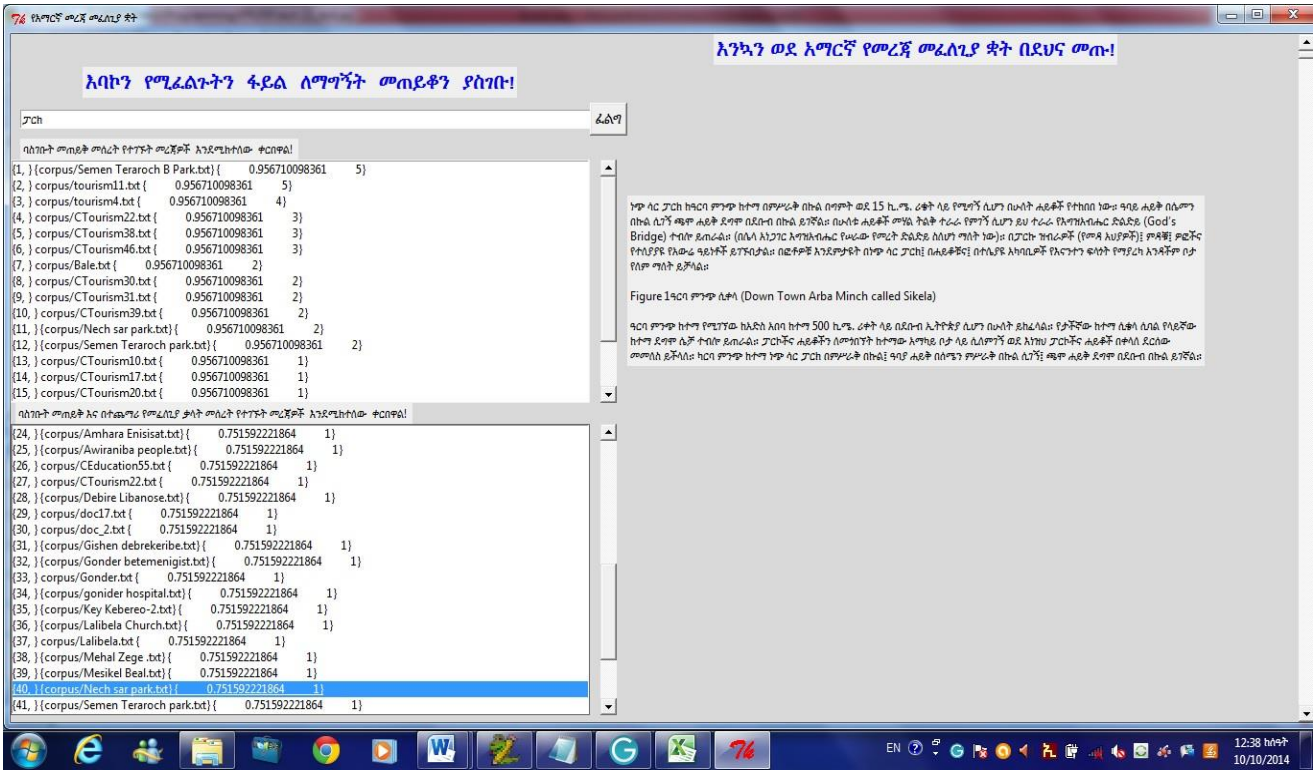

Figure 3.2 User interface for searching

3.3 System Architecture

The architectural model of the proposed system is depicted in figure 3.3. It is a modified version of the orginal IR system model. The modified part is clearly shown in the figure.

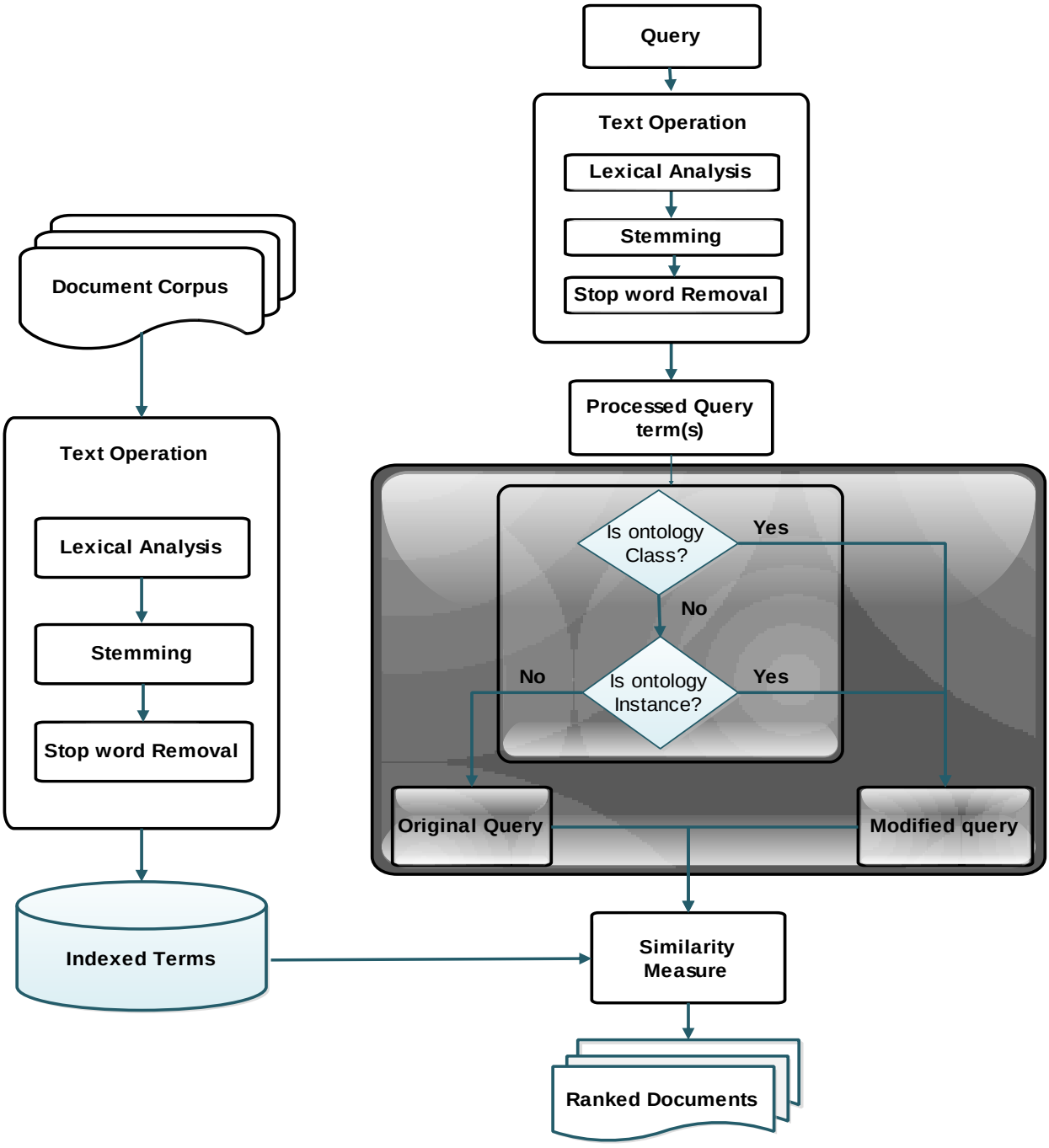

Figure 3.3: System architecture of the proposed ontology based query expansion IR system. The shaded area represents the focus of this research.

3.4 Flow Chart

The flow chart of the proposed system is depicted in figure 3.4. The figure illustrates how the query expansion process is carried on. When the user enters the query representation of his/her information need into the system, the system sends (1) the query to the query operation module. The query operation module performs text operation on the query and identifies the candidate terms and passed those recognized candidate query terms to the query expansion module (2). In query expansion module, firstly, a query term will be checked if it is a concept in the ontology (see figure 3.3). If it is, its subclasses, equivalent classes, and all sub-instances will be listed as expansion words (Expand 1 in figure 3.3). If the term is not a concept in the ontology, it will be checked if it is an instance under the classes in the ontology. If it is, its neighbor instances will be listed as expansion words (Expand 2 in figure 3.3). If the expansion module gets expansion words from the ontology, it integrates the original query terms with expansion terms found from the ontology and sends the new integrated terms to the IR system (3) (see figure 3.1 the original IR system model). If it doesn't get the term in the ontology, it sends the original query to the IR system (4). The IR system generates output (5) based on refined query (3) or original query (4).

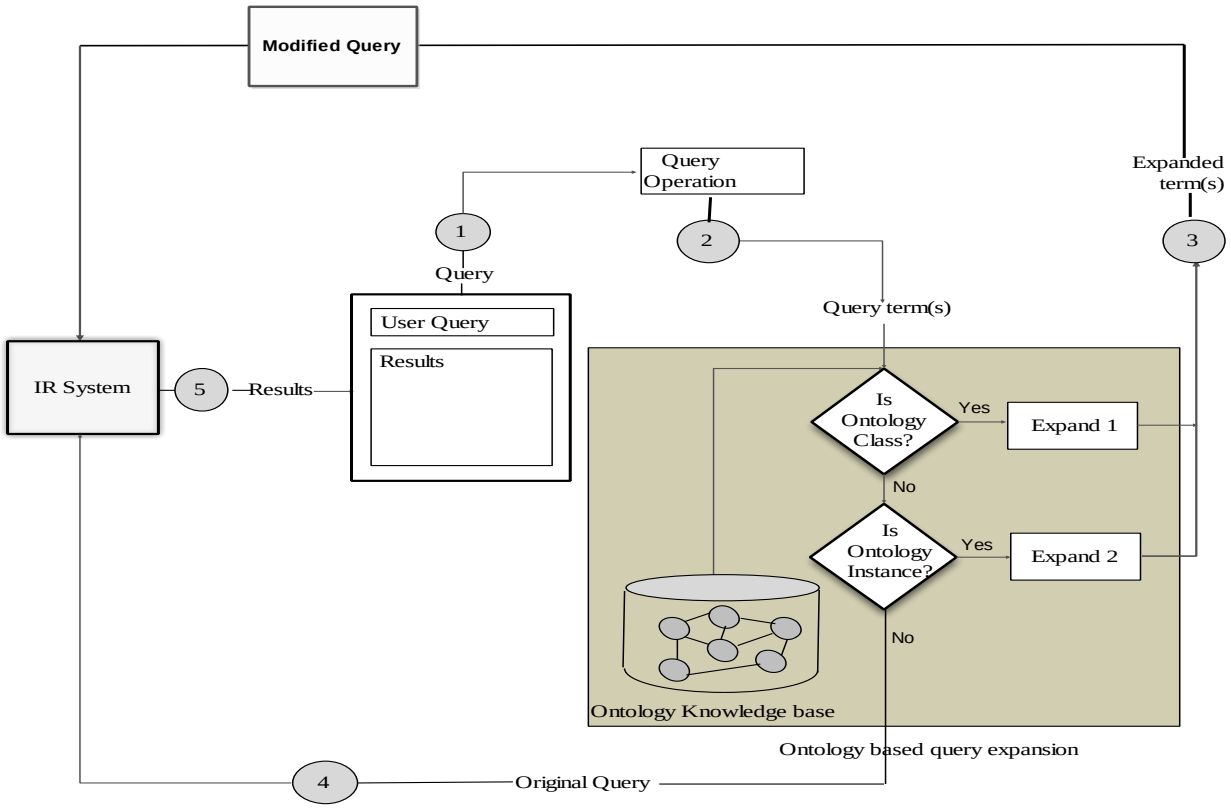

Figure 3.4: Flow chart of the proposed ontology based query expansion IR system.

3.5 Amharic Ontology Development

Initially we had a plan to use an existing Amharic ontology in our experiment. However, we couldn't find any usable Amharic ontology and we changed our plan to construct Amharic domain specific ontology automatically or semi-automatically from our testing corpus (i.e. Tourism corpus, this is because a suitable corpus that we have for domain specific ontology is tourism sector) by performing semantic analysis on the corpus. To fulfill this plan (i.e. Automatic or semi-automatic construction of ontology) we have implemented the following series steps.

3.5.1 Text preprocessing

As we have seen in section 2.3, the first step in ontology construction is text preprocessing. It includes sentence segmentation, tokenization, part of speech tagging, stop word removal, and stemming activities.

Sentence segmentation is carried on by Amharic punctuation rule. All Amharic sentences are ended by double colon symbol (i.e. ::). Using this symbol we segmented the document into lines of character. After sentence segmentation is completed, we tokenized the segmented sentence into words using blank space between words as a separator.

Next to sentence and word segmentation, Part of speech tagging (POS) is followed. POS is important to filter out the necessary terms in the domain. We have hardly found an automatic Amharic POS tagger which can tag an Amharic sentence automatically. Because of that we are forced to use manually tagged corpus. This manually tagged corpus is tagged by linguist in the language area. Figure 3.5 below shows sample of manually tagged Amharic sentence.

በቅድመ <NP> ታሪክ <N> መስክ <N> የሚከናወኑ <VREL> የምርምር <NP> ተግባራት <N> ለእድገት <NP> ወሳኝነት <N> ኢንዳላቸው <VP> ተገለጸ <V>

Figure 3.5 Amharic parts of speech tagged sentence.

The other preprocessing task is stop word removal. To remove stop words from our testing corpus, we use Algorithm 3.1 below

Algorithm 3.1: Stop word removal algorithm

```
1:  read all stop words from the file and store in memory
2:      while end of file
3:          read words from the corpus
4:              if the word found in the stop word list then
5:                  discard the word
6:              else
7:                  keep the word
8: Output: list of non-stop words
```

The last activity in text preprocessing is stemming. We have used a steamer designed by Alemayehu [4]. The steamer we used for our project did not give us the result that we expect (for detail see Alemayehus' work) . We tried to modify the stemming algorithm, but we could not improve too much in the resulting output. This is because of the fact that Amharic language is a morphologically rich language [57] and needs a very big project and effort to come up with a better stemming algorithm.

The second step in ontology construction is concepts extraction from the corpus. This step includes linguistic filtering, single and multi-word candidate term extraction, synonym word identification and merging, and taxonomy generation.

Linguistic filtering is a process of identifying terms which satisfy a certain linguistic criteria. The part of speech tagging activity, which is done in the preprocessing step above, is used to setup the criteria or rules which can help to identify those important terms in a document. If a term fulfills a certain criteria or rule, we keep the word otherwise we left out from the list. In short, linguistic filtering is employed to remove words from word classes that are suggested by a linguist to hardly constitute a term. This linguist filtering can be applied in two ways, closed filter and open filter [48]. Closed filter is the one in which only those that firmly satisfies the linguistic criteria are passed, whereas open filter can be considered as a somehow relaxed filtering technique. Berhanu [48] choose an open filter in his research so that open filter prevents only word classes known to hardly constitute a concept, which results in augmented recall over precision: According to Berhanu [48], the word classes suggested by a linguist to be filtered out

from Amharic sentences are 'PREP', 'PUNC', 'PRONP', 'AUX', 'PRON', 'V', 'VP', 'VC','VREL. We adopted Berhanu's rules to filter out words in this research. After linguistic filter is completed, single and multi-word candidate term selection comes as a next step.

3.5.2 Single word selection

To extract a single word, we used information retrieval measure TF/IDF (Term frequency/ inverse document frequency) method discussed in section 2.4. The algorithm for selecting a single word from the corpus is given in Algorithm 3.2. See the python implementation in appendix A.3

Algorithm 3.2: Single word selection algorithm

1: for each d in a documents do
2: for each word in d do
3: stem (word)
4: if not (stopword(word))
5: keep word
6: TF (word)= $\frac{f(t,d)}{max\{(f(w,D):w\in D\}}$ (word)
7: IDF (word) = log $\frac{N}{df(t)}$ (word)
8: TF/IDF (word)=TF*IDF
9:
10: else
11: Drop word
12: end for
13: if (TF/IDF (word) > t then // t=threshold value of TF/IDF value (i.e. 0.25)
14: Keep word
15: end for
16: Output: list of words with their tf-idf value

The execution of the above algorithm gives out the list of words which are sorted out based on their tf-IDF value. A sample of the top 50 single words with their tf-IDF values are summarized in table 3.1 below.

Single word	Tf-IDF value
አቡነ	0.0941599964557
ቅሬተ	0.0917178437681
ድንጋይ	0.0895946632942
አበባ	0.0859978204425
ቁጥር	0.0859756949287
ምክር	0.0857661721868
ትብቅ	0.0846626250167
ግብር	0.0811116629602
ኤድንበርግ	0.0778736409351
ነብር	0.0762365974993
ፖርኩ	0.0754240787936
ደባርቅ	0.0717791820794
አፋር	0.0710726896324
አዲስ	0.0710073629172
ገሪማ	0.0701028105234
ሙዚቃ	0.0693664364632
አማካሪ	0.068788382826
ለማጠናከር	0.0685931537533
ነብር	0.0679391435319
ጉሙዝ	0.0671959974706
የቀይ	0.0671213521462
ቆርኬ	0.0655817470032
ትውልድ	0.0635833094346
ኢትዮጵያ	0.0634163119606
ፖርክ	0.0632536853573
ያቤሎ	0.0632536853573
ሀገር	0.0623760314073
ስላሴ	0.0611452291787
ታደሰ	0.0608358824837
እሳያስ	0.0598159850661
ለማዋል	0.0597522583807
ህዝብ	0.0593765807447
የደንቆሮ	0.0593699576046
የዶርዜ	0.0589614709938
አገሩ	0.0585433045328
ኢትዮጵያ	0.0582929832445
አብያተ	0.0574540945968
በመስኩ	0.0550307062608
ኮሚሽነሩ	0.0531196937415
በግዥ	0.0529141406354
ባህል	0.0528091755968
የመቅደላ	0.0527968551555
ፕሮፌሰር	0.0527968551555
ውጪ	0.05275286248 የ
ሀገረ	0.0525771078925

Table 3.1 Single words extracted from the corpus sorted by their Tf-IDF values

3.5.3 Multiple-word selection

To select multi-word concepts, we use python nltk tool [59] and the C - value method discussed in section 2.4.

Step 1 (Candidate terms extraction)

In this step we generate all possible combinations of word sequences in every sentence using bigram and trigram nltk tools. To minimize complexity of finding multi-word, we restrict our term length (word count) in a candidate term to 3.

E.g. Given Sentence: = {የእንስሳት ልማት እንቅስቃሴ የሚያሳየውን እምርታ}

Candidate terms: = { የእንስሳት ልማት}, {የእንስሳት ልማት እንቅስቃሴ}, {ልማት እንቅስቃሴ} ,

{ልማት እንቅስቃሴ የሚያሳየውን},{እንቅስቃሴ የሚያሳየውን},{እንቅስቃሴ የሚያሳየውን እምርታ}, and {የሚያሳየውን እምርታ}

Step 2: (Computing C-value)

After the candidate terms are identified, we use the techniques detailed in Section 2.4 to calculate the C-value measure of a term. The algorithm for this work is shown in Algorithm 3.3. The algorithm is adopted from Berhanu Mengiste work [48]

Algorithm 3.3: Multi-word selection algorithm

```
1: for each word in a documents do
2:     for each word in d do
3:         stem (word)
4:         if not(stopword(word)
5:             wordlist.append(word)
6:         else
7:             Drop word
8:         end if
9:     end for
10: end for
11: bi_word= nltk.bigrams (wordlist)
12: tri_word=nltk.trigrams (wordlist)
13: for each word in bi_word do
14:     calculate C_value (word)
15: end for
16: for each word in tri_word do
17:     calculate C_value (word)
18: end for
19: Output: list of words with their C-value
```

When the C-value computing program is run it gives us 75 multi-word terms with C-value greater than 0 and the sample selected multi-words are sorted and summarized in table 3.2 below.

Concept	C-Value
የዱር እንስሳት	17.33333
ባህል ቱሪዝምና ማስታወቂያ	4.509775
ተራሮች ብሄራዊ ፓርክ	4.094738
ባህል ማስታወቂያና ቱሪዝም ቢሮ	4
ባህል ቱሪዝምና ማስታወቂያ ቢሮ	4
የዞኑ ንግድ እንዱስትሪና ቱሪዝም መምሪያ	3.965784
ተራሮች ብሄራዊ	3.5
ባህል ማስታወቂያና ቱሪዝም	3.424813
ንግድ እንዱስትሪና ቱሪዝም	3.08985
የዞኑ ንግድ እንዱስትሪና ቱሪዝም	3
ንግድ እንዱስትሪና ቱሪዝም መምሪያ	3
የዱር እንስሳት ከፍተኛ ባለሙያ	3
ብሄራዊ ፓርክ	3
በቢሮው የባህል መምሪያ ሃላፊ	3
የቅርስ ትናትና ትበቃ ባለስልጣን	3
የኢትዮጵያ ቱሪዝም ኮሚሽን	2.924813
ለዋልታ ኢንፎርሜሽን ማእከል	2.924813
የቅርስ ትናትና ትበቃ	2.83985
ቱሪዝም ኮሚሽን	2.666667
መምሪያ ሃላፊ	2.666667
ኢንፎርሜሽን ማእከል	2.666667
የአለም ቱሪዝም ቀን	2.33985
የጽህፈት ቤቱ ሃላፊ	2.33985
ውቅር አብያተ ክርስቲያናት	2.33985

ማስታወቂያና ቱሪዝም ቢሮ	2.33985
ቱሪዝምና ማስታወቂያ ቢሮ	2.33985
የቀይ ቀበሮዎች ቁጥር	2.33985
የዞኑ ንግድ እንዱስትሪና	1.754888
እንዱስትሪና ቱሪዝም መምሪያ	1.754888
ቅዬ የዱር እንስሳት	1.754888
የዱር እንስሳት ከፍተኛ	1.754888
እንስሳት ከፍተኛ ባለሙያ	1.754888
የሱፍ አብዱላሂ ሱከር	1.754888
የዱር እንስሳት እና	1.754888
ቅርሶችን የመጎብኘት ባህሉ	1.754888
የአዲስ አበባ ሙዚየም	1.754888
የሰሜን ተራሮች ብሄራዊ	1.754888
በቢሮው የባህል መምሪያ	1.754888
የባህል መምሪ ያሃላፊ	1.754888
እንፎርሜሽን ማእከል መግለጫ	1.754888
በመቅደላ ቶርነት ወቅት	1.754888
ትናትና ትበቃ ባለስልጣን	1.754888
የባሌ ተራሮች ብሄራዊ	1.754888
የብጹእ ወቅዱስ ፓትሪያርክ	1.754888
የቱሪዝም አማካሪ ምክር	1.754888
የድንጋይ ዘመን ሰዎች	1.754888
በቤንሻንጉል ጉሙዝ ክልል	1.754888

Table 3.2 Multi-words extracted from the corpus sorted by their C-Value

The above three activities (linguistic filtering, single word selection and multi-word selection) are intended to identify the candidate terminologies which can potentially conceptualize the domain area. Using those identified concepts (terminologies), we tried to construct the taxonomy of the domain.

3.5.4 Taxonomy Construction

Once relevant concepts (terminologies) are extracted using the techniques presented in previous sections there is a need to identify the relationship between concepts and sub concepts. Using the relationship found between concepts and sub concepts, we can construct the taxonomy of the domain. Taxonomy is the hierarchical structure of a given domain.

We tried to construct the taxonomy structure by using a method called Latent Semantic Indexing (LSI), which primarily involves decomposing the term-document matrix into three other matrixes called term matrix, document matrix, and diagonal matrix using Singular Value Decomposition (SVD) method.

Latent Semantic indexing is a statistical method that links terms into a useful semantic structure without syntactic or semantic natural language analysis and without manual human intervention. Latent Semantic Indexing uses Singular Value Decomposition (SVD) to accomplish its goal as described thoroughly by Berry et al. [58]. Algorithm 3.4 depicted the algorithm we used to identify relationships between concepts and sub concepts.

Algorithm 3.4: Taxonomy building using LSI algorithm

```
1: for each word in word list do
2:      create n*m term-document matrix // A [len (word) * no (document)]
3:      u, s, v=linalg. svd (A) // using scipy ntlk module decompose the term-document matrix.
4: end for
5: for each column  i in matrix u do
6:      Term1=u[i][0]
7:      Term2=u[i][0]
8:      for j in range(1,len(u))
9:          if(u[i][j]<term1)
10:             term1=u[i][j]
11:             term2=term1
12:     end for
13:     concept[1]=term1+' _'+ term2
14: end for
15: Output: list of concepts
```

The output of the above algorithm is shown in the table 3.3 below. The two columns show there is a relationship between those two concepts. However, from the list we found that, the result is not showing a convincing relationship between those two columns. Due to that, we were not able to construct the taxonomy using the output and use that taxonomy in our query expansion experiment.

ታቦት	ቅርሶች
የድንጋይ	ባህል
የድንጋይ	ጉሙዝ
በፓርክነት	አቡነ
በፖርኩ	ለማጠናከር
የድንጋይ	ጉሙዝ
አቡነ	ቁጥር
ታቦት	በኤድንበርግ
ቁጥር	ቦታዎች
ሙዚቃ	ስህፈትቤት
ቢኖሩም	ለቱሪዝም
ትገና	ቢኖሩም
በደባርቅ	ሀብት
ቦታዎች	አበባ
የአፋር	ህዝብ
ምክር	አማካሪ
አግዚቢሽን	በስዊድን
አግዚቢሽን	በስዊድን
በስዊድን	አግዚቢሽን
ታደሰ	ኑብር
በደባርቅ	ተወካዩ
ለመግታት	በታሪካዊ
ለማቋቋም	ወይዘሮ
የዶርዜ	ለክልሉ
እሳያስ	እላት

ገሪማ	ከዘርፉ
በፖርኩ	ኑብር
የቀይ	አይነት
ሜትር	ስፖርት
የሀገር	ስፖርት
ክልሉን	ቅርሶች
ሙዚየም	ግንባታው
ወይዘሮ	ተግባራዊ
የስዌን	ቆርኬ
ሙዚየም	ግብር
የስዌን	ቆርኬ
ግብር	አገሩ
አገሩ	ክርስቲያን
መሰረተ	መስህቦች
በመስኩ	በአማራ
በቡግና	በመስኩ
ለኑብኝዎች	ደኑን
ክርስቲያናት	ቦታዎች
የዶርዜ	በኤድንበርግ
አመትና	የኑብር
አገሩ	አበባ
አለማየሁ	የኑብር

Table 3.3 Related concept generated by our algorithm

The un-satisfactory result may happen due to various reasons. In the following few paragraphs we tried to figure out the possible causes of the poor result to accomplish automatic ontology construction from Amharic corpus.

As we mentioned above, to perform automatic ontology construction, part of speech tagging is a crusial step. Since we don't have the automatic tagger for Amharic language, we use manually tagged corpus. This might contribute negatively to the output result. In addition to POS, there are no good works in many areas of linguistic for Amharic language. For example the stemmer that we found does not give the expected result.

To construct the taxonomy of the ontology we used Latent Semantic Indexing that uses the single value decomposition method. The output concepts of the SVD matrix decomposition method are shown in table 3.3. As we can see from the output result in most cases we couldn't get concepts mapped together correctly. There is no clue on the literature that SVD is language dependent. However, this should be investigated in the future, whether it works for Amharic language or not.

Finally, when we reached to a point where automatic or semi-automatic ontology construction is impossible in this point of time in this research, we decided to construct a small domain specific Amharic ontology manually. The tourism sector is our domain area, this is because a suitable corpus that we have for domain specific ontology is tourism sector, and the development process is discussed below.

3.5.5 Manually constructed Tourism Ontology

The tourism ontology is constructed manually to test our ontology based query expansion model. The ontology by itself aims to provide a conceptualized description of information about the tourism domain. Since doing ontology is a hard job [60], it covers few important aspects in the tourism area. It includes tourist attraction area (የቱሪስት መስህብ), accommodation (የቱሪስት ማረፊያ), tourist events (የቱሪስት መዝናኛ), tour guide (አስጎብኚ), service (የአገልግሎት አቅርቦት), and transportation (ትራንስፖረት አቅርቦት). The ontology contains 7 top level classes, 20 subclasses (concepts) and several instances per classes. The class hierarchy has a maximum depth of 3 words (see figure 3.6). This is because as we describe before constructing ontology is a very hard and challenging job and needs a strong effort.

The ontology has 4 main classes (concepts) namely የቱሪስት መስህብ (tourist attraction area), የቱሪስት ማረፊያ (accommodation), የቱሪስት መዝናኛ (tourist events), አስጎብኚ (tour) and ትራንስፖረት (transport) at the top and the other subclasses are grouped under this 4 top level class.

- Tourist Attraction area (የቱሪስት መስህብ) ፡ the places where tourists are interested to visit. This class has 9 subclasses and 54 instances under these classes:
    - People and culture: It is a class which consolidates peoples and group of peoples and their cultures. It includes instances of cultures which can have a potential to attract tourist. For example, surima/'ሱርማ/ people and their cultural festival.

- Park /ፓርክ/: This class contains information about the different parks in the country. It includes many individual parks from different regions. Bale national park/ ባሌ በሔራዊ ፓርክ/ is an example of individual parks in this class.
- Heritage/ ቅርስ/ ፡ this class conceptualize the concept of heritages in the country. There are many kinds of heritage in the country. Religious, cultural and natural heritage instances are grouped under this class. The country has around 10 registered heritages.
- Endemic animal/ ብርቅዬ እንስሳት/፡ animals and birds that can be of interest to tourists for visiting are categorized under this class. For example, red fox/ ቀይ ቀበሮ/ is an instance of endemic animal under this class.
- Waterfall/ ፏፏቴ/: any waterfall instances that attract tourists are grouped under this class.
- Museum/ ሙዚየም/፡ this class contains different areas that are organized for tourists so that they can visit many things in one area.
- Holidays/ በአላት/፡ this class contains holiday festival that tourists may want to see and celebrate with the society. Epiphany is an example of these types of holidays.
- Churches /አብያተ ክርስቲያናት/ ፡ Churches which attract tourists are grouped under this class.

- Accommodation / የቱሪስት ማረፊያ ቦታዎች/፡ places which provide an accommodation for tourist. This class has three subclasses and around 30 instances under those classes:
  - Hotel / ሆቴል/፡ place which provides all accommodation services for tourists are grouped under this class. The service includes food, room, Gym, etc. This concept sub divided into four subclasses namely five stars /አምስት ኮኮብ/, four stars /አራት ኮኮብ/, three stars /ሶስት ኮኮብ/ and ordinary/ መደበኛ/ hotels.
  - Motel / ፔኒሲዮን/፡ this class contain places which gives only bed service.
  - Resort/ ሪዞርት/፡ this class consolidates places which give more services for tourists. Most resorts are built near to lake and have boat visit to tourists.
  - Restaurant / የምግብ እና የመጠጥ ቦታዎች/፡ this class includes all areas which can provide prepared food and drinks. It has a subclass namely: Restaurant, cafe, club and bar.

- Tourist events/ የቱሪስት መዝናኛ/: the place where tourist can get different entertaining events. This class has the following subclasses:
    - Cinema / ሲኒማ/: this class contains individual places which provide movie, shows, and concert service.
    - Theater /ቲያትር/ ፡ this class includes places where the tourist can get real stapled plays.
    - Sport / ስፖርት ማዘውተርያ/፡ contain places that a tourist may perform sport activities. It includes stadiums, gymnasium.
    - Night bar / የምሽት መዝናኛ/፡ this class has individual places that can give night service for tourists. This class has two subclasses called modern night bar / ክለብ መዝናኛ/ and cultural night bar/ የባህል ምሽት/.
- Tour guide/ አስጎብኚ/፡ individual or agency that facilitates the tourist activity. This class has 3 subclasses:
    - Individual guide / የግል አስጎብኚ/፡ this class refers the private individual tour guide.
    - Tour agents/ አስጎብኚ ድርጅት/ ፡ this class refers the tour agents that provide services for tourists.
    - Tourism minister / የባህልና ቱሪዝም ሚኒስቴር/: this class refers the regulator body which controls the overall activity in the industry.
- Infrastructure /አገልግሎት መስጫዎች/: the place that a tourist can get services. This class includes 3 sub classes:
    - Hospital /ሆስፒታል/፡ this class comprise information about the hospitals that provide medication for a tourist if there is a health related issue.
    - Bank /ባንክ/፡ it comprises information about the banks that provide financial issues.
    - Transport /ትራንስፖርት አገልግሎት/: agency that gives transport from city to city. This class encompasses all information about the transportation.
- City /ከተሞች/ ፡ cities in the country those which have a connection with the tourism industry.

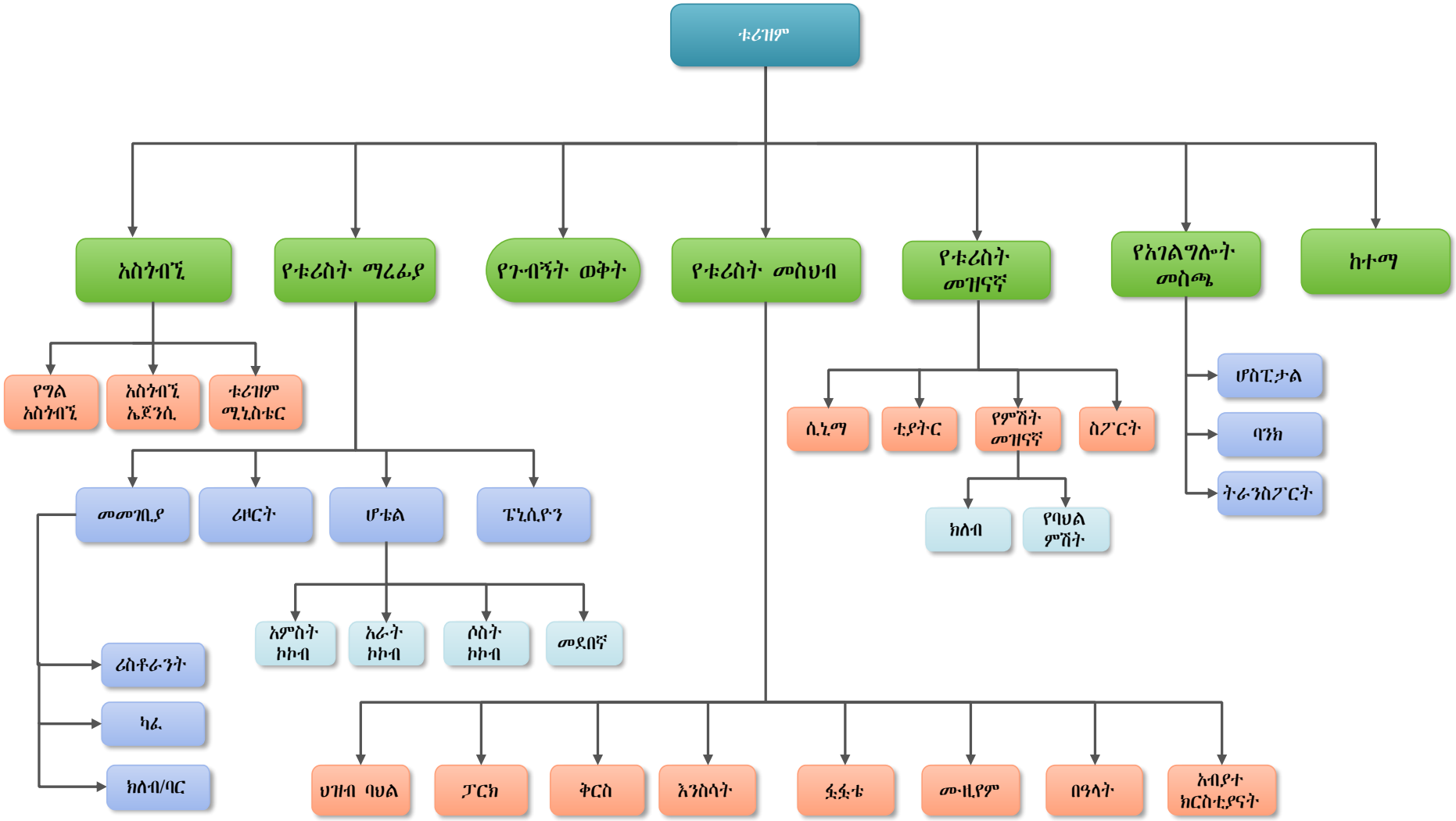

Figure 3.6: The Submission/ Hierarchical / structure of the tourism ontology

Table 3.4: shows individual/ instances/ in each subclass in the hierarchical diagram shown in figure 3.4 above.

Table 3.4: Individual/ instances/ in each subclass

Supper Class	Subclass1	Subclass 2	Subclass 3	Individual/ Instance
ቱሪዝም	አስጎብኚ	የግል አስጎብኚ		ኢትዮ ላንድ፣ቻፕተር
		አስጎብኚ ድርጅት		ዮኤል፣ዮል፣ሶፍ ዑመር፣ኢትዮ ላንድ
		የቱሪዝም ሚኒስቴር		
	ማረፊያ	መመገቢያ እና መጠጫ	ሪስቶራነት	ኮብ ቺክን፣ ቻይና ባር፣ ሙላቱ አስታጥቄ፣ ሀበሻ፤፣ መሶብ ሀበሻ፣ ህብር ኢትዮጲያ
			ካፌ	ላፓሪዝን፣ ሳባ ፣ ቢሎስ፣ አዲስ ጣፋጭ፣ ሀሁ
		ሆቴል	አምስት ኮከብ	ሸራት፣ ሂልተን፣ራዲሰን ብሉ
			አራት ኮከብ	ግዮን ፣ አዲስ ሆቴል፣ ኢሊሌ ኢንተርናሽናል፣ ኢንተርኮንትንትኔንታል
			ሶስት ኮከብ	ሆምላንድ፣ፕላኔት፣አክሱም
			መደበኛ	ጠአይቱ፣ሶሊሽ ኢንተርናሽናል
		ሪዞርት		ባቦ ጋያ ፣ኩሪፊቱ፣አዶላላ
		ፔኒሲን		
	የቱሪስት መስህብ	ህብረተሰብና ባህል		አውራ አንባ፣ኮንሶ፣
		ፓርክ		ሰሜን ተራሮች፣ነጭ ሳር፣ጋምቤላ ብሄራዊ
		ቅርስ		አክሱም ሐውልት፣ፋሲል ግቢ፣ላሊበላ፣ኦሞ ሸለቆ፣ ጥያ ትክል ድንጋይ፣ሀረር
		እንስሳት		ጭላዳ ዝንጀሮ፣ቀይ ቀበሮ፣ዋልያ
		ፏፏቴ		ጣና ሐይቅ፣ጢስዓባይ፣ትስኪ
		ሙዝየም		የኢትዮጲያ ብሄራዊ፣ አዲስ አበባ ሙዚየም፣
		በዓላት		መስቀል፣ጥምቀት፣ አሸንዳ
		አብያተ ክርስቲያት		መንበር ፀባዖት ቅድስት ሥላሴ፣ደብረ ብርሃን ስላሴ፣ ጣና ቂርቆስ፣ዳጋ እስጢፋኖስ፣
	የቱሪስት መዝናኛ	ሲኒማ		ኤድናሞል፣ አለም ፣አንፔር፣አባሳደር፣ኢትዮጵያ ሲኒማ፣ ሆሊይ ሲቲ
		ቲያትር		በሔራዊ ትያትር፣ ሃገር ፍቅር፣ አዲስ አበባ ማዘገጃ ቤት
		ስፖረት		አዲስ አበባ ስቴድየም፣ አለም ፊትነስ፣ቦሌ ሮክ
		የምሽት መዝናኛ	ክለብ	ጃዝ አምባ፣ አራራት፣ፕላቲኒየም፣ ፍለርት ላውንጅ
	አገልግሎት	ሆስፒታል		ጥቁር አንበሳ፣ ኮርያ፣ ዘውዲቱ፣አሴር
		ባንክ		ንግድ ባንክ፣ ንብ፣ወጋገን፣አንበሳ፣ዳሸን፣አሮምያ፣
		ትራንስፖርት		ሰላም፣ሰካይ፣
	ከተማ			ባህርዳር፣ አዋሳ፣ ጎንደር፣መቀሌ፣ሐረር፣ላሊበላ….

3.5.6 RDFs representation

The manually constructed ontology should be stored as RDFs. RDFs is generated with concepts and their relations in the ontology by using the protégé application tool. Here also we faced a problem to use this tool because of language limitations of the tool. As the best knowledge of the researcher the tool doesn't support Amharic languages. To overcome this problem we construct the RDFs using "English- Amharic" words. For example the word 'አስጎቢኚ' in Amharic represented 'Asigobigni' in the protégé tool. After we finish this we generate the owl file and replace "English- Amharic" word with the exact word to the .owl file (see figure 3.7 and 3.8). Actually, this is only for display purpose. Because of the language limitation, we were not be able to use the reasoner tool found in the package.

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.semanticweb.org/ontologies/tourism.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     ontologyIRI="http://www.semanticweb.org/ontologies/tourism.owl">
    <Prefix  name="rdf"  IRI="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Annotation>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <Literal  datatypeIRI="&rdf;PlainLiteral">A  tourism  ontology
that describe varies tourist attraction place </Literal>
    </Annotation>
    <Declaration>
        <Class IRI="#Abiyatkiristiyanat"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Amist_kokobi"/>
    </Declaration>
```

```
        <NamedIndividual IRI="#Yebale_terarawoch_park"/>
    </Declaration>
    <SubClassOf>
        <Class IRI="#Abiyatkiristiyanat"/>
        <Class IRI="#Yturist_Mesihibi"/>

    <Class IRI="#Park"/>
        <NamedIndividual IRI="#Yebale_terarawoch_park"/>
    </ClassAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>#Yturist_Mesihibi</IRI>
        <Literal                datatypeIRI="&rdf;PlainLiteral">Yturist
medatesh</Literal>
    </AnnotationAssertion>
</Ontology>
```

Figure 3.7 Sample tourism ontology generated by the OWL API

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.semanticweb.org/ontologies/tourism.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     ontologyIRI="http://www.semanticweb.org/ontologies/tourism.owl">
    <Prefix   name="rdf"   IRI="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Annotation>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <Literal  datatypeIRI="&rdf;PlainLiteral">A  tourism  ontology
that describe varies tourist attraction place </Literal>
    </Annotation>
    <Declaration>
        <Class IRI="#አብያተ ክርስቲያናት"/>
    </Declaration>
    <Declaration>
```

```
        <Class IRI="#አምስት ኮከብ"/>
    </Declaration>

        <NamedIndividual IRI="#ባሌ በሔራዊ ፓርክ"/>
    </Declaration>
    <SubClassOf
        <Class IRI="#አብያተ ክርስቲያናት"/>
        <Class IRI="#የቱሪሰት መስህብ"/>

    <Class IRI="#ፓርክ"/>
        <NamedIndividual IRI="#ባሌ በሔራዊ ፓርክ"/>
    </ClassAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>#የቱሪሰት መስህብ</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">የቱሪሰት መስህብ</Literal>
    </AnnotationAssertion>
</Ontology>
```

Figure 3.8 Sample tourism ontology generated by the OWL API with Amharic text

Because of the limitation of protégé application tool for Amharic language, we put each concept and its instances in a file with a format shown in figure 3.9 below for our experiment. Each line represents a single concept and all its individuals. For example the first line in figure 3.9 shows one concept which conceptualizes part of tourism domain with a sub concept of a private tour guide. The '@' symbol separate all classes and subclasses. The '#' symbol separate instances in that sub concept.

```
ቱሪዝም@አስጎብኚ@የግል አስጎብኚ@@ኢትዮ#ላንድ#ቻፕተር
ቱሪዝም@አስጎብኚ@አስጎብኚ ድርጅት@@ዮኤል#ዮል#ሶፍ ዑመር#ኢትዮ ላንድ
ቱሪዝም@አስጎብኚ@የቱሪዝም ሚኒስቴር@@#
ቱሪዝም@ማረፊያ@መመገቢያ እና መጠጫ@ሪስቶራነት@ኮብ ቺከን# ቻይና ባር# ሙላቱ አስታጥቄ# ሀበሻ፤#መሶብ ሀበሻ# ህብር ኢትዮጲያ
ቱሪዝም@ማረፊያ@መመገቢያ እና መጠጫ@ካፌ@ላፓሪዝን# ሳባ # ቢሎስ# አዲስ ጣፋጭ# ሀሁ
ቱሪዝም@ማረፊያ@ሆቴል@አምስት ኮከብ@ሸራት# ሂልተን#ራዲሰን ብሉ#ሊዊ
ቱሪዝም@ማረፊያ@ሆቴልአራት@ኮከብ@ግዮን # አዲስ ሆቴል# ኢሊሌ ኢንተርናሽናል#
ኢንተርኮንትንትኔንታል#ክራውን#ሶርአምባ
ቱሪዝም@ማረፊያ@ሆቴል@ሶስት ኮከብ@ሆምላንድ#ፕላኔት#አክሱም
ቱሪዝም@ማረፊያ@ሆቴል@መደበኛ@እቴጌ ጣይቱ#ሶሊሽ ኢንተርናሽናል
ቱሪዝም@ማረፊያ@ሪዞርት@@ባቦ ጋያ #ኩሪፊቱ#አዶላላ
```

ቱሪዝም@ማረፊያ@እንግዳ ማረፊያ@@አብድር አዳማ #ታዚና
ቱሪዝም@መስህብ@መስህብ@@ዳሎል#ኤርታሌ#መልካ ቁንጥሬ#ሪፍት ቫሌ#ስምጥ ሸለቆ#ድሬ ሸክ ሁሴን
ቱሪዝም@መስህብ@ህብረተሰብና ባህል@@አውራ አንባ#ኮንሶ#ሙርሲ
ቱሪዝም@መስህብ@ፓርክ@@ሰሜን ተራሮች#ነጭ ሳር#ጋምቤላ #ማንጎ#ሰሜን ሸለቆ#አሞ#አብያታ ሻላ#ባሌ ተራሮች
ቱሪዝም@መስህብ@ቅርስ@@አክሱም ሐውልት#ፋሲል ግቢ#ጎንደር አብያተ መንግስት#ጀጎል ግንብ#ላሊበላ#ኦሞ ሸለቆ#ጥያ
ትክል ድንጋይ#ሀረር#ሰሜን ተራሮች#ታችኛው አዋሽ#ኮንሶ ባህላዊ መንደሮች#የመስቀል በዓል
ቱሪዝም@መስህብ@ብርቅዬ@እንስሳት@ጭላዳ ዝንጀሮ#ቀይ ቀበሮ#ዋልያ# አቦሸማኔ#ደጋ ድኩላ#ስዋይን
ቆርኬ#አንበሳ#ነብር#የሜዳ አህያ#ዝሆን#አእዋፍ
ቱሪዝም@መስህብ@ፏፏቴ@@ጢስዓባይ#ትስኪ
ቱሪዝም@መስህብ@ሙዝየም@@ብሄራዊ# አዲስ አበባ ሙዚየም#ስማዕታት ሐውልት#እንጦጦ
ቱሪዝም@መስህብ@በዓላት@@መስቀል#ጥምቀት#አሸንዳ
ቱሪዝም@መስህብ@ገዳም@@ደብረ ሊባኖስ#ደብረ ብርሃን ስላሴ#ጣና ቂርቆስ#ዳጋ እስጢፋኖስ#አዝዋ ማርያም#ይጋንዳ
ተክለሃይማኖት#ዘጌ ጊዮርጊስ#መስቀለ ክርስቶስ#አዳዲ ማርያም#ሳማ ሰንበት#ደብረ ዳሞ#ክርስቶስ ሰምራ
ቱሪዝም@መስህብ@አብያተ ክርስቲያን@@ደብረ ሊባኖስ#መንበ ፀባያት ቅድስት ሥላሴ#ደብረ ብርሃን ስላሴ#ጣና
ቂርቆስ#ዳጋ እስጢፋኖስ#አንዋር መስጊድ#አዝዋ ማርያም#ይጋንዳ ተክለሃይማኖት#ዘጌ ጊዮርጊስ#መስቀለ
ክርስቶስ#ግሸን#አዳዲ ማርያም#አክሱም ጺዮን#ሳማ ሰንበት#ደብረ ዳሞ#ክርስቶስ ሰምራ
ቱሪዝም@መዝናኛ@ሲኒማ@@ኤድናሞል# አለም #አንፔር#አባሳደር#ኢትዮጵያ ሲኒማ# ሆሊይ ሲቲ
ቱሪዝም@መዝናኛ@ቲያትር@@በሔራዊ ትያትር# ሃገር ፍቅር# አዲስ አበባ ማዘገጃ ቤት
ቱሪዝም@መዝናኛ@ስፖረት@@አዲስ አበባ ስቴድየም# አለም ፊትነስ#ቦሌ ሮክ
ቱሪዝም@መዝናኛ@የምሽት መዝናኛ ክለብ@@ጃዝ አምባ# አራራት#ፕላቲኒየም# ፍለርት ላውንጅ
ቱሪዝም@አገልግሎት@ሆስፒታል@@ ጥቁር አንበሳ# ኮርያ# ዘውዲቱ#አሴር#አማኑኤል
ቱሪዝም@አገልግሎት@ባንክ@@ንግድ ባንክ# ንብ#ወጋገን#አንበሳ#ዳሸን#አሮምያ#
ቱሪዝም@አገልግሎት@ትራንስፖርት@@ሰላም#ሰካይ#
ቱሪዝም@ከተማ@@@ባህርዳር# አዋሳ# ጎንደር#መቀሌ#ሐረር#ላሊበላ

Figure 3.9: Manually constructed ontology in a file.

To extract the concept and instances of a concept from the file that contains many lines with the above format we used algorithm 3.5.

Algorithm 3.5: Vocabularies loaded to the memory

1: Open the ontology file

2: while end of file do

3: read each line from a file

4: Split the line with '@' separator

5: Split the last string with '#' separator

6: end

7: Close file

8: Output: list of concepts and instance in a concept

After the ontology is loaded on the file, another algorithm is used to select concepts and individual instances, which are going to match with the user query, from the ontology that reside in the memory.

3.6 Query expansion.

As we discussed in section 2.4 in the previous chapter, Query expansion (QE) is the process of reformulating a seed query to improve retrieval performance in an information retrieval operations. In the context of web search engines, query expansion involves evaluating a user's input (i.e. what words were typed into the search query area) and expanding the search query to match additional documents.

We invoked query expansion in our model to increase the quality of search result and to improve the efficiency of the retrieval process. The very hard thing in query expansion is selecting the best expansion terms that are used to expand the original query. Different methods and approaches were proposed on various researches. Previous Amharic language query expansion researches used relevance feedback [4], co-occurrence analysis and Bi-gram [5] and, wordnet with word sense disambiguation [6] methods to identify the terms that can be used in the expansion. The query expansion model designed in this research is an automatic query expansion model which uses domain specific Amharic ontology to extract best expansion terms. We assumed that the submitted user queries are within a particular domain area (i.e. Tourism domain).

3.6.1 Term selection

Our model uses an ontology as a support knowledge source to select the right terms for the expansion. One of the definitions of ontology states that, "Ontologies provide consistent vocabularies and world representations necessary for clear communication within knowledge domains" [37, 44]. Therefore, this clear communication nature in the domain area helps to minimize the ambiguity of words during the searching process. Our assumption is, due to the use of specific domain ontology, the problem of synonymous and polysemous words is minimized. Since our ontology is not full representative we can only minimized the effect rather eliminated at all. The overall purpose of our experiments is to assess the usefulness of ontology based query expansion in providing contextual information to resolve ambiguous queries and improve the search results.

In the model when the user inserts the query to the GUI (graphical user interface), the IR model sends the query to the query preprocessor. The query preprocessor performs the text operation on the given query in the same fashion as the document is preprocessed. Once the query is processed the query terms are forwarded to the query expansion module. The query expansion module implements the algorithm 3.6 to identify the best expanding term from the ontology.

Algorithm 3.6: Expansion term selection algorithm

```
1: for each t in query list do
2:      for c in concept list do
3:          If (c = = t) then
4:                new_term=add(s, e, i) // subclass, equivalent class and individuals under subclass
5:          Else if (t=i) then
6:                new_ term=add (i, I) // individual, neighbor individuals
7:          else
8:                new_ term =add (t)
9:          end if
10:     end for
11: Output: list of terms (new_term)
```

For each term *t* in a query the algorithm finds similar concepts *c* from the ontology class. If the term *t* matches with any concept *c* in the ontology all sub concepts *s*, equivalent sub concepts *e* and sub instances *I* in the sub concepts are added to the expanded list for the expansion. If term *t* is not found in the concept list, then it will be checked in the instance list. If the term *t* matches with any instance *i* under sub concept in the ontology, all brother instances *I* are added to the expansion list. If the term is not found in the ontology at all, the algorithm returns the original query *q*. Based on the returned query the IR model retrieves documents and rank them based on their relevance. However, instead of using all the selected term in the process of expanding we implement weighting mechanism to get the best expanding term.

3.6.2 Filtering Candidate terms

Using all the candidate terms which are found from the expansion model might not yield a good result. Therefore, we implement a filtering mechanism to choose the best terms among the list given by our expansion model. The function of filter module is filtering out the words which have less potential to differentiate documents well. Our filtering relies on the TF/IDF (Term frequency/ inverse document frequency) of the word in document. If the tf-idf of a word is higher than a specified value, the word will be kept. If the TF/IDF is lower than the threshold value, the word will be filtered out.

Chapter Four
EXPERIMENTATION

In this research, an effort has been made to design ontology based automatic query expansion technique for Amharic IR system. A sample Amharic ontology was manually constructed to experiment the proposed model on the tourism domain. The proposed expansion system is integrated with a prototype IR system, which uses the Probabilistic Retrieval Model. The IR model is a modified version of what was developed by Amanuel [9]. The proposed architecture (see Figure 3.2) starts by receiving a user query from the user interface. The received query is passed to the query preprocessor module to clean the query itself from unnecessary words, for example stop words. The preprocessor module passes the cleaned query to the expansion module so that the user query will be expanded by the terms found from the ontology. Therefore, in this research, user queries' are expanded by using already available knowledge from the ontology. After we expand the user query, searching is carried out and the result is displayed to the user.

The goal of this research is to evaluate the performance of the query expansion model which uses ontology as a supportive source knowledge. The retrieval performance is evaluated based on precision, recall and F-measures. The precision, recall and F-measures are computed before augmenting the original user query and after integrated the model and modified the original user query.

4.1 Dataset Preparation

Domain specific ontology need to be used in this experiment to expand a user query. There should be a close match between the ontology and the document collection in order to get the most effective result. If the concepts contained in the document collection are not covered by ontology, ontology based query expansion will not produce a better performance. To create a domain specific corpus for this purpose, domain specific text files are gathered from the World Wide Web and the internet. The corpus consists of 195 documents and 5632 unique tokens (the vocabulary size become very big because of the poor efficiency of the Amharic stemmer. There

is duplication of the same word in different format). The documents include news, articles, which talks about tourism domain, and address of the places where tourists want to go. These documents are saved under a common folder in plain text format. Plain text is used for document representation to achieve the benefits of saving memory. Moreover, plain text is supported by most programming languages.

To conduct the experimentation, a SAMSUNG laptop computer with specification, AMD A6-420M APU CPU having a processor speed of 1.5 GHz, 4 Giga Bytes RAM, and Windows 7 Ultimate edition 64-bit operating system is used. The prototype is developed using the Python programming language.

In order to test the performance of the prototype ontology based automatic query expansion system, ten test queries are selected. The central idea of this thesis "The use of ontology based query expansion to improve the retrieval effectiveness" is tested by these selected queries. The selected query contains words that are concepts in the tourism ontology. Table 4.1 shows the test queries used in this study.

No	Test Query
1	የቱሪስት መስህብ
2	አለም አቀፍ ቅርስ
3	ብርቅዬ እንስሳት
4	የመመገቢያ እና ማረፊያ
5	አብያተ ክርስቲያን
6	ሆቴል
7	ገዳም
8	ፓርክ
9	የምሽት ክበብ
10	ፏፏቴ

Table 4.1: Sample test queries used for the experiment

4.2 Description of the Query Expansion System

Query words are known to be ambiguous. However, our assumption in this work is that the user inserts his/her query to a domain specific search engine which minimizes or avoids the ambiguity of words.

For each query term, we find out an additional expanding term which has a potential to improve the performances of the system, from the ontology. Our ontology file consists of all sampled concepts in the tourism domain. Each main concept, sub concepts and individual instance in a concept are saved in a single line. Concepts are separated by '@' symbol, and '#' symbol is used to separate individual instance in a concept. For example, the following ontology line

" ቱሪዝም@የቱሪስት መስህብ@ፓርክ@ሰሜን ተራሮች#ነጭ ሳር#ጋምቤላ ብሄራዊ"

Gives one main concept in the tourism domain and instances in that specific concept. "ቱሪዝም" is the main concept and "የቱሪስት መስህብ" and "ፓርክ" are sub concepts under the main concept. "ሰሜን ተራሮች","ነጭ ሳር"and "ጋምቤላ ብሄራዊ"are instances in a sub concept.

The envisaged system first load the ontology structure defined and saved in a file. When it loads the structure in a memory, first, it read one line at a time from the file, and then it breakdown the line into terms , finally it store the individual terms in the structured dictionary which makes easy to use in future.

During searching, Search the query term submitted by users is searched from the structured dictionary, if it found equivalent concept, the expansion model use the lower part of the ontology tree to identify the expanding terms which can enable to empower the user query.

Finding an equivalent term becomes a challenging task when some of the ontology have no in between concepts. In that case the memory assigned for the sub concept become null.

4.3 Performance Evaluation

The selected ten test queries were used to evaluate the performance of the prototype IR system with and without the proposed query expansion model. Precision, recall and the F-measures are used for the evaluation.

The experiments were carried out before and after the query expansion process to evaluate the performance of the IR system. Table 4.2 shows the initial performance of the IR system before query expansion is applied.

Query	**Retrieved**	**Relevant Retrieved**	**Relevant**	**Precision**	**Recall**	**F-measure**
የቱሪስት መስህብ	30	18	56	0.60	0.32	0.42
አለም አቀፍ ቅርሶች	46	14	33	0.30	0.42	0.35
ብርቅዬ አራዊት	32	15	17	0.47	0.88	0.61
የመመገቢያ እና ማረፊያ	20	1	7	0.05	0.14	0.07
አብያተ ክርስቲያናት	15	1	12	0.07	0.08	0.07
ሆቴል	21	8	10	0.38	0.80	0.52
ገዳም	9	6	13	0.67	0.46	0.55
ፓርክ	19	16	27	0.84	0.59	0.70
የምሽት ክበብ	7	2	3	0.29	0.67	0.40
ፏፏቴ	8	6	8	0.75	0.75	0.75
AVERAGE				0.44	0.54	0.44

Table 4.2 Performance of IR system retrieval without query expansion

The IR system showed 44% average precision, 54% average recall and44% average f-measure without adding expansion terms in the user query.

The second test is carried out by placing the query expansion module in the prototype IR system. In this second test we concatenated the original query, submitted by the user, with terms found from the ontology. Precision, recall, and F- measure also calculated for this second test. Table 4.3 shows the performance of the IR system after the query expansion is applied.

Query	**Retrieved**	**Relevant Retrieved**	**Relevant**	**Precision**	**Recall**	**F-measure**
የቱሪስት መስህብ	139	65	76	0.47	0.86	0.60
አለም አቀፍ ቅርሶች	132	31	33	0.14	0.94	0.36
ብርቅዬ እንስሳት	39	17	17	0.44	1	0.61
የመመገቢያ እና ማረፊያ	34	5	7	0.15	0.71	0.24
አብያተ ክርስቲያናት	96	10	12	0.10	0.83	0.19
ሆቴል	23	9	10	0.39	0.90	0.55
ገዳም	28	12	13	0.43	0.92	0.59
ፓርክ	44	22	27	0.50	0.81	0.62
የምሽት ክበብ	7	2	3	0.29	0.67	0.40
ፏፏቴ	8	8	8	1.00	1.00	1.00
AVERAGE				0.40	0.86	0.52

Table 4.3 Result of experimenting of IR system retrieval with query expansion

The result of the experiment, having expansion module, showed a drop of precision and an increase on recall. Recall is increased from 0.54 to 0.86 which shows 32% performance improvement. F-measure also increased from 0.44 to 0.52. The overall performance improved by around 7%.

4.4 Finding and Challenges

In this research, we attempted to evaluate the performance of the IR system by integrating ontology based query expansion module to the Amharic IR system.

The Experimental result shows that ontology based query expansion, improved the performance of the system in terms of recall. An average 86% recall is registered and F-measure is improved by 7%. The use of ontology based query expansion has only increased mean average recall but the precision is usually identical to the baseline or sometimes even below the baseline. The reason for this is that more ontology child terms are retrieved but all the retrieved terms are actually relevant, thus having minimum impact on precision. The retrieval effectiveness is highly

affected by the ambiguity of terms found from the ontology for the expansion. The selected candidate terms become ambiguous. For example, when the system searchs for additional terms for the query 'የቱሪስት መስህብ', ' ሙዚየም' is one of the sub concepts in the ontology tree of 'መስህብ' concept. Therefore, the system picks that concept and start looking for sub-classes and instances of the ontology if any. In our ontology there is 'የኢትዮጵያ በሔራዊ' as an instance, under 'ሙዚየም' concept. Thus, the system used this instance to expand the query term. However, as we can understand from the result the word 'የኢትዮጵያ' affect the performance of the system. This is because the word 'የኢትዮጵያ' become expanding word and unable to discriminate the document in the output result.

In the process, there were many challenges that are encountered. The first challenge was the construction of ontology for our experiment. To undertake the query expansion experiment, getting ontology was a crucial step. However, we didn't find any existing ontology. We had tried to construct it ourselves automatically from our corpus, but as we explained in the previous chapter we failed to get a workable ontology output. For the sake of the experiment we developed a small scale manual ontology. The second challenge is the small number of concepts in the ontology and lack of richness in concept in the domain and the necessary relationships that exist between the concepts. The other challenge is the limited amount of the test data used for indexing and searching. We have faced a big problem to collect and set up domain specific dataset to our experiment. To have reliable experimental output results, there should be a standard data set. Finally, as Amharic is an under resourced language, it was a challenge to come across a good stemmer and part of speech tagger.

Chapter Five
CONCLUSION and RECOMMENDATION

5.1 Conclusion

Information retrieval engines have been used to carry out information as per the request of information seekers. Information on the web is very huge. When combined with the ambiguity of the languages, a long list of results is returned for a query, much of which is not always relevant to the user's information needs. Because of this long list, the user community is suffering from information overload. To avoid unnecessary document and to increase the number of relevant documents retrieved, users' queries need to be empowered with additional words, which have more power to retrieve relevant documents.

Query expansion has been used for the purpose of improving the searching capability of the user queries. Different types of query expansion methods have been tested in many researches. In this research an attempt is made to design ontology-based query expansion in order to improve the performance of the Amharic language IR system.

Experiments were done to explore the extent to which ontology based query expansion techniques improve performance of the system. The first experiment was carried out by concatenating user query and the terms found from the ontology. The first experiment results shows that the performance of the system improved by 42% in terms of recall and 7% overall system improvement. It shows a decline of precision of the system. The second experiment result showed the same improvement result without affecting precision of the system. Using terms found from the ontology helps more to retrieve more document. However, to increase the precesion of the system more work need to be done on the area..

Our experiments prove the central idea of this research "Does the ontology based query expansion method improve the effectiveness of the Amharic IR system?" However, more work needs to be done on improving the precision results for ontology based query expansion.

5.2 Recommendation

Ontology becomes one of the key areas to support human knowledge. Very limited research work has been attempted in regards to building ontology for Amharic language.

In this research ontology based query expansion model integrated with the IR system to test how far it can improve the performance of the Amharic IR system. Although, it has given a valuable lesson on using ontology in query expansion for the purpose of improving the system performance, there are some issues that need to be further studied to design an effective and efficient ontology based query expansion and IR system. The following are recommended for further research.

1. This research assumes the use of ontology to minimize the ambiguity of words in the expansion process. However, the finding shows that the performance of the system is affected by ambiguous words. Thus more work need to be done to minimize the ambiguity of words that are extracted from the ontology itself.
2. In this study, the terms selected from the ontology are not evaluated to measure their importance before we use them for the expansion. This should be further studied for enhancing the performance of the IR system.
3. To get the desired performance an advanced and rich ontology is required. Automatic or semi-automatic ontology extraction method should be studied and designed.
4. To enhance the retrieval performance of the Amharic IR system, more research work should be done to design an effective and workable stemmer, part of speech tagger for Amharic language -
5. This research assumes the use of ontology to minimize the ambiguity of words in the expansion process. However, the finding shows that the performance of the system is affected by ambiguous words. Thus more work need to be done to minimize the ambiguity of words that are extracted from the ontology itself.
6. Having standard corpus and test queries for evaluating the effectivnes of an IR system is a must. At present, there is no standard corpus for such experiment at all. Thus, the development of the standard Amharic corpus shall be further explored

References

[1]. A. Andreou, “Ontologies and Query Expansion”, MSc Thesis, School of Informatics, University of Edinburgh, United Kingdom, 2005.

[2]. Ed Greengrass, ‘Information Retrieval: A Survey”, 2000.

[3]. Baeza-Yates R. and Ribeiro-Neto B., “Modern information retrieval”, 2nd ed, Addison-Wesley, 1999.

[4]. Alemayehu, "Application of query expansion for Amharic information retrieval system”, MSc Thesis, Addis Ababa University, Ethiopia, 2010.

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

[6]. Iman M. " Query Expansion on proper sense disambiguation for Amharic language”, MSc Thesis, Addis Ababa University, Ethiopia, 2013.

[7]. Buckland, M. “Translingual information management using domain ontologies”, available at http://metadata.sims.berkeley.edu/GrantSupported/tides.html , Accessed Date; 15/12/2014, 2013

[8]. Tewodros H. “Amharic text retrieval: an experiment using semantics indexing (LSI) with singular value decomposition” MSc Thesis, School of Information Science, Addis Ababa University, Ethiopia, 2003.

[9]. Amanuel Hirpa, “Probabilistic Information Retrieval System for Amharic Language", MSc Thesis, School of Information Science, Addis Ababa University, Ethiopia, 2012.

[10]. Hofstede A.H.M. -, Proper H.A., and Weide Th.P. van der, “Query formulation as an information retrieval problem”, The Computer Journal, 39(4), pp. 255-274, 1996

[11]. Ben He and Iadh Ounis, “Studying Query Expansion Effectiveness“, Department of Computing Science, University of Glasgow, United Kingdom,2009

[12]. C.R. Kothari, Research Methodology: Methods and Techniques, 2nd ed. India: New Age International Publishers, 2004.

[13]. Stephen Morse, “5 things you need to do to get search right”, available at http://easydita.com/5-things-you-need-to-do-to-get-search-right.

[14]. Vannevar Bush, “As We May Think”, Atlantic Monthly, 176:101–108, 1945.

[15]. H.P. Luhn, “A statistical approach to mechanized encoding and searching of literary information”, IBM Journal Research and Development, Vol. 1, pp. 309-317, 1957..

[16]. Gerard Salton,”The SMART Retrieval System—Experiments in Automatic Document Retrieval”. Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[17]. Christopher. D. Manning, P. Raghavan and H. Schütze, "An Introduction to Information Retrieval", Cambridge UP, Available at: http://nlp.stanford.edu/IRbook/pdf/irbookonlinereading.pdf, Accessed Date; 3/4/2013, 2009.

[18]. G. Salton, J. McGill, “Introduction to modern in formation retrieval”, McGraw Hill, 1993.

[19]. C. J. van rijsbergen ,”Information retrieval,” Department of Computing Science University of Glasgow.

[20]. Luis M. de Campos, Juan M. Fernandez, Juan F. Huete,”Query Expansion in Information Retrieval Systems using a Bayesian Network-Based Thesaurus”, E.T.S.I. Information University de Granada SPAIN.

[21]. Amanda Spink, Dietmar Wolfram, Major Jansen, and Tefko.Saracevic, “Searching the web: the public and their queries,” Journal of the American Society for Information Science and Technology, vol. 52, no. 3, pp. 226–234, 2001.

[22]. Mohameth-François Sy, Sylvie Ranwez, Jacky Montmain, Armelle Regnault, Michel Crampes, and Vincent Ranwez, "User centered and ontology based information retrieval system for life sciences ," From Semantic Web Applications and Tools for Life Sciences (SWAT4LS), Berlin Germany, 2010

[23]. Wu. F. Wu. G. Fu. X.,"Design and Implemntation of Ontology-Based query expansion for information retrieval,” in IFlP International Federation for Information Processing, Volume 254. Research and Practical Issues 01' Enterprise Information Systems II Volume I pp. 293-298, 2007.

[24]. W. Bruce Croft, Michael Bendersky, Hang Li2, Gu Xu2, “Query Representation and Understanding Workshop report,” SIGIR conference , vol 4(2), pp 48-53, 2010

[25]. Wollersheim D., Rahayu W. J., “Ontology Based Query Expansion Framework for Use in Medical Information Systems” J. WEB INFOR. SYST. vol. 1(1), 2005

[26]. Jinxi Xu and W. Bruce Croft, “Query expansion using local and global document analysis”, Center for Intelligent Information Retrieval Computer Science Department University of Massachusetts, Amherst, MA 01003-4610, USA, In SIGIR '96, pages 4-11, 1996.

[27]. Gerard Salton and Chris Buckley, "Improving retrieval performance by relevance feedback", Journal of the American Society of Information Science, vol 41(4):288-297, 1990.

[28]. J. J. Rocchio,"Relevance feedback in information retrieval." In The SMART Retrieval System: Experiments in Automatic Document Processing, pages 313-323. Prentice-Hall Inc., 1971.

[29]. Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma, "Query Expansion by Mining User Logs", IEEE Transactions on knowledge and data engineering, vol 15(4), 2003.

[30]. Yuanhua Lv, ChengXiang Zhai,"Positional Relevance Model for Pseudo-Relevance Feedback", SIGIR'10, Geneva, Switzerland, 2010.

[31]. Fu, G., C. B. Jones and A. I. Abdelmoty, "Ontology-based spatial query expansion in information retrieval". In Proceedings OTM Confederated International Conferences CoopIS, DOA, and OOBASE, pp. 1466-1482, 2005.

[32]. Hazra Imran and Aditi. Sharan, "Thesaurus and Query Expansion", International Journal of Computer science & Information Technology, vol. 1(2), pp. 89–97, 2009.

[33]. Y. Qiu and H. P. Frei, "Improving the retrieval effectiveness by a similarity thesaurus," Technical Report No. 225, Dept. Computer Science, Swiss Federal Institute of Technology (ETH), 1995.

[34]. Yonggang. Qiu and H. P. Frei, "Concept based query expansion," in Proceedings of the 16th ACM SIGIR International Conference on Research and Development in Information Retrieval, pp. 160-169, 1993.

[35]. Lipika Dey, Shailendra Singh, Romi Rai, and Saurabh Gupta. Ontology aided query Expansion for retrieving relevant texts. In Advances in Web Intelligence Third International Atlantic WebIntelligence Conference, pages 126–132, 2005.

[36]. Yu Zheng, Wenxiang Dou, Gengfeng Wu1, and Xin Li2, "Automated Chinese Domain Ontology Construction from Text Documents", Springer-Verlag Berlin Heidelberg LSMS, LNCS 4688, pp. 639 – 648, 2007.

[37]. Eva Blomqvist, "Semi-automatic Ontology Construction based on Patterns", Department of Computer and Information Science Linköping University, Dissertation, 2009.

[38]. G. A. Miller, WordNet: a lexical database, Available at http://wordnet.princeton.edu/. Accessed date 02/04/2014

[39]. B. Padmaja Rani and Ramakrishna Kolikipogu, Information Retrieval in Indian Languages: A Survey on Query Expansion Techniques by taking Telugu as a Case Study, International Journal of Advanced Computing (IJAC), Vol. 4, Issue 3 & 4, 2012.

[40]. D. E. Losada, Query Expansion Using Wordnet With a Logical Model of Information Retrieval, IADIS International Conference Applied Computing, Vol, 2, p.487-494, 2005.

[41]. A. Andreou, Ontologies and Query Expansion, MSc Thesis, School of Informatics, University of Edinburgh, United Kingdom, 2005.

[42]. Sack, H. (2005): NPBibSearch: An ontology augmented bibliographic search. In Proceedings 2nd Italian Semantic Web Workshop, Trento, Italy.

[43]. J. F. Sowa. Knowledge Representation - Logical, Philosophical, and Computational Foundations. Brooks/Cole, 2000.

[44]. Nenadic, Goran, Irena Spasic, and Sophia Ananiadou, Automatic discovery of term similarities using pattern mining, COLING-02 on COMPUTERM 2002: second international workshop on computational terminology-Volume 14, Association for Computational Linguistics, 2002.

[45]. S. Staab and R. Studer (Eds.), Handbook on Ontologies, Springer Dordrecht Heidelberg London New York,2009.

[46]. Carola Carstens, Ontology Based Query Expansion – Retrieval Support for the Domain of Educational Research, PhD Dissertation, Department III (Linguistics and Information Science) of the University of Hildesheim, Germany, 2011.

[47]. Grigoris Antoniou and Frank van Harmelen, Web Ontology Language: OWL.

[48]. Berhanu Mengiste, Automatic Ontology Learning From Unstructured Amharic Tex , MSc Thesis, Addis Ababa University, Ethiopia, 2012.

[49]. P. Cimiano. Ontology Learning and Population from Text - Algorithms, Evaluation and Applications. Springer, 2006.

[50]. Ramos, J., Using tf-idf to determine word relevance in document queries, 2003.

[51]. K. Frantzi, S. Ananiadou, and J. Tsuji. The c-value/nc-value method of automatic recognition for multi-word terms. In Proceedings of the ECDL, pp 585-604, 1998.

[52]. Andreia Dal P. and José Maria P. , Simple Method for Ontology Automatic Extraction from Documents, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No. 12, 2012.

[53]. Sarah Saad Eldin , Ahmed Sharaf eldin Ahmed and Adel Elsayed, Using of conceptual representation approach for query expansion in information retrieval, International Journal of Engineering & Computer Science IJECS-IJENS Vol:12 No:04,2012.

[54]. Million Meshesha, Recognition and Retrieval from Document Image Collections, PhD Dissertation, International Institute of Information Technology, India, 2008.

[55]. http://www.spellingsociety.org/journals/j19ethiopic.php //amharic_language, Accessed date 03/01/2014

[56]. Biniam Asnake, Retrieval from real-life Amharic document images, MSc Thesis, Addis Ababa University, Ethiopia, 2012.

[57]. Martha Yifiru, "Morphology-Based Language Modeling for Amharic," PhD Dissertation, Department Informatik, Universität Hamburg, 2010.

[58]. M.W. Berry , S. T. Dumais, G. W. O' Brien, Using linear algebra for Intelligent Information Retrieval, Germany, 1994.

[59]. Payton 2.7 NLTK tool, available at http://www.nltk.org/download, Accessed date 03/03/2014

[60]. Shiyan Ou1, Viktor Pekar, Constantin Orasan, Christian Spurk, and Matteo Negri, "Development and Alignment of a Domain-Specific Ontology for Question Answering", Research Group in Computational Linguistics, University of Wolverhampton, UK.

[61]. Govind Reddy M., Chakravarthi S., Sadanand Srivastava, and James Gil , "Ontology Extraction from text documents by Singular Value Decomposition", Bowie State University, 14000, Jericho park road, Bowie, MD, 20715, 2001

APPENDICES

Appendix A: Relevance judgment used in the experiment

No.	Documents	የቱሪስት መስህብ	አለም አቀፍ ቅርሶች	ብርቅዬ እንስሳ ት	ማረፊያ እና መመገቢያ	አብያተ ክርስቲያ ን	ሆቴል	ገዳም	ፓርክ	የምሽት መዝናኛ	ፏፏቴ
1	Abiyot Kiris.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
2	Akisum Holit Tarik.txt	relevant	non-relevant	non-relevant	Non - relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
3	Akisum.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
4	Alem Fitines Jim.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
5	Amanuel Hosipital.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
6	Amhara Enisisat.txt	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
7	Arrat kileb.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant
8	Aser Hosipita.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
9	Awiraniba people.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
10	Aziwa Mariyam.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
11	Bale Key.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
12	Bale.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
13	Bank2.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
14	Birkiye Mulu ken.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
15	bole roke Jim.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
16	cack bet.txt	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
17	Cafe.txt	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
18	CEducation42.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
19	CEducation43.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
20	CEducation44.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
21	CEducation45.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
22	CEducation46.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
23	CEducation47.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
24	CEducation48.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant

No.	Documents	የቱሪስት መስህብ	አለም አቀፍ ቅርሶች	ብርቅዬ እንስሳ ት	ማረፊያ እና መመገቢያ	አብያተ ክርስቲያ ን	ሆቴል	ገዳም	ፓርክ	የምሽት መዝናኛ	ፏፏቴ
25	CEducation49.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
26	CEducation50.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
27	CEducation51.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
28	CEducation52.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
29	CEducation53.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
30	CEducation54.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
31	CEducation55.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
32	CEducation56.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
33	CEducation57.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
34	CEducation58.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
35	CEducation59.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
36	CEducation60.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
37	CEducation61.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
38	Charpter Asigobign.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
39	Chilada zinijero.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
40	Commertial Bank.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
41	CTourism1.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
42	CTourism10.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant
43	CTourism11.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant
44	CTourism12.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
45	CTourism13.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
46	CTourism14.txt	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
47	CTourism15.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
48	CTourism16.txt	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
49	CTourism17.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
50	CTourism18.txt	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
51	CTourism19.txt	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
52	CTourism2.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant

No.	Documents	የቱሪስት መስህብ	አለም አቀፍ ቅርሶች	ብርቅዬ እንስሳ ት	ማረፊያ እና መመገቢያ	አብያተ ክርስቲያ ን	ሆቴል	ገዳም	ፓርክ	የምሽት መዝናኛ	ፏፏቴ
53	CTourism20.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
54	CTourism21.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
55	CTourism22.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
56	CTourism23.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
57	CTourism24.txt	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
58	CTourism25.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
59	CTourism26.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
60	CTourism27.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
61	CTourism28.txt	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
62	CTourism29.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
63	CTourism3.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
64	CTourism30.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
65	CTourism31.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
66	CTourism32.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
67	CTourism33.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
68	CTourism34.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
69	CTourism35.txt	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
70	CTourism36.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
71	CTourism37.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant
72	CTourism38.txt	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
73	CTourism39.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
74	CTourism4.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant
75	CTourism40.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
76	CTourism41.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
77	CTourism42.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
78	CTourism43.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
79	CTourism44.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
80	CTourism45.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant

No.	Documents	የቱሪስት መስህብ	አለም አቀፍ ቅርሶች	ብርቅዬ እንስሳ ት	ማረፊያ እና መመገቢያ	አብያተ ክርስቲያ ን	ሆቴል	ገዳም	ፓርክ	የምሽት መዝናኛ	ፏፏቴ
81	CTourism46.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
82	CTourism5.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
83	CTourism6.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
84	CTourism7.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
85	CTourism8.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant
86	CTourism9.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
87	Daga Estifanos .txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
88	Debire birihan silase.txt	relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
89	Debire Libanose.txt	relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
90	doc1.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
91	doc10.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
92	doc11.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
93	doc12.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
94	doc13.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
95	doc14.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
96	doc15.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
97	doc16.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
98	doc17.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
99	doc18.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
100	doc19.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
101	doc2.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
102	doc20.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
103	doc3.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
104	doc4.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
105	doc5.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
106	doc6.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
107	doc7.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
108	doc8.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant

No.	Documents	የቱሪስት መስህብ	አለም አቀፍ ቅርሶች	ብርቅዬ እንስሳ ት	ማረፊያ እና መመገቢያ	አብያተ ክርስቲያ ን	ሆቴል	ገዳም	ፓርክ	የምሽት መዝናኛ	ፏፏቴ
109	doc9.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
110	doc_1.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
111	doc_2.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
112	doc_3.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
113	doc_4.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
114	doc_5.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
115	Enisisat 3.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
116	Enisisat 4.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
117	Enisisat 5.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
118	Enivestiment.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
119	Etegetehayitu Hotel.txt	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant
120	Ethio Land Asigobign.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
121	Five Star hotel.txt	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant
122	Four Star Hotel.txt	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant
123	Gishen debrekeribe.txt	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
124	Gitim.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
125	Golif Kileb.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
126	Gonder betemenigist.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
127	Gonder.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
128	gonider hospital.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
129	Gust House.txt	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant
130	Homland_hotel.txt	non-relevant	non-relevant	non-relevant	non-relevant	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant
131	Hotel (2).txt	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant
132	Hotel.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant
133	Jazi amba Kibeb.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant
134	Jegol Ginib.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
135	Key Kebereo-2.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
136	key keberoo.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant

No.	Documents	የቱሪስት መስህብ	አለም አቀፍ ቅርሶች	ብርቅዬ እንስሳ ት	ማረፊያ እና መመገቢያ	አብያተ ክርስቲያ ን	ሆቴል	ገዳም	ፓርክ	የምሽት መዝናኛ	ፏፏቴ
137	Keyi kebro.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
138	Kirisi Baladera.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
139	Koniso Bahilawi Menider.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
140	Lalibela Church.txt	relevant	relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
141	Lalibela.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	relevant	relevant	non-relevant	non-relevant	non-relevant
142	Mehal Zege .txt	relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
143	Meniber Tebaote Kidisit silase.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
144	Mesigid.txt	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
145	Mesikel 2.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
146	Mesikel 3.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
147	Mesikel Beal.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
148	Mesikel Kiristos.txt	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
149	Mezingna.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
150	Murisi Culture.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
151	Muziyem .txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
152	muziyem.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
153	Nech sar park.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
154	Nech sar.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
155	Omo Sheleko (2).txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
156	Omo Sheleko.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
157	Pilanet jim.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
158	Pilatiniyem Kileb.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant
159	Restorant and cafe.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
160	Restorant.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
161	Semen Teraroch B Park.txt	relevant	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
162	Semen Teraroch park.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
163	Sinima House.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
164	Sofumer Asigobign.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant

No.	Documents	የቱሪስት መስህብ	አለም አቀፍ ቅርሶች	ብርቅዬ እንስሳ ት	ማረፊያ እና መመገቢያ	አብያተ ክርስቲያ ን	ሆቴል	ገዳም	ፓርክ	የምሽት መዝናኛ	ፏፏቴ
165	Solish hotel.txt	non-relevant	non-relevant	non-relevant	non-relevant	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant
166	Tana Hiyik 1.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
167	Tana Hiyik.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant
168	Tana kirikosi.txt	relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
169	Tekilehayimanot.txt	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant
170	Tiru sina Mesigid.txt	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
171	Tisi Abay.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant
172	Tisiki fafute.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant
173	Tiya tikil Dinigy.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
174	Tiya.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
175	tourism10.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
176	tourism11.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
177	tourism12.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
178	tourism13.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
179	tourism14.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
180	tourism15.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
181	tourism2.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
182	tourism3.txt	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
183	tourism4.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
184	tourism5.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
185	tourism6.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
186	tourism7.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
187	tourism8.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
188	tourism9.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
189	Waliya 2.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
190	Waliya_3.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
191	Wild animals.txt	relevant	non-relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	relevant	non-relevant	non-relevant
192	Yegara Kirise.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant

No.	Documents	የቱሪስት መስህብ	አለም አቀፍ ቅርሶች	ብርቅዬ እንስሳ ት	ማረፊያ እና መመገቢያ	አብያተ ክርስቲያ ን	ሆቴል	ገዳም	ፓርክ	የምሽት መዝናኛ	ፏፏቴ
193	Yetachignaw Awash.txt	relevant	relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
194	Yoel Asigobign.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant
195	Yol Asigobign.txt	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant	non-relevant

Appendix B: Python code for Probabilistic Amharic IR

B.1. Indexing

```
# -*- coding: utf-8 -*-
import os
import math
import codecs
import string
import re
import glob
import sys
class amhariIndexing:
    def index(self, path):
        def condition1(affix, tempp):
            if len(tempp)> 2:
                if affix.__eq__("ከ"):
                     return 1
                elif affix.__eq__("በ"):
                      return 1
                elif affix.__eq__("ኞች"):
                      return 1
                elif affix.__eq__("ን"):
                      return 1
                elif affix.__eq__("ት"):
                      return 1
                elif affix.__eq__("ነት"):
                      return 1
                elif affix.__eq__("ዎች"):
                      return 1
                elif affix.__eq__("ች"):
                      return 1
                elif affix.__eq__("ችን"):
                      return 1
                elif affix.__eq__("ችሁ"):
                      return 1
                elif affix.__eq__("ቸው"):
                      return 1
                elif affix.__eq__("ቱ"):
                      return 1
                elif affix.__eq__("ቹ"):
                      return 1
                elif affix.__eq__("ዊ"):
                      return 1
                elif affix.__eq__("ዊት"):
                      return 1
```

```
            elif affix.__eq__("ውያን"):
                return 1
            elif affix.__eq__("ቷል"):
                return 1
            elif affix.__eq__("ም"):
                return 1
            elif affix.__eq__("ት") or affix.__eq__("አ"):
                return 1
            elif affix.__eq__("ኛ"):
                return 1
            elif affix.__eq__("ነት") or affix.__eq__("እ"):
                return 1
            elif affix.__eq__("ም") or affix.__eq__("እ"):
                return 1
            elif affix.__eq__("ለት"):
                return 1
        else:
            return 0

    def condition2(affix,tempp):
        if len(tempp) > 2:
            if affix.__eq__("ከ"):
                if tempp.__eq__("ከብት") or tempp.__eq__("ከለከለ") or tempp.__eq__("ከነከነ") or
tempp.__eq__("ከረከረ") or tempp.__eq__("ከመከመ") or tempp.__eq__("ከሰከሰ") or
tempp.__eq__("ከተከተ") or tempp.__eq__("ከተማ") or tempp.__eq__("ከረን") or
tempp.__eq__("ከሰል") or tempp.__eq__("ከፍተኛ") or tempp.__eq__("ከብት"):
                    return tempp
##                  elif len(tempp)==3:
####                    if (check(str))
##                  return tempp
                else:
                    tempp = tempp[1: ]
                    tempp = preffix(tempp);
                    return tempp
            elif affix.__eq__("በ"):
                if tempp.__eq__("በሽታ") or tempp.__eq__("በጀት") or tempp.__eq__("በረበረ") or
tempp.__eq__("በሰበሰ"):
                    return tempp
                else:
                    tempp = tempp[1: ]
                    tempp = preffix(tempp);
                    return tempp
            elif affix.__eq__("ኛ"):
                if tempp.__eq__("በሽታ"):
                    return tempp
                else:
```

```
                tempp = tempp[ :-1]
                tempp = preffix(tempp);
                return tempp
            elif affix.__eq__("ን"):
                if tempp.__eq__("ዘመን") or tempp.__eq__("ቡድን") or tempp.__eq__("ሽፋን") or
tempp.__eq__("ወገን") or tempp.__eq__("ቡድን") or tempp.__eq__("ሽፋን") or
tempp.__eq__("ወገን") or tempp.__eq__("ጭቁን") or tempp.__eq__("ህጻን") or
tempp.__eq__("እውን") or tempp.__eq__("እምን") or tempp.__eq__("ዘፈን"):
                    return tempp
                else:
                    tempp = tempp[ :-1]
##                      tempp=sadisConverter(tempp)
                    tempp = suffix(tempp)
                    return tempp
            elif affix.__eq__("ለት"):
                if tempp.__eq__("አለት") or tempp.__eq__("እለት") or tempp.__eq__("ሁለት"):
                    return tempp
                else:
                    tempp = tempp[ :-2]
                    #change to sades
                    tempp = suffix(tempp);
                    return tempp
            elif affix.__eq__("ት"):
                if tempp.__eq__("አባት") or tempp.__eq__("አያት") or tempp.__eq__("ህይወት") or
tempp.__eq__("አራዊት") or tempp.__eq__("ሰኣት") or tempp.__eq__("መብት") or
tempp.__eq__("መሰረት") or tempp.__eq__("ወቅት") or tempp.__eq__("ጥረት") or
tempp.__eq__("ትርፍ") or tempp.__eq__("ሁለት") or tempp.__eq__("ሶስት") or
tempp.__eq__("አራት") or tempp.__eq__("አምስት") or tempp.__eq__("ስድስት") or
tempp.__eq__("ሰባት") or tempp.__eq__("ስምንት") or tempp.__eq__("ትምህርት") or
tempp.__eq__("ጥቅምት") or tempp.__eq__("ሳምንት") or tempp.__eq__("ሰራዊት"):
                   return tempp
                else:
                    tempp = tempp[ :-1]
                    #change to sades
                    tempp = suffix(tempp);
                    return tempp
            elif affix.__eq__("ነት"):
                if tempp.__eq__("እውነት") or tempp.__eq__("እምነት"):
                    return tempp
                else:
                    tempp = tempp[ :-1]
                    #change to sades
                    tempp = suffix(tempp);
                    return tempp
            elif affix.__eq__("ዎች"):
                if tempp.__eq__("ሰዎች"):
```

```
                str1 = tempp[-1: ]
##                  str2 = tempp[ :len(tempp)-2]
##                  str3 = (u'\u12C9')
##                  tempp = str2 + str3
##                  #change to sades
                return tempp
            else:
                tempp = tempp[ :-2]
                #change to sades
                tempp = suffix(tempp)
                return tempp
        elif affix.__eq__("ኞች"):
            if tempp.__eq__("ዳኞች"):
                tempp = tempp[ :-1]
                ##change to sades
                return tempp
            else:
                tempp = tempp[ :-1]
                #change to sades
                tempp = suffix(tempp)
                return tempp
        elif affix.__eq__("ች"):
                tempp = tempp[ :-1]
                ##change to sades
                tempp = suffix(tempp)
                return tempp
        elif affix.__eq__("ችን"):
                tempp = tempp[ :-2]
                ##change to sades
                tempp = suffix(tempp)
                return tempp
        elif affix.__eq__("ችሁ"):
                tempp = tempp[ :-2]
                ##change to sades
                tempp = suffix(tempp)
                return tempp
        elif affix.__eq__("ቸው"):
                tempp = tempp[ :-2]
                ##change to sades
                tempp = suffix(tempp)
                return tempp
        elif affix.__eq__("ቱ"):
                ##change to sades
##                  tempp = suffix(tempp)
                return tempp
        elif affix.__eq__("ቹ"):
                tempp = tempp[ :-1]
```

```
                ##change to sades
                tempp = suffix(tempp)
                return tempp
            elif affix.__eq__("ዊ"):
                tempp = tempp[ :-1]
                ##change to sades
                tempp = suffix(tempp)
                return tempp
            elif affix.__eq__("ዊት"):
                if tempp.__eq__("አራዊት"):
                    return tempp
                else:
                    tempp = tempp[ :-2]
                    #change to sades
                    tempp = suffix(tempp);
                    return tempp
            elif affix.__eq__("ውያን"):
                tempp = tempp[ :-3]
                ##change to sades
                tempp = suffix(tempp)
                return tempp
            elif affix.__eq__("ቷል"):
                if tempp.__eq__("ሞቷል"):
                    tempp = tempp[ :-1]
                    ##change to sades
                    return tempp
                else:
                    tempp = tempp[ :-2]
                    #change to sades
                    tempp = suffix(tempp)
                    return tempp
            elif affix.__eq__("ም"):
                if tempp.__eq__("ገዳም") or tempp.__eq__("ሰላም") or tempp.__eq__("አለም"):
                    return tempp
                else:
                    tempp = tempp[ :-1]
                    #change to sades
                    tempp = suffix(tempp);
                    return tempp
        else:
            return tempp
    def punctuationremove(str):
        if str.endswith('\n'):
            str=str.rstrip('\n')
        if str.startswith(' '):
            str=str.lstrip(' ')
        p=codecs.open("Punctuation.txt",'r', encoding = 'utf8')
```

```
    Punctuation = p.read()
    for pc in Punctuation:
        if str.endswith(pc):
            str=str.rstrip(pc)
        if str.startswith(pc):
            str=str.lstrip(pc)
    for j in Punctuation:
        for i in range(1,len(str)):
            if j in Punctuation and j in str:
                str=str.strip(j)
    return str
def normal(str):
    sim=codecs.open("similarWord.txt",'r', encoding = 'utf8' )
    similarWord=sim.read()
    li1=[]
    for m in similarWord.split():
        li1.append(m)
    for i in range(1,len(li1),2):
        if str==li1[i] or str==li1[i+1]:
            str=li1[i+1]
    temp=codecs.open("similarChar.txt",'r', encoding = 'utf8' )
    similarChar=temp.read()
    li=[]
    for j in similarChar.split():
        li.append(j)
    for i in range(1,len(li),2):
        str=str.replace(li[i],li[i+1])
    return str
def preffix(word):
    if len(word)<3:
        return word
    else:
        while len(word)>=3:
            tempp=word[ :3]
            p=codecs.open("prefix.txt",'r', encoding = 'utf8' )
            pr = p.read()
            y=pr.split()
            p.close()
            if y.__contains__(tempp):
                if (condition1(tempp,word)):
                    word=condition2(tempp,word)
                    return word
                else:
                    word=word[3: ]
                    preffix(word)
            else:
                tempp=word[ :2]
```

```
            if  y.__contains__(tempp):
                if (condition1(tempp,word)):
                    word=condition2(tempp,word)
                    return word
                else:
                    word=word[2: ]
                    preffix(word)
            else:
                tempp=word[ :1]
                if  y.__contains__(tempp):
                    if (condition1(tempp,word)):
                        word=condition2(tempp,word)
                        return word
                    else:

                        word=word[1: ]
                        preffix(word)

                else:
                    return word
        #print (word + ", ")
    return word
def suffix(word):
    if len(word)<3:
        return word
    else:
        while len(word)>=3:
            tempp=word[len(word)-3: ]
            s=codecs.open("sufix.txt",'r', encoding = 'utf8' )
            su = s.read()
            x=su.split()
            s.close()
            if x.__contains__(tempp):
                if (condition1(tempp,word)):
                    word=condition2(tempp,word)
                    return word
                else:
                    word=word[ :len(word)-3]
                    suffix(word)
            else:
                tempp=word[len(word)-2: ]
                if  x.__contains__(tempp):
                    if (condition1(tempp,word)):
                        word=condition2(tempp,word)
                        return word
                    else:
                        word=word[ :len(word)-2]
```

```
                suffix(word)
            else:
                tempp=word[len(word)-1: ]
                if  x.__contains__(tempp):
                    if (condition1(tempp,word)):
                        word=condition2(tempp,word)
                        return word
                    else:
                        word=word[ :len(word)-1]
                        suffix(word)

                else:
                    return word


    return word
def fileReader(filename):
    indexterms=[]
    infile=codecs.open(filename, encoding='utf8')
    lterms=infile.readlines()
    infile.close()
    for str in lterms:
        if str.endswith('\n'):
            str=str.rstrip('\n')
        indexterms.append(str)
    return indexterms
def docsReader(pst):
    ldocid=[]
    infile=open(pst)
    lterms=infile.readlines()
    infile.close()
    for str in lterms:
        if str.endswith('\n'):
            str=str.rstrip('\n')
        ldocid.append(str)
    return ldocid

# loads stop words from file into memory
def getStopWords():
    stop_words=codecs.open(''stopw.txt'','r',encoding = 'utf8')
    n=stop_words.read()
    s=n.split()
    return s

def indexTerm(ldocid):
    fileterms=[]
    List2 = []
```

```
        List3 = []
        L7 = []
        List4 = []#store terms in each document in sorted order??
        List22 = []#store unique terms from the document
        eng_list=list('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ')
        indexterm=[]
        cff = {}#store terms and its collection frequency in a dictinay
        tff = {}#store terms and its term frequency in a dictinay
        dff = {}#store terms and its document frequency in a dictinay
        posFile=[]#store docid and tf of terms
        docID = 0#store docid
        L9 = []#L9 store term, docid and tf for all document
        L10=[]#L10 stor the docid and tf

        s=getStopWords() # loads stop words to memory
        docs=r'D:\MSC\Thesis\Programming\PR\PIR\corpus2'
        # the loop that reads the id of each document togather
        for docid in ldocid:
            docID = docID + 1
            docs_file_name= docs +'\\'+ 'doc_' + str(docID) + '.txt'
            fone=codecs.open(docs_file_name,'w', encoding = 'utf8')
            List5 = []
            aList1 = []#store the key of dictionay which is unique index terms as a list
            aList2 = []#store the value(tf)
            aList3 = []#store the tf for further purpose
            res = []
            L8 = []#store term,docid and tf
            #fileterms store terms in each documents
            fileterms=fileReader(docid)
            for term in fileterms:
                List1 = []#store every index in every docs terms as a list
                L6 = []#have terms

#---------------Stop word removal and stemming ----------------------------------------------
                for index in term.split():
                    indexterms=[]
                    if index.isalpha() and eng_list.count(index[0])==0:
                        if s.count(index)==0:
                            index=normal(suffix(preffix(punctuationremove(index))))
                            if (len(index))>1:
                                indexterms.append(index)
                                fone.write(index)
                                fone.write(' ')
#-------------------------------------------------------------------------------------------
                    List1 = []
                    for k in indexterms:
                        List1.append(k)
```

```
                List4 = sorted(List4 + List1)
                List22 = list(set(List4))
                List5 = sorted(List5 + List1)

#----------------term frequency count loop-----------------------------------------------------
            for tf in List5:
                try:
                    tff[tf] = tff[tf] + 1
                except:
                    tff[tf] = 1
                tfff = sorted(tff.items())
            for keyValue in range(len(tfff)):
                aList2.append(tfff[keyValue][0])#index terms
                aList1.append(tfff[keyValue][1])#tf of terms
            for kk in aList1:
                aList3.append(kk)#aList3 store tf for futher process purpose
                posFile.append((docID, kk))
            ListDict = dict(zip(aList2,aList3))
            for key in ListDict:
                L6.append(key)#have terms
                L8.append((key, docID, ListDict[key]))#store term,docid and tf for a single doc
            L7 = sorted(L7 + L6)
            tff = {}
            L9 = sorted(list(L9 + L8))#L9 store term, docid and tf for all document
#-------------------------------------------------------------------------------------------------------
            fone.write('\n')
        fone.close()
#---------loop to count the collection frequency of terms-----------------------------------------------
        for cf in List4:
            try:
                cff[cf] = cff[cf] + 1
            except:
                cff[cf] = 1
            cfreq = sorted(cff.items())
        y=[]#store terms by extracting from cf
        for kk2 in range(len(cfreq)):
            y.append(cfreq[kk2][0])
#-------------------------------------------------------------------------------------------------------
#---------loop to count the document frequency of terms-------------------------------------------------
        for df in L7:
            try:
                dff[df] = dff[df] + 1
            except:
                dff[df] = 1
            dfreq2 = sorted(dff.items())
#-------------------------------------------------------------------------------------------------------
        TDFCF = []#store index term df and cf as a single list
```

```
for k1 in range(len(dfreq2) and len(cfreq)):
    TDFCF.append((dfreq2[k1][0], dfreq2[k1][1], cfreq[k1][1]))
L33 = []
ID = 0
L44=[]
f=codecs.open('doc_in_one.txt', 'r', encoding = 'utf8')
ldocid=f.readlines()
for j in ldocid:
    ID = ID + 1
    dic={}
    for term in y:
        print term
        ind_found=0
        if term not in dic:
            found=j.find(term)
            while found >-1:
                if term not in dic:
                    dic[term]=found
                    found=j.find(term, found+1)
                    ind_found=1
                else:
                    dic[term]=((str(dic[term]))+ "    "+(str(found)))
                    found=j.find(term, found+1)
                    ind_found=1
        if ind_found >-1:
            for k in range(len(L9)):
                if term == L9[k][0] and ID == L9[k][1]:
                    L10.append((term, ID, L9[k][2], dic[term]))
                    L33.append((term,ID))
                    L44=sorted(L10)
                    break;
                else: continue
    f.close()

L11=[]#store term, df and cf
for i in range(len(TDFCF)):
    L22 = []
    max_tfidf=0
    #L11.append((normal(TDFCF[i][0]), TDFCF[i][1], TDFCF[i][2]))
    for j in range(len(L10)):
        if TDFCF[i][0] == L10[j][0]:
            sim=float(float(ID)/float(TDFCF[i][1]))
            simm=math.log((sim),10)
            tfidf=float(L10[j][2]*simm)
            if tfidf>max_tfidf:
                max_tfidf=tfidf
            # L22.append(L33[j][0])
```

```
                else: continue
            L11.append((normal(TDFCF[i][0]), TDFCF[i][1], TDFCF[i][2],max_tfidf))
            #L11.append(tuple(L22))
        fiLe=codecs.open("collectionfrequency.txt", 'w', encoding = 'utf8')
        fiLe.write(str("Term")+ "\t" + str("CF") + "\t" + "\n")

        for i in range(len(L11)):
            if L11[i][3]>0.25:
                fiLe.write(L11[i][0]  + "\t" + str(L11[i][2]) + "\t")
                fiLe.write("\n")
            else: continue
        fiLe.close()
        fiLe=codecs.open("vocabularyFile.txt", 'w', encoding = 'utf8')
##          fiLe.write(str("Term") + "\t" + str("DF") + "\t" + str("CF") + "\t" + "\n")
        for i in range(len(L11)):
            if L11[i][3]>0.25:
                fiLe.write(L11[i][0] + "\t" + str(L11[i][1]) + "\t" + str(L11[i][2]) + "\t")
                fiLe.write("\n")
            else: continue
        fiLe.close()
        fiLe=codecs.open("postingFile.txt", 'w', encoding = 'utf8')
        #fiLe.write(str("docID") + "\t" + str("TF") + "\n")
        for i in range(len(L11)):
            for key in range(len(L44)):
                if L44[key][0]==L11[i][0] and  L11[i][3]>0.25:
                    fiLe.write(L44[key][0])
                    fiLe.write("\t")
                    aaa=str(L44[key][1])
                    fiLe.write(aaa)
                    fiLe.write("\t")
                    bbb=str(L44[key][2])
                    fiLe.write(bbb)
                    fiLe.write("\t")
                    ccc=str(L44[key][3])
                    fiLe.write(ccc)
                    fiLe.write('\n')
                else: continue
        fiLe.close()

    def test(indexlist, ldocid):
        totDocs = int(len([fiLe for fiLe in os.listdir("./corpus") if
os.path.isfile(os.path.join("./corpus", fiLe))]))
    return test(indexTerm(docsReader(path)), docsReader(path))
ob=amhariIndexing()
f1=open("documentsID.txt",'w')
for i in os.listdir('./corpus'):
  docs="corpus"
```

```
    docs=docs + '/' + i
    f1.write(docs)
    f1.write("\n")
f1.close()
ob.index("documentsID.txt")
```

B.2 : Searching Python Code

```
from Tkinter import *
import Tkinter as tk
import tkFont
import tkMessageBox as box
import codecs
import os
import math
import string
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer("[\w’]+", flags=re.UNICODE)
from ttk import Frame, Button, Style
from operator import itemgetter, attrgetter


class App(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.initUI()
        self.load_file()
        self.ontology=self.load_ontology()



    def initUI(self):
        self.parent.title("የአማርኛ መረጃ መፈለጊያ ቋት")
        self.style = Style()
```

```
        self.style.theme_use("default")
        self.pack(fill=BOTH, expand=1)
        self.fonts = tkFont.Font(family='Nyala',size=16, weight='bold')
        label = tk.Label(self,
                         anchor="w",fg="Blue",text='እንኳን ወደ አማርኛ የመረጃ መፈለጊያ ቋት በደህና
መጡ!',font=self.fonts)
        label.grid(column=2,row=0 )
        label = tk.Label(self,
                         anchor="w",fg="Blue",text='እባኮን የሚፈልጉትን ፋይል ለማግኘት መጠይቆን
ያስገቡ!',font=self.fonts)
        label.grid(padx=2,pady=2)
        #frame = Frame(self, relief=RAISED, borderwidth=1)
        #frame.pack(fill=BOTH, expand=1)
        self.entry = tk.Entry(self)
        self.entry.grid(padx=10,pady=10 ,columnspan=2,sticky='EW')
        self.var = IntVar()
        self.var3 = IntVar()
        self.label2 = tk.Label(self, text="", textvariable=self.var)
        self.label2.grid(padx=10,pady=0,sticky=W)
        button = tk.Button(self,text=u"ፈልግ" ,font='Nyala',command=self.readIndexData)
        button.grid(column=1,row=2)
        self.yScroll = tk.Scrollbar(self, orient=tk.VERTICAL)
        self.yScroll.grid(row=4, column=1, sticky=tk.N+tk.S)
        self.lb = Listbox(self, width=100, height=15,yscrollcommand=self.yScroll.set )
        self.lb.bind("<<ListboxSelect>>", self.onSelect)
        self.lb.grid(row=4, column=0, sticky=tk.N+tk.S+tk.E+tk.W,)
        self.yScroll['command'] = self.lb.yview
        self.label3 = tk.Label(self, text="", textvariable=self.var3)
        self.label3.grid(padx=5,pady=0,sticky=W)
        self.yyScroll = tk.Scrollbar(self, orient=tk.VERTICAL)
        self.yyScroll.grid(row=6, column=1, sticky=tk.N+tk.S)
```

```
        self.lbb = Listbox(self, width=100, height=18,yscrollcommand=self.yyScroll.set )
        self.lbb.bind("<<ListboxSelect>>", self.onSelect)
        self.lbb.grid(row=6, column=0, sticky=tk.N+tk.S+tk.E+tk.W)
        self.yyScroll['command'] = self.lbb.yview

        self.var2 = IntVar()
        self.label1 = Label(self, text="", textvariable=self.var2, wraplength=600,justify=LEFT)
        self.label1.grid(column=2,row=4,columnspan=1,rowspan=1, sticky=W)

    def onSelect(self, val):
        sender = val.widget
        idx = sender.curselection()
        value = sender.get(idx)
        value2=value[1]
        a=codecs.open(value2, 'r', encoding = 'utf-8')
        #for i in a:
        b=a.read()
        self.var2.set(b)
    def load_file(self):
        flag=0
        self.L1=[]
        L2,L01,L02=[],[],[]
        DocL=[]
        DocN=1
        DocL2=[]

        self.N = int(len([fiLe for fiLe in os.listdir("./corpus") if os.path.isfile(os.path.join("./corpus",
fiLe))]))
        a=codecs.open("vocabularyFile.txt", 'r', encoding = 'utf-8')
        b=a.readlines()
        for j in b:
```

```
    c=j.split()
    f=c[0],int(c[1])
    L01.append(f)
for i in L01:
    self.L1.append((i[0],i[1]))
a.close()
n=codecs.open("postingFile.txt", 'r', encoding = 'utf-8')
m=n.readlines()
for i in m:
    s=i.split()
    f2=int(s[1]),int(s[2])
    L02.append(f2)
for i in L02:
    L2.append((i[0],i[1]))
n.close()
L3,L6=[],[]
self.vector=[]
d,h,g,a,m,c=0,0,0,0,0,0
while c < (len(self.L1)):
        L=[]
        L30=[]
        for h in range(self.L1[d][1]):
            L.append((self.L1[m][0], L2[a][0],L2[a][1],a))
            a = a + 1
        m = m + 1
        d = d + 1
        c=c+1
        self.vector = self.vector + L
f1=open("documentsID.txt", 'r')
f2=f1.readlines()
for i in f2:
```

```
        record=i.split('\n')
        DocL.append(record)
        DocL2.append((DocN,DocL[DocN-1][0]))
        DocN=DocN+1
    self.docDict=dict(DocL2)

def condition1(self,affix, tempp):
        if len(tempp)> 2:
            if affix.__eq__("ከ"):
                 return 1
            elif affix.__eq__("በ"):
                 return 1
            elif affix.__eq__("ኞች"):
                 return 1
            elif affix.__eq__("ን"):
                 return 1
            elif affix.__eq__("ት"):
                 return 1
            elif affix.__eq__("ነት"):
                 return 1
            elif affix.__eq__("ዎች"):
                 return 1
            elif affix.__eq__("ች"):
                 return 1
            elif affix.__eq__("ችን"):
                 return 1
            elif affix.__eq__("ችሁ"):
                 return 1
            elif affix.__eq__("ቸው"):
                 return 1
            elif affix.__eq__("ቱ"):
```

```
                return 1
            elif affix.__eq__("ቹ"):
                return 1
            elif affix.__eq__("ዊ"):
                return 1
            elif affix.__eq__("ዊት"):
                return 1
            elif affix.__eq__("ውያን"):
                return 1
            elif affix.__eq__("ቷል"):
                return 1
            elif affix.__eq__("ም"):
                return 1
            elif affix.__eq__("ት") or affix.__eq__("አ"):
                return 1
            elif affix.__eq__("ኛ"):
                return 1
            elif affix.__eq__("ነት") or affix.__eq__("እ"):
                return 1
            elif affix.__eq__("ም") or affix.__eq__("እ"):
                return 1
            elif affix.__eq__("ለት"):
                return 1
        else:
            return 0

    def condition2(self,affix,tempp):
        if len(tempp) > 2:
            if affix.__eq__("ከ"):
                if tempp.__eq__("ከብት") or tempp.__eq__("ከለከለ") or tempp.__eq__("ከነከነ") or
tempp.__eq__("ከረከረ") or tempp.__eq__("ከመከመ") or tempp.__eq__("ከሰከሰ") or
```

```
tempp.__eq__("ከተከተ") or tempp.__eq__("ከተማ") or tempp.__eq__("ከረን") or
tempp.__eq__("ከሰል") or tempp.__eq__("ከፍተኛ") or tempp.__eq__("ከብት"):
                return tempp
            else:
                tempp = tempp[1: ]
                tempp = self.preffix(tempp);
                return tempp
        elif affix.__eq__("በ"):
            if tempp.__eq__("በሽታ") or tempp.__eq__("በጀት") or tempp.__eq__("በረበረ") or
tempp.__eq__("በሰበሰ"):
                return tempp
            else:
                tempp = tempp[1: ]
                tempp = self.preffix(tempp);
                return tempp
        elif affix.__eq__("ኛ"):
            if tempp.__eq__("በሽታ"):
                return tempp
            else:
                tempp = tempp[ :-1]
                tempp = self.preffix(tempp);
                return tempp
        elif affix.__eq__("ን"):
            if tempp.__eq__("ዘመን") or tempp.__eq__("ቡድን") or tempp.__eq__("ሸፋን") or
tempp.__eq__("ወገን") or tempp.__eq__("ቡድን") or tempp.__eq__("ሸፋን") or
tempp.__eq__("ወገን") or tempp.__eq__("ጭቁን") or tempp.__eq__("ህጻን") or
tempp.__eq__("እውን") or tempp.__eq__("እምን") or tempp.__eq__("ዘፈን"):
                return tempp
            else:
                tempp = tempp[ :-1]
##                  tempp=sadisConverter(tempp)
```

```
            tempp = self.suffix(tempp)
            return tempp
        elif affix.__eq__("ለት"):
            if tempp.__eq__("አለት") or tempp.__eq__("እለት") or tempp.__eq__("ሁለት"):
                return tempp
            else:
                tempp = tempp[ :-2]
                #change to sades
                tempp = self.suffix(tempp);
                return tempp
        elif affix.__eq__("ት"):
            if tempp.__eq__("አባት") or tempp.__eq__("አያት") or tempp.__eq__("ህይወት") or
tempp.__eq__("አራዊት") or tempp.__eq__("ሰኣት") or tempp.__eq__("መብት") or
tempp.__eq__("መሰረት") or tempp.__eq__("ወቅት") or tempp.__eq__("ጥረት") or
tempp.__eq__("ትርፍ") or tempp.__eq__("ሁለት") or tempp.__eq__("ሶስት") or
tempp.__eq__("አራት") or tempp.__eq__("አምስት") or tempp.__eq__("ስድስት") or
tempp.__eq__("ሰባት") or tempp.__eq__("ስምንት") or tempp.__eq__("ትምህርት") or
tempp.__eq__("ጥቅምት") or tempp.__eq__("ሳምንት") or tempp.__eq__("ሰራዊት"):
               return tempp
            else:
                tempp = tempp[ :-1]
                #change to sades
                tempp = self.suffix(tempp);
                return tempp
        elif affix.__eq__("ነት"):
            if tempp.__eq__("እውነት") or tempp.__eq__("እምነት"):
                return tempp
            else:
                tempp = tempp[ :-1]
                #change to sades
                tempp = self.suffix(tempp);
```

```
        return tempp
    elif affix.__eq__("ዎች"):
        if tempp.__eq__("ሰዎች"):
            str1 = tempp[-1: ]
            return tempp
        else:
            tempp = tempp[ :-2]
            #change to sades
            tempp = self.suffix(tempp)
            return tempp
    elif affix.__eq__("ኞች"):
        if tempp.__eq__("ዳኞች"):
            tempp = tempp[ :-1]
            ##change to sades
            return tempp
        else:
            tempp = tempp[ :-1]
            #change to sades
            tempp = self.suffix(tempp)
            return tempp
    elif affix.__eq__("ች"):
            tempp = tempp[ :-1]
            ##change to sades
            tempp = self.suffix(tempp)
            return tempp
    elif affix.__eq__("ችን"):
            tempp = tempp[ :-2]
            ##change to sades
            tempp = self.suffix(tempp)
            return tempp
    elif affix.__eq__("ችሁ"):
```

```
            tempp = tempp[ :-2]
            ##change to sades
            tempp = self.suffix(tempp)
            return tempp
        elif affix.__eq__("ቸው"):
            tempp = tempp[ :-2]
            ##change to sades
            tempp = self.suffix(tempp)
            return tempp
        elif affix.__eq__("ቱ"):
            ##change to sades
##                  tempp = suffix(tempp)
            return tempp
        elif affix.__eq__("ቹ"):
            tempp = tempp[ :-1]
            ##change to sades
            tempp = self.suffix(tempp)
            return tempp
        elif affix.__eq__("ዊ"):
            tempp = tempp[ :-1]
            ##change to sades
            tempp = self.suffix(tempp)
            return tempp
        elif affix.__eq__("ዊት"):
           if tempp.__eq__("አራዊት"):
            return tempp
           else:
            tempp = tempp[ :-2]
            #change to sades
            tempp = self.suffix(tempp);
            return tempp
```

```
        elif affix.__eq__("ውያን"):
                tempp = tempp[ :-3]
                ##change to sades
                tempp = self.suffix(tempp)
                return tempp
        elif affix.__eq__("ቷል"):
            if tempp.__eq__("ሞቷል"):
                tempp = tempp[ :-1]
                ##change to sades
                return tempp
            else:
                tempp = tempp[ :-2]
                #change to sades
                tempp = self.suffix(tempp)
                return tempp
        elif affix.__eq__("ም"):
            if tempp.__eq__("ገዳም") or tempp.__eq__("ሰላም") or tempp.__eq__("አለም"):
                return tempp
            else:
                tempp = tempp[ :-1]
                #change to sades
                tempp = self.suffix(tempp);
                return tempp
    else:
        return tempp

def punctuationremove(self,str):
    if str.endswith('\n'):
        str=str.rstrip('\n')
    if str.startswith(' '):
        str=str.lstrip(' ')
```

```
        p=codecs.open("Punctuation.txt",'r', encoding = 'utf8')
        Punctuation = p.read()
        for pc in Punctuation:
            if str.endswith(pc):
                str=str.rstrip(pc)
            if str.startswith(pc):
                str=str.lstrip(pc)
        for j in Punctuation:
            for i in range(1,len(str)):
                if j in Punctuation and j in str:
                    str=str.strip(j)
        return str
    def normal(self,str):
        sim=codecs.open("similarWord.txt",'r', encoding = 'utf8' )
        similarWord=sim.read()
        li1=[]
        for m in similarWord.split():
            li1.append(m)
        for i in range(1,len(li1),2):
            if str==li1[i] or str==li1[i+1]:
                str=li1[i+1]
        temp=codecs.open("similarChar.txt",'r', encoding = 'utf8' )
        similarChar=temp.read()
        li=[]
        for j in similarChar.split():
            li.append(j)
        for i in range(1,len(li),2):
            str=str.replace(li[i],li[i+1])
        return str
    def preffix(self,word):
        if len(word)<3:
```

```
    return word
else:
    while len(word)>=3:
        tempp=word[ :3]
        p=codecs.open("prefix.txt",'r', encoding = 'utf8' )
        pr = p.read()
        y=pr.split()
        p.close()
        if y.__contains__(tempp):
            if (self.condition1(tempp,word)):
                word=self.condition2(tempp,word)
                return word
            else:
                word=word[3: ]
                self.preffix(word)
        else:
            tempp=word[ :2]
            if  y.__contains__(tempp):
                if (self.condition1(tempp,word)):
                    word=self.condition2(tempp,word)
                    return word
                else:
                    word=word[2: ]
                    self.preffix(word)
            else:
                tempp=word[ :1]
                if  y.__contains__(tempp):
                    if (self.condition1(tempp,word)):
                        word=self.condition2(tempp,word)
                        return word
                    else:
```

```
                    word=word[1: ]
                    self.preffix(word)

                else:
                    return word
        return word
    def suffix(self,word):
        if len(word)<3:
            return word
        else:
            while len(word)>=3:
                tempp=word[len(word)-3: ]
                s=codecs.open("sufix.txt",'r', encoding = 'utf8' )
                su = s.read()
                x=su.split()
                s.close()
                if x.__contains__(tempp):
                    if (self.condition1(tempp,word)):
                        word=self.condition2(tempp,word)
                        return word
                    else:
                        word=word[ :len(word)-3]
                        self.suffix(word)
                else:
                    tempp=word[len(word)-2: ]
                    if  x.__contains__(tempp):
                        if (self.condition1(tempp,word)):
                            word=self.condition2(tempp,word)
                            return word
                        else:
                            word=word[ :len(word)-2]
```

```
                self.suffix(word)
            else:
                tempp=word[len(word)-1: ]
                if  x.__contains__(tempp):
                    if (self.condition1(tempp,word)):
                        word=self.condition2(tempp,word)
                        return word
                    else:
                        word=word[ :len(word)-1]
                        self.suffix(word)

                else:
                    return word
    return word
def load_ontology(self):
    fread=codecs.open('ontology.txt','r+',encoding="utf8")
    ontology={}
    j=0
    for line in fread:
        line_num='line'+ str(j)
        ontology[line_num]={'superclass': [], 'subclasss1':[] ,
              'subclasss2': [],'subclasss3': [],'individual': []}

        line=line.split('@')
        term=[]
        for i in line[0].split():
            term.append(self.normal(self.suffix(self.preffix(self.punctuationremove(i)))))
            ontology[line_num]['superclass']=term
        term=[]
        for i in line[1].split():
            term.append(self.normal(self.suffix(self.preffix(self.punctuationremove(i)))))
```

```
            ontology[line_num]['subclasss1']=term
        term=[]
        for i in line[2].split():
            term.append(self.normal(self.suffix(self.preffix(self.punctuationremove(i)))))
            ontology[line_num]['subclasss2']=term
        term=[]
        for i in line[3].split():
            term.append(self.normal(self.suffix(self.preffix(self.punctuationremove(i)))))
            ontology[line_num]['subclasss3']=term
        line=line[4].split('#')
        term=[]
        for i in line:
            for y in i.split():
                term.append(self.normal(self.suffix(self.preffix(self.punctuationremove(y)))))
                ontology[line_num]['individual']=term
        j=j+1
    fread.close()
    return ontology
def getStopWords(self):
    stop_words=codecs.open("stopw.txt",'r',encoding = 'utf8')
    n=stop_words.read()
    s=n.split()
    return s
def readIndexData(self):
    eng_list=list('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ')
    flag=0
    quer_doc=[]
    vector=self.vector
    querylist=[]
    queryterms=[]
    pro_vector=[]
```

```
UserQuery=""
UserQuery=self.entry.get()# rececive uder input query here
userInput=tokenizer.tokenize(UserQuery)#UserQuery.split()
s=self.getStopWords()
for query in userInput:
    if query.isalpha() and eng_list.count(query[0])==0:
        if s.count(query)==0:
            x=self.normal(self.suffix(self.preffix(self.punctuationremove(query))))
            if len(x)>1:
                queryterms.append(x)
ontology=self.ontology
querylist=queryterms
org_query= querylist
exp_query=[]
for quer in org_query:
    for i in ontology.keys():
        if (ontology[i]['superclass']==[quer]):
            print query
        elif(ontology[i]['subclasss1']==[quer]):
            for v in ontology[i]['subclasss2']:
                    exp_query.append(v)
            if(ontology[i]['subclasss3']<>u''):
                for v in ontology[i]['subclasss3']:
                    exp_query.append(v)
                for j in ontology[i]['individual']:
                    exp_query.append(j)
            else:
                for t in ontology[i]['individual']:
                    exp_query.append(t)
        elif(ontology[i]['subclasss2']==[quer]):
            if(ontology[i]['subclasss3']<>u''):
```

```
                for v in ontology[i]['subclasss3']:
                    exp_query.append(v)
                for f in ontology[i]['individual']:
                    exp_query.append(f)
            else:
                for f in ontology[i]['individual']:
                    exp_query.append(f)
        elif(ontology[i]['subclasss3']==[quer]):
            for v in org_query:
                exp_query.append(v)
            for k in ontology[i]['individual']:
                exp_query.append(k)
        elif(1):
            for u in ontology[i]['individual']:
                if u==quer:
                    for v in org_query:
                        exp_query.append(v)
                    for c in ontology[i]['individual']:
                        exp_query.append(c)
                    break
        out=codecs.open("output.txt", 'w', encoding = 'utf-8')
for o in(range (0,2)):
  # quer_doc=[]
   quer_doc_per_query=[]
   pro_vector=[]
   provector=[]
   count=0
   rank=[]
   rank_count=1
   forweight=[]
   if o==1:
```

```
            querylist=[]
            for i in exp_query:
                if i not in querylist and len(i)>1:
                    querylist.append(i)
        for n in range(len(querylist)):
            found=0
            quer_doc=[]

            for i in range(len(self.L1)):
                if querylist[n]==self.L1[i][0]:
                    quer_doc.append((querylist[n],self.L1[i][1]))

            for t in range(len(vector)):
                if querylist[n]==vector[t][0]:
                    for j in range(quer_doc[0][1]):
                        provector.append((vector[t+j][0],vector[t+j][1],vector[t+j][2]))
#quer_doc[j][1],1))
                        quer_doc_per_query.append((vector[t+j][0],vector[t+j][1],vector[t+j][2]))
                    forweight.append((querylist[n],quer_doc[0][1]))
                    rank.append((querylist[n],rank_count))
                    rank_count=rank_count+1
                    pro_vector.append(quer_doc[0][1])
                #if found==(len(quer_doc_per_query)):
                    break


        weight=[]
        for i in range(len(forweight)):
            sim1 = float(((self.N)-(forweight[i][1])) + 0.5)
            sim2=float((forweight[i][1]) + 0.5)
            sim=float(sim1/sim2)
```

```
            simm=math.log((sim),10)
            weight.append(simm)
        #count=1
        # rank=[]
        #count=1
        score=[]
        score2=[]
        for i in range(len(forweight)):
            l_list=[]
            for k in range(len(score2)):
                l_list.append(score2[k][0])
            for j in range(len(provector)):
                if forweight[i][0]==provector[j][0]:
                    for m in range(pro_vector[i]):
                        if l_list.count(quer_doc_per_query[j+m][1])==0:
                            score.append((quer_doc_per_query[j+m][1],rank[i][1]))
                            score2=sorted(score)
                    break
        add=0
        scoree=[]

        for k in range(len(weight)):
            for j in range(len(quer_doc_per_query)):
                for i in range(len(score2)):
                    if querylist[k]==quer_doc_per_query[j][0] and
score2[i][0]==quer_doc_per_query[j][1] and score2[i][1]==(k+1):
                        add=(weight[(score2[i][1])-1])
                        scoree.append((score2[i][0],add,quer_doc_per_query[j][2]))
                add=0

        if len(userInput)>0 and len(userInput)<10:
```

```
if len(userInput)>=1 and len(userInput)<10:
    #finalRes={}
    #finalRes=dict(scoree)
    Ranking=[]
    Ranking1=[]
    #Ranking=finalRes.keys()
    pathList=[]
    chekpoint=0

    Ranking1=sorted(scoree, key=itemgetter(1,2), reverse=True)
    for i in range(len(Ranking1)):
        Ranking.append(Ranking1[i][0])
    for i in range(0,len(Ranking)):
        if Ranking1[i][1] !=0:
            chekpoint=1
    if chekpoint==1:
        if o==0:
            v ="ባስገቡት መጠይቅ መሰረት የተገኙት መረጃዎች እንደሚከተለው ቀርበዋል! "
            self.var.set(v)
            self.lb.delete(0,(self.lb.size()-1))
            rank_doc=[]
            str1,str2,str3="","",""
            for i in range(len(Ranking)):
                if Ranking1[i][1]>0:
                    pathList.append(self.docDict[Ranking[i]])
                    j=i+1
                    str1=str(j )+', '
                    str2= self.docDict[Ranking[i]]
                    str3='          ' + str(Ranking1[i][1])+ '          ' + str(Ranking1[i][2])
                    rank_doc=(str1, str2,str3)
                    self.lb.insert(END,rank_doc)
```

```
                    out.write(self.docDict[Ranking[i]])
                    out.write('\n')
                else:
                    v ="ባስገቡት መጠይቅ እና በተጨማሪ የመፈለጊያ ቃላት መሰረት የተገኙት መረጃዎች
እንደሚከተለው  ቀርበዋል! "
                    self.var3.set(v)
                    self.lbb.delete(0,(self.lbb.size()-1))
                    rank_doc=[]
                    str1=""
                    for i in range(len(Ranking)):
                        if  Ranking1[i][1]>0:
                            pathList.append(self.docDict[Ranking[i]])
                            j=i+1
                            str1=str(j )+', '
                            str2= self.docDict[Ranking[i]]
                            str3='          ' + str(Ranking1[i][1])+ '          ' + str(Ranking1[i][2])
                            rank_doc=(str1, str2,str3)
                            self.lbb.insert(END,rank_doc)
                            out.writelines(self.docDict[Ranking[i]])
                            out.write('\n')
            else:
                if o==0:
                    v = "ይቅርታ! ከጠየቁት መጠይቅ ጋር የሚመሳሰል መረጃ አልተገኘም።"
                    self.var.set(v)
                else:
                    v = "ያስገቡት መጠይቅ ተጨማሪ የመፈለጊያ ቃላት አልተገኘለት!"
                    self.var3.set(v)


        else:
            v="ምንም ዓይነት መጠይቅ አላስገቡም፤ ወይም መጠይቆ ከሚፈቀደዉ መጠን በላይ ነዉ! እባክዎ መጠይቆን
እንደገና ያስገቡት! "
```

```
            self.var.set(v)
        out.close()
# create the application

def main():

    root = Tk()
    scrollbar = Scrollbar(root)
    scrollbar.pack( side = RIGHT, fill=Y )
    root.geometry("1360x690+0+0")
    app =  App(root)
    root.mainloop()

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

B.3 : Single word Extraction Python Code

```
#-*- coding: utf-8 -*-
import sys
sys.path.append('C:\Python27\Lib\site-packages')
import re
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk import bigrams, trigrams
import math
import codecs
from nltk.corpus import PlaintextCorpusReader

tokenizer = RegexpTokenizer("[\w']+", flags=re.UNICODE)

def stopword():
    fread = codecs.open(r'D:\MSC\Thesis\Programming\Stopword\Amharic','r+',encoding="utf8")
    stopword=[]
    for line in fread:
        stopword.extend(line.split())
```

```
    fread.close()
    return stopword

def freq(word, doc):
    return doc.count(word)


def word_count(doc):
    return len(doc)


def tf(word, doc):
    return (freq(word, doc) / float(word_count(doc)))


def num_docs_containing(word, list_of_docs):
    count = 0
    for document in list_of_docs.keys():
        l_list=list_of_docs[document]['token']
        if freq(word, l_list) > 0:
            count += 1
    return 1+ count


def idf(word, list_of_docs):
    return math.log(len(list_of_docs) /
            float(num_docs_containing(word, list_of_docs)))


def tf_idf(word, doc, list_of_docs):
    ll_doc_word=doc['token']
    return (tf(word, ll_doc_word) * idf(word, list_of_docs))

def tpcw(word,list_of_bi_word_docs,list_of_docs):
    ll_word=tokenizer.tokenize(word)
    return float(2* (freq(word,list_of_bi_word_docs)))  /  float(freq(ll_word[0],list_of_docs) +
freq(ll_word[1],list_of_docs))

#Compute the frequency for each term.
vocabulary = []
bi_vocabulary = []
dict_of_word_doc={}
bi_dict_of_word_doc={}
docs = {}
bi_docs = {}
all_tips = []
list_of_word_doc=[]
```

```
bi_list_of_word_doc=[]
corpus_path=r'D:\MSC\Thesis\Programming\corpus'
file_list_path=PlaintextCorpusReader(corpus_path,'.*')
file_path=file_list_path.fileids()
stopwords=stopword()
j=0
for i in file_path:
    file_name=corpus_path +'\\'+ i
    f=codecs.open(file_name,encoding='utf-8')
    dict_of_word_doc[file_name]={'token':[]}
    bi_dict_of_word_doc[file_name]={'token':[]}
    for line in f:
        file_tokens= tokenizer.tokenize(line)
        list_of_word_doc.extend(file_tokens)
        bi_list_of_word_doc=bigrams(file_tokens)
    dict_of_word_doc[file_name]['token']=list_of_word_doc
    bi_dict_of_word_doc[file_name]['token']=bi_list_of_word_doc
    list_of_word_doc=[]
    bi_list_of_word_doc=[]
    f.close()

for tip1 in dict_of_word_doc.keys():
    tokens =dict_of_word_doc[tip1]['token']
##    stem_token=[]
##    for i in tokens:
##        stem_token.append(suffix((punctuationremove(i))))
##    tokens=stem_token
##    tokens = [token.lower() for token in tokens if len(token) > 2]
    tokens = [token for token in tokens if token not in stopwords]
    # single term words
    final_tokens = []
    final_tokens.extend(tokens)

    # double term words
    bi_tokens=bigrams(tokens)
    bi_final_token=[]
    bi_tokens = [' '.join(token).lower() for token in bi_tokens]
    bi_tokens = [token for token in bi_tokens if token not in stopwords]
    bi_final_token.extend(bi_tokens)

    docs[tip1] = {'freq': {}, 'tf': {}, 'idf': {},
                  'tf-idf': {}, 'tokens': []}
    bi_docs[tip1] = {'freq': {}, 'tf': {},
                  'tpcw': {}, 'tokens': []}
    for token in final_tokens:
        #The frequency of each term in a document
        docs[tip1]['freq'][token] = freq(token, final_tokens)
```

```
        #The term-frequency (Normalized Frequency)
        docs[tip1]['tf'][token] = tf(token, final_tokens)
        docs[tip1]['tokens'] = final_tokens

    for token in bi_final_token:
        #The frequency of each term in a document
        bi_docs[tip1]['freq'][token] = freq(token, bi_final_token)
        #The term-frequency (Normalized Frequency)
        bi_docs[tip1]['tf'][token] = tf(token, bi_final_token)
        bi_docs[tip1]['tokens'] = bi_final_token
        bi_docs[tip1]['tpcw'][token] = tpcw(token, bi_final_token, final_tokens)

for d1 in docs.keys():
    for token in docs[d1]['tf']:
        #The Inverse-Document-Frequency
        docs[d1]['idf'][token] = idf(token, dict_of_word_doc)
        #The tf-idf
        docs[d1]['tf-idf'][token] = tf_idf(token, dict_of_word_doc[d1], dict_of_word_doc)


words = {}
bi_words={}
fout = codecs.open(r'D:\MSC\Thesis\Programming\Outputs\tf_idf.txt','w+',encoding="utf8")
for doc in docs:
    for token in docs[doc]['tf-idf']:
        if token not in words:
            words[token] = docs[doc]['tf-idf'][token]
        else:
            if docs[doc]['tf-idf'][token] > words[token]:
                words[token] = docs[doc]['tf-idf'][token]
    vocabulary.extend(words)

for doc in bi_docs:
    for token in bi_docs[doc]['tpcw']:
        if token not in bi_words:
            if  bi_docs[doc]['tpcw'][token]>1:
                bi_words[token] = bi_docs[doc]['tpcw'][token]
        else:
            if bi_docs[doc]['tpcw'][token] > bi_words[token]:
                if  bi_docs[doc]['tpcw'][token]>1:
                    bi_words[token] = bi_docs[doc]['tpcw'][token]

    bi_vocabulary.extend(bi_words)

for item in sorted(bi_words.items(), key=lambda x: x[1], reverse=True):
    fout.write( str(item[1]) +'\t'+ item[0] +'\t'+ '\n')
fout.close()
```

Declaration

I declare that the thesis is my original work and has not been presented for a degree in any other university.

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

_____

Dereje Teferi (Ph.D)