****

**ADDIS ABABA UNIVERSITY COLLEGE OF NATURAL AND COMPUTATIONAL SCIENCE SCHOOL OF INFORMATION SCIENCE**

**ISR Group Assignment**

**Group Member ID**

**Frezer Gebeyaw UGR/7381/15**

**Abigiya getachew UGR/0128/15**

**Yekoye Admassue UGR/3202/13**

**Mariamawit tewodros UGR/7952/15**

**Submitted to: Dr. Martha Y.**

**Assignment One**

**Introduction and Overview:**

For this assignment, we selected Amharic as our language of focus and we used ኤክሶድያ fiction book and have the word about 21926 . Our methodology involved tokenizing the text, calculating word frequencies and ranks, plotting graphs, and employing Python with NLTK for tokenization and Matlab library for graph plotting to demonstrate frequency distributions. Our analysis aimed to explore Zipf’s distribution and Luhn’s idea.

**Challenges**

The availability of reliable tools and libraries designed for certain languages is often crucial for the study of linguistic data. One major challenge we faced while researching Amharic text for our assignment was the noticeable lack of Amharic support in the Python packages that were already available. When we began to use the resources at our disposal to examine texts written in Amharic, this proved to be a very difficult task. Our normal array of Python-based natural language processing tools and libraries was unable to handle the complexities of Amharic text due to the lack of support. This constraint was a significant challenge, demanding the development of substitute tactics and modifications to our procedures in order to overcome it

**how the challenges were solved**

To overcome the significant obstacle of the Amharic language not being supported by any of the current Python libraries, we utilized a combination of manual implementation and strategic use of existing resources. Utilizing built-in features and utilizing Python libraries like Matlabplot, we developed a customized method to meet the unique requirements of Amharic text analysis. This required us to use relevant Python modules to support our efforts in addition to manually implementing some operations. By using this hybrid technique, we successfully overcame the obstacle and advanced our research by filling in the gap left by the lack of direct support for the Amharic language.

**Different Methodologies and Process Descriptions:**

Our process commenced with the collection of raw Amharic text from various sources. Subsequently, we preprocessed the collected text to eliminate irrelevant content. Tokenization, facilitated by the Python NLTK Library, segmented the text into individual words. Statistical analysis encompassed calculating word frequencies through counting algorithms, ranking words by frequency, and utilizing the Matlab plotting library to depict frequency versus rank for observing Zipf's Law. To further analyze the distribution of word occurrences, we calculated the product of rank and frequency.

**Statistics Analysis:**

Frequency Distribution: The analysis yielded calculated frequencies for each word within the text.

Ranked Words: We compiled a list of Amharic words ranked according to their frequency within our analysis.

Frequency vs. Rank Plot: Utilizing Matlab Plot, we generated a graph illustrating the relationship between word frequency and its respective rank.

Product of Rank and Frequency: Employing Python functions, we computed the product of rank and frequency for each word.

Distribution Explanation: Through the output generated by our code, we observed and interpreted Zipf's law and Luhn’s idea, providing insights into the distribution characteristics and indexing considerations.

**Assignment two**

Introduction:

Our objective is to choose a variety of Amharic writings. First, we standardize the text by standardizing its structure or format, or normalization. After that, the text is tokenized, or divided into individual words or tokens. After tokenization, we remove Amharic-specific stop words, which are terms that are frequently used but have little or no meaning. In order to improve text processing and analysis, we apply stemming to the words, reducing them to their root or base form.

Challenges:

When attempting to employ various text operations such as stemming and tokenization, we encountered the challenge that Amharic is not supported by several libraries. Consequently, we had to devise alternative Python code to implement these text operations effectively.

Special cases we encountered while implementing

**Limited Language Support**: One major challenge was the lack of support for Amharic language processing in existing libraries. This necessitated custom solutions tailored specifically for Amharic text operations.

**Unique Stop Words**: Amharic language has its own set of stop words, which are different from those in other languages. Identifying and removing these stop words required careful consideration.

**Complex Morphology**: Amharic has a complex morphology with various affixes and morphological patterns. This complexity posed challenges during stemming, as existing algorithms designed for other languages were not suitable for Amharic.

**Resource Availability**: Availability of linguistic resources for Amharic was limited compared to more widely spoken languages. This scarcity impacted the accuracy and effectiveness of text processing tasks.

# 

# 

# **Assignment Three: Inverted Index for Amharic Text**

# **Introduction**

# The purpose of this project is to develop a system that creates an inverted index for a collection of Amharic text documents. An inverted index is a data structure used in information retrieval systems to map content to its locations within a set of documents. This system is particularly useful for search engines, allowing for efficient full-text searches.

## **Methodology**

### **Libraries and Resources**

The code uses the following libraries and resources:

* json: To handle JSON operations for reading and writing files.
* stop\_word.txt: A file containing a list of stop words to be excluded from the index.
* doc1.txt, doc2.txt, doc3.txt: Sample text documents used to demonstrate the inverted index creation.

### 

### 

### 

### 

### **Step-by-Step Process**

#### **1. Define Stop Words**

A list of stop words is defined and read from a file named stop\_word.txt. These words are excluded from the index to improve search efficiency and relevance.

#### **2. Define Prefix and Suffix Patterns**

Prefix and suffix patterns relevant to the Amharic language are defined. These patterns are used in the stemming process to normalize tokens by removing common prefixes and suffixes.

#### **3. Stemming Function**

A custom stemming function is implemented to remove defined prefixes and suffixes from tokens. This function also performs additional character replacements to further normalize the tokens. The replacements are based on common variations in Amharic words.

#### **4. Create Inverted Index**

The create\_inverted\_index function processes a list of documents to create the inverted index:

* Each document is split into tokens.
* Tokens are filtered to remove stop words.
* Remaining tokens are stemmed using the stemming function.
* The inverted index is updated with each token, recording the document ID and frequency of the token in each document.

oc\_id

# **Assignment Four: Document Similarity Calculation Using Cosine Similarity for Amharic Text Documents.**

## **Introduction**

In this assignment, we explore a process and methodology to calculate the similarity between a set of Amharic text documents and a query using cosine similarity. The objective is to create document vectors based on term frequency, vectorize a query, and compute the cosine similarity to rank the documents by their relevance to the query.

## **Methodology**

### **Libraries and Resources**

The code utilizes the following libraries and resources:

* collections: For creating default dictionaries.
* math: For mathematical operations.
* stop\_word.txt: A file containing a list of stop words to exclude from the analysis.
* doc1.txt, doc2.txt, doc3.txt: Sample text documents for the analysis.
* query.txt: A sample query to compare against the documents.

### **Step-by-Step Process**

#### **1. Define Stop Words**

Stop words are read from stop\_word.txt and stored in a set for efficient exclusion from token processing. These words are common and do not contribute significantly to the content of the documents.

#### 

#### **2. Define Prefix and Suffix Patterns**

Prefix and suffix patterns specific to the Amharic language are defined to normalize tokens during the stemming process.

#### **3. Stemming Function**

The stemming function is implemented to remove prefixes and suffixes from tokens and perform specific character replacements to normalize the tokens. This process helps in reducing the variations of words to their root forms, enhancing the effectiveness of the term frequency calculations.

#### **4. Compute Document Vectors**

The compute\_document\_vectors function processes each document to create a vector based on term frequency:

* Each document is split into tokens.
* Tokens are filtered to remove stop words.
* Remaining tokens are stemmed.
* Term frequencies are calculated for the stemmed tokens, creating a document vector.

#### **5. Compute Cosine Similarity**

The compute\_cosine\_similarity function calculates the cosine similarity between two vectors. The formula for cosine similarity is given by:

#### **6. Vectorize Query**

The vectorize\_query function processes the query similarly to the documents, creating a term frequency vector for the query.

#### **7. Rank Documents by Similarity**

Similarities between the query vector and each document vector are calculated and sorted in descending order to rank the documents by their relevance to the query.