School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Arabic Stemmer

Bashaer Mohammed Almashy

Submitted in accordance with the requirements for the degree of
MSc in Advanced Computer Science (Artificial Intelligence)

2020/2021

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Deliverables 1	*Report*	*SSO (12/11/21)*
Deliverables 2	*URL:* *https://github.com/m-bashayer/Arabic_Stemmer*	*Supervisor, assessor (12/11/21)*

Type of Project: Exploratory software.

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student): BASHAER ALMASHY

Summary

Stemming is a technique used to reduce inflected and derived words to their basic forms or also called the root. Arabic language stemming can have different difficulties as the language has one of the most challenging morphologies in NLP as it is both morphologically rich and highly ambiguous.

This research aims to improve the Arabic language stemming techniques by implementing a system for stemming Arabic texts to produce the surface word before the addition of any inflectional affixes or the stem. By proposing a way based on the extraction of prefixes and suffixes from the word to then compare it to a list of 135 valid Arabic patterns depending on the extracted word's length to find the right pattern and therefore its root.

Acknowledgments

The satisfaction that accompanies the successful completion of this project would be incomplete without the mention of the people who made it possible and under those whose constant guide the project was completed.

I would like to dedicate this work to family especially my wonderful parents for all the support that I have been receiving from them for my entire life and throughout my master degree journey.

I would also like to thank my friend Wed Alfotawi for helping me gathering my thoughts and fixing my Arabic grammer issues.

Finally, I would like to thank my supervisor Dr. Ammar Alsalka for his patience, motivation and support in this project.

Table of Content

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Introduction

The Arabic language presents one of the most challenging morphologies in NLP as it is both morphologically rich and highly ambiguous. The Arabic language is based on 28 alphabet letters but words can have complicated forms with suffixes and affixes. Most of the formations in Arabic come from the root word consisting of three letters. In addition, Arabic has complicated grammatical rules and it is very rich in its derivational system. These features make the language challenges in computational processing and morphological analysis because in most cases, exact keyword matching between documents and user queries, is inadequate. Some of the language's difficulties as stated by Mustafa, M., Eldeen, A.S., Bani-Ahmad, S., & Elfaki, A.O. (2017) are as follows:

- Complex Morphology.
  Different forms can be derived from one word after being attached to different affixes as shown in table 1 below.

Arabic Word	Meaning
حسب	Compute (a tri-literal root)
يحسب	He computes
حسبنا	We compute
حسبن	They compute (plural feminine)
يحسبون	They compute (plural masculine)
حسبا	They compute (dual masculine)
حاسوب	Computer (Machine name)
حسّب	He computes (for intensifying verbs)

Table 1. Derivatives from the root for the Arabic word حسب

- Words and morphological variations are derived from roots using patterns.
  Arabic is usually constructed from 3 consonants (tri-literals) and it is possible that 4 consonants (quad-literals) or 5 consonants (pent-literals) are used. Words are formed by expanding the root with affixes using well-known morphological patterns as some are shown in Table 2 below. The main pattern is فعل and more regular patterns can be derived from the main pattern.

3-Lettered	4-Lettered	5-Lettered	6-Letterd
فُعْل	فعلن	فعلوا	فعلوهم
فُعَل	فعلت	افعال	تفعيلة
فَعِل	فعول	فعلين	مفاعيل
فَعْل	فعيل	فعلات	يفاعلن
فِعَل	افعل	لنفعل	ليفعلن
فِعِل	فعال	لأفعل	سنفعله
فَعَل	فعلة	ليفعل	فعيلته

Table 2. Regular pattern in Arabic

- Complex structures can be derived from patterns by adding affixes.
  Different kinds of affixes that were collected by Saad, M., & Ashour, W. (2010) are shown in table 3 below.

Antefixes	Prefixes	Suffixes	Postfixes
وبال, وال, بال, فال, كال, ولل, ال, وب, لل, فس, فب, فل, وس, ك, ف, ب, ل	ا, ن, ي, ت	تا, وا, ين, ون, ان, ات, تان, تين, يون, تما, تم, و, ي, ا, ن, ت, نا, تن	ي, ه, ك, كم, هم, نا, ها, تي, هن, كن, هما, كما

Table 3. Arabic Affixes

Nouns and verbs in Arabic are generated from a set of roots, about 11,347 roots classified according to the length of their letter as classified by Al-Omari, A., & Abuata, B.M. (2014) where:

- 115 roots for words with two letters in Arabic
- 7,198 roots for words with three letters in Arabic
- 3,739 roots for words with four letters in Arabic
- 295 roots for words with five letters in Arabic

Arabic roots provide a valuable feature that can be extended for various tasks, where a large number of studies have been devoted to the analysis of the best approach to index Arabic words. Where each word can have more than one root as shown in Figure 1 from El-Defrawy, M., El-Sonbaty,Y., & Belal, N.A. (2016) research.

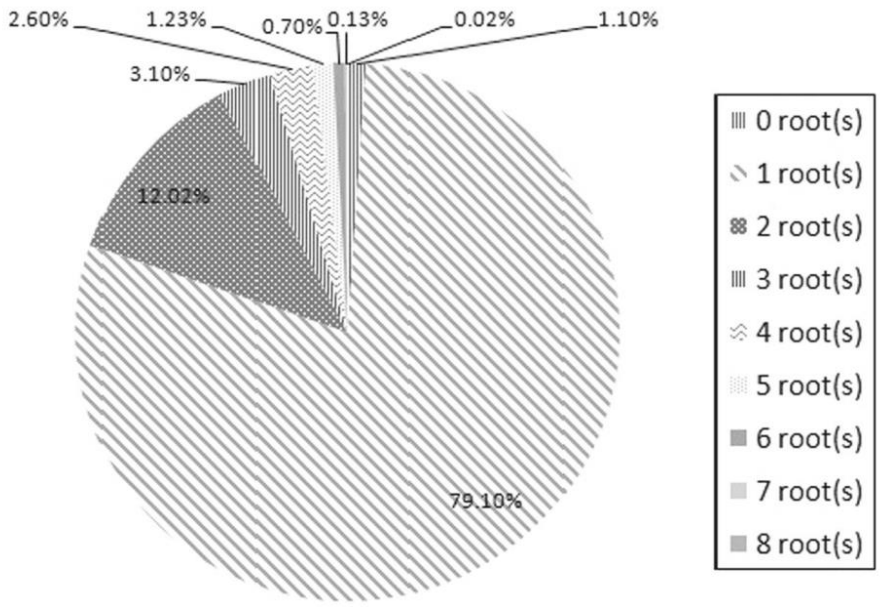

Figure 1. Number of roots corresponding to words percentage

The stemmer is a computational process of removing all the suffixes and prefixes of a word to generate the stem or root. Stemming is an important process used in many fields of natural language processing like information retrieval systems, Web search engines, Question Answering Systems, textual classifiers, and more. The main task in the stemmer is to return the word-formation to the basic word root or stem. For an example by Younes Jaafar, Driss Namly, Karim Bouzoubaa, & Abdellah Yousfi (2017), the word: "كتب" (he wrote), "كتبوا" (they wrote), "سيكتب" (he will write), "أكتبتم" (have you written?) have the same root as "كتب" (he wrote).

There are different methods for stemming which are mostly language-dependent as some stemmers are only applicable to particular languages. As there many algorithms and methods presented by researchers to find the appropriate Arabic root of different Arabic words, they all face different issues including:

- Incorrect suffix and prefix removal
- Words losing their meaning
- Words can have different roots but the stemmer only provides one root

Arabic language stemming algorithms can be categorized as either light-based or rule-based depending on the desired level of analysis.
In this study, an Arabic Stemmer technique is proposed, implemented, and tested. The rest of this research is organized as follows:

- Chapter 2 presents the related work on the topic along with the different available approaches.
- Chapter 3 presents the system requirements of the project.
- Chapter 4 presents the implementation phase of the project.
- Chapter 5 presents the testing phase of the project.
- Chapter 6 presents the conclusion and future work for the project.

1.2 Project Aim

This project aims to improve the Arabic language stemming techniques by implementing a system for stemming Arabic texts to produce the surface word before the addition of any inflectional affixes or the stem.
In order to achieve the goal of indexing the most adequate Arabic root from a word, the proposed system will use a dataset for some morphological patterns in the language and it will focus on tri-literals roots .

1.3 Objectives

Several objectives are essential to achieve the aim of the project as follows:

- Understand the different challenges of current Arabic Stemmers
- Explore available researches on the topic
- Collect the required dataset
- Propose a new Arabic stemmer.
- Implement a high accuracy Arabic stemming system
- Validate the implemented system using different test cases

1.4 Deliverables

At the end of this project, a fully functional software will be provided with testing results along with full research on the project. The project will deliver the following:

- Full research that includes the background of the project, the implementation phase, and the testing phase.
- Related researches on the topic
- A fully functional software for Arabic stemming
- A dataset with some the Arabic morphological patterns
- Test cases to validate the system

1.5 Ethical, Legal, and Social Issues

Since the beginning of the Artificial Intelligence revolution, an extreme threat to the privacy and safety of people was created, as many businessmen and scientists predicated Artificial Intelligence to become a threat to humans. Therefore, it is very important to avoid any ethical issues in this project's implementation. There is no use of any personal data and no external libraries were used in the development.

Chapter 2. Background Research

2.1 Literature Survey

2.1.1 Arabic Stemmer System Based on Rules of Roots

Abuzaid, N., & Al-Abweeny, N. (2018) implemented an Arabic stemmer based on mathematical rules and several relations between letters instead of the Arabic root patterns which aim to handle most Arabic morphological issues. Arabic rules were collected and extracted from the book Almojaz in Arabic Language Grammar.

Methodology:

The method in the research works by conducting pre-processing steps to get ready for the stemming phase, these steps are as follows:

- Remove the diacritics.
- Remove the stopping word like separated pronouns, prepositions, and the letters of the monument and assertion.
- Removing the " ال " which translates to "The".
- Check the words "الله" and return it without any process.

For the main stemming phase, the process works by checking the length of the word as the first step and then going to the suitable case depending on the number of letters. Different cases are as follows:

- Case 1: The system returns the word if it contains three letters.
- Case 2: The word goes through a loop for finding its root if it contains four letters. Finding the root depends on the trilateral rules that were extracted before.
- Case 3: The word goes through many loops if it contains five letters until it reaches the base root of the word after the system matches the word with the suitable trilateral and quadrilateral rules.

- Case 4: If the word contains six letters, the prefix letters are removed based on the suitable rule to then find the trilateral or quadrilateral rule of the word if that is applicable.
- Case 5: If the word contains more than six letters, our system matches all the rules to extract the trilateral, quadrilateral, quintuple and hexagonal.
- Case 6: If the entered word does not match with any rule, then the system returns the exact word, so it may be an anomalous word.

Results:

The system returns all the roots of the entered word with high efficiency with a 95.3% accuracy percentage. For more accuracy and to increase the accuracy percentage, enhancement and improvement can be done to the research's system by handling the bilateral, anomalous words, connected prepositions with words.

2.1.2 An Intelligent Use of Stemmer and Morphology Analysis for Arabic Information Retrieval

Alnaied, A., Elbendak, M., & Bulbul, A. (2020) aim to extract Arabic stems in a new approach because other existing schemes extract the roots by removing affixes from a word without distinguishing whether the removed letters are actually core letters of the root or not.

Methodology:

The research states that a single root can generate 1000 words, which led to the method of root extraction based on morphology features by matching the word with all possible affixes and patterns attached to it. Therefore, the research proposes a technique that attempts to extract the Arabic root based on a validation of the letters before removing affixes by building an AMIR dictionary that generates over 1400 words from each root. Arabic Morphology Information Retrieval Dictionary (AMIR) is constructed from several Arabic grammatical rules to extract stems regarding the relationship among Arabic letters to find the root of the respective words. AMIR dictionary is composed of two main phases:

- Phase 1: Add patterns to the root where eight patterns can be added to each root as the AMIR rules.

- Phase 2: Add affixes to indicate the inflectional morpheme like pronouns, gender, prepositions, and stop words.
-

Results:

The AMIR performance provided highly accurate results and can dissect a plural word and then get its singular form, unlike other techniques it was compared with.

2.1.3 Comparative Study of Arabic Stemming Algorithms for Topic Identification

Naili, M., Chaibi, H., & Ghezala, H.H.B. (2019) provided different algorithms to overcome Arabic stemming problems to compare between them. Several Arabic stemmers have been proposed, yet none of these stemmers have shown a perfect or stable performance in different research fields. Each stemmer has its own advantages, weakness, and limitations.

Methodology:

The research provides several approaches to compare between them including:

1. Root-based approach

   The root-based approach extracts the root of the words by removing the longest suffix and prefix. To then match it with verbal and nouns patterns based on the dictionary.

   Advantages:

   - Using roots in topic categorization is better than using stems.
   - Solves repetition problem.
   - Closest simulations to manual root extraction.

   Disadvantages:

   - Increase word ambiguity
   - Lose form of words
   - May produce wrong roots

2. Light-stem based approach

The light-stem approach eliminates a greater number of suffixes and prefixes from stems. It removes some defined prefixes and suffixes and some strings that would be found as affixes far more often than they would be found as the beginning or end of an Arabic word without affixes.

Advantages:

- Eliminate a greater number of prefixes and suffixes which decreases word ambiguity.
- Highest information retrieval performance.
- More efficient to calculate the similarity between Arabic words than root-based stemmer.
- Maintain the meaning of words.
- Reduce repetition.
- Fast and simple algorithm.

Disadvantages:

- False form of words.
- Low linguistic accuracy.
- Does not handle irregular plural.
- Removes affixes without linguistic rules.

Results:

In the research, it was concluded that using light stems to the presented topic is more efficient than using the full forms of words, roots, or stems in order to avoid repetition and loss of meaning problems. But each stemming technique has its own results and advantages even if they share the same goal.

2.1.4 A New Stemming Algorithm Dedicated for Arabic Documents Classification

Hamid, Z., & Khafaji, H.K. (2020) proposed an Arabic stemming technique to help solve errors occurring during the stemming process. By using a list of collected words taken from 1200 Arabic documents that involves sport, economic, law, and news topics.

Methodology:

This technique works by removing additional letters from the word to extract its root. As it is found that 80% of the Arabic words belong to a root consisting of three letters which are (ع, ل , ف) for the pattern (فعل) where additional letters for popular forms are then added to form Arabic words. In the research affixes that may be attached to the root of Arabic words were put in a list as shown in table 4.

Affixes position and length	Affixes
One Prefix	ا , ب , س , و , ي , ت , ن , ل , ف
One Suffix	ن , ا , ت , ي , ك , هـ , ة
Two Prefixes	ال , لل
Two suffixes	هن , ها , كن , ما , ية , ات , ون , ان , ين , كم , وا , تن , نا يه , هم , يا , تم , ني , ته
Three Prefixes	الا , ولل , بال , وال , كال , فال
Three Suffixes	تهم , تمل , تان , كمل , تين , همل

Table 4. New Stemming Algorithm Research Affixes List

The technique works in three main steps including:

- Hamza Normalization

In this first step, the stemmer normalizes the various forms of “Hamza” (أ, ؤ , ئ , آ, ى) to aleph (ا).

- ت Normalization

Normalize (ت) to (ة)

- sort the list of patterns under consideration from longest word to the shortest word.
- Perform stemming process

If the word is not equivalent to any pattern with the same length, the stemmer will determine the suffix and prefix to remove compare them again to a different pattern’s length until no patterns matched and it gets to three letters presenting the root.

Results:

The process of stemming relays on the possibilities of the number of affixes in the word and its position. Wrong results were found when five and six letters’ words have only one suffix or prefix removed in different research, but in the proposed technique this issue was solved because the stemmer does not remove suffixes and prefixes until after matching the word with predefined patterns which helps in avoiding the deletion of primary letters in the word.

2.2 Methods and Techniques

For the Arabic stemmer project to be fully functional and achieve the best efficiency and highest accuracy level, different approaches should be studied and evaluated to select the best technique for the implementation as follows.

2.2.1 Light-Stem Based Approach

The light stemming approach as written by Atwa, J., Wedyan, M, & Al-Zoubi, H. (2019) aims not to produce the linguistic root of a given Arabic word, but to remove the most frequent suffixes and prefixes. It works by stripping off a small set of prefixes and suffixes by going through the list of suffixes once in right to left order. This approach eliminates a larger number of prefixes and suffixes which causes a serious issue in stemming Arabic words because it does not differentiate between root characters letters and affix letters.

Light-Stem based technique has four main steps as stated Abd, D.H., Khan, W., Thamer, K.A., & Hussain, A.J. (2021) are shown in the model in Figure 2 below.

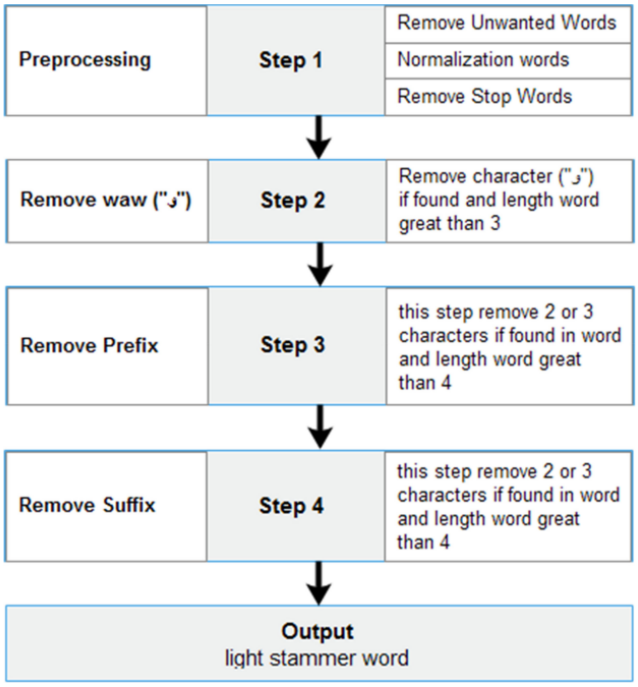

Figure 2. Light-Stem Based Model

2.2.2 Root-Based Approach

Al-Kabi, M.N., Kazakzeh, S.A., Abu Ata, B.M., Al-Rababah, S.A., & Alsmadi, I.M. (2015) summarized the Root-based stemming technique, also called the Morphological Analysis, which aims to extract the basic form for any given word by performing heuristic and linguistic morphological analysis. This approach first removes the longest prefixes and suffixes, to then match the word with predefined patterns. The searched pattern depends on the length of the word after extraction. The three phases for the Root-based stemming techniques are as follows where the pseudo-code is shown in Figure 3.

- Remove Affixes
- Identify verb pattern
- Refine the proposed root

Input: A text file that contains the Arabic words

Output: Arabic Triliteral Verb/Verbs

1. Remove Arabic prefix(es) from each word
2. Normalize 3 shapes of (Alif, "أ، إ، آ") to (Bare Alif, "ا")
3. Remove suffix(es) from each word
4. Determine word length after removing affixes (prefixes and suffixes)
5. Identify Arabic patterns having same lengths to word length in step 4.
6. Compare each pattern identified in step 5 with extracted word from step 3
7. Select the closest pattern:
    a. Choose the pattern from the set of Arabic patterns having same lengths to word length which has the highest number of common Arabic letters with the Arabic word extracted from step 3.
    b. Determine the pattern which has the largest matching corresponding letters with the generated word from step 3 which is considered as the right pattern, where the corresponding Arabic letters within the extracted word from step 3 will not be compared with three Arabic letters (Faa', "ف"), (Ayn, "ع"), (Laam, "ل") within the pattern under consideration.
8. Eliminate all matched letters in step 7. The Arabic letters of the Arabic word extracted from step 3 which corresponds to the Arabic letters (Faa', "ف"), (Ayn, "ع"), and (Laam, "ل") in the selected pattern (found in step 7.a) are selected to constitute the extracted Arabic root.
9. Refine the extracted Arabic root by converting some of the Arabic letters.

Figure 3. Root-based Approach Pseudocode

2.2.3 Similarity Measures Approach

The similarity measure Al-Ramahi, M.A., & Mustafa, S.H., (2012) wrote about in their research is used to measure the similarity of the meaning between two words, sentences, or documents. It is a challenging task, especially for short texts. Text similarity aims to determine the degree of likeness or closeness of two pieces of texts at the sentence level. Examples of similarity measures approach can be found in Machine translation, summarization, question answering, and automatic short answer grading, and conversational systems. To start with the similarity measures approach different aspects goes into the study as follows:

- Selecting the stemming algorithm
- Selecting the machine learning algorithm
- Select the similarities features

To further explain the similarity measures approach, the following flowchart by Alhawarat, M.O., Abdeljaber, H., & Hilal, H. (2021) illustrates the steps of the method.

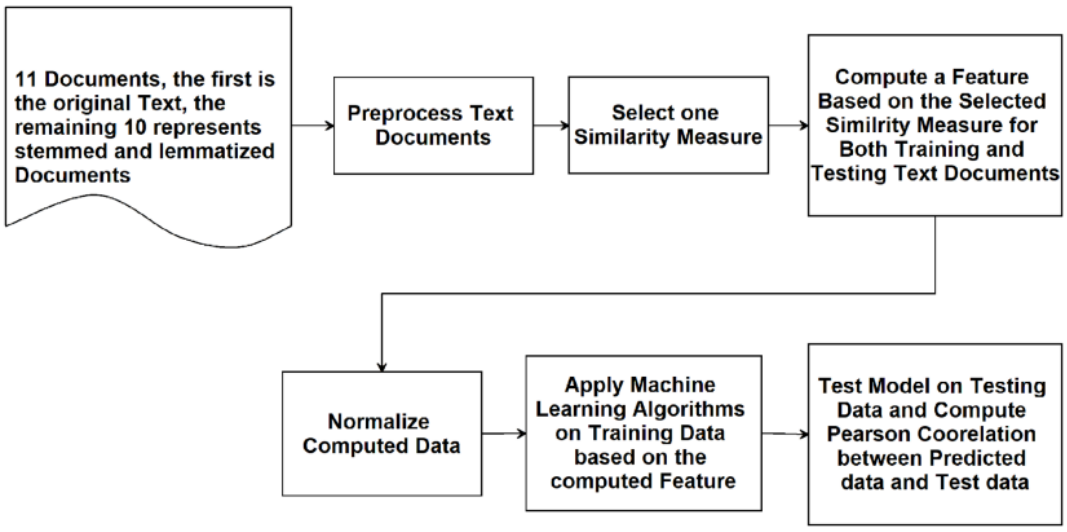

Figure 4. Similarity Measures Approach Flowchart

2.2.4 Methods Comparison

Approach	Advantages	Disadvantages
Root-based approach	• Remove suffixes and prefixes • Match the extracted root against a list of valid roots. • Employ morphological rules	• Can cluster different words into a single root • Lose meaning of the word • Can remove affixes that are part of the word • May produce wrong roots
Light-stem based approach	• Preserve word meaning • Fast and simple • Reduce repetition • High performance	• Removes affixes without linguistic rules • Does not handle irregular plural
Similarity Measures Approach	• Sentence based • High accuracy level • Useful for paragraphs translations	• Complicated task • Long implementation process

Table 5. Methods Comparison

2.3 Choice of Methods

To overcome the weaknesses in the nowadays Arabic Stemming techniques and consequently improve the search effectiveness, and with the use of previous stemming techniques studied, a new method is proposed with the aim of improving and testing another way of Arabic Stemming.

The proposed technique removes known singular and plural prefixes and suffixes from the word, to then use the extracted word in the second step. If the extracted word's length is three letters, it is considered in the measure of 'فعل'. In the next step, the word is compared with the Arabic known patterns with the same length as the extracted word. The system will contain a list of 135 valid Arabic pattern words. The extracted word and the pattern word are compared with each other letter by letter and remove the similar letters at the same index except for the letters "ف, ع, ل" in the pattern. The root of the word consists of the letters left after the third step.

Advantages:

- Compare words with valid Arabic patterns
- Handles irregular plurals
- Keeps word meaning

Chapter 3. System Requirements and System Design

3.1 Software Requirements

Any software system needs a programming language, and this language is the core of the system which means that without a programming language there is no code and without the code, there is no system. Therefore, choosing a suitable programming language is the most critical part of building a software system and it can be one of the major reasons for a software system's success or failure. In this project, the Python programming language was the best choice instead of the other available programming languages because of many reasons including its flexibility, and the wide Python community.

To successfully use Python as a programming language, the hardware requirements of the computer must have the following minimum requirements which are considered easy to meet:

- Modern Operating System including:
    - Windows 7 or 10
    - Mac OS X 10.11 or higher, 64-bit
    - Linux: RHEL 6/7, 64-bit (almost all libraries also work in Ubuntu)
- x86 64-bit CPU (Intel / AMD architecture)
- 4 GB RAM
- 5 GB free disk space

3.2 System Design

Defining the requirements of the project contributes to creating a clearer idea and gives the needed information that describes the behavioral and non-behavioral features of the system.

In this section, the functional requirements and the non-functional requirements for this system will be defined. Along with the needed UML diagrams to describe and analyze the system to make the researches technique clearer and provide a good visualization of the system requirements and flow.

3.2.1 Functional Requirements

FR1: Remove known singular and plural prefixes and suffixes from the word.	
Priority	High It is very important for the system to successfully extract the suffixes and prefixes from the word.
Description	The system should remove known suffixes and prefixes from the word.
Pre-Condition	Providing an Arabic word for extraction.
Post-Condition	The extracted version of the word.

Table 6. FR1

FR2: Compare extracted words with the same length patterns.	
Priority	High The main functionality of the system is comparing the extracted word with the same length patterns.
Description	The system should compare extracted words with the same length patterns available in the developed dataset if the word's length is more than three letters.
Pre-Condition	Extracted word with removed suffixes and prefixes.
Post-Condition	Find the correct pattern for the word for comparison.

Table 7. FR2

FR3: Compare extracted words with the selected pattern.	
Priority	High The system should compare the extracted word with the selected pattern to find the word's root which is the main goal.
Description	The extracted word and the pattern word are compared with each other letter by letter and remove the similar letters at the same index except for the letters "ف, ع, ل" in the pattern.
Pre-Condition	Extracted word with selected matching pattern.
Post-Condition	Finding the root of the word.

Table 8. FR3

3.2.2 Non-Functional Requirements

NFR-01: Performance (response Time)	
Priority	High Response time is an important feature of the system.
Description	For the system to be efficient it must handle the functionalities easily and respond as fast as possible.

Table 9. NFR-01 Performance

NFR-02: Reliability	
Priority	High The system should not crash in any situation.
Description	The system is reliable as it will not crash at any function.

Table 10: NFR-02 Reliability

NFR-03: Security	
Priority	High To protect data privacy and information from hacking and unauthorized access.
Description	The data in the system is secured and will not be shared or viewed by an unauthorized person.

Table 11: NFR-03 Security

NRF-04: Maintainability	
Priority	High To make sure that the system can grow, and more features are added.
Description	The system can be simply fixed the cause of any error or bug and it's updatable.

Table 12: NFR-04 Maintainability

NRF-05: Usability	
Priority	High To make sure that the system can be used easily by any person.
Description	The system should be learnable and easy to use and work on for future updates.

Table 13: NFR-07 Usability

NRF-07: Scalability	
Priority	High To make sure that the system can provide more usability.
Description	The system should be extensible to provide more usability.

Table 14: NFR Scalability

3.2.3 Flowchart Diagram

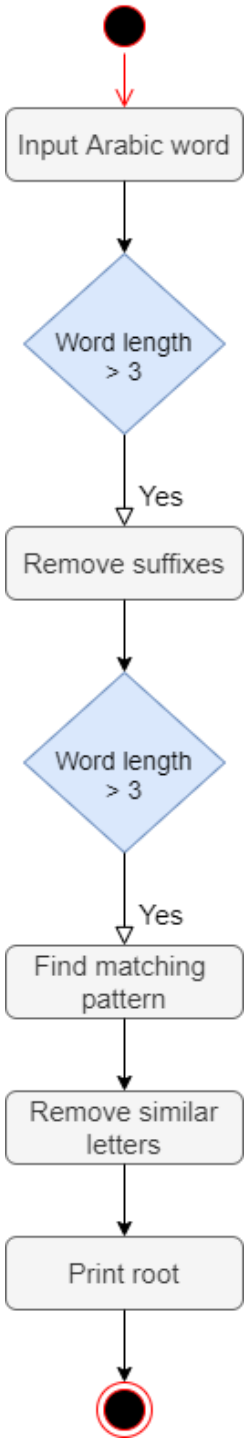

Figure 5. Flowchart Diagram

The process of extracting the root from the word “بكلمات” is shown in Figure 6 below.

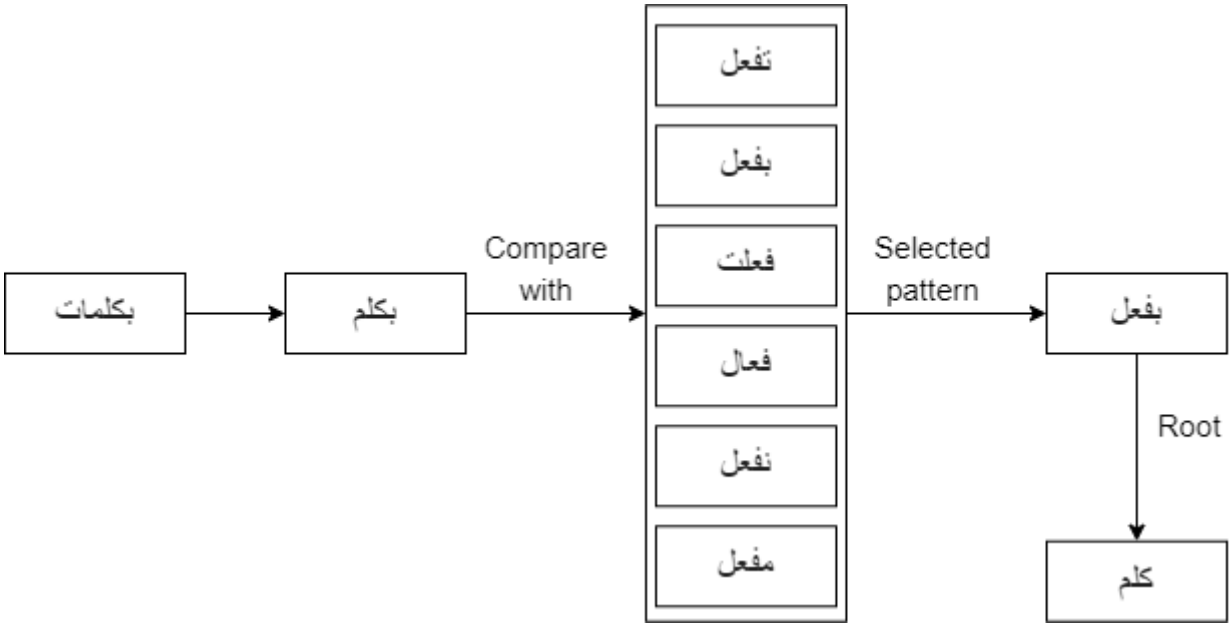

Figure 6. Root Extraction Example

Chapter 4. Software Implementation

The aim of this research is to build a fully functional Arabic stemmer to extract Arabic roots from the input words. The proposed technique undergoes different steps to successfully extract the root at the end. This section provides full details and code of the proposed system.

- Phase 1

  In the first step, the entered word goes through suffixes and prefixes extraction in order to make the word ready for the next step. This is done by removing known suffixes if available in the word depending on the words' length not all at once but twice to ensure no main letters are lost from the word as follows:

  1. Word's length is greater than or equals to five letters
     - وا
     - ون
     - ها
     - هن
     - هم
     - ان
     - كن
     - ات

```
for x in range(5):
  if len(word):

   if len(word)>=5:
     if word.endswith("وا"):
        word=word.replace("وا","")
     elif word.endswith("ون"):
        word=word.replace("ون","")
     elif word.endswith("ها"):
         word=word.replace("ها","")
     elif word.endswith("هن"):
         word=word.replace("هن","")
     elif word.endswith("هم"):
         word=word.replace("هم","")
     elif word.endswith("ان"):
          word=word.replace("ان","")
     elif word.endswith("كن"):
          word=word.replace("كن","")
     elif word.endswith("ات"):
          word=word.replace("ات","")
```

Figure 7. Words with five letters suffixes extraction

2. Word’s length is greater than or equals to six letters

    - هما

```
if len(word)>=6:
    if word.endswith("هما"):
     word=word.replace("هما","")
```

Figure 8. Words with six letters suffixes extraction

3. After previous suffixes removal, if the word’s length is still greater than or equals to five letters, more affixes will be removed as follows.
    - ال
    - فل
    - لي

```
if word[0]=="ا" and word[1]=="ل":
  word=word.replace("ال","")
elif word[0]=="ف" and  word[1]=="ل":
  word=word.replace("فل","")
elif word[0]=="ل" and  word[1]=="ي":
  word=word.replace("لي","")
```

Figure 9. Words with five letters suffixes extraction step 2

4. After previous suffixes removal, if the word's length is still greater than or equals to six letters, more affixes will be removed as follows.
    - وست
    - ولل
    - كال
    - وال
    - فال
    - بال
    - لل

```
if len(word)>=6:
    if word[0]=="و" and  word[1]=="س" and word[2]=="ت":
      word=word.replace("وست","")
    elif word[0]=="و" and  word[1]=="ل" and word[2]=="ل":
      word=word.replace("ولل","")
    elif word[0]=="ك" and word[1]=="ا" and word[2]=="ل":
      word=word.replace("كال","")
    elif word[0]=="و" and word[1]=="ا" and word[2]=="ل":
      word=word.replace("وال","")
    elif word[0]=="ف" and word[1]=="ا" and word[2]=="ل":
      word=word.replace("فال","")
    elif word[0]=="ب" and word[1]=="ا" and word[2]=="ل":
       word=word.replace("بال","")
    elif word[0]=="ل" and word[1]=="ل":
      word=word.replace("لل","")
```

Figure 10. Words with six letters suffixes extraction step 2

- Phase 2

In this step, the correct pattern for the word is searched for from a dataset of 135 valid patterns depending on the word's length. The patterns used are as follows:

فعيلتها","فعيلته","يتفعل","تتفعل","افتعاليه","افتعالية","افتعالي","تفعلت","فعلة","افعلت","فعلي","بفعل","سنفعله","ستفعلونه","ستفعلوه","سيفعلونه","سيفعلوه","ستفعله","سيفعله","فعله","نفعله","تفعلونه","تفعلوه","يفعلونه","يفعلوه","تفعله","يفعله","متفعلة","متفعل","فعلية","استفعالات","مفاعل","مفعلة","مفعلات","انفعالات","لنفعل","ليفعلن","ليفعلوا","لتفعلوا","لأفعل","ليفعل","لتفعلي","لتفعل","يفعل","تفعل","افعل","اتفعل","نفعل","يفعلوا","يفعلن","فعلل","يفعلل","تفعلل","افعلل","فعللة","فاعل","فاعلة","فاعلان","فاعلون","استفعل","استفعال","استفعلت","مستفعل",

"استفعلوا","انفعل","انفعلت","انفعال","انفعلوا","انفعلن","مفتعل","مفتعلة","افتعال","افتعل","افتعلت","مفعول","مفاعيل","تفعيل","فعيلة","مفعل","مفعلة","مفعلون","افعل","افعلي","افعلوا","افعلن","تفاعل","تفاعلات","يفاعل","افاعل","يفاعلوا","يفاعلن","يتفاعل","تتفاعل","يتفاعلوا","يتفاعلن","اتفاعل","ينفعل","تنفعل","انفعل","ينفعلوا","انفعلوا","انفعلن","ينفعلن","يفتعل","تفتعل","يفتعلوا","افتعلوا","افتعلن","يفتعلن","يستفعل","تستفعل","يستفعلوا","استفعلوا","يستفعلون","استفعلن","استفعل","استفعلت","فعال","فعالة","فعالين","فعالون","افتعالة","فعول","الفعول","فواعل","متفاعل","متفاعلة","متفاعلون","متفاعلين","متفاعلات","افعال","فعلا","مفاعلة","افعلة","فعلات","فعلين","افعال","تفعيلة","فعلوهم","فعل","فعلت","فعلوا","فعلن

After selecting the correct pattern for the extracted word, the extracted word and the pattern word are compared with each other letter by letter to remove the similar letters at the same index except for the letters "ف, ع, ل" in the pattern.

```
root=""
lenght=len(word)
M_list=[]
for x in The_list:
  if len(x)==lenght:
    M_list.append(x)
position=0
remain=0
for x in M_list:
  for y in x:
    if y!=('ف') and y!=('ع') and y!=('ل'):
      if y==word[position]:
        remain+=1
    position+=1
```

Figure 11. Extracted word and pattern comparison

- Phase 4

After deleting similar letters at the same index between the word and the pattern, the root of the word consists of the letters left in it at the positions of (ف,ع,ل) in the pattern.

```
if remain==lenght-3:
  r1 = x.index('ف')
  r2 = x.index('ع')
  r3 = x.index('ل')
  root=word[r1]+word[r2]+word[r3]
  print("the root of the word '"+ The_word +"' is :" +root)
  break
else:
  position=0
  remain=0
```

Figure 12. Printing the word's root

4.1 Another Solution

To get to the final code of the proposed solution, different attempts were made first to successfully confirm the selected technique discussed previously. One of the different solutions that were tested aimed to extract the root of the Arabic word by focusing more on the main three-letters patterns to extract different words formed from the selected pattern.

To explain this solution in detail, different steps were made as follows:

1. Collect the dataset

   An online dataset for Arabic words was used as it was saved in a list for editing and later uses in the code.

```
example1 = "/content/new.txt"
```

```
with open(example1, "r") as file1:
    FileasList = file1.readlines()
```

```
counter=0
for x in FileasList:
  FileasList[counter]=x.replace("'\n'","")
  FileasList[counter]=x.strip("\'")
  counter+=1
#FileasList
```

Figure 13. Another Solution Dataset

Some editing was done on the used dataset to remove duplicates and remove words that contained less than three letters as shown below.

```
for word in FileasList:
  if len(word)>=5:
    #وا
    if word.endswith("وا"):
      FileasList[counter_x]=word.replace("وا","")
    #ون
    elif word.endswith("ون"):
      FileasList[counter_x]=word.replace("ون","")
    #ها
    elif word.endswith("ها"):
      FileasList[counter_x]=word.replace("ها","")
    #هن
    elif word.endswith("هن"):
      FileasList[counter_x]=word.replace("هن","")
    #هم
    elif word.endswith("هم"):
      FileasList[counter_x]=word.replace("هم","")
    #ان
    elif word.endswith("ان"):
      FileasList[counter_x]=word.replace("ان","")
    #كن
    elif word.endswith("كن"):
      FileasList[counter_x]=word.replace("كن","")

  if len(word)>=6:
    #هما
    if word.endswith("هما"):
      FileasList[counter_x]=word.replace("هما","")
```

```
if len(word)>=4:
  #ها
  if word.endswith("ة"):
    FileasList[counter_x]=word.replace("ة","")

if len(word)>3:
  #ال
  if word[0]=="ا" and word[1]=="ل":
    FileasList[counter_x]=word.replace("ال","")
  #لل
  elif word[0]=="ل" and word[1]=="ل":
    FileasList[counter_x]=word.replace("لل","")
  #بال
  elif word[0]=="ب" and word[1]=="ا" and word[2]=="ل":
    FileasList[counter_x]=word.replace("بال","")
  #فال
  elif word[0]=="ف" and word[1]=="ا" and word[2]=="ل":
    FileasList[counter_x]=word.replace("فال","")
  #كال
  elif word[0]=="ك" and word[1]=="ا" and word[2]=="ل":
    FileasList[counter_x]=word.replace("كال","")
  #وال
  elif word[0]=="و" and word[1]=="ا" and word[2]=="ل":
    FileasList[counter_x]=word.replace("وال","")
  #فل
  elif word[0]=="ف" and  word[1]=="ل":
    FileasList[counter_x]=word.replace("فل","")
  #ولل
  elif word[0]=="و" and  word[1]=="ل" and word[2]=="ل":
    FileasList[counter_x]=word.replace("ولل","")
  #وست
  elif word[0]=="و" and  word[1]=="س" and word[2]=="ت":
    FileasList[counter_x]=word.replace("وست","")
```

Figure 14. Another Solution Dataset Filtering

2. Remove Affixes and Prefixes

The selected word goes through this step for affixes and prefixes extraction with the aim of getting the word's root.

```
if len(the_word)>=5:

    if the_word[0]=="ا" and the_word[1]=="ل":
      the_word=the_word.replace("ال","")
    elif the_word[0]=="ل" and the_word[1]=="ل":
      the_word=the_word.replace("لل","")
    elif the_word[0]=="ب" and the_word[1]=="ا" and the_word[2]=="ل":
      the_word=the_word.replace("بال","")
    elif the_word[0]=="ف" and the_word[1]=="ا" and the_word[2]=="ل":
      the_word=the_word.replace("فال","")
    elif the_word[0]=="ك" and the_word[1]=="ا" and the_word[2]=="ل":
      the_word=the_word.replace("كال","")
    elif the_word[0]=="و" and the_word[1]=="ا" and the_word[2]=="ل":
      the_word=the_word.replace("وال","")
    elif the_word[0]=="ف" and  the_word[1]=="ل":
      the_word=the_word.replace("فل","")
    elif the_word[0]=="و" and  the_word[1]=="ل" and the_word[2]=="ل":
      the_word=the_word.replace("ولل","")
    elif the_word[0]=="و" and  the_word[1]=="س" and the_word[2]=="ت":
      the_word=the_word.replace("وست","")
```

Figure 15. Another Solution Affixes Extraction

3. Cosine Similarity Algorithm

The cosine Similarity technique used by Froud, H., Lachkar, A., & Ouatik, S.A. (2012) is a popular way to measure the similarity in text documents irrespective of their size. It is mainly based on counting the maximum number of common words between the two documents. Using this algorithm, the extracted word was compared with all the Arabic words in the dataset to collect the similar words available.

```
def word2vec(word):
    from collections import Counter
    from math import sqrt

    cw = Counter(word)
    sw = set(cw)
    lw = sqrt(sum(c*c for c in cw.values()))

    return cw, sw, lw

def cosdis(v1, v2):
```

```
iv=word2vec(the_word)
for x in FileasList:
  xv=word2vec(x)
  if cosdis(xv,iv) > 0.9:
   wordslist.append(x)
   print(x + ":" + str(cosdis(xv,iv)) )
```

Figure 16. Another Solution Cosine Similarity

4. Remove duplicated letters

After collecting a list of similar Arabic words to the extracted word, each similar word goes through a step of deleting any duplicate letters in the word to be able to count the unique letters in each word.

```
from collections import Counter
all=[]
for x in wordslist:
  all.append("".join(set(x)))
all ="".join(all)
alpha={'0:'ت', 0:'ب', 0:'ا', 0:'أ',
'0:'ز', 0:'ر', 0:'ذ', 0:'د', 0:'خ', 0:'ح', 0:'ج', 0:'ث',
'0:'ع', 0:'ظ', 0:'ط', 0:'ض', 0:'ص', 0:'ش', 0:'س',
'0:'غ',
'0:'ؤ', 0:'ء' ,0:'ئ', 0:'ي', 0:'و', 0:'ه', 0:'ن', 0:'م', 0:'ل', 0:'ك', 0:'ق', 0:'ف'}
counter_x=0
for x in alpha:
  alpha[x]=all.count(x)
print(alpha)
c = Counter(alpha)
```

Figure 17. Another Solution Duplicate Letters Removal

5. Common letters

In the last step of this technique, the top three common letters were selected and organized to match the extracted word. These three letters selected are considered the final root of the word.

```
the_root=c.most_common(3)
print(the_root)
root_letters=[]
for x in the_root:
  root_letters.append(x[0])
print(root_letters)
ff=""
for x in the_word:
  if x in root_letters:
    ff=ff+x
print("the root of "+the_word+" is: "+ ff)
```

Figure 18. Another Solution common letters

Results:

This technique was not considered to be efficient because of different issues faced including:

- The Dataset

  One of the failure reasons was the dataset used because of many reasons such as:

  1. The words contained many affixes which affected the duplicate letters extraction step.
  2. Many roots in the dataset were limited to a few options of words.
  3. The dataset words were too general and contained all types of roots with different words' lengths.

- Cosine Similarity

  The cosine similarity algorithms only focus on similar letters which make words lose their meaning and give wrong similar words. Although the algorithm could be efficient in different projects, trying it in this Arabic stemming approach was not successful and is mostly because Cosine similarity works best on comparing paragraphs better than single words.

Chapter 5. Software Testing and Evaluation

As mentioned earlier, the proposed Arabic stemming technique was implemented using the Python programming language. The system reads the entered Arabic word to extract its root after comparing the word with one of the 135 Arabic patterns in the dataset depending on its length. The output results for some Arabic words that were extracted successfully are as follows:

Inputted Word	Number of letters	Extracted root	Result
وقوف	Four	وقف	Correct
تصبيرة	Six	صبر	Correct
كاتب	Four	كتب	Correct
مكتوب	Five	كتب	Correct
خدمات	Five	خدم	Correct
عروض	Four	عرض	Correct
فضيلة	Five	فضل	Correct
استخرج	Six	خرج	Correct
يستخدم	Six	خدم	Correct
مقدمة	Five	قدم	Correct
الانفجارات	Ten	فجر	Correct
جحور	Four	جحر	Correct
مطعم	Four	طعم	Correct

معلمة	Five	علم	Correct
جريمة	Five	جرم	Correct
الاستفسارات	Eleven	فسر	Correct
الأفضل	Six	فضل	Correct
طباعة	Five	طبع	Correct
يتذكر	Five	ذكر	Correct
يدرسون	Six	درس	Correct
استخدموا	Eight	خدم	Correct

Table 15. Correct Extracted Words Results

After testing the implemented code on 92 different Arabic words, 89 roots were extracted correctly with only three words resulting in the wrong root. The following table shows the wrong results that would help for future updates.

Inputted Word	**Number of letters**	**Extracted root**	**Result**
مكتبة	Four	كبة	Wrong
لتخرج	Five	-	Wrong
انتقد	Five	تقد	Wrong

Table 16. Wrong Extracted Words Results

One of the tested words that resulted in a wrong root was the word "انتقد", the implemented code found two patterns for the word which led to choosing the wrong pattern as it was difficult to compare the patterns for the best option in some cases.

To verify the efficiency of the proposed technique, and after collecting the results of different roots from the tests made, the proposed solution's experimental results achieving success as it returned 96.7% accuracy rate.

Chapter 6. Conclusion and Future Work

6.1 Conclusions

The Arabic language is an extremely rich language with its morphology, derivational system, and grammatical rules. Through this research, it was clear how stemming techniques could have a significant impact on improving retrieval performance.

To be able to determine the right technique and features of the proposed system, a full search for similar work from other researchers was done to review, analyze, and discuss their work. Other similar systems that have been conducted to resolve Arabic stemming problems were studied to list their main features and limitations to then compare it with the proposed solution. After analyzing other systems, the next step was to determine the system's main features and functionalities based on the studied topics.

Which led to the implementation of the Arabic Stemmer project using a list of valid Arabic patterns to use on words after the extraction of suffixes and prefixes depending on the extracted word length.

As a conclusion, the main objectives and deliverables of this project have been met and what can be finally confirmed is that this technique is efficient and resulted in stemming the correct root from words without losing its meaning.

6.2 Future Work

The proposed project's results proved that it is fully functional and accurate based on the results of the test. However, future updates suggestions could be made for higher efficiency including:

- Expanding the current patterns' dataset for more accurate results.
- Testing more Arabic roots to find better improvements for the code.
- Add more suffixes and prefixed that could be found in Arabic words.

- Improve choosing the right pattern for the word.

List of References

Abd, D.H., Khan, W., Thamer, K.A., & Hussain, A.J. 2021. Arabic Light Stemmer Based on ISRI Stemmer. *Part of the Lecture Notes in Computer Science book series.* 12838, pp. 32-54.

Abuzaid, N., & Al-Abweeny, N. 2018. Arabic Stemmer System based on Rules of Roots. *International Journal of Information Technology and Language Studies*. 2(1), pp.19-26.

Al-Afghani, S. 2003. *Almojaz in Arabic Language Grammar.* Beirut: Dar Al Fikr.

Alhawarat, M.O., Abdeljaber, H., & Hilal, H. 2021. Effect of Stemming on Text Similarity for Arabic Language at Sentence Level. *PeerJ Computer Science.*

Al-Kabi, M.N., Kazakzeh, S.A., Abu Ata, B.M., Al-Rababah, S.A., & Alsmadi, I.M. 2015. A Novel Root Based Arabic Stemmer. *Journal of King Saud University - Computer and Information Sciences.* (2), pp. 94-103.

Alnaied, A., Elbendak, M., & Bulbul, A. 2020. An intelligent use of stemmer and morphology analysis for Arabic information retrieval. *Egyptian Informatics Journal*. 21(4), 209–217.

Al-Omari, A., & Abuata, B.M. 2014 Arabic Light Stemmer. *Journal of Engineering Science and Technology.* 9(6), pp. 702-717.

Al-Ramahi, M.A., & Mustafa, S.H., 2012. N-Gram-Based Techniques for Arabic Text Document Matching; Case Study: Courses Accreditation. *ABHATH AL-YARMOUK: "Basic Sci. & Eng."* 21(1), pp. 85-105.

Attia, M. 2015. Arabic Wordlist for Spellchecking. [Online]. [Accessed 4 November 2021]. Available from: https://sourceforge.net/projects/arabic-wordlist/

Atwa, J., Wedyan, M, & Al-Zoubi, H. 2019. Arabic Text Light Stemmer. *Fourth International Conference on Trends in Computing and Information Technology (ICTCIT 2019).* 8(2), pp. 17-23. /337089412_Arabic_Text_Light_Stemmer

El-Defrawy, M., El-Sonbaty,Y., & Belal, N.A. 2016. A Rule-Based Subject-Correlated Arabic Stemmer. *Arabian Journal for Science and Engineering.* 41(8), pp. 2883-2891.

Enthought Knowledge Base. 2020. Python Minimum Hardware Requirements [Online]. [Accessed 2 November 2021]. Available from: https://support.enthought.com/hc/en-us/articles/204273874-Enthought-Python-Minimum-Hardware-Requirements

Froud, H., Lachkar, A., & Ouatik, S.A. 2012. Stemming versus Light Stemming for measuring the simitilarity between Arabic Words with Latent Semantic Analysis model. *Colloquium in Information Science and Technology.* pp. 69-73.

Hamid, Z., & Khafaji, H.K. 2020. A new stemming algorithm dedicated for Arabic documents Classification. *2020 2nd Annual International Conference on Information and Sciences (AiCIS).*

Jaafar, Y., Namly, D., Bouzoubaa, K., & Yousfi, A. 2017. Enhancing Arabic stemming process using resources and benchmarking tools. *Journal of King Saud University - Computer and Information Sciences.* 29(2), pp. 164-170.

Mustafa, M., Eldeen, A.S., Bani-Ahmad, S., & Elfaki, A.O. 2017. A Comparative Survey on Arabic Stemming: Approaches and Challenges. *Intelligent Information Management*. 9, pp. 39-67.

Naili, M., Chaibi, H., & Ghezala, H.H.B. 2019. Comparative Study of Arabic Stemming Algorithms for Topic Identification. *Procedia Computer Science*. 159, pp. 794–802.

Saad, M., & Ashour, W. 2010. Arabic Morphological Tools for Text Mining. *6th ArchEng International Symposiums, EEECS'10 the 6th International Symposium on Electrical and Electronics Engineering and Computer Science.*