AUTOMATIC VOCALIZATION OF ARABIC TEXT

BY

Yahya Mohammad Suleiman Khraishi

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

May 2016

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN - 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Yahya M. Khraishi** under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

[illegible]
H.M

Dr. Husni A. Al-Muhtaseb
(Advisor)

[illegible]

Dr. Moustafa Elshafei
(Member)

[illegible]

Dr. Wasfi Al-Khatib
(Member)

عنه/د.خالد الجاسر [illegible] ١٤٣٨/٢/٢٠

Dr. Abdulaziz Alkhoraidly
Department Chairman

[illegible]

Dr. Salam A. Zummo
Dean of Graduate Studies

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES

24/11/16

Date

Dedication

To my family…

ACKNOWLEDGMENTS

I would like to give my deep thanks and gratitude to my advisor Dr. Husni Al-Muhtaseb who supervised this work, for his continuous guidance and supervision, which made this research possible. I also would like to thank the committee members: Dr. Mustafa Elshafei and Dr. Wasfi Al-Khatib.

In addition, I would like to acknowledge the Deanship of research at KFUPM as the work is part of DSR projects with numbers RG1104-1 and RG1104-2 under Dr. Husni Al-Muhtaseb.

Also, I would like to give my thanks to the authors of the many tools I have used, and I would like to acknowledge their work. I have used several tools and libraries in the research, which assisted me greatly in the work. My personal appreciation and thanks to the authors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

• **AATD**	Arabic Automatic Text Diacritization
• **ARFF**	Attribute Relation Format File
• **ASR**	Automatic Speech Recognition
• **ATD**	Automatic Text Diacritization
• **AUC**	Accuracy
• **CR**	Correct Ratio
• **CE**	Case Ending
• **CSV**	Comma Separated Vectors
• **DER**	Diacritic Error Rate
• **DLL**	Dynamic Link Library
• **DT**	Diacritization Tool
• **HMM**	Hidden Markov Models
• **IG**	Information Gain
• **JAR**	Java Archive
• **POS**	Part of Speech Tagging
• **ROC**	Receiver Operation Characteristic
• **WER**	Words Error Rate
• **WS**	Window Size

ABSTRACT

Full Name	:	Yahya Mohamad Suleiman Khraishi
Thesis Title	:	Automatic Vocalization of Arabic Text
Major Field	:	Computer Science
Date of Degree	:	May 2016

Diacritical marks in Arabic play a major role in understanding the meaning of the words, their pronunciations and the overall meaning of the context. A word could have different forms of diacritics and thus different meaning for each form. While native Arabic speakers face no problems in reading and understanding Arabic text with no diacritics, non-native speakers find it difficult. Arabic computer applications such as speech recognition applications or text to speech applications need the Arabic words to be vocalized. Otherwise, using unvocalized text on such applications would add ambiguity to the process and may have a negative impact on the results.

Automatic vocalization is the process of inserting diacritics to unvocalized or partially vocalized text. This process, sometimes called "diacritic restoration", is common with different levels in several languages including some Latin and Semitic languages.

This research work reports the development process of the updated fully diacritized corpus and Arabic text vocalization using decision trees algorithms.

For the corpus development, we redeveloped a previously built corpus, SENTENCES3. The SENTENCES3 corpus was normalized to have it consistent and fully vocalized. We have named the newly corpus as “Tashkeel-2016” Furthermore, we introduced a new corpus which is the MUSHAF, for being accurately vocalized and consistent to be used in text vocalization. A third corpus was also developed. The new corpus targeted Modern Standard Arabic (MSA) in news since the nature of the “Tashkeel-2016”corpus was mostly Classical Text. The news corpus, named “Akhbar-2016”, contains over 10 million words.

The second part of the work was Arabic text vocalization. Features extraction was done to come up with features that would help in the classification process. After applying feature extraction, feature selection was performed to select the best set of features.

To prepare an appropriate setup for vocalization, several modules were developed. The developed modules communicate together to form the vocalization system. The vocalization system uses decision tree algorithm for classifications. The system also applies a post-processing step of vocalization using N-Gram word models.

Many experiments were conducted trying to achieve the highest accuracy and the best performing model. The highest result achieved was for the MUSHAF corpus. Results of

6% and 9% for diacritic error rate (DER) without case ending and a DER with case ending were achieved respectively. For “Tashkeel-2016”experiments, results of 18% and 28% was achieved for both the word error rate (WER) without case ending and WER with case ending respectively.

ملخص الرسالة

الاسم الكامل: يحيى محمد سليمان خريشي

عنوان الرسالة: تشكيل النص العربي آليا

التخصص: علوم الحاسب الآلي

تاريخ الدرجة العلمية: مايو 2016

تمثل علامات التشكيل (الحركات) في اللغة العربية دورا رئيسا في فهم معاني الكلمات وصحة لفظها وفهم المعنى العام للنص، حيث أن الكلمة الواحدة قد تحتمل عدة أوجه في التشكيل، لكل وجه معنىً مختلف. ولهذا السبب يواجه غير العربي صعوبة في قراءة وفهم النصوص العربية غير المُشَكَّلة.

وتتطلب العديد من تطبيقات اللغة العربية أن تكون النصوص والكلمات العربية مُشَكَّلة كي تكون نتائج هذه التطبيقات مقبولة، ومن هذه التطبيقات أنظمة التعرف الآلي على الكلام العربي.

نعرف التشكيل الآلي على أنه عملية إضافة علامات التشكيل المناسبة إلى النصوص غير المُشَكَّلة أو المُشَكَّلة جزئياً. وتسمي هذه العملية أحيانا باسترجاع علامات التشكيل. وعملية التشكيل الآلي شائعة في عدة لغات منها بعض اللغات اللاتينية واللغات السامية.

قمنا في هذا العمل البحثي بتطوير مكنز عربي مشكل تشكيلا كاملا، وطورنا طرقا للتشكيل الآلي للنص العربي. اعتمدنا في عملية تطوير المكنز على مكنز قد سبق إنشاؤه وسمي ب "SENTENCES3"، حيث قمنا بتعريض المكنز إلى معالجات تصحيحية للتأكد من صحة كلمات المكنز واكتمال التشكيل. وكانت النتيجة الوصول إلى كنز جديد أسميناه "تشكيل-2016" كما قمنا بالعمل على مكنز جديد وهو مكنز المصحف، ولقد اخترنا العمل عليه لتأكدنا من دقة المحتوى والتشكيل. عدا عن ذلك، قمنا بتطوير مكنز جديد استهدفنا في محتواه النصوص العربية المعاصرة، حيث إن مكنز "تشكيل-2016" اعتمد على النصوص التقليدية. سمّينا المكنز الجديد بمكنز "أخبار-2016" حيث انه اعتمد على النصوص الإخبارية فقط، ويحتوي المكنز على اكثر من 10 ملايين كلمة.

ويتركز الجزء الثاني من هذا العمل البحثي على تطوير طرق لتشكيل النص العربي آليا. واعتمد البحث على استنباط الخصائص التي تساهم في عملية التشكيل ودقته ومن ثم اختيار افضل مجموعة من هذه الخصائص، وللبدء في عملية التشكل تم تطوير برمجية تحتوي على عدة وحدات مترابطة مع بعضها مشكلة نظام التشكيل الآلي. ويستعمل نظام التشكيل أشجار القرار وبالأخص خوارزمية "WEKA J48" يضاف إليها مرحلة "بعد المعالجة" نستخدم فيها نماذج التكرار "N-Gram".

لقد تم إجراء العديد "من التجارب للحصول على أدق النتائج الممكنة وافضل نماذجٍ للتشكيل حيث حصل مكنز المصحف على افضل النتائج مقارنة بمكنز "تشكيل-2016".

كانت أفضل النتائج لأقل نسبة في الخطأ التشكيلي على مستوى الحرف لمكنز المصحف بنسبة 6% في حالة التشكيل دون تشكيل آخر حرف في الكلمة و9% في حالة التشكيل الكامل. وأما بالنسبة إلى مكنز "تشكيل-2016" فكانت نسبة الخطأ التشكيلي على مستوى الكلمة 18% في حالة التشكيل دون تشكيل آخر حرف في الكلمة و28% في حالة التشكيل الكامل.

CHAPTER 1

INTRODUCTION

Arabic is the fifth largest language in the world in terms of native speakers with about 300 million speakers [1]. It is one of the Semitic languages, which includes Amharic, Ethiopian, Assyrian, Babylonian, Hebrew and other languages [2]. Some of these languages use diacritic marks. Diacritic marks are added to letters to resolve possible word ambiguity. Adding such marks to an unvocalized word may change the pronunciation of the word and its meaning.

Arabic consists of 28 basic letters and 8 additional letters which represent writing variations. Table 1 shows the basic Arabic letters while Table 2 shows the writing variations. It is to note that an Arabic letter may have different shapes depending on its position in the text and surrounding letters. Table 3 shows the basic shapes that the letter "Ein" may have depending on its position in the word (first, internal or last).

Table 1: Arabic letters

#	Letter	Unicode Letter Name
1	ا	ARABIC LETTER ALEF
2	ب	ARABIC LETTER BEH
3	ت	ARABIC LETTER TEH
4	ث	ARABIC LETTER THEH
5	ج	ARABIC LETTER JEEM
6	ح	ARABIC LETTER HAH
7	خ	ARABIC LETTER KHAH
8	د	ARABIC LETTER DAL
9	ذ	ARABIC LETTER THAL
10	ر	ARABIC LETTER REH

#	Letter	Unicode Letter Name
11	ز	ARABIC LETTER ZAIN
12	س	ARABIC LETTER SEEN
13	ش	ARABIC LETTER SHEEN
14	ص	ARABIC LETTER SAD
15	ض	ARABIC LETTER DAD
16	ط	ARABIC LETTER TAH
17	ظ	ARABIC LETTER ZAH
18	ع	ARABIC LETTER AIN
19	غ	ARABIC LETTER GHAIN
20	ف	ARABIC LETTER FEH
21	ق	ARABIC LETTER QAF
22	ك	ARABIC LETTER KAF
23	ل	ARABIC LETTER LAM
24	م	ARABIC LETTER MEEM
25	ن	ARABIC LETTER NOON
26	ه	ARABIC LETTER HEH
27	و	ARABIC LETTER WAW
28	ي	ARABIC LETTER YEH

Table 2: Letters writing variations

#	Letter	Unicode Letter Name
1	ء	ARABIC LETTER HAMZA
2	آ	ARABIC LETTER ALEF WITH MADDA ABOVE
3	أ	ARABIC LETTER ALEF WITH HAMZA ABOVE
4	ؤ	ARABIC LETTER WAW WITH HAMZA ABOVE
5	إ	ARABIC LETTER ALEF WITH HAMZA BELOW
6	ئ	ARABIC LETTER YEH WITH HAMZA ABOVE
7	ة	ARABIC LETTER TEH MARBUTA
8	ى	ARABIC LETTER ALEF MAKSURA

Table 3: Different shapes of the letter Ein

In word Example	Shape
عالم	عـ
مصعب	ـعـ
ربيع	ـع
شجاع	ع

Arabic has 14 diacritical marks. Table 4 shows these diacritics.

Table 4: Diacritics

#	Diacritic	Unicode Diacritic Name		#	Diacritic	Unicode Diacritic Name
1	◌َ	ARABIC FATHA		**8**	◌ٌ	ARABIC DAMMATAN
2	◌ِ	ARABIC KASRA		**9**	◌َ◌ّ	ARABIC SHADDA WITH FATHA
3	◌ُ	ARABIC DAMMA		**10**	◌ِ◌ّ	ARABIC SHADDA WITH KASRA
4	◌ْ	ARABIC SUKUN		**11**	◌ُ◌ّ	ARABIC SHADDA WITH DAMMA
5	◌ّ	ARABIC SHADDA		**12**	◌ً◌ّ	ARABIC SHADDA WITH FATHATAN
6	◌ً	ARABIC FATHATAN		**13**	◌ٍ◌ّ	ARABIC SHADDA WITH KASRATAN
7	◌ٍ	ARABIC KASRATAN		**14**	◌ٌ◌ّ	ARABIC SHADDA WITH DAMMATAN

Diacritics can be categorized into four groups:

1. Short vowel diacritics, called "Tashkil". They are "Arabic Fatha", "Arabic Kasra" and "Arabic Damma”.
2. No vowel mark. The “Sukoon” (Arabic Sukun) denotes that the letter has no vowel.
3. Nunation diacritics called "Tanween". They are "Tanween-Fath" (Arabic Fathatan), "Tanween-Kasr" (Arabic Kasratan) and "Tanween-Damm" (Arabic Dammatan).
4. Gemination diacritics consonant marks called "Shaddah" (Arabic Shadda). It includes "Shaddah" with "Fatha", "Shaddah" with "Kasra", "Shaddah" with "Damma", "Shaddah" with "Fathatan", "Shaddah" with "Kasratan", and "Shaddah" with "Dammatan".

The remaining of this chapter is divided into 4 sections. Section 1.1 addresses the problem statement. Section 1.2, states the motivation behind the study. Thesis contribution is presented in Section 1.3. Section 1.4 layouts the thesis structure.

1.1 Problem Statement

Modern Arabic text is rarely written with diacritics. The absence of diacritical marks could cause ambiguity for the reader. Usually, native Arabic speakers face no difficulties in reading unvocalized text because they can deduce the diacritics from the context of the text, syntax of the language and their knowledge of the morphology of the words. However, non-native speakers would have difficulties in understanding the unvocalized text as a single word with different diacritics may have different word forms with different meanings and different pronunciations.

Restoring diacritics from unvocalized text is an active research area. Several studies have been pursued with different languages including French, Spanish, Persian, Arabic and Hebrew. Computer applications that rely on text processing are another reason that motivates researchers to tackle this problem. Examples of these applications are speech recognition [3] [4] [5], text to speech [6] [7], and automatic translation [8].

On the other hand, corpora developed to support Arabic text vocalization research need to be consistent in terms of content and vocalization as they should represent general Arabic text.

In this research work, we normalize a previously built corpus SENTENCES3 [9] and introduce the MUSHAF corpus [10] for being fully vocalized and prepared for automatic Arabic text vocalization. Also, we work on building a new news corpus (“Akhbar-2016”) based on MSA as the SENTECNES3 corpus is 90% classical.

Furthermore, we propose a way to automatically vocalize Arabic text using decision trees classifier. Many studies have been conducted on the area of text vocalization. The Studies showed that decision trees were scarcely used in vocalizing foreign text and up to our knowledge was never used for Arabic text vocalization.

1.2 Motivation

As mentioned before, Modern Arabic text is almost never written with diacritics. Ignoring diacritical marks in writing could cause difficulties in understanding the unvocalized text for non-native Arabic speakers, since a single word could have different diacritics and that would change the meaning and pronunciation of the different forms of the word.

Other motivations behind the study are the various computer applications that require the Arabic text to be vocalized. Applications such as speech recognition [3] [4] [5], text to speech [6] [7] and many other applications require text to be vocalized. Providing vocalized text for such applications will have a positive impact on functionality and performance of these applications.

1.3 Objectives

The main objectives of this thesis are:

1. Conduct an in depth study on the vocalization techniques of Arabic text.
2. Investigate and propose feature selection towards efficient vocalization and develop the needed tools to extract and prepare proper features.
3. Use decision trees to build a model for vocalization. Literature reported very little studies done on text vocalization using decision trees in general and Arabic on specific.
4. Determine the effectiveness of the proposed prototype.

5. Expand previously developed vocalized corpus [9] with at least 10 million extra vocalized words from modern Arabic text (MSA).

1.4 Thesis Structure

Figure 1 shows an overview of the thesis work. The work consists of two main parts: corpus development and text vocalization.

The rest of the thesis is organized as follow: Chapter 2 presents previous studies in the

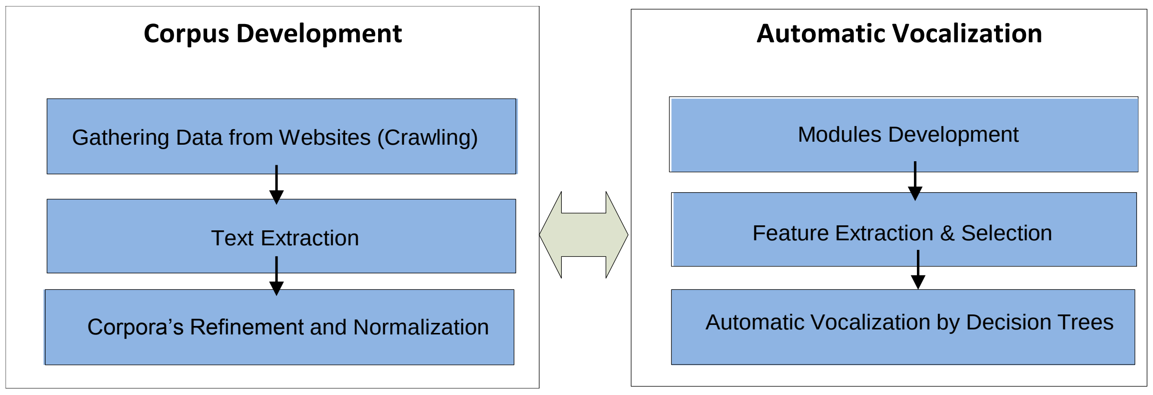

Corpus Development
Gathering Data from Websites (Crawling)
Text Extraction
Corpora's Refinement and Normalization
Automatic Vocalization
Modules Development
Feature Extraction & Selection
Automatic Vocalization by Decision Trees

Figure 1 Thesis work overview

subject. Chapter 3 describes the process of building and developing the new corpus and the improvement of other two corpora. Chapter 4 discuss features selection and extraction and introduces the developed models. Chapter 5 introduces the performance metrics used for evaluation and results compared with other relative work. Finally, the conclusion and future work are given in Chapter 6.

CHAPTER 2

LITERATURE REVIEW

Many studies have been conducted on the area of vocalization exploring different languages. In this section we briefly address related studies highlighting core information.

Section 2.1 presents previous work on Arabic and foreign languages. Section 2.2 introduces a previous study done at KFUPM.

2.1 Text Vocalization

Mihalcea [11], used learning techniques for the restoration of diacritics for Romanian text. The learning process was done at the letter level instead of the word level. The reasons behind choosing letters in the learning process instead of words were:

- Lack of large fully vocalized corpora.
- Unavailability of supportive tools such as morphological analyzers and syntactic analyzers.

Part of author experiment was to build a corpus based on Romanian. Articles were downloaded from the internet in HTML format and converted to text files. The size of the resulted corpus was around 3 million words. The learning algorithm chosen was based on instance learning. The features used were on the letter level and depended on its surrounding letters. The reported average accuracy was over 99%.

Same approach was used and tested on four other languages [12]. The authors used Czech, Hungarian, Polish and Romanian languages. The result was an average accuracy of over 98%.

Ya'akov [13], used Hidden Markov Model (HMM) for diacritics restoration for both Arabic and Hebrew. Phonetic accuracy was measured for Hebrew as part of the research. Two models were designed, unigram and bigram models. The corpus used for Hebrew was the Hebrew Bible and the corpus used for Arabic was the MUSHAF. In both models, 90% of each corpus was used as a training set and the remaining 10% went through undiacritization process and then used as a test set. The hidden state for both models were the diacritized words, each hidden state was linked to its corresponding undiacritized word which represented the observation. Using the unigram model, a word accuracy of 68% and 74% were achieved for Hebrew and Arabic respectively. As for the bigram model, an 81% accuracy and an 87% phonetic accuracy were achieved for Hebrew, while Arabic achieved an accuracy of 86%.

Crandall [14], used three approaches in his study on accent restoration of Spanish text. He used Bayesian framework, HMM bigram model and a combination of both. Due to the lack of large comprehensive corpora, the author created his own corpus using the typical approach of crawling and processing websites extracted information. The corpus created contained around 35 million words.

Bayesian framework was used to determine the word accentuation by looking at the surrounding words. Using the framework and testing with different window sizes, the best accuracy achieved was 99.1% with a window size of two (-/+). Increasing the window size decreased the accuracy, so a method for selecting the best window size was devised. Selecting the optimal window size resulted in a 99.2% accuracy. As for the second approach, HMMs bigram was used for the purpose of performing part of speech tagging (POS). Matching rules were applied to produce the data for training. Viterbi decoding was used for testing. An accuracy of 99.1% was achieved. Finally, the hybrid approach was used. This approach alternates between both approaches and takes the best result. An accuracy of 99.24% was achieved.

Elshafei et al [15], used HMMs unigram, bigram and trigram for diacritics restoration. They used a fully vocalized corpus of the MUSHAF. They extracted all the words in the corpus into a list, then constructed a frequency table with distinct words. To determine identical words, a metric was developed. A database was generated from mapping each unvocalized word to its possible vocalized forms. Another database was generated by constructing a two words sequence table (bigram).

The hidden state for the HMMs were the possible vocalized words and the observations were the unvocalized sequences of the words. To determine the best transition state, Viterbi algorithm was used. An error rate of approximately 4.1% was resulted. The authors stated that an error rate of 2.5% could be achieved by using trigrams and a preprocessing phase to clear out some of the error roots.

Same experiment was conducted on a different corpus [16]. The corpus used was developed by (KACST) and contains about 102 thousand words. Two databases were generated "unvowled word database" and "bigram database". Applying HMMs to this corpus resulted in a WER of 5.5%.

EL-Harby et al [17], conducted a research on the diacritics restoration of MUSHAF. The authors proposed two systems. The first was based on a unigram model while the other was based on bigram HMMs. The corpus used was a fully vocalized MUSHAF text.. A frequency table was constructed for each word in the corpus. The unigram system implemented determines the correct vocalized word by looking for the same word structure disregarding its diacritics. The result would be a set of different vocalized forms for the word. Then the word with the highest frequency is selected as the correct vocalized word. As for the bigram system, the hidden states represent the last letter diacritic of a vocalized word, while the observation represent none vocalized words. Viterbi algorithm was used to obtain the best transition sequence. Both systems were tested on different parts of different sizes of the corpus. The unigram based system was tested on 25%, 50%, 75% and 100%, and the results of accuracy were 94%, 94.3%, 93.4% and 92.5% respectively, while the bigram system resulted in 95.2%, 94.8%, 93.7% and 93% respectively.

Maher et al [18], presented their work on the diacritics restoration of Sindhi language. Sindhi is similar to Arabic. Missing diacritics on letters could change the word meaning and the context it is in. The corpus used was a book with the name "Shah Jo Risalo". The system implemented used n-gram models (unigram, bigram and trigram). The system resulted in three probabilities. A fourth probability was calculated from the multiplication of the previous probabilities. The system had three phases, the first was the tokenization, which breaks the input text into segments. Usually white spaces are used to segment words but some words in Sindhi are a combination of a word and a white space. This means that a word of two syllable exists. This led the authors to devise a new method for tokenization.

Second phase involves calculating the probabilities of the n-gram models and then the multiplication of their probabilities. For the third phase, Viterbi algorithm is applied to find the most likely path between the probabilities. The results obtained were a word error rate (WER) of 0.71% and diacritization error rate (DER) 3.21%.

Khorsheed [19], used HMMs and designed 14 models where each model represented a diacritic with an addition model that represented no diacritic. All 15 models formed what he called the global model. Hidden Markov Model toolkit (HTK) [20] was used for the development of these models. The corpus was built by the author and it contained over 24,000 sentences. The developed system consisted of two main phases. First, the Arabic text was coded into a sequence readable by HTK. Features were extracted for all the Arabic characters and diacritics along with the white space character which resulted in a 110 features. Another feature was added which represented the starting and the ending of a sentence. In the second phase, HTK was used to perform the experiments. Two sets of experiments were conducted. In each experiment the system was trained and tested with 10,000 sentences and a 20,000 sentences. The first set of experiments resulted in an average Correct Ratio (CR) of 72.76% and 72.80% and the second set resulted in a CR of 72.50% and 72.67% for the 10,000 and the 20,000 sentences respectively.

Hifny [21], used a statistical approach (bigram models) for Arabic diacritics restoration. He used higher n-gram models. The proposed algorithm (dynamic lattice search) calculates the probabilities of transition in lattices at run time. In his experiment, he used a corpus named "Tashkeela" which consisted of classical Arabic text. SRILM toolkit was used to build the language model. The best results achieved were a WER of 8.9% with case ending and a WER of 3.4% without case ending (WER2). These results were achieved using n-gram of order four.

In a related study, Hifny [22], applied smoothing techniques to improve the vocalization accuracy. The smoothing was suggested to be applied when the tested words are not in the training set. The smoothing techniques takes a portion of the observed n-gram probability and distributes it to the ambiguous n-grams. The author tried using three different smoothing techniques which were Katz Smoothing, Absolute Discounting, and Kneser-Ney smoothing. The results showed that using smoothing techniques may yield better accuracy.

Bebah et al [23], used a combination of both morphological analysis and HMMs hybrid approach. The morphological analyzer used was "AlkhalilMorpho". It was adjusted by adding a new lexicon. The lexicon included the most frequent words from all the available Arabic corpora which resulted in a 16,200 words corpus. Output of the analyzer was changed to produce the possible vocalizations and ignore other outputs. The analyzer was used to get all the possible vocalized words out of context. After that, HMMs were used to remove the ambiguity.

Two HMMs were used. The first model used unvocalized words as its observations and vocalized words as its hidden states. The second model had the same observations as the first model but for the hidden states possible diacritics were used. The system was tested using the original analyzer and using the modified version.

For the first model, testing resulted in a 21.11% of word error rate with case ending (WER1), 9.93% of word error rate without case ending (WER2), 7.37% of diacritic error rate with case ending (DER1) and 3.75% of diacritic error rate without case ending DER2. The results of the second model were a 21.41% WER1, 10.59% WER2, 7.47% DER1 and 3.95% DER2.

Harrat et al [24], used statistical machine translation for diacritics restoration of Algiers dialects. The main purpose of their study was to develop a speech translation system which translates from modern Arabic to Algerian dialect. Due to the importance of vocalization in the speech translation system, a vocalizer was needed. Two experiments were performed. First conducted experiment was on Arabic using two corpora "Tashkeela" and “LDC Arabic Tree Bank". Second experiment was done on Algerian where they created their own corpus since there were no resources on Algerian dialects. A statistical machine translation system based on the phrase level was built. Testing with corpora, "Tashkeela" corpus yielded 16.2% WER and 4.1% DER while "LDC" yielded 23.1% WER and 5.7% DER. For the Algerian corpus, a result of 25.8% WER and 12.8% DER were achieved.

Alghamdi and Muzafar [25], built an Arabic vocalizer using quad-gram probability model on the letter level. The corpus used was KDATD. The corpus was analyzed and a frequency table was constructed for four consecutive letters. The extracted frequency table was then used in the vocalization process. Testing with KDATD corpus a DER of 7.64% was achieved. Testing with another set of data that was taken from a newspaper resulted in an 8.87% DER. Both resulted in an average DER of 8.52%.

Shaalan et al [26], used a hybrid approach in building an Arabic vocalizer. The hybrid approach consisted of three methods which were "lexicon retrieval, bigram and SVM-statistical prioritized techniques" [26]. The corpus used was “LDC Arabic Tree Bank”. The first method (lexicon retrieval) takes an unvocalized word as its input and tries to find a single vocalized match for it in the lexicon. If a match is found then it is considered to be the correct vocalized word. If more than one match is returned then the second method (bigram) is used. As for the third method, POS tags are used to find the right vocalization. The best results achieved by the system were a WER of 12.16% and a DER of 3.78% both with case ending.

Rashwan et al [27], introduced a stochastic hybrid system for automatic Arabic vocalization. The hybrid system used two vocalizers, the first was morphological and the second was based on full form words. Using an Arabic corpus, a dictionary of full form words was built. The input is searched in the dictionary and if found, then all its possible vocalization is returned. To determine the most likely path for the sequence of vocalization, both n-gram and a lattice search are used. In case if the input was not found, words were factorized into their morphological possibilities, then again n-gram and a lattice search is performed to disambiguate the possibilities and find the most likely vocalization sequence. Testing the hybrid system on the “LDC Arabic Tree Bank” corpus resulted in a 12.5% WER and 3.8% DER with case ending. Without case ending, a results of 3.1% WER and 1.2% DER were achieved.

Haraty et al [28], designed a vocalization engine "Shakkel" for the restoration of diacritics of Arabic text. Corpus of the University of Leeds was used with several modifications mainly for POS tagging. The "Shakkel" engine receives the user input and then tokenizes it into words, then assigns a POS tags to the words. After that each word is searched in the corpus and a tuple value that corresponds to that word is retrieved. If a word is not found, then it will be tagged with "None". When the whole input words are tagged, the list of all

POS tags will be reprocessed again to assign tags to the "None" words tag. HMMs and rule based approaches were used in the implementation of the system. Metrics such as word error rate or accuracy were not used. Instead a program using the "Shakkel" engine was implemented using Python. Authors reported that the results were promising.

Rosenfeld [29], conducted a study on the restoration of both capitalization and diacritics of the Portuguese language. The study compared classifications between Naïve Bayes and Decision Trees classifier on the letter level. WEKA was used for the classification and both the Naïve Bayes and J48 classifiers were used. The data used in the experiments were from a recent Portuguese Wikipedia dump taken at the time of the study and it consisted of 83,610 words. Different window sizes (N) (on each side of the letter) were used in the experiments. For diacritics restoration, J48 classifier achieved 97.61% accuracy at (N=3) while NaiveBayes scored 95.19% at (N=1). As for the capitalization, an accuracy of 49.35% using NaiveBayes and 51.67% using J48 were achieved.

Al-Thwaib [30], carried out a research on Arabic text classification. She used two feature selection techniques: text summarization and term frequency. Each technique was applied separately on a set of documents. WEKA SVM was used in the classification process to predict the class of these documents. Results showed a remarkable increase in accuracy, precision and recall but suffered from execution time for the devised method.

Almuhareb et al [31], worked on Arabic text classification and specifically on poems. Their target was to be able to determine and classify a text as a poem or not. Several features were selected based on the nature of Arabic poems such as average-line-length, line-repetition-rate, diacritics-rate…etc. WEKA Decision Trees and Naive Bayes classifiers were used in classification. It was shown that decision tree classifier performed the best with 99.81% accuracy using all features proposed.

Table 5 summarizes the studies done on foreign languages and Table 6 summarizes the studies done on Arabic.

From the surveyed work, we can see that decision trees were used scarcely in the area of vocalization and as far as our knowledge, that decision trees have not been used in Arabic vocalization. Thus we focused on using decision trees techniques in our work.

Table 5: Studies on foreign languages

Author	Approach	Languages	Corpus	Size (words)	Accuracy	WER	DER
Mihalcea [11], 2002	Learning algorithms	Romanian	Custom	3,000,000	99% at letter level	N/S	N/S
Mihalcea et al [12], 2002	Learning algorithms	Czech, Hungarian, Polish and Romanian	Czech, Hungarian, Polish, Romanian	1,460,000, 1,720,000 2,500,00 and 3,000,000	98% at letter level	N/S	N/S
Gal [13], 2002	HMMs - unigram and bigram	Hebrew	Westminster Hebrew Morphologica	300,000	68% (unigram) and 81% (bigram) at word level	N/S	N/S
Crandall [14], 2005	Bayesian framework, HMM bigram and hybrid of both	Spanish	Custom	35,318,775	99.211% - 99.0501- 99.2433% (at letter level)	N/S	N/S
MAHER et al [18], 2011	HMMs - unigram, bigram and trigram	Sindhi	Shah Jo Risalo	27,360	Not specified	0.71 %	3.21 %
Gal [32], 2011	HMMs - unigram and bigram	Hebrew	Westminster Hebrew Morphological	300,000	80% at word level	N/S	N/S
Rosenfeld [29], 2014	WEKA Decision Tree & Naïve Bayes	Portuguese	Portuguese Wikipedia dump	83,610	97.61 for DT, 95.16 for NB at letter level	N/S	N/S

Table 6: Studies on Arabic

Author	Approach	Corpus	Size (words)	Accuracy	WER (CE)	WER (WCE)	DER (CE)	DER (WCE)
Gal [13], 2002	HMMs - unigram and bigram	MUSHAF	90,000	74% unigram, 86% bigram at word level	N/S	N/S	N/S	N/S
Elshafei et al [15], 2006	HMMs - unigram, bigram and trigram	MUSHAF	78,672	N/S	4.1%	N/S	N/S	N/S
Elshafei et al [16], 2006	HMMs - unigram and bigram	KACST	102,000	N/S	5.5%	N/S	N/S	N/S
EL-Harby et al [17], 2008	Unigram model and HMM bigram	MUSHAF	87,803	95.2% at word level	N/S	N/S	N/S	N/S
Khorshed [19], 2012	HMMs	Custom	200,000	72.80%	0.71 %	N/S	3.21 %	N/S
Hifny [21], 2012	Bigram model	Tashkeela	6,149,72 6	N/S	8.9%	3.4%	N/S	N/S
Hifny [22], 2012	Bigram model with smoothing techniques	Tashkeela	6,149,72 6	N/S	8.9%	3.4%	N/S	N/S
Bebah et al [23], 2014	Morphological analysis and HMMs	Tashkeela& RDI	2,463,35 1	N/S	21.1 1%	9.93%	7.37 %	3.75%
Harrat et al [24], 2012	SMT	Tashkeela and LDC Arabic Tree Bank	6,000,00 0 and 340,000	N/S	16.2 %	N/S	4.1 %	N/S
Alghamdi and Muzafar [25], 2007	Quad-gram model	KDATAD	N/S	N/S	N/S	N/S	7.64 %	N/S
Shaalan et al [26], 2009	Lexicon retrieval, bigram model and SVM	LDC Arabic Tree Bank	340,000	N/S	12.1 6%	31.86 %	3.78 %	7.92%
Rashwan et al [27], 2009	Morphological and full form words vocalizers	LDC Arabic Tree Bank	340,000	N/S	12.5 %	3.1%	3.8 %	1.2%
Haraty et al [28], 2013	HMMs and rule based approach	Corpus of University of Leeds	N/S	N/S	N/S	N/S	N/S	N/S

2.2 Automatic Diacritics Restoration for Arabic Text at KFUPM

Shaaban [9], proposed a hybrid system for automatic restoration of Arabic text. Part of his work was the development of new comprehensive corpus. The development resulted in a "General Corpus". This corpus was a synthesis of both vocalized and unvocalized words containing a 1,587,511,592 billion words. According to the author it was the largest and most comprehensive corpus up to the date of the study. Vocalized text was extracted which resulted in another vocalized corpus that have 30,169,610 million words.

The other part of the research work was the development of a prototype for diacritics restoration and the experimental work related to the development. The experiments he conducted used a hybrid approach which combined, rule-based and statistical approaches. For the statistical part, a custom N-gram extraction tool was built to better meet the author's needs. The tool built aimed to generate letter-grams, word-grams and POS-grams. It also had the option of including non-Arabic words, numbers or punctuation. The tool had the capability to deals with POS-grams in a proper manner in case of a word having multiple POS tags for the same diacritical form.

After applying the N-gram extraction and to get the best possible vocalization, a greedy algorithm without backtracking was applied for both letter and word grams. The same was also applied for the POS with a difference of handling multiple diacritical forms for a single POS tag.

By developing the vocalized corpus, the author was able to infer a set of rules. These inferred rules were used in the rule-based part and were divided into three parts as quoted by the author [9]:

- Diacritics assumed to be missing and not part of the feature set.
- Diacritics assumed to be missing with the addition of contextual feature such as Previous Letter, Previous Word, Next Letter or Next Word.
- Diacritics are part of the feature set and the current letters diacritic was removed.

The vocalizer system consists of six main components which are:

- User interface
- N-gram statistical component
- Rule-based component
- The vocalizer component which handles the use of both the n-gram and the rule-based components.
- Utilities which include many tools that perform different functionality such as: the tokenization of words, a normalizer for diacritization and other tools.

The vocalizer system starts by receiving input text from the user along with specifying the vocalization methods (Statistical or Rule-based) order. The order has major effect on the final result. For the evaluation of the system, four systems were considered for comparison. The systems were: 1) Arabi NLP [33] 2) Mishkal [34] 3) AraDiac [35] 4) Sakhr [36]

The results showed that the implemented hybrid system have a DER (case ending) with a 3.511% which is better than a 6.577% and 11.663% for both Arabi-NLP and Mishkal respectively. On the other hand, it was found the author's system performed worse than Sahkr's with a 2.905%. For WER (case-endings) it was shown that the hybrid system performed better than the other two systems but not the third respectively. As for the vocalization level, it achieved an 81.672% which outperform the first two but behind the third as it has a 99.26% diacritization level.

2.3 Summary

In this chapter we have surveyed some of the related studies on vocalization. The studies covered both Arabic and non-Arabic languages. Many solutions and techniques have been introduced. We have noticed that WEKA was not used much on our target research area. Using WEKA will be the primary focus of our thesis. Also, we presented a previous study that was done at KFUPM. We will use the built corpus as a bases to our corpus.

CHAPTER 3

CORPUS DEVELOPMENT

In this chapter we explain the normalization process of the previously built corpus (SENTENCES3) [9]. We also introduce the MUSHAF corpus as another valid and consistent resource. Furthermore, the process of developing a new corpus will be detailed. The new corpus in which we called “AKHBAR-2016” corpus will contain only Modern Standard Arabic (MSA). The reason for specifying the content to be MSA is because the SENTENCES3 corpus had 90% of its content as Classical Text.

In section 3.1 we enumerate through all the steps that creates a fully vocalized and consistent “Tashkeel-2016” corpus depending on the previously prepared SENTENCES3 corpus. In section 3.2, we present statistics on the “Tashkeel-2016” corpus. Section 3.2, introduces the MUSHAF corpus and the adjustments we made to it, while in section 3.4, we present some statics on the MUSHAF corpus. Section 3.5 describes the criteria used in developing the “AKHBAR-2016” corpus. The process of selecting the target websites for data gathering, the tool used for crawling, text extraction and the vocalization process are described. Finally, we conclude with summary in section 3.6.

3.1 “Tashkeel-2016” Corpus

Our objective from studying this corpus was to ensure that we have a trusted and a consistent resource of Arabic text. Upon investigating the corpus we have found many inconsistencies. Some of these where: unvocalized words, wrong vocalization, long words, foreign letter, and others. The following subsections explains some of these inconsistencies and how they were addressed.

3.1.1 Unvocalized Words

In Arabic, some letters tend to be normally unvocalized. However, since we are going to use this corpus for classification, then it is important that every letter in a word is explicitly vocalized. To ensure that words are fully and consistently vocalized, we automatically add diacritics to some of the letters by applying the following rules:

- If a letter has a "Fatha" diacritic and is followed by an unvocalized letter Alef ("ا"), then the letter Alef will be vocalized with "Sukoon".
- If a letter has a "Kasra" diacritic and is followed by an unvocalized letter Yaa ("ي"), then the letter Yaa will be vocalized with "Sukoon".
- If a letter has a "Fatha" diacritic and is followed by an unvocalized letter Alef-Maqsora ("ى"), then the letter Alef-Maqsora will be vocalized with "Sukoon".
- If a letter has "Tanween-Fath" diacritic and is followed by an unvocalized letter Alef ("ا"), then the letter Alef will be vocalized with "Sukoon".
- If a letter has "Tanween-Fath" diacritic and is followed by an unvocalized letter Alef-Maqsora ("ى"), then the letter Alef-Maqsora will be vocalized with "Sukoon".
- If a letter has "Damma" diacritic and is followed by an unvocalized letter Waw ("و") then a letter Alef ("ا"), then both letters Waw and Alef will be vocalized with "Sukoon".
- If a letter Alef ("ا") without Hamza ("ء") is appearing at the beginning of a word and this letter Alef is followed by a letter Lam ("ل") with “Sukoon” then we put “Sukoon” on the letter Alef.

3.1.2 Incorrect Vocalization

This is cleaning position violations of Diacritics. One example was the diacritic “Shadda” at the first letter of a word. All starting letters of words were checked for “Shadda” diacritic and if found, was removed.

Furthermore, we encountered diacritics that were scattered around the lines and were not placed on any letter. We searched for all of these diacritics and removed them. Table 7 shows two words with “Shadda” violations at the first letter.

Table 7: Examples of “Shadda” violation at the start of the word

Word
سُّكَّانِ
تَّغَيَّرَ

3.1.3 Long Words and Lines

We identified words that have more than 10 letters excluding diacritics. Upon examination of these words, we found that they were a combination of two words or more concatenated together. We removed all lines containing these words. Table 8 shows two examples of long words.

Table 8: Long words

Word
اَلْبَيَانَاتِاُكْتُبْ
التَّعَاوُّنِيُّاَلْتَّدْرِيبُ

We also noticed that some lines were very long in term of words, other lines contain multiple lines (more than one full-stop mark). To address the issue, we decided that each line should contain a number of words between eight and twenty. The normalization was done by applying the following set of rules in the same order (only one rule is applied per line):

- Check for multiple full-stop marks. If found the line is split by the full-stop mark.
- Count the number of words in a line and if they exceeds twenty words, then we look for several punctuation marks to split the line by them. Table 9 shows the punctuation marks used for splitting. Note that priority for using a mark for splitting a line, was given in the same top-down order in Table 9. A study of choosing better precedence of punctuation marks may be needed in related future work.
- The resulted lines were checked again until all lines that match these rules were collected and others were rejected.

3.1.4 Foreign Letters and Words

Non-Arabic words and letters were found between the Arabic texts. We searched for all Non-Arabic letters and removed them.

3.1.5 Words with Numbers

Some words were concatenated with numbers. These words were split and a whitespace was added between the number and the word.

3.1.6 Punctuation Marks Inconsistences

We encountered some lines starting with a punctuation mark or a bullet. We removed the occurrence of punctuation marks and bullets at the beginning of lines. Also, consecutive punctuation marks were found, so we kept only one of these marks.

After filtering, we traversed the corpus and identified all the lines that contains words that were not fully vocalized and removed them. We claim that our corpus has 100% fully vocalized text.

Table 9: Split marks

Mark
;
،
?
!
,

3.2 "Tashkeel-2016" Statistics

We present in this section some statistics of the updated "Tashkeel-2016" Corpus.

Table 10 shows the corpus the number of times a diacritic occurs. Table 11 and Table 12 show letters - diacritics distributions in the updated "Tashkeel-2016" Corpus. The tables show the occurrence percentage for a letter and all its possible diacritics.

Table 10: “Tashkeel-2016” diacritics frequencies

Diacritic	Frequency
◌َ	4742398
◌ِ	1910834
◌ُ	1270679
◌ْ	3800880
◌ً	107358
◌ٍ	147140
◌ٌ	112266
◌ّ◌َ	307745
◌ّ◌ِ	73974
◌ّ◌ُ	43974
◌ّ◌ً	6662
◌ّ◌ٍ	9074
◌ّ◌ٌ	7732

Table 11: “Tashkeel-2016” Letters - diacritics distribution part1

Letter	Diacritic						
	◌َ	◌ُ	◌ِ	◌ْ	◌ً	◌ٌ	◌ٍ
ء	25.382%	13.104%	30.030%	0.002%	4.169%	13.874%	13.439%
آ	100.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
أ	90.055%	5.724%	0.095%	3.905%	0.094%	0.077%	0.048%
ؤ	28.471%	26.173%	0.133%	44.471%	0.322%	0.173%	0.259%
إ	0.000%	0.000%	99.968%	0.000%	0.000%	0.000%	0.032%
ئ	9.336%	4.345%	70.952%	5.110%	9.506%	0.299%	0.452%
ا	0.000%	0.000%	0.000%	100.000%	0.000%	0.000%	0.000%
ب	35.997%	9.119%	39.814%	8.772%	1.055%	0.993%	1.474%
ة	15.908%	12.970%	32.087%	0.005%	12.360%	12.308%	14.363%
ت	53.470%	13.694%	14.829%	11.475%	0.292%	0.478%	1.103%
ث	38.430%	27.959%	13.659%	14.507%	1.595%	1.356%	1.581%
ج	41.618%	19.023%	18.081%	15.280%	0.268%	0.327%	0.421%
ح	51.769%	9.656%	15.569%	17.232%	0.563%	0.866%	0.936%
خ	47.090%	13.452%	14.036%	22.860%	0.278%	0.404%	0.469%
د	34.814%	12.506%	19.740%	12.099%	2.506%	2.911%	4.886%
ذ	70.577%	5.937%	10.863%	9.051%	0.308%	0.169%	1.157%
ر	40.235%	12.472%	22.402%	11.955%	2.026%	2.250%	3.392%
ز	46.010%	13.962%	23.081%	10.528%	0.923%	1.516%	1.017%
س	42.397%	8.644%	15.481%	28.703%	0.760%	0.840%	1.569%
ش	55.272%	6.666%	9.453%	25.717%	0.408%	0.457%	1.232%
ص	42.025%	9.386%	20.330%	21.235%	0.572%	0.567%	0.661%
ض	38.431%	13.272%	25.486%	9.653%	5.848%	1.578%	3.776%

Letter	Diacritic						
	◌َ	◌ُ	◌ِ	◌ْ	◌ً	◌ٌ	◌ٍ
ط	45.763%	10.242%	16.902%	19.773%	0.867%	1.115%	1.751%
ظ	46.855%	18.077%	16.697%	12.798%	0.912%	0.787%	1.587%
ع	59.136%	8.834%	12.184%	16.572%	1.167%	0.948%	1.058%
غ	72.171%	7.921%	7.260%	11.736%	0.277%	0.189%	0.328%
ف	52.592%	5.254%	30.587%	7.374%	0.785%	0.857%	0.910%
ق	59.590%	11.080%	13.644%	9.735%	0.786%	0.938%	0.809%
ك	58.465%	19.606%	9.839%	8.028%	0.342%	0.795%	0.789%
ل	39.060%	6.131%	17.020%	30.754%	0.753%	0.903%	0.925%
م	39.303%	15.784%	20.078%	15.887%	0.800%	0.914%	1.304%
ن	31.694%	4.792%	10.413%	37.109%	0.737%	0.705%	1.173%
ه	19.495%	45.634%	31.201%	2.978%	0.102%	0.203%	0.236%
و	57.428%	1.372%	1.603%	37.753%	0.053%	0.029%	0.054%
ى	0.00%	0.000%	0.000%	100.000%	0.000%	0.000%	0.000%
ي	27.246%	8.480%	0.477%	56.246%	0.230%	0.040%	0.042%

Table 12: “TASHKEEL-2016” letters - diacritics distribution part2

Letter	Diacritic					
	◌ًّ	◌ٌّ	◌ٍّ	◌َّ	◌ُّ	◌ِّ
ء	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
آ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
أ	0.000%	0.000%	0.000%	0.001%	0.000%	0.000%
ؤ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
إ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ئ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ا	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ب	0.037%	0.037%	0.030%	1.358%	0.559%	0.756%
ة	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ت	0.017%	0.011%	0.034%	4.233%	0.120%	0.244%
ث	0.005%	0.013%	0.013%	0.417%	0.067%	0.397%
ج	0.051%	0.028%	0.128%	3.026%	0.529%	1.220%
ح	0.003%	0.002%	0.004%	1.694%	1.354%	0.351%
خ	0.000%	0.001%	0.007%	0.846%	0.071%	0.487%
د	0.219%	0.134%	0.167%	6.995%	1.253%	1.769%
ذ	0.016%	0.047%	0.013%	1.156%	0.134%	0.572%
ر	0.145%	0.211%	0.198%	3.017%	0.668%	1.029%
ز	0.024%	0.017%	0.080%	1.997%	0.293%	0.553%
س	0.024%	0.005%	0.013%	0.895%	0.155%	0.514%
ش	0.009%	0.019%	0.019%	0.454%	0.059%	0.235%
ص	0.244%	0.230%	0.162%	3.120%	0.803%	0.665%
ض	0.018%	0.016%	0.006%	1.440%	0.129%	0.348%
ط	0.044%	0.016%	0.028%	2.245%	0.689%	0.564%
ظ	0.104%	0.133%	0.112%	0.841%	0.327%	0.771%

Letter	Diacritic					
	◌ًّ	◌ٌّ	◌ٍّ	◌َّ	◌ُّ	◌ِّ
ع	0.000%	0.000%	0.000%	0.083%	0.004%	0.013%
غ	0.000%	0.000%	0.000%	0.082%	0.003%	0.034%
ف	0.023%	0.012%	0.031%	1.111%	0.086%	0.379%
ق	0.129%	0.195%	0.174%	1.164%	0.728%	1.028%
ك	0.019%	0.013%	0.040%	1.414%	0.138%	0.510%
ل	0.037%	0.046%	0.075%	2.428%	0.762%	1.107%
م	0.035%	0.037%	0.075%	4.897%	0.288%	0.596%
ن	0.031%	0.016%	0.026%	12.545%	0.150%	0.608%
ه	0.000%	0.000%	0.000%	0.104%	0.021%	0.027%
و	0.014%	0.015%	0.018%	1.303%	0.079%	0.279%
ى	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ي	0.318%	0.423%	0.446%	3.193%	0.975%	1.884%

3.3 MUSHAF Corpus

The motive behind working on the MUSHAF corpus is because we are sure that the corpus is accurately diacritized. The text of MUSHAF can be acquired at [10]. The website provided several versions of the MUSHAF text from simple text to Uthmani text. At the current of accessing it, the site also provided the option of downloading the MUSHAF in different file formats. When downloading the MUSHAF, the user is given the option to include the following in the MUSHAF text:

- Pause marks.
- Sajdah signs (۩).
- El-hizb signs (۞).
- Superscript alefs (like in إِلَىٰ).

The version we have chosen was the simple version in text format. We excluded all the options mentioned above, as the inclusion of these characters is beyond our work.

As the simple text includes the diacritics reflecting TAJWEED rules, we needed to filter these to reflect MSA writing. We analyzed the corpus and made some adjustments to the text:

- Some words first letters had “Shadda” diacritic which reflect TAJWEED rules. Thus we removed the “Shadda” in that case.
- We found that some words were not fully vocalized as some letters were without diacritics reflecting TAJWEED rules. We generated a list of the target words and letters and based on our analysis we decided to vocalize all unvocalized letters with “Sukoon”. This processes included the unvocalized letters due to “Edgham” (إدغام), “Ekhfaa” (إخفاء), and "Eqlab" (إقلاب).
- We generated a list of all unvocalized words and letters, then we analyzed the list and decided to vocalize all remaining unvocalized letters with “Sukoon”.
- Due to the fact that the version of the corpus we downloaded was written as a simple text and not in Uthmani writing, we found six cases that needed to be fixed. Addressing these cases were also essential so that the corpus would be in line with Quran morphology corpus which was used for the MUSHAF POS tagger in which will be explained in the next chapter 4.3.3.2). The cases were:
    - All words that start with (يَاْ) were followed by a space. This does not match the original writing in MUSHAF. i.e. (يَاْ أَيُّهَاْ). To fix this the whitespace is removed from any word that is (يَاْ) followed by a whitespace.
    - All words that start with (هَاْ) were followed by a space. This does not match the original writing in MUSHAF. i.e. (هَاْ أَنْتُمْ). To fix this the whitespace is removed from any word that starts by (هَاْ) and followed by a whitespace.

- All words that start with (وَيَاْ) were followed by a space. This does not match the original writing in MUSHAF. i.e. (وَيَاْ آدَمُ). To fix this the whitespace is removed from any word that starts by any (وَيَاْ) and followed by a whitespace.
- Removed whitespace from (يَاْبْنَ أُمَّ).
- Removed whitespace from (إِلْ يَاْسِيْنَ).
- Removed white space from (وَأَنْ لَوِ)

After fixing all the issues mentioned above, we indexed the corpus. The indexing was done on each line and not by Sora. We added an index number for each line followed by a character "|".The indexing was done because it was necessary for using the MUSHAF POS tagger.

3.4 Some Statistics on the MUSHAF Corpus

We present in this section some statistics of the MUSHAF's Corpus.

Table 13 shows the frequency of each diacritic in the corpus.

Table 13: MUSHAF diacritics frequencies

Diacritic	Class Value
◌َ	111915
◌ِ	43551
◌ُ	36216
◌ْ	117087
◌ً	3472
◌ٍ	2542
◌ٌ	2385
◌َ◌ّ	9523
◌ِ◌ّ	2419
◌ُ◌ّ	1104
◌ً◌ّ	270
◌ٍ◌ّ	91
◌ٌ◌ّ	134

Table 14 and Table 15 show letters - diacritics distribution. They show the occurrence percentage for each letter and all its possible diacritics.

Table 14: MUSHAF letters - diacritics distribution part1

Letter	Diacritic						
	◌َ	◌ُ	◌ِ	◌ْ	◌ً	◌ٌ	◌ٍ
ء	37.833%	20.722%	17.934%	0.000%	5.513%	3.359%	14.639%
آ	0.000%	0.000%	0.000%	100.000%	0.000%	0.000%	0.000%
أ	84.713%	9.628%	0.000%	5.582%	0.044%	0.033%	0.000%
ؤ	4.903%	16.345%	0.149%	77.860%	0.446%	0.297%	0.000%
إ	0.000%	0.000%	99.824%	0.000%	0.000%	0.000%	0.176%
ئ	9.306%	7.614%	64.975%	10.237%	7.445%	0.000%	0.423%
ا	0.000%	0.000%	0.000%	100.000%	0.000%	0.000%	0.000%
ب	29.319%	9.181%	35.550%	9.956%	1.897%	1.680%	1.297%
ة	14.761%	10.452%	21.800%	0.000%	21.630%	15.102%	16.254%
ت	47.006%	20.751%	16.302%	6.644%	0.437%	0.466%	1.749%
ث	34.441%	35.078%	11.174%	16.054%	1.202%	0.141%	0.990%
ج	46.518%	12.662%	14.652%	19.837%	1.296%	0.693%	0.784%
ح	45.797%	9.396%	16.570%	24.444%	1.884%	0.821%	0.845%
خ	51.822%	7.809%	16.139%	21.986%	0.160%	0.200%	0.040%
د	23.736%	21.666%	20.915%	16.875%	5.358%	2.320%	2.270%
ذ	38.909%	6.427%	36.740%	9.874%	0.831%	0.020%	1.480%
ر	36.999%	17.028%	17.504%	14.722%	4.684%	3.564%	1.871%

Letter	Diacritic						
	◌َ	◌ُ	◌ِ	◌ْ	◌ً	◌ٌ	◌ٍ
ز	37.273%	12.070%	27.142%	12.320%	1.438%	1.751%	0.688%
س	45.509%	13.789%	15.452%	21.790%	0.532%	0.516%	0.948%
ش	59.840%	7.439%	9.981%	19.492%	0.282%	0.047%	0.188%
ص	44.788%	9.749%	22.490%	18.967%	0.338%	0.241%	0.483%
ض	31.969%	13.938%	31.435%	13.879%	1.720%	1.008%	3.677%
ط	46.976%	9.662%	21.838%	11.783%	1.964%	1.257%	2.907%
ظ	41.266%	24.150%	19.578%	10.434%	1.407%	1.055%	0.821%
ع	55.736%	9.899%	10.494%	21.414%	1.329%	0.734%	0.298%
غ	52.580%	11.712%	5.487%	28.911%	0.491%	0.491%	0.328%
ف	51.446%	8.643%	28.810%	7.465%	0.903%	0.583%	0.412%
ق	50.796%	22.633%	11.715%	7.378%	1.351%	0.611%	0.810%
ك	44.908%	38.220%	8.164%	6.192%	0.200%	0.229%	0.095%
ل	37.925%	6.172%	11.550%	33.589%	0.979%	0.382%	0.479%
م	29.991%	14.206%	19.488%	28.218%	1.227%	1.679%	1.231%
ن	45.985%	6.975%	6.909%	26.142%	0.693%	0.524%	0.785%
ه	21.623%	45.926%	27.771%	3.845%	0.222%	0.229%	0.114%
و	47.423%	0.512%	0.592%	50.296%	0.125%	0.020%	0.000%
ى	0.000%	0.000%	0.000%	100.000%	0.000%	0.000%	0.000%
ي	26.983%	7.814%	0.332%	59.833%	0.086%	0.036%	0.000%

Table 15: MUSHAF letters - diacritics distribution part2

Letter	Diacritic					
	◌ً◌ّ	◌ٌ◌ّ	◌ٍ◌ّ	◌َ◌ّ	◌ُ◌ّ	◌ِ◌ّ
ء	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
آ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
أ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ؤ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
إ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ئ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ا	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ب	0.096%	0.009%	0.009%	2.524%	2.132%	6.353%
ة	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ت	0.000%	0.000%	0.000%	6.407%	0.086%	0.152%
ث	0.071%	0.000%	0.000%	0.636%	0.071%	0.141%
ج	0.030%	0.000%	0.030%	1.990%	0.452%	1.055%
ح	0.000%	0.000%	0.000%	0.193%	0.000%	0.048%
خ	0.000%	0.000%	0.000%	1.522%	0.000%	0.320%
د	0.267%	0.017%	0.017%	3.388%	2.003%	1.168%
ذ	0.000%	0.000%	0.000%	3.021%	0.020%	2.676%
ر	0.218%	0.145%	0.097%	1.637%	0.669%	0.863%
ز	0.188%	0.000%	0.000%	4.378%	0.500%	2.251%

Letter	Diacritic					
	◌ًّ	◌ٌّ	◌ٍّ	◌َّ	◌ُّ	◌ِّ
س	0.017%	0.000%	0.000%	0.965%	0.283%	0.200%
ش	0.000%	0.000%	0.000%	0.659%	0.047%	2.024%
ص	0.000%	0.000%	0.000%	1.593%	0.627%	0.724%
ض	0.000%	0.000%	0.000%	1.601%	0.652%	0.119%
ط	0.000%	0.000%	0.000%	2.907%	0.079%	0.628%
ظ	0.469%	0.000%	0.234%	0.117%	0.000%	0.469%
ع	0.011%	0.000%	0.000%	0.043%	0.021%	0.021%
غ	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ف	0.091%	0.000%	0.034%	1.212%	0.091%	0.309%
ق	0.256%	0.242%	0.114%	1.251%	1.024%	1.820%
ك	0.029%	0.010%	0.133%	1.372%	0.029%	0.419%
ل	0.060%	0.089%	0.045%	7.499%	0.456%	0.775%
م	0.097%	0.011%	0.007%	3.602%	0.082%	0.161%
ن	0.015%	0.022%	0.000%	10.957%	0.180%	0.814%
ه	0.000%	0.000%	0.000%	0.182%	0.000%	0.088%
و	0.081%	0.093%	0.012%	0.709%	0.012%	0.125%
ى	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
ي	0.469%	0.137%	0.123%	1.288%	1.120%	1.779%

3.5 The “AKHBAR-2016” Corpus

The first step in developing the corpus is to choose the target domains to be crawled. Since we already decided that we want only MSA text, we have chosen to go with news websites. News websites cover several domains as they do not only contain news, but also sport, art…etc.

To select the websites to crawl, we tried to be subjective in the selection process. We used Alexa [37], a website ranking system based on traffic metrics. We searched through Alexa to get Arabic news websites that have the highest global ranking. Table 16 shows the list of websites selected with their ranking at the time they were accessed.

Table 16: Crawled websites list

#	Website	Url	Rank	Last Accessed
1	BBC News - Arabic	http://www.bbc.com/arabic/	107	September, 2015
2	Alwakeel News	http://www.alwakeelnews.com/	3,323	September, 2015
3	Saraya News	http://www.sarayanews.com/	4,449	September, 2015
4	Arabian Business - Arabic	http://www.arabic.arabianbusiness.com/	10,407	September, 2015

After identifying the list of websites to be crawled, a tool was needed for the crawling process. The tool we used in which we have found to be efficient and flexible is called WinHTTrack-Website-Copier [38]. The tool can download a website and built the same website structure allowing you to navigate offline. In crawling the websites, we limited the files to be downloaded to HTML files by specifying the files extensions. Also, we found that some of the navigated URLs are irrelevant so we prevented the crawling of these URLs by specifying certain query parameters.

3.5.1 Text Extraction

While there are several tools for text extraction, we decided to build our own tool. One of the main reasons that lead us to this decision was to avoid extracting text that would make the corpus inconsistent in term of content. Since we are dealing with news websites then it is expected that each page could have a comment section. The comment section could contain slang Arabic which is called “Aamieh”. Extracting such text will cause problems since many of the slang words are not available in Arabic dictionaries. Also the text could contain content from advertisements or any kind of unrelated content.

To extract specific portions of the text from the crawled HTML pages, we faced several challenges. Some of these challenges were:

1. We needed to identify the HTML tag that holds the news details.
2. The identified HTML tag could change if the website went through some updates. Since the amount of crawled data was large, it covered around one to three years of content. Thus we needed to check a range of the crawled pages to make sure that if there were any changes to be taken into consideration.
3. Each crawled website used a different HTML tag for its news description. Hence, we needed to identify the tag for each website.

After the identification of the target HTML tags for each website, the tool was built to iterate through all the crawled HTML pages and extract the text. To ensure the consistency of the extracted text, the text went through several filtration processes which are explained in the following subsections.

3.5.1.1 Foreign Words and Letters

We removed HTML tags in the text. We found out that sometime tags that make the text bold or italic or tags that change the font size were injected inside the text. Also, we took out escape words, extra spaces between words, symbols and foreign letters (Non-Arabic).

Furthermore, we replaced Arabic letters in different fonts or formats with standard format letters, the reason for replacing these letters is that they were saved in different Unicode presentations (using the Unicode of the shape of the letter). This assures that the text is in standard Unicode. Similarly, Arabic numbers were replaced with English numbers for the same reason.

3.5.1.2 Punctuation Marks

The full-stop punctuation mark (.) is supposed to be connected to the last letter in a sentence, but this was not always the case as we found some lines having space(s) before the (.) mark. We removed all the spaces and connected the mark with the last word. Furthermore, we handled space issues with parentheses brackets, i.e. space inside the bracket at the start or end of the bracket or spaces before brackets…etc.

3.5.1.3 Lines

We removed any line that have less than 100 characters. We also removed empty and duplicate lines.

Table 17 shows basic statistics on the corpus after extracting text and addressing all inconstancies mentioned. As we can see from the statistics that the corpus have a very low diacaritzititon level. This is expected due to the nature of the domain selected.

Table 17: Crawled corpus statistics

Lines Count	278,562
Words Count	10,579,257
Unique Words Count	591,318
Letters Count	50,393,237
Diacritized Letter Count	122,875
Diacritization Level	0.2%

3.5.2 Corpus Vocalization

To vocalize the corpus, the vocalization steps mentioned in section 3.1.1 were applied. After that, best performing models as described in chapter 5.3 were chosen. Both voting and N-Gram vocalization were used for the best results possible. The steps to vocalize a text file is enumerated in Appendix I.

3.6 Summary

In this chapter, we presented the steps followed in normalizing the SENTENCES3 corpus leading to the new fully vocalized “Tashkeel-2016” corpus. We also presented the preparation of MUSHAF corpus for text vocalization, the statistics of letters and diacritics in the two corpora, and the approach followed in the development of “AKHBAR-2016” corpus in term of websites selection, tools for crawling and the process of vocalization.

CHAPTER 4

Text Vocalization

When building a model for classification, we are basically describing a dataset. The dataset can be described by the attributes (features) it has. The number of attributes and their types varies depending on the nature of the data. Thus extracting features is very important as it reflects the characteristics of the data. Feature extraction is the process of creating new features or the use of existing ones to come up with new features. If the amount of information (features) used to describe a dataset is big, then it would be hard to have a clear understanding of the data [39].

Having too many features may include irrelevant or correlated ones. Irrelevant features are considered as noise. Using such features when building a model will most likely increase the size of the model. Thus, more computational power will be needed in building the model. Correlated features on the other hand, are several attributes that are trying to describe the same thing. As a consequence, they may contribute nothing or they may end up reducing the predictive power of the built model.

Feature selection is the process of extracting an optimal number of features that are not redundant nor correlated but are descriptive and accurate [40]. This would reduce the number of features and as a consequence could reduce the model size and the time needed to build the model. It may also increase the model accuracy [41].

While features are an essential and critical part of classifications, other parts are important as well.

Figure 2 shows the proposed vocalization system architecture. The system consists of different modules: Input, Settings, Instant Diacritizer, Data Preparation, and Feature Extraction, Post-processing, and Output modules. The system works as follows: first, an input text that is line formatted is provided to be vocalized. Several settings are set and given to the diacritizer, along with the input. The instant diacritizer delegates the input and the required settings to the data preparation module. The data preparation module sends the input to the feature extraction module. The features will be extracted and then sent back. Afterward, the training and testing files will be generated. These files, will be used for classifications. The vocalized text will be built from the result of the classifications. If post-processing option was selected, then the vocalized text will go through extra processing to enhance its vocalization.

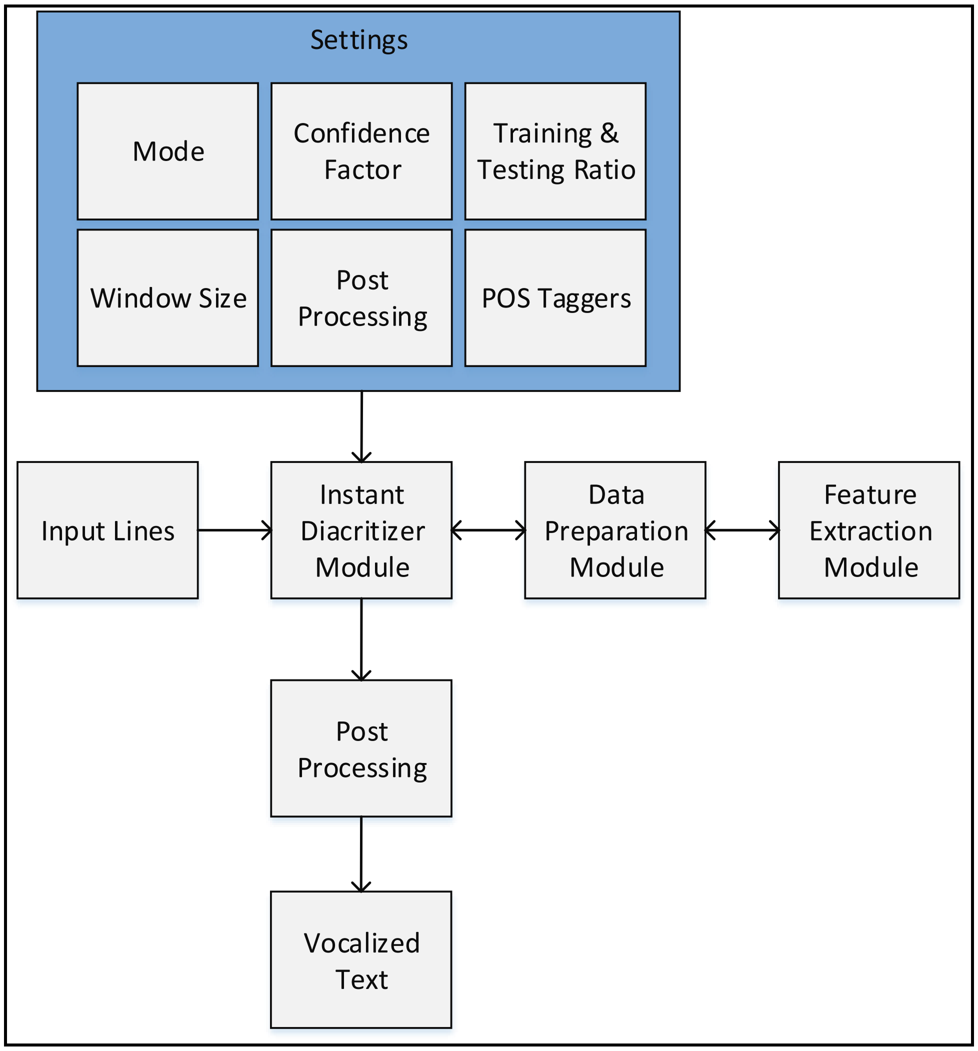


Figure 2 Vocalization System Architecture

The remaining of this chapter presents some technical details related to the vocalizer and its modules. In section 4.1 WEKA data mining software is introduced. In section 4.2, we describe the features we extracted. Section 4.3 presents the modules of the vocalization system. The selection process of the best set of features is discussed in section 4.4. Section 4.5 introduces the post-processing used to enhance vocalization. Some impletion issues is discussed in Section 4.6. Finally, the summary of the chapter is presented in section4.7.

4.1 WEKA Data Mining Software

WEKA [42] is an open source data mining tool t developed in Java programming language. It contains a vast number of classification algorithms. It also contains tools for data processing, regression, clustering…etc. it is considered a suitable environment for the development of new classifiers [42].

WEKA can be used in different ways. It can be used from either the interface or command lines. Furthermore, it can be used by importing its library, then accessing its functionality through coding. Our main interest in WEKA is the use of a Decision Trees (DT) classifier called the J48. The next subsection will introduce the classifier.

4.1.1 J48 Decision Tree Classifier

The J48 classifier is an implementation of the open source C4.8 algorithm [43] which is also an improvement of the ID.3 algorithm [44]. The idea of the ID.3 algorithm is to generate a DT from a training dataset (S) that contains instances that are already classified (their classes are already known)

$S = s_1, s_2, s_3, \dots$

Each classified instance S_i contains a set of attributes.

$$s_i = \{a_{i,1}, a_{i,2}, a_{i,3}, \dots\}$$

The algorithm iterates through each unused attribute for all S and calculates its information gain $IG(a)$, and then the attribute with the highest information gain is chosen to split the dataset into smaller subsets. After that, unused attributes will be iterated in the subsets, splitting them into more subsets. The same process will keep recurring until all the attributes are traversed and the DT is created.

C4.8 uses the same concept but with some improvements that include the ability to handle discrete and continuous attributes and the use of missing value attribute which indicates that the data haven’t been used in forming the DT.

4.2 Features Extraction

Features extraction is the process of creating and coming up with new features. As mentioned earlier, features are critical in the classification process. Although we experimented with different types of features, the created features were based on the letter level. The following subsections lists all the features we extracted.

4.2.1 Character

Character represents the base feature in which all other features are related to.

4.2.2 Position

Position indicates whether the letter is at the beginning of the word, internal or at the end of the word.

4.2.3 Connection

Connection feature indicates the letter connection with its adjacent letters. A letter can be left connected, right connected, left-right connected or it can be neither (isolated).

4.2.4 Letter Position

Letter Position indicates the index of the letter in a word. For example, a four letters word means that it has four indices: first, second, third and fourth. If the letter is the second one in the word its index is 2.

4.2.5 Current Word Length, Previous Word Length, and Next Word Length

These three features indicate the number of letters in current word, previous word and next words, respectively.

4.2.6 Word’s First Letter, Second Letter, Before Last Letter, and Last Letter

These four features indicate the letter being either first, second, before last letter or if it is the last letter. When a word consists of two letters, the second letter is only considered last.

4.2.7 Next Word First Letter and Next Word Last Letter

These two features indicate the next word first and last letters.

4.2.8 Previous Word First Letter and Previous Word Last Letter

These two features indicate the previous word first and last letters.

4.2.9 Next Letter and Next-Next Letter

These two features indicate the next and the next-next letters of the current letter being traversed, i.e. in a word of three letters and while the pointer points to the first letter, the next letter would be the second letter and the next-next letter would be the third letter.

4.2.10 Previous and Previous-Previous Letter

These two features indicate the previous letter and the letter before the previous letter of the current letter being traversed, i.e. in a word of three letters and while the pointer points to the third (last) letter, the previous letter would be the second and the previous-previous would be the first letter.

4.2.11 Current Word Part of Speech Tag (POS), Previous Word POS Tag, and Next Word POS Tag

These three features were implemented using the Stanford POS tagger [45] for normal text and the MUSHAF Morphology Tagger [46] for MUSHAF text. Both taggers will be introduced later in this chapter. See sections (4.3.3.1 and 4.3.3.2).

4.2.12 Features Sum

Feature Sum is the summation of the used features. It sums all numeric representations of selected features into a single number.

As a general note on related features, when deciding the previous and the next words to extract features from, if the next word starts with a non-letter or the previous word ends with a non-letter, then they are considered null.

To extract the needed features from a text, we had to develop our own system as there are no systems available to automate this process. The next section will present the modules/ tools we have developed and describe how each works. Also, we will list the third party tools we have used and integrated with our tools.

4.3 Modules

The tools we developed were for feature extraction, data preparation and real time vocalization. The next subsections describes briefly each developed tool and how it operates.

4.3.1 Characters – Diacritics Generator Module

This tool main purpose is to generate three lists:

1. All Arabic letters without any diacritics.
2. All diacritics.
3. Every possible combination of Arabic letters and diacritics.

The lists have been generated by iterating through all Arabic letters and diacritics Unicode representations. The generated lists contained some letters with diacritics that could never occur in Arabic and thus were filtered out. These generated lists were used in all the modules we have developed.

4.3.2 Feature Extraction Module

Most of machine learning tools require their input to be converted into discreet sequences. Since we were working with WEKA, the same concept applies. To extract features from Arabic text, we developed a module to handle the conversion process. Before starting the development we determined all the characters that we may encounter in the text besides Arabic letters, i.e. punctuation marks, numbers, special characters…etc. Other characters that occurs without being specified were considered as a foreign characters and were

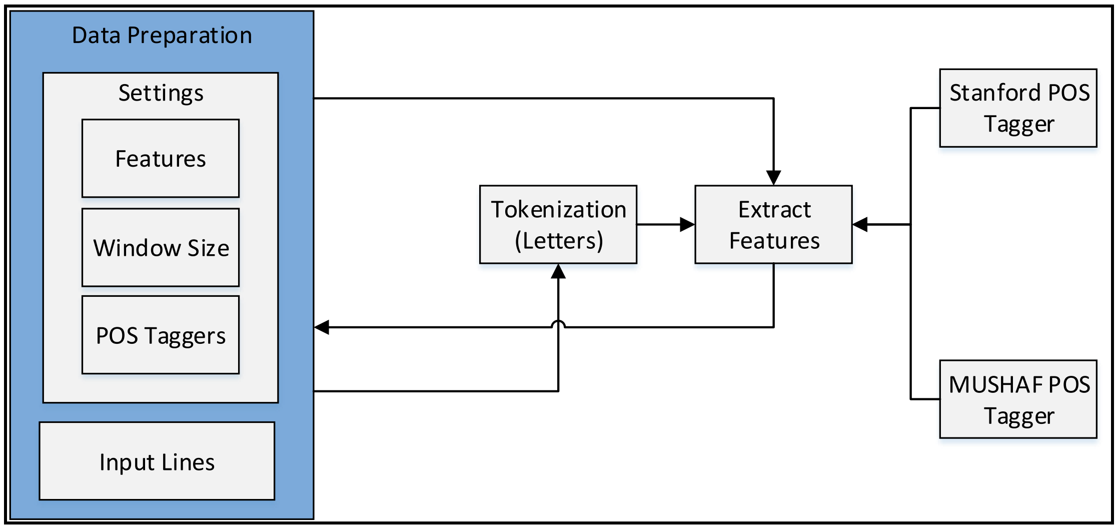


Figure 3 Feature extraction module

ignored. All these characters were added to the characters list generated earlier (See Section 4.3.1). Figure 3 shows an overview of the process of this module.

The following steps illustrate in details how the module operate:

First, the characters list is used. For each character inside the list, a unique numeric code is assigned. The same applies for the diacritics list. Table 18 shows an example of four Arabic letters with their assigned numeric codes.

Table 18: Characters numeric

Character	**Numeric Code**
ب	105
ت	136
ث	151
ج	166

Second, we determine the set of features to be extracted from the input text on the letter level (for characters). Each extracted feature from a character is mapped to the unique numeric code that was assigned for each character. Numeric codes will be converted into binary based on a specific number of bits determined for each feature.

Third, after the extraction of all features for a single character, all the extracted binary codes are accumulated (appended) together to form a single unique numeric number. Due to the large numbers resulted (high number of bits) and in case we increased the number of features, we used Big Integers which are integers that have no boundaries. The result will be a sequence of numeric numbers such that each number represents a set of features for a single character.

Fourth, the output will be used as input to another module to format the data as required by WEKA. We can also reverse back the sequence of numbers into their original text.

Table 19 shows an example of features extraction for the word "ذَهَبَ". Each numeric sequence represent a letter with all its defined features. This example assumes nine features for each letter.

Table 19: Features extraction for the word "ذَهَبَ"

Word	Feature Extraction Result (Numeric Sequence)
ذَهَبَ	1668510200572751041 1985 357325361977357387886 41 786067683826372261 9841

4.3.3 POS Tagger Module

The POS tagger module consists of two taggers. The Stanford POS tagger and the MUSHAF POS tagger. The next two subsection will describe each one respectively.

4.3.3.1 Stanford POS Tagger

A part of speech (POS) tagger is a software that is used to tag words with their part of speech such as a verb, a noun, an adjective… etc. The Stanford POS tagger is a java implemented software [45] that has been developed, enhanced and improved by several researchers [47]. The tagger supports a variety of languages such as English, Arabic and French…etc. Also, the software is available in different programming languages such as C#, F# or Ruby…etc.

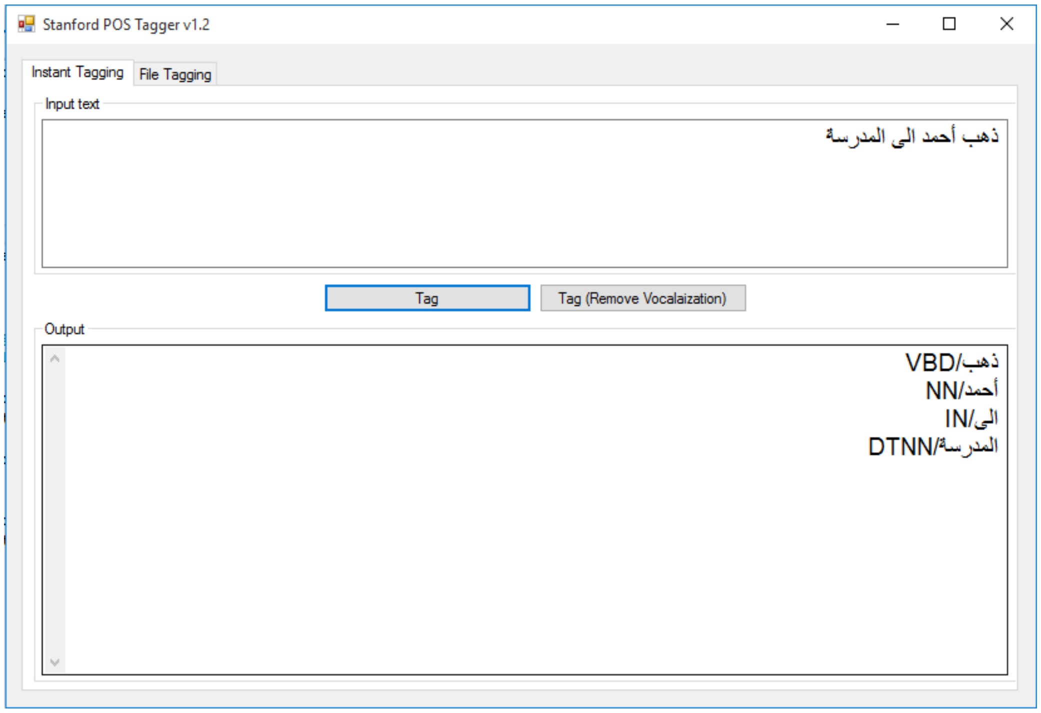
To use the Stanford POS tagger, we have downloaded the C# version of the software. Since the software is implemented in java, a Dynamic Link Library (DLL) was already complied and used in the project. The DLL used was the result of conversion of the Java Archive (JAR) file to DLL assembly using IKVM [48] library. We created a separate tool and referenced Stanford assembly and the IKVM assemblies. The IKVM assemblies were needed because functions calls in Stanford library requires java objects.

The tool we developed can tag sentences on the fly and can also tag a text file. In tagging we added two options which are tagging with/without vocalization. The reason for implementing these two features is because we noticed that tagging results for unvocalized words were better than vocalized words.

The first step of the integration was to determine all possible tags that the tagger uses.

Table 20 and Table 21 show a list of all tags that Stanford tagger uses along with their definitions. After determining these tags, a unique number was assigned to each tag. These unique numbers were used in forming the related features.

Figure 4 shows an example for tagging the sentence "ذهب أحمد إلى المدرسة" using the tool.



ذهب أحمد الى المدرسة

ذهب/VBD
أحمد/NN
الى/IN
المدرسة/DTNN

Figure 4 Tagging example using Stanford Tagger tool

Table 20: Stanford tags list – part 1

Tag	Definition
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
DTJJ	Determiner + Adjective
DTJJR	Determiner + Adjective, comparative
DTNN	Determiner + Noun, singular or mass
DTNNP	Determiner + Proper noun, singular
DTNNPS	Determiner + Proper noun, plural
DTNNS	Determiner + Noun, plural
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
NN	Noun, singular or mass
NNP	Proper noun, singular
NNPS	Proper noun, plural
NNS	Noun, plural
NOUN	Noun

Tag	Definition
PRP	Personal pronoun
PRP$	Possessive pronoun
PUNC	Punctuation
RB	Adverb
RP	Particle

Table 21: Stanford tags list - part 2

Tag	Definition
UH	Interjection
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VN	Verbal noun
WP	Wh-pronoun
WRB	Wh-adverb
,	Punctuation
.	Punctuation
:	Punctuation

4.3.3.2 MUSHAF POS Tagger

The development of a POS tagger for MUSHAF was inspired by a project that many people contributed to. The origin of the project is an open source project [46] named “Quranic Arabic Corpus”, which was started by the University of Leeds. The project included “POS tagging, morphological segmentation and a formal representation of the Quranic syntax using dependency graphs” [46].

The corpus includes each word in the MUSHAF mapped from Arabic to English via the “Buckwalter Morphological Analyzer”, each word have its tag and also a set of features that specify the word’s properties. Each word is indexed by the chapter and verse number.

Table 22 and Table 23 show a list of all tags used by the Quran Morphological corpus.

Table 22: Quran Morphological corpus tags list - part 1

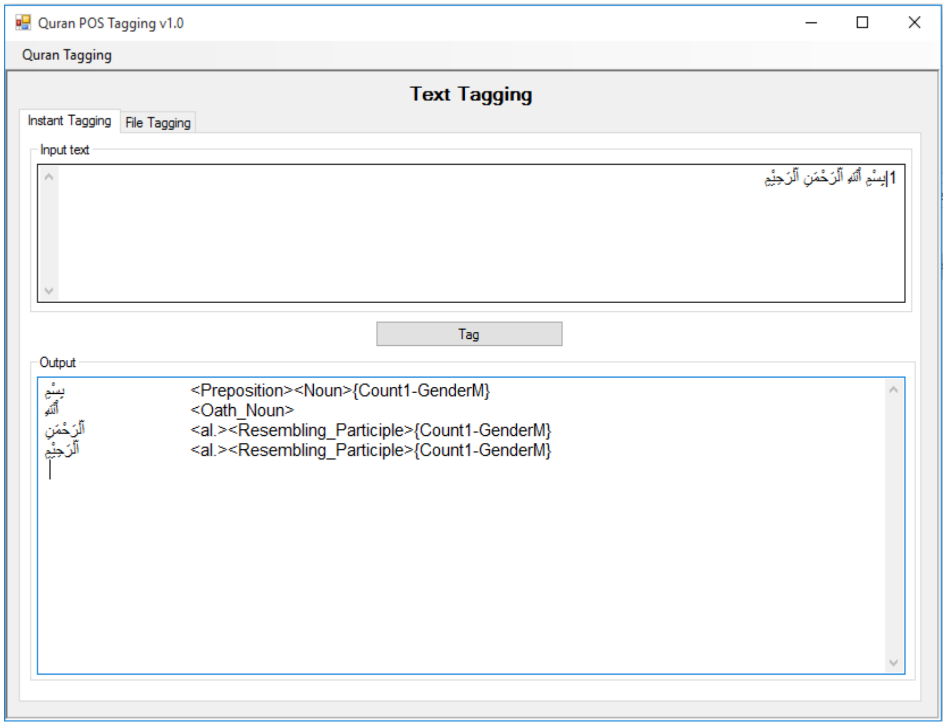
Tag	Definition
N	Noun
PN	Proper noun
ADJ	Adjective
IMPN	Imperative verbal noun
PRON	Personal pronoun
DEM	Demonstrative pronoun
REL	Relative pronoun
T	Time adverb
LOC	Location adverb
V	Verb
P	Preposition
EMPH	Emphatic lām prefix
IMPV	Imperative lām prefix
PRP	Purpose lām prefix
CONJ	Coordinating conjunction
SUB	Subordinating conjunction
ACC	Accusative particle
AMD	Amendment particle
ANS	Answer particle
AVR	Aversion particle
CAUS	Particle of cause
CERT	Particle of certainty

Table 23: Quran Morphological corpus tags list - part 2

Tag	Definition
CIRC	Circumstantial particle
COM	Comitative particle
COND	Conditional particle
EQ	Equalization particle
EXH	Exhortation particle
EXL	Explanation particle
EXP	Exceptive particle
FUT	Future particle
INC	Inceptive particle
INT	Particle of interpretation
INTG	Interrogative particle
NEG	Negative particle
PREV	Preventive particle
PRO	Prohibition particle
REM	Resumption particle

Tag	Definition
RES	Restriction particle
RET	Retraction particle
RSLT	Result particle
SUP	Supplemental particle
SUR	Surprise particle
VOC	Vocative particle
INL	Quranic initials

The version of the Morphological corpus we used was the enhanced version. We used the MUSHAF corpus we have and tagged the whole corpus. This resulted in a fully tagged MUSHAF corpus. We then started developing a tool for the goal of tagging only MUSHAF text. Both the tagged corpus and text corpus were indexed so that we can retrieve either the tag or the text. Figure 5 shows an example of tagging a verse through the tool.



Input text: 1بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

```
بِسْمِ              <Preposition><Noun>{Count1-GenderM}
ٱللَّهِ              <Oath_Noun>
ٱلرَّحْمَٰنِ          <al.><Resembling_Participle>{Count1-GenderM}
ٱلرَّحِيمِ           <al.><Resembling_Participle>{Count1-GenderM}
```

Figure 5 MUSHAF POS tagging example

4.3.4 Data Preparation Module

The purpose of the data preparation module is to provide a proper input to WEKA learning scheme. WEKA's most common files formats are comma separated vector (CSV) format and Attribute Relation File Format (ARFF). We choose the ARFF format since it is the native format used by WEKA. This module integrates with the feature extraction module discussed earlier and uses its output for the creation of ARFF files. Figure 6 shows how the module works in order to generate the ARFF files.

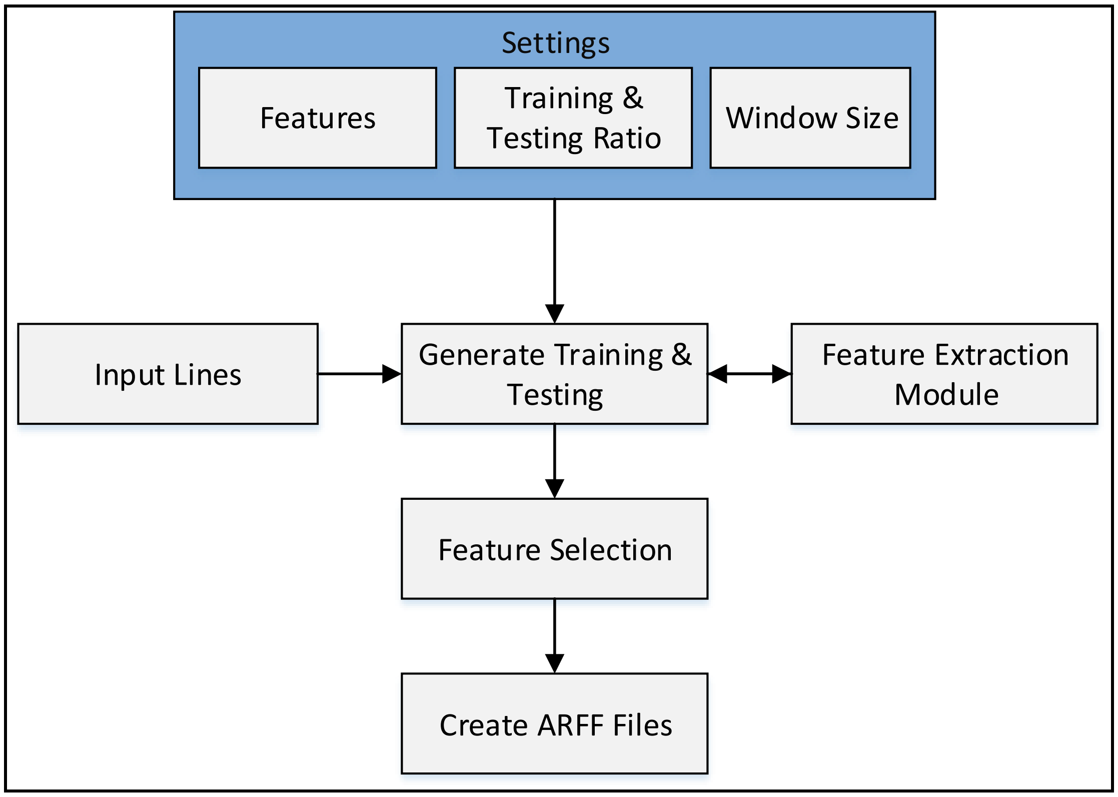


Figure 6 Data preparation module

The following steps explain how the input files for WEKA are prepared:

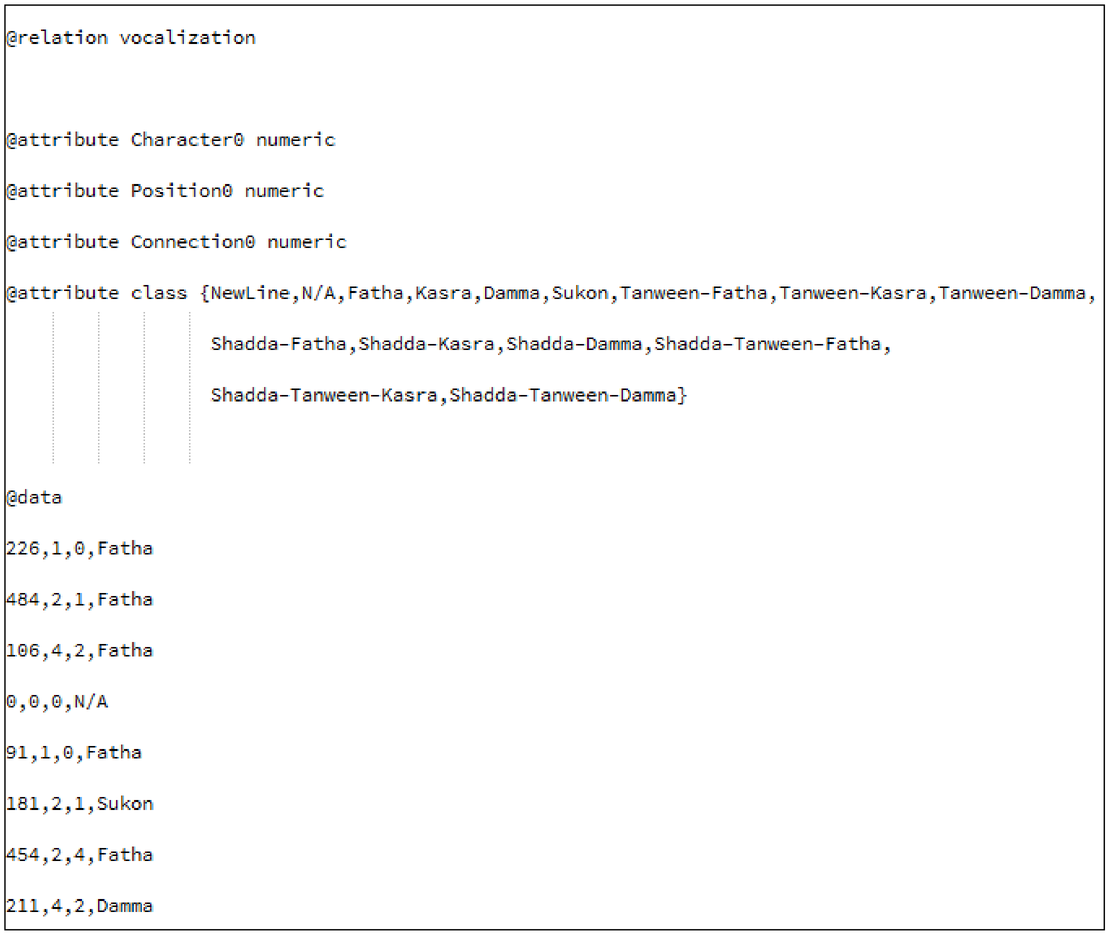
First, settings which include training and testing ratios, window size and features are specified. The window size refers to the number of characters traversed to extract features for a single character. For example, a window size of three, means that we are going to extract the features for the first, second and then the third character. All of these will represent the features of the characters diacritic.

Second, the input (Arabic text) is selected. The input has to be line formatted. The data selection for training and testing is in term of lines. This means that the ratio of training and testing will split the input by lines and not words. We used lines instead of words because choosing words may breaks the natural aspect of the language. Selection of data can be either random or static. For random selection, lines are selected randomly based on the training and testing ratios. In static selection, selected lines for training and testing are the same during the same cycle of training and testing.

Third, the input will go through feature extraction. When the extraction process is done, the results will be processed again. Each generated numeric number (similar to Table 19) encapsulates several features. All features will be extracted for all the generated numeric sequences. Then, the output will be written into an ARFF files. An ARFF file format is broken down into two sections:

1. Header section, in which the "@Attribute" tags are specified. Each tag corresponds to a feature.
2. Data section, which includes comma separated values. The number of values corresponds to the number of attributes used in the header.

Figure 7 shows an example of ARFF file.



```
@relation vocalization


@attribute Character0 numeric

@attribute Position0 numeric

@attribute Connection0 numeric

@attribute class {NewLine,N/A,Fatha,Kasra,Damma,Sukon,Tanween-Fatha,Tanween-Kasra,Tanween-Damma,

                  Shadda-Fatha,Shadda-Kasra,Shadda-Damma,Shadda-Tanween-Fatha,

                  Shadda-Tanween-Kasra,Shadda-Tanween-Damma}


@data

226,1,0,Fatha

484,2,1,Fatha

106,4,2,Fatha

0,0,0,N/A

91,1,0,Fatha

181,2,1,Sukon

454,2,4,Fatha

211,4,2,Damma
```

Figure 7 An ARFF file example

When an attribute is flagged with the "class" property, it means that this attribute will be the target for classification and since we are predicting diacritics, all diacritics were specified as possible values for predication.

Table 24 lists all diacritics and their corresponding class values. These values are used in the ARFF file. The last two class values correspond to a new line and no diacritic respectively. The new line class value was added at a later stage because it was needed to construct the output and its diacritics. As for the no diacritic class, it was needed for punctuation marks and special characters that have no diacritic.

There are two things to note here. First, when selecting a window size larger than one, the features generated will be different as opposed to selecting a window size of one. Let us assume a word of three letters and a window size of two was chosen for generating training and testing features. The features will be generated in the following manner:

1. The first and the second letter features would be generated and for both, the class value would be the diacritic of the first letter.
2. The second and the third letter features would be generated and for both, the class value would be the diacritic of the second letter.
3. Since the third letter has no next letter, then it will be represented by a null character feature that we define. The class value will be the diacritic of the third letter its class value will be the diacritic of the letter.

Second thing to note is that when generating training and testing files using vocalized data, the output for the classification is already known and is included in the testing file. WEKA ignores the classification in the testing file while performing predictions and then uses these values later to determine the accuracy rate. When generating only testing files from unvocalized data, then question marks (?) are used for the classification output since real values are unknown and we want to predict them. The question mark is also referred to as missing value.

Table 25 shows an example of both cases. The last value for both letters is the class value.

Table 24: Diacritics and their class values

Diacritic	Class Value
◌َ	Fatha
◌ِ	Kasra
◌ُ	Damma
◌ْ	Sukoon
◌ً	Tanween-Fath
◌ٍ	Tanween-Kasr
◌ٌ	Tanween-Damm
◌َ◌ّ	Shadda-Fatha
◌ِ◌ّ	Shadda-Kasra
◌ُ◌ّ	Shadda-Damma
◌ً◌ّ	Shadda-Tanween-Fath
◌ٍ◌ّ	Shadda-Tanween-Kasr
◌ٌ◌ّ	Shadda-Tanween-Damm
-	N/A
-	NewLine

Table 25: ARFF format for known classification value and for "to be predicated" value

Mode	Letter	Feature Extraction	ARFF Format
Training and Testing	ذَ	994509390599649	264, 1, 2, 1, 3, 1
Testing	ه	212982943996929	264, 1, 2, 1, 3, ?

4.3.5 Instant Diacritizer and Trainer Module

This module has three main functionalities:

1. Instantly diacritize text using decision trees models.
2. Automate the process of training and testing.
3. Build incremental classification models.

We have integrated the instant diacritizer module with the feature extraction and data preparation modules.

The next subsections present the functionality of the instance diacritizer.

Figure 8 shows how the module operates in training and testing mode, while Figure 9 shows how it operates using testing mode only.

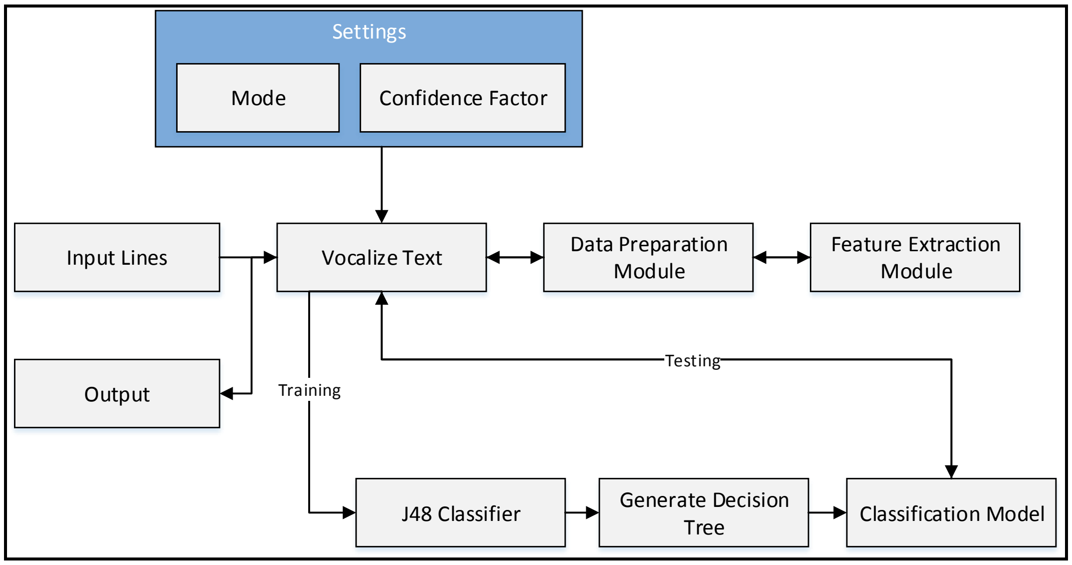


Figure 8 Instant diacritizer training and testing

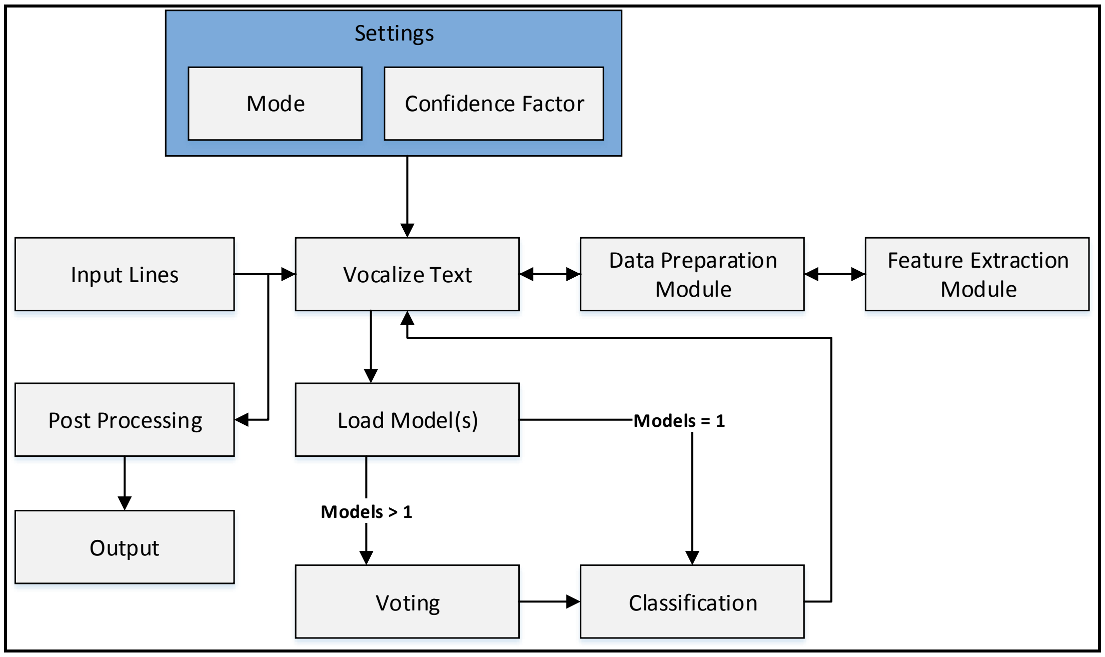


Figure 9 Instant diacritizer testing

4.3.5.1 Instant Diacritizer (Testing)

The process of instant Diacritization needs to go into four procedures. These procedures are:

First, since we want our text to be diacritized, testing mode is set by default. Also, the ratio of training and testing is set to 0 and 100 respectively. It is important to note that same used parameters (window size, features and class index) generating the training file must be used for the testing file as choosing different settings will result in incompatibility between the files.

Second, the built model(s) has/ have to be selected and then loaded. If more than on model is loaded, then voting will be triggered, voting will be explained in the following subsection. Once loading finishes, the user enters the text to be vocalize.

Third, before vocalizing the text, two options are provided. The first option is to normalize the output and remove the “Sukoon” diacritic based on specific rules. The second option, is to apply an extra layer of vocalization on top of WEKA by using n-gram models. We call both of these options post-processing since they are done after classification. Both of these options will be detailed at the end of this chapter.

Fourth, the system will automatically generate a testing file and classifies its contents against the loaded model(s) and process the output if post-processing option was selected.

4.3.5.1.1 Voting

Voting technique is activated when more than one model is loaded. One of the motives behind implementing such technique is the inability to train the whole database at once. Moreover, using more than one model for prediction at the same time means that these models learned from different information. The voting technique simply works by loading more than one model such that all participate in making the final predictions by majority. Note this is done to each instance being predicted.

Figure 10 gives an overview of how does voting work.

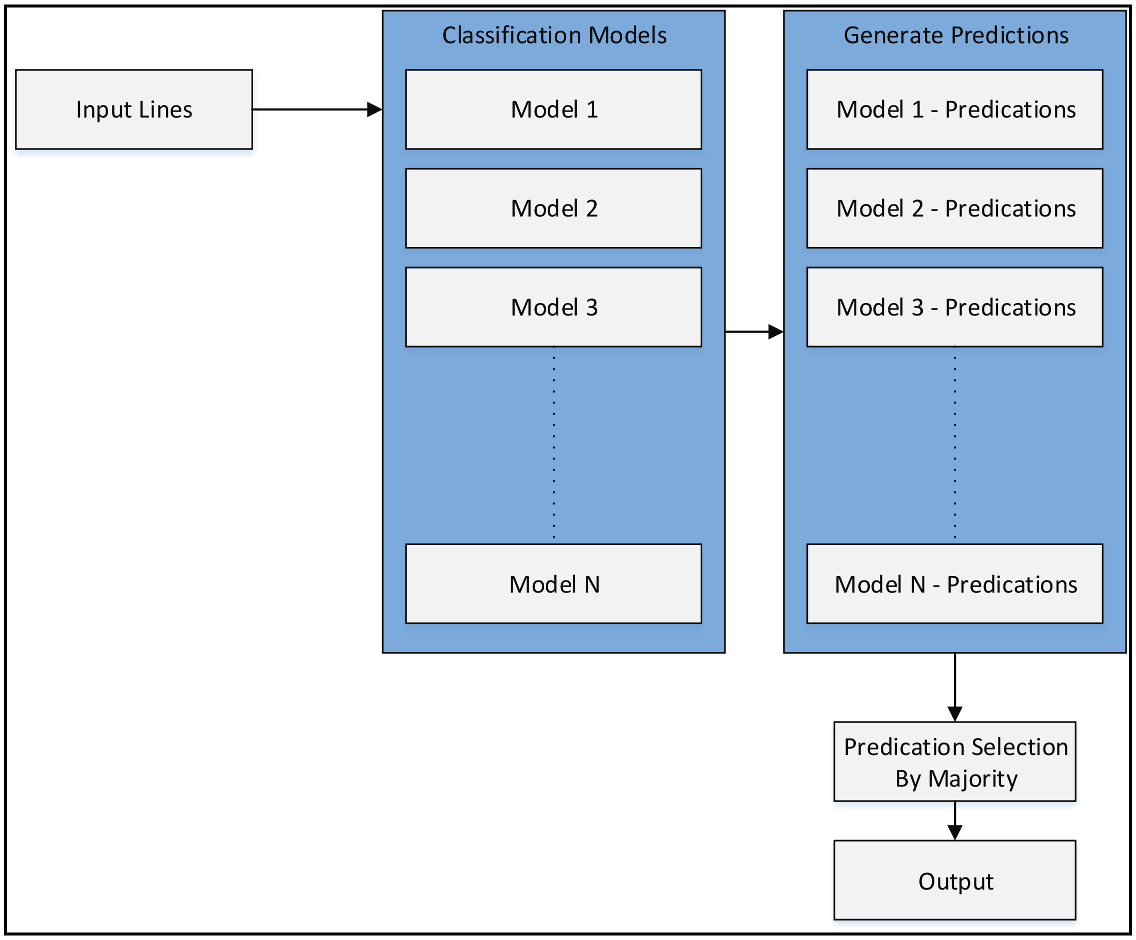


Figure 10 Voting

Table 26 shows an example of vocalizing the letter "أ" from the word "أحمد". The ID column represents the letter number assigned from the character list. Four models are used in the prediction. The result of the voting will the diacritic "Fatha" by majority as it has 3 votes out of 4.

Table 26: Voting example

Model	Letter	ID	Predicted
M1	أ	91	Fatha
M2	أ	91	Fatha
M3	أ	91	Damma
M4	أ	91	Fatha

4.3.5.2 Trainer for Training and Testing

As stated earlier, the purpose of the trainer module is to automate the process of training and testing. The automation process is done in the following manner:

First, other than the previously mentioned settings, the mode and classifier are to be set. There are three options for the mode: 1) testing, 2) building model, and 3) building model and testing. In "building" mode each trained model is saved. As for the classifier option, a set of classifiers are listed to select from. All available classifiers are incremental learners except for the J48 classifier. In our extensively pursued experiments, we have found that J48 classifier performs the best for Arabic text vocalization.

Second, the input file (line formatted) has to be selected. If an incremental classifier was chosen and the input file size exceeds 4MB then the file will be split into smaller chunks.

Third, based on the selected mode, the process of training and testing will commence, and for each input file(s), the classification results will be recorded and logged.

We want to note that in testing mode, if the text entered was partially diacritized, then all the diacritics are left untouched and only the letters without diacritics are predicated.

4.4 Feature Selection

As mentioned in the beginning of this chapter, feature selection is the process of extracting an optimal or sub-optimal subset of features from a set of features.

The main goal behind performing feature selection is to meet one or more of the following [49]:

1. Select an optimal or a sub-optimal subset of features that could increase the predictive power of the model.
2. Reduce the computational requirements and time needed to build and test a model.
3. Identify a subset of features that are related to the domain of the problem being worked on.

The basic process of feature selection can be summarized as follow [50]:

1. Generation, in which different subsets of features will be generated for evaluation.
2. Evaluation, in which the subsets generated will be evaluated and the best performing subset will be selected.
3. Stopping criterion, which represent the condition(s) for stopping the feature selection process. A condition could be based on either the generation or evaluation step. It could be a specific number of generated subsets or a number of iterations.
4. Validation, which represents testing the resulted subset. This step is not considered part of the process when we want to vocalize text in real time without having its ground truth vocalized text.

Feature selection techniques can be categorized into two groups which are filter method and wrapper method [51]. The filter method produces a subset of features based on the data properties and characteristics without using any learning scheme, while the wrapper method applies a learning scheme (classifier) to find the best subset.

WEKA provides a feature called “Attribute Selection” [42] . The process of attribute (feature) selection is composed of two parts:

1. Attribute evaluator, in which features are assessed and evaluated.
2. Search method, in which the space of attributes is searched.

Feature selection in WEKA can be either supervised or unsupervised [42] . Supervised feature selection uses features correlation with the class value when evaluating the features, while the other uses the distribution or the variance of the data for evaluation. We will be using supervised feature selection since we know that our features are correlated with the class value.

The attribute evaluator in WEKA, implements nine evaluators; seven of them are used for ranking individual features, while the remaining two evaluators identify the best features subsets. These two evaluators are:

1. Correlation-based feature subset (CFS) evaluation. It evaluates features based on their accuracy in prediction and prefers the features with the high correlation with the class and low correlation between other features.
2. Wrapper subset evaluator. It evaluates features by using a classifier in which a model will be generated for each subset of features searched, and the evaluation will be based on a specific measurement criteria which can be accuracy, recall, f-measure, and others.

Using the wrapper evaluator has an advantage on the other evaluators. Studies showed [52] [53] that by using the wrapper evaluator, the extracted features will be optimized for the used classifier with the wrapper. As a consequence this should yield a better result on classification. Since our work is based on using the decision tree classifier J48, we are going to use the same classifier with the wrapper.

The feature extraction done was broken into two phases. The first phase was done at an early stage of our work, so it covered only the first ten features mentioned earlier, while the second phase covers all features. Details of the work done is explained in the next two subsections.

4.4.1 Feature Extraction Phase 1

This phase was implemented before finishing the process of feature extraction. We decided that we are going to perform feature selection even though we still had ideas for other features. This decision was chosen to evaluate the features we had at that time and to identify the best performing features.

To determine the best set of features and the best sliding window size, we conducted many experiments. In those experiments, the space of possible features was explored using the hill climbing algorithm in which we started from the most basic feature "Character" until we traversed through all features. Table 27 shows all the combinations of features that were generated and tested.

The text lines used for training and testing were selected from "Tashkeel-2016" corpus. We created a validation dataset for this purpose. The set consisted of 500 lines. In generating the training and testing sets, a ratio of 80/20 was used respectively. The generation of the sets were not random as the first 400 lines were always taken for training and the reminder 100 lines were taken for testing. The reason for this is to eliminate any factor that could manipulate the prediction results. This minimizes the possibility of choosing the wrong features subset. In the pursed experiments we have covered six sliding window sizes (from one to six).

Table 27 shows the features space to be searched for feature selection. The table shows a list of feature sets with a code assigned to each set.

To perform these experiments, we used our developed module “Instant Diacritizer and Trainer” which we introduced earlier (See4.3.5). Figure 11, Figure 12, Figure 13, Figure 14, Figure 15, and Figure 16 show the results of the experiments in terms of features tested and their accuracy rates using different sliding window sizes. Each one of these five figure represents the results for a single sliding window size. Figure 17 shows the performance of all features across all used sliding window sizes.

Table 27: Features space for feature selection.

#	Features
F1	Character + Position
F2	Character + Connection
F3	Character + LetterPosition
F4	Character + WordFirstLetter
F5	Character + CurrentWordLength
F6	Character + NextWordLength
F7	Character + PreviousWordLength
F8	Character + WordSecondLetter
F9	Character + WordBeforeLastLetter
F10	Character + WordLastLetter
F11	Character + Position + Connection
F12	Character + Position + LetterPosition
F13	Character + Position + WordFirstLetter
F14	Character + Position + CurrentWordLength
F15	Character + Position + NextWordLength
F16	Character + Position + PreviousWordLength
F17	Character + Position + WordSecondLetter
F18	Character + Position + CurrentWordBeforeLastLetter
F19	Character + Position + WordBeforeLastLetter
F20	Character + Position + WordLastLetter
F21	Character + Position + Connection + WordFirstLetter
F22	Character + Position + Connection + CurrentWordLength
F23	Character + Position + Connection + NextWordLength
F24	Character + Position + Connection + PreviousWordLength
F25	Character + Position + Connection + WordSecondLetter
F26	Character + Position + Connection + WordBeforeLastLetter
F27	Character + Position + Connection + WordLastLetter
F28	Character + Position + Connection + LetterPosition + WordFirstLetter
F29	Character + Position + Connection + LetterPosition + CurrentWordLength
F30	Character + Position + Connection + LetterPosition + NextWordLength
F31	Character + Position + Connection + LetterPosition + PreviousWordLength
F32	Character + Position + Connection + LetterPosition + WordSecondLetter
F33	Character + Position + Connection + LetterPosition + WordBeforeLastLetter
F34	Character + Position + Connection + LetterPosition + WordLastLetter
F35	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength
F36	Character + Position + Connection + LetterPosition + WordFirstLetter + NextWordLength
F37	Character + Position + Connection + LetterPosition + WordFirstLetter + PreviousWordLength
F38	Character + Position + Connection + LetterPosition + WordFirstLetter + WordSecondLetter
F39	Character + Position + Connection + LetterPosition + WordFirstLetter + WordBeforeLastLetter
F40	Character + Position + Connection + LetterPosition + WordFirstLetter + WordLastLetter
F41	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength
F42	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + PreviousWordLength
F43	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + WordSecondLetter

F44	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + WordBeforeLastLetter
F45	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + WordLastLetter
F46	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + PreviousWordLength
F47	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + WordSecondLetter
F48	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + WordBeforeLastLetter
F49	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + WordLastLetter
F50	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + PreviousWordLength + WordSecondLetter
F51	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + PreviousWordLength + WordBeforeLastLetter
F52	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + PreviousWordLength + WordLastLetter
F53	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + PreviousWordLength + WordSecondLetter + WordBeforeLastLetter
F54	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + PreviousWordLength + WordSecondLetter + WordLastLetter
F55	Character + Position + Connection + LetterPosition + WordFirstLetter + CurrentWordLength + NextWordLength + PreviousWordLength + WordSecondLetter + WordBeforeLastLetter + WordLastLetter

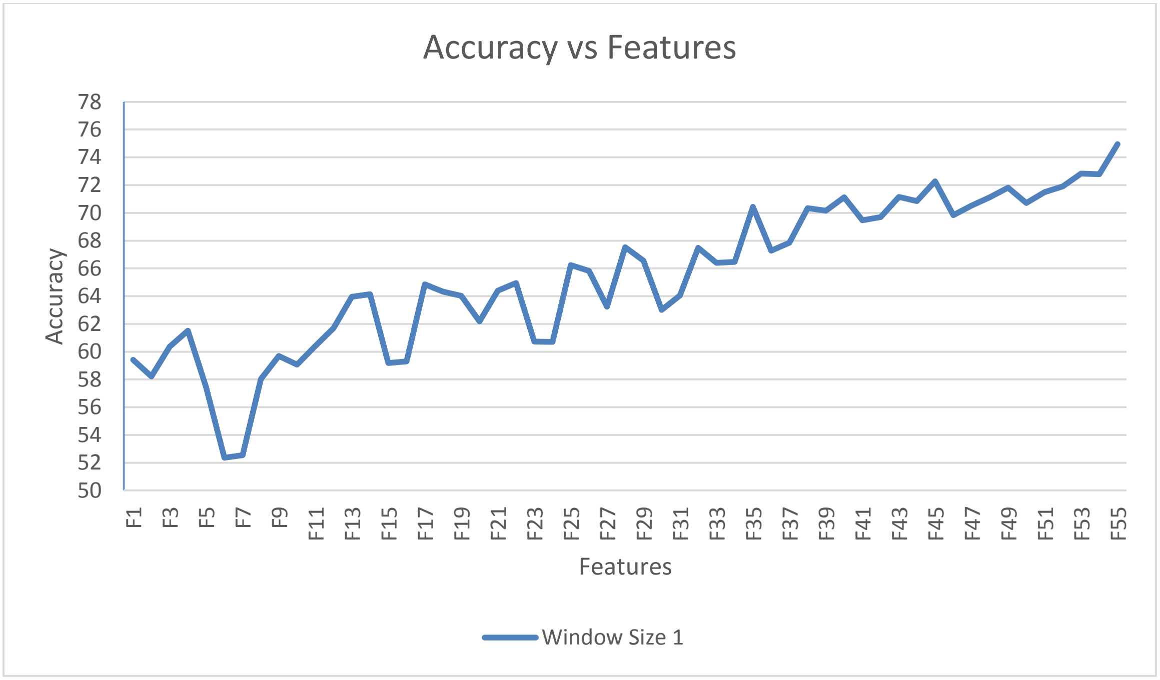


Figure 11 Accuracy for sliding window of size 1

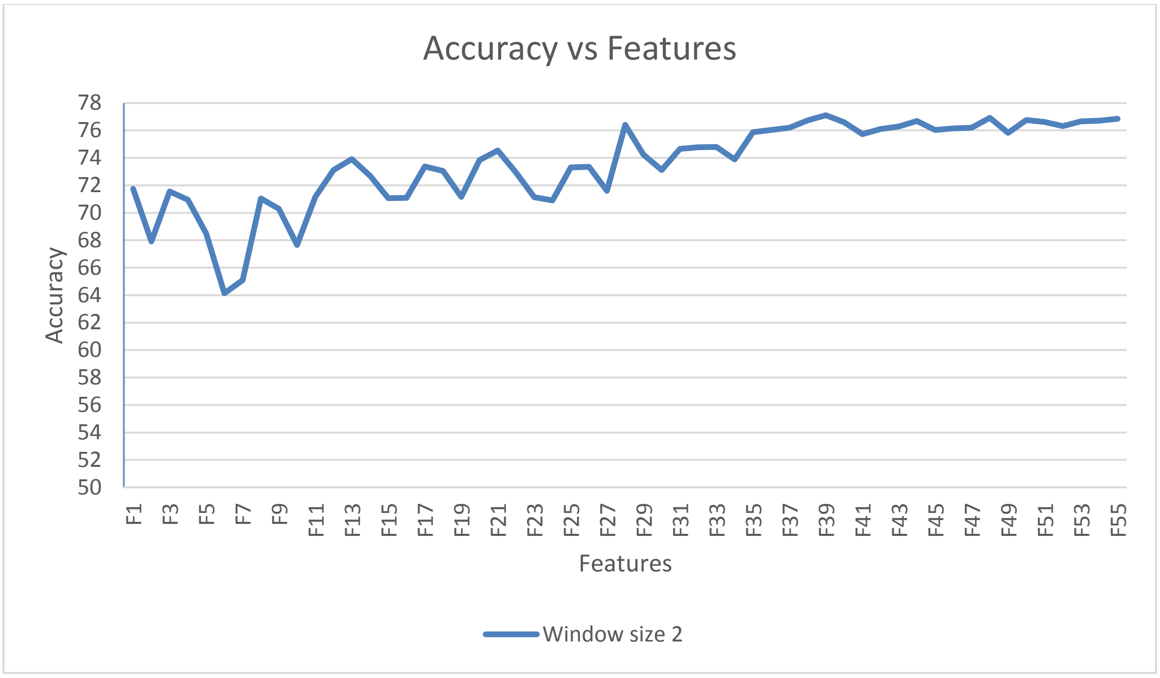


Figure 12 Accuracy for sliding window of size 2

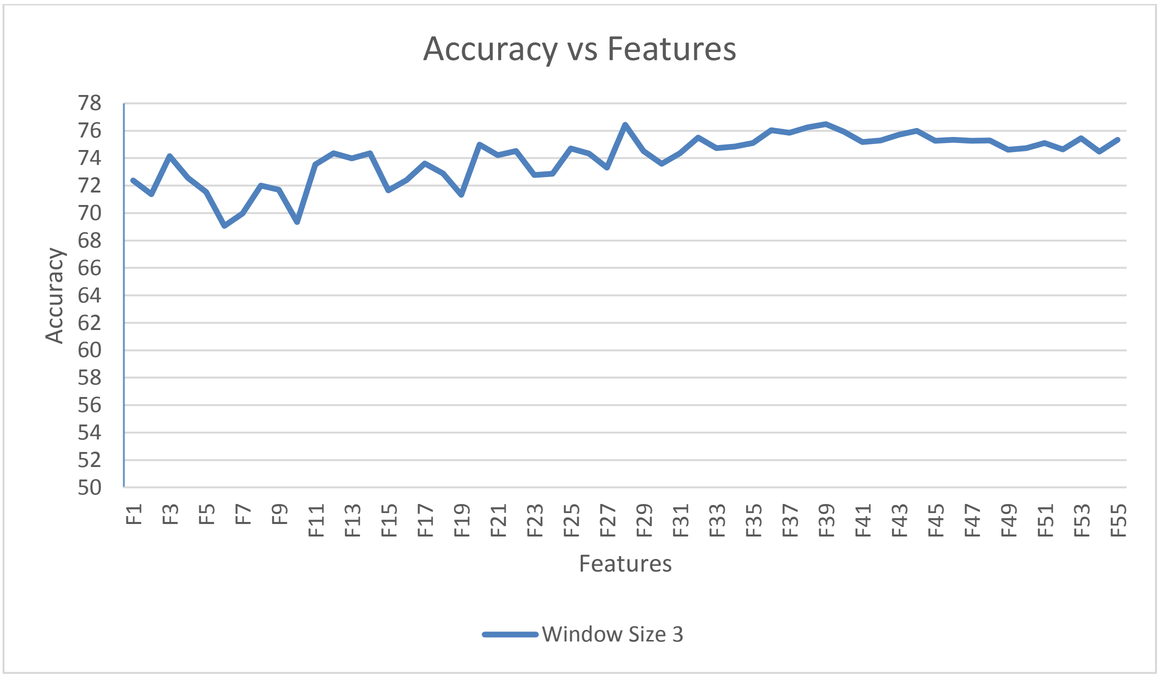


Figure 13 Accuracy for sliding window of size 3

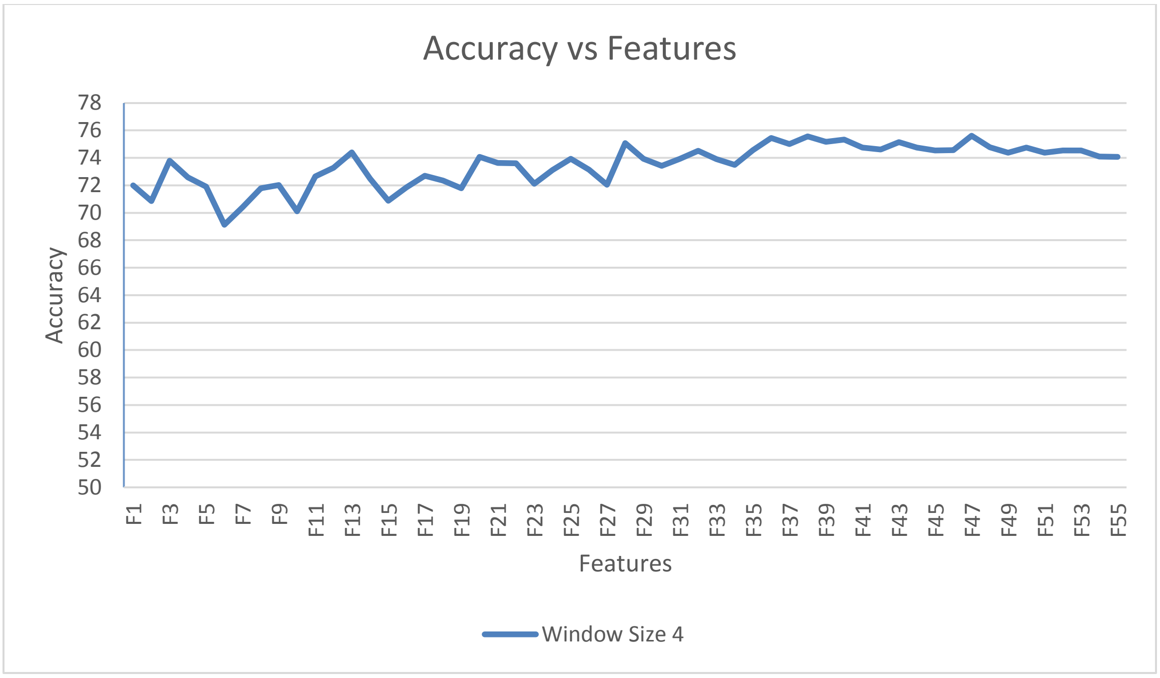


Figure 14 Accuracy for sliding window of size 4

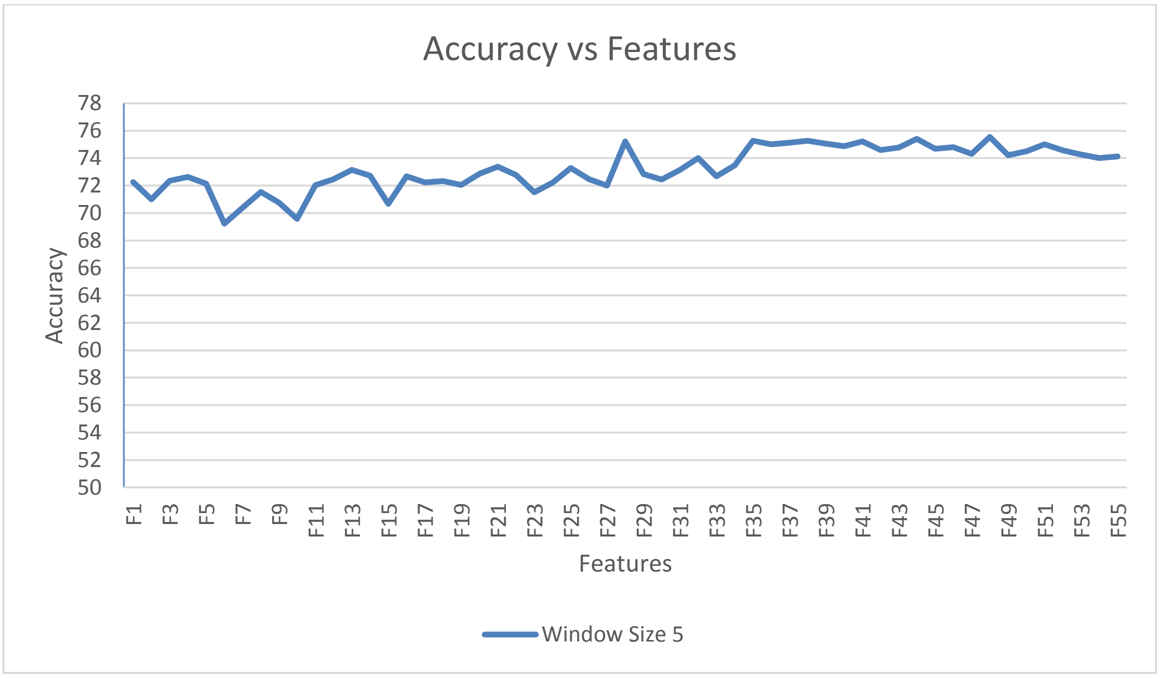


Figure 15 Accuracy for sliding window of size 5

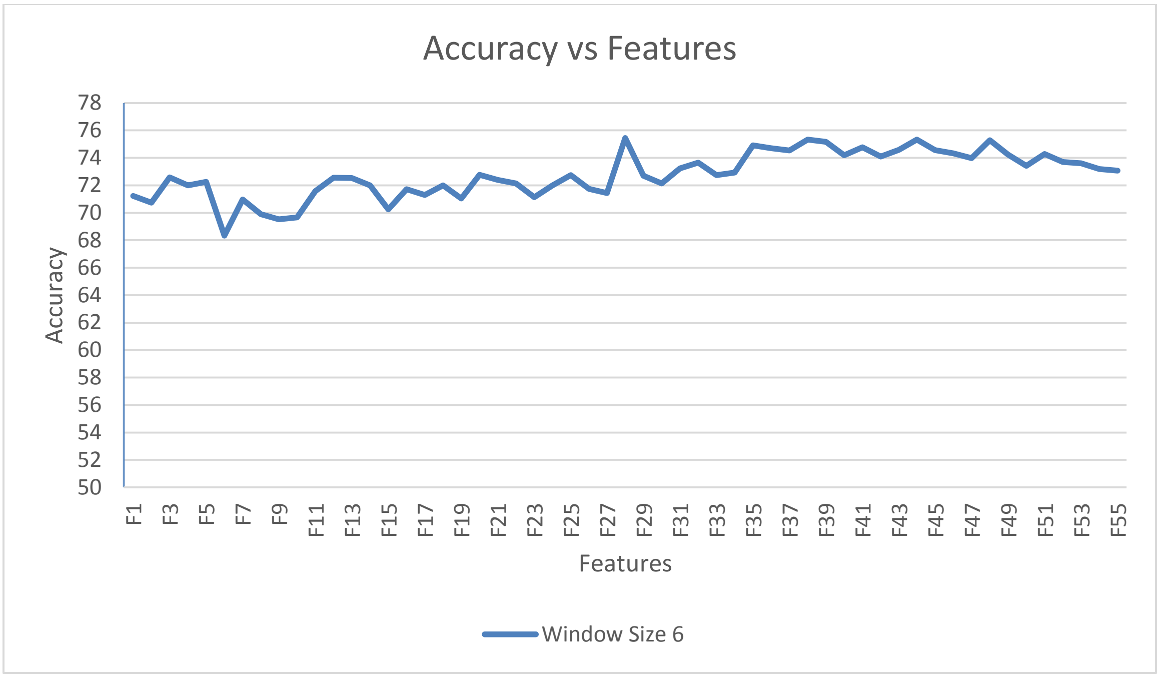


Figure 16 Accuracy for sliding window of size 6

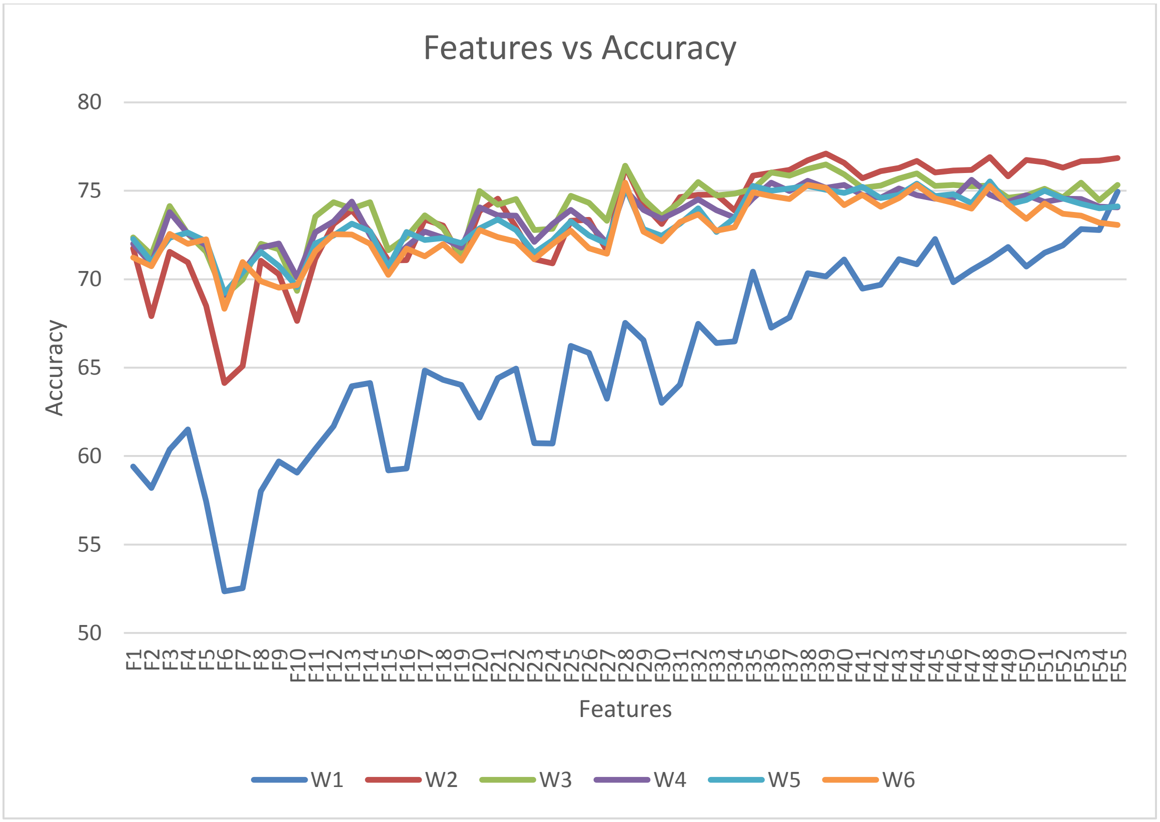


Figure 17 Accuracy for all used sliding window sizes

From the results we conclude the following:

- The fluctuation in the line represent features either doing good or bad.
- Increasing the window size would not necessarily increase classification results.
- The lowest classification was resulted from features set F6 with window size one. It scored 52.351%.
- The highest classification was achieved by features set F39 scoring 77.114% using window of size three. The Feature set F39 has the features: Character, Position, Connection, LetterPosition, WordFirstLetter, and WordBeforeLastLetter.

4.4.2 Feature Extraction Phase 2

After completing the process of feature extraction, we came up with 23 features, excluding the diacritic since it represents the class value for prediction. In this phase we used WEKA from its interface to perform the feature selection. The features evaluator we used was the wrapper along with the J48 classifier as a learning scheme. As for the search method, we used the Greedy Stepwise search. WEKA provides another native method which is the Best-First search. The reason for choosing the Greedy Stepwise is due to the time taken in searching the features space. The Greedy Stepwise implementation provides a critical feature, the user can specify the number of CPU cores to use which reduces computation time greatly.

One key difference between the feature selection in this phase and the feature selection in phase I, is that a feature significance was determined by using it in all sliding window sizes. For example, assume that we have a set of features and we want to test a newly created feature “X” across different sliding window sizes. We tested this feature against a sliding window of size two and found that the feature did not perform well. The issue here is that our conclusion could be wrong because we assumed that the “X0” (Window 1) and “X1” (Window 2) are bad while may be if one of them was tested alone, accuracy may get better.

The dataset we used, was the same validation dataset used before. Similar settings to the previous feature selection were used.

Table 28 shows the results of feature selection per sliding window size. We can see from the results that a sliding window with width one scored the lowest while the highest results were scored by sliding window with widths three, four, five and six. Furthermore, we can notice that in sliding windows with sizes three, four, five and six, the widening of the sliding window has no effect on accuracy. Also the selected features remained unchanged. Comparing these results with the previous results of feature selection, we can see that the accuracies here dominate all the results from the previous phase. Hence we choose the features subset with sliding window of size three to be our optimal subset.

Table 28: Wrapper feature selection per window

Window Size	Accuracy	Features
1	80.3%	Character0 + Connection0 + LetterPosition0 + CurrentWordLength0 + WordBeforeLastLetter0 + NextLetter0 + NextNextLetter0 + PreviousLetter0 + WordPOS0
2	80.4%	Character0 + Position0 + Connection0 + LetterPosition0 +CurrentWordLength0 + WordBeforeLastLetter0 + NextLetter0 + NextNextLetter0 + PreviousLetter0 + WordPOS0 + Character1 + Position1 + WordBeforeLastLetter1 + WordPOS1
3	0.783%	Character0 + Connection0 + LetterPosition0 + CurrentWordLength0 + WordBeforeLastLetter0 + NextLetter0 + PreviousLetter0 + WordPOS0 + LetterPosition1 + PreviousLetter1 + Character2 + Position2 + LetterPosition2
4	80.5%	Character0 + Connection0 + LetterPosition0 + CurrentWordLength0 + WordBeforeLastLetter0 + NextLetter0 + PreviousLetter0 + WordPOS0 + LetterPosition1 + PreviousLetter1 + Character2 + Position2 + LetterPosition2
5	80.5%	Character0 + Connection0 + LetterPosition0 + CurrentWordLength0 + WordBeforeLastLetter0 + NextLetter0 + PreviousLetter0 + WordPOS0 + LetterPosition1 + PreviousLetter1 + Character2 + Position2 + LetterPosition2
6	80.5%	Character0 + Connection0 + LetterPosition0 + CurrentWordLength0 + WordBeforeLastLetter0 + NextLetter0 + PreviousLetter0 + WordPOS0 + LetterPosition1 + PreviousLetter1 + Character2 + Position2 + LetterPosition2

4.5 Post-processing

To further enhance the vocalization, another step was added after classification. Basically what will happen is that at first, the best features subset will be used to generate a model using the decision tree classifier, then the model will be used for text vocalization. After text vocalizing, post-processing will occur. Post-processing consist of two parts in which will be described in the following subsections. The aim of the first part (N-Gram Vocalization) is to enhance vocalization while the second part aim ("Sukoon" Diacritic

Normalization) is to make the output text more consistent. Figure 18 shows how post-processing works.

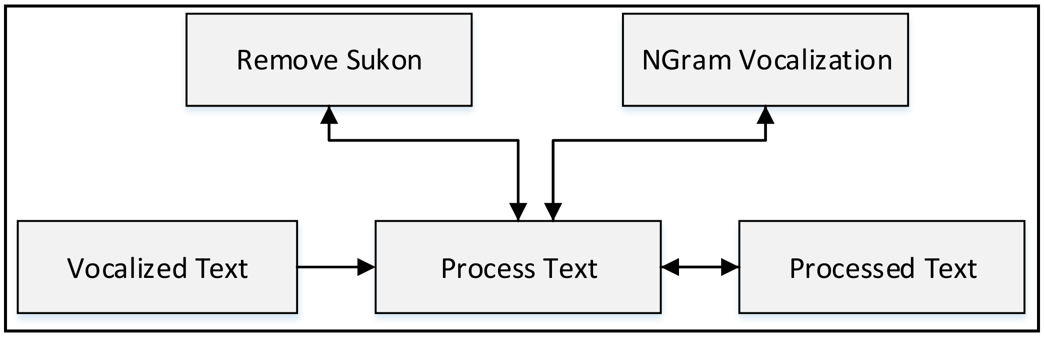


Figure 18 Post-processing

4.5.1 N-Gram Vocalization

To enhance the vocalization, n-gram models were constructed for this goal. Three n-gram models were created, bi-grams, tri-grams and quad-grams models. We used the SENTNCES3 corpus in building these models. The n-gram model format used is the word vocalized without its cased ending and a list of its previous unvocalized words along with the frequency of each possible case. When vocalizing a word in a sentence, the word and its previous words are collected based on the n-gram size and then all of their diacritics will be striped. After that a search through each model is initiated starting from the highest n-gran model. If a match is found, then the word is vocalized with the matched word while keeping the original diacritic of the word case ending. If no match is found, then searching through lower n-grams models takes place. Note that in the construction of the models, while traversing words, if we find that the current token we are traversing starts with a punctuation mark such i.e. “(“ or “]” …etc, Then we consider this word as the first word of a sentence. Similarly, if while going over the previous words, we encounter a word that ends with a punctuation mark, then we stop looking for any words behind that word. This feature was added to Instant Diacritizer tool explained before.

4.5.2 “Sukoon” Diacritic Normalization

In the corpus development chapter, we detailed the process followed in making sure that the corpus is fully vocalized. One of the main things we did, was applying "Sukoon" diacritic to all unvocalized letters. Although, this step is necessary for classification, the appearance of "Sukoon" on some letters does not look natural in Arabic, e.g. (اْلْطَاْلب). So to make the vocalization output more consistent, we removed the “Sukoon” diacritic. The removal was done by reversing the rules used in applying the “Sukoon” diacritic. The rules used in the removal are:

1. If letter "ا" vocalized with "Sukoon" is followed by letter "ل" vocalized with "Sukoon" at the start of a word, then we remove "Sukoon" from both letters.
2. If a letter "ا" vocalized with "Sukoon", then we remove the "Sukoon".
3. If a letter has a "Kasra" diacritic and is followed by a letter "ي" vocalized with "Sukoon", then we remove the "Sukoon" from the letter "ي".
4. If a letter has a "Fatha" diacritic and is followed by a letter "ى" vocalized with "Sukoon", then we remove the "Sukoon" from the letter "ى".
5. If a letter has a "Tanween-Fath" diacritic and is followed by a letter "ى" or "ا" vocalized with "Sukoon", then we remove "Sukoon" from either letters.
6. If a letter has a "Damma" diacritic and is followed by letter "و" and/or "ا" vocalized with "Sukoon", we remove "Sukoon" from either or both letters.

4.6 Implementation Issues

This section discusses some implementation issues related to this research work.

4.6.1 Features Combinations

One of the implementation issues we had to deal with was the time consumption of the searching process to find the best set of features using WEKA the search algprithms provided through WEKA, namely, the Best-First search and the Greedy-Stepwise search. . While both algorithms try to find the best set features, one critical feature was available in the Greedy-Stepwise and not in the other algorithm. The feature was to be able to set the number of CPU cores. Although using this feature decreased the search time by ¾, the process of finding the best set of features took over 24 hours on a HPC machine. The HPC machine we used had two CPU's and 45 GB of RAM.

4.6.2 Experiments

In our work, we conducted hundreds of experiments. Some implementation issues related to these experiments were:

1. We wanted to make the process of experimentation automated. This led us to develop our own tools.
2. The tools developed integrate with WEKA, which is built on Java, while the programming language we used was C#, we had to convert the Java library into a DLL in order to use it.
3. To conduct the experiments we used the same HPC machine used before, and even though, the experiments we did took days to finish.

4.7 Summary

In this chapter, we discussed features extraction and selection. We listed all features we were produced. We also introduced the modules we have developed to automate text diacritization. Feature selection process was divided into two phases, one that was done at an early stage and the second phase was pursued after completing extracting all considered features. An optimal features subset was chosen and discussed, and finally we highlighted the major implementation issues we faced during our work

CHAPTER 5

EVALUATION

The evaluation phase of any implemented system is critical as it defines the boundary between success and failure. In this chapter, we discuss the used evaluation metrics in Section 5.1. In Section 5.2, we discuss the experiments we performed using the best reached settings along with the results. Section 5.3 presents applying performance tuning to possibly enhance the vocalization accuracy. . Comparing our work with some other related work is presented in Section5.4. Finally, Section 5.5 is the summary of the chapter.

5.1 Performance Metrics

There are many metrics that can be used to measure the performance of a diacritizer. We choose common metrics that researchers often apply with the addition of specific metrics that WEKA provides. The metrics are:

- Diacritics Error Rate (DER)
- Word Error Rate (WER)
- Accuracy
- KAPPA
- Receiving Operating Characteristics Curve (ROC)

In the next subsections, a brief description of each metric will be given along with their calculation methods.

5.1.1 Diacritic Error Rate (DER)

Diacritic Error Rate is the ratio of wrongly diacritized letters to the total number of letters [54], as denoted by Equation 1

$$DER = \frac{number\ of\ incorrectly\ diacritized\ letters}{total\ number\ of\ letters}$$

Equation 1: Diacritic-error rate (DER)

5.1.2 Word Error Rate (WER)

Word Error Rate is the ratio of wrongly diacritized words to the total number of words as denoted by Equation 2.

$$WER = \frac{number\ of\ incorrectly\ diacritized\ words}{total\ number\ of\ words}$$

Equation 2: Word error rate (WER)

5.1.3 Accuracy

The accuracy represents the ratio of correctly classified instances. Equation 3 shows how accuracy is calculated.

$$AUC = \frac{number\ of\ correcly\ classified\ instances}{total\ number\ of\ instances\ to\ be\ classified}$$

Equation 3: Accuracy (AUC)

5.1.4 KAPPA

KAPPA measures the chance of agreement between what have been classified and the actual results of the classification. It means that it estimates the degree of whether the classification was done by chance or not. A KAPPA value of zero would mean that a classifier is classifying instances completely by chance (random). A value greater than zero indicates that a classifier is doing better than chance. A value of one, means that the classifier is sure of what the classification result would be. Note that the classifier being sure of the result, does not mean that the classification would be right. This can be represented as “I have learned” versus “I am sure of what I have learned”. To explain it in a more easy way, assume that a student has been asked a question. The student answers the question and he is sure of his answer, but the problem is that his answer might be wrong.

5.1.5 Receiving Operating Characteristics Curve (ROC)

The ROC measures the model predictive ability. i.e., a model ability to separate classes and distinguish them. A model with a ROC value of 50% means that its prediction is random, much like a coin toss, while a higher value indicates a better prediction. Based on a point system, we can say that for example an ROC of 90%, means that its ability to distinguish between classes is excellent, as opposed to 50% which is actually failing in distinction.

5.2 Experiments

After determining the best settings (features and sliding window size) through feature selection, we pursued more experiments to obtain higher classification accuracy and good models to be used for vocalization. In these experiments, both the “TASHKEEL-2016” and the MUSHAF corpus were used. The following subsections describes the experiments done on each corpus.

5.2.1 Experiments with “TASHKEEL-2016” Corpus

As a start, we tried training the whole corpus but the generated training file was big for used tool and resources to handle. As a consequence, we had to split the corpus to smaller chunks. To split the corpus, we first decided on the sizes of the training sets in terms of lines per file. We have chosen sizes ranging from 1000 to 15000 lines. For each size, the whole corpus was used to generate the datasets. E.g. for a 1000 lines size, about 300 sets were generated. The generation of the lines in the datasets, were random. Since the corpus covers different domains of text, ignoring random generation for the data may result in training sets that covers only a specific domain. Thus, we used random generation expecting that our training data would be diverse.

Table 29 shows the highest results achieved for each lines size. Note that for these results we did not measure DER and WER because when we generated the training files for WEKA, we ignored both new lines and spaces between words, so it was not possible to reconstruct the output and calculate these metrics. Figure 19, Figure 20, and Figure 21 show the results in term of size, accuracy, KAPPA and ROC respectively.

We notice from the results the gradual increase in all metrics as the number of lines increases. The highest accuracy and KAPPA achieved were with a dataset of 14000 lines, while the highest ROC was scored by a dataset of 15000 lines. In general we can conclude the bigger that data, the better the results. The highest accuracy and KAPPA achieved were 86.68% and 81.77% respectively, while the highest ROC achieved was 96.3%. The highest KAPPA achieved implies that the model has a good precision (reliability). Similarly, the highest ROC indicates that the model predictions is not random and can distinguish between classes very well.

Table 29: "TASHKEEL-2016" Experimental Results

Lines Size	Accuracy	KAPPA	Average ROC
1000	81.60%	74.70%	92.60%
2000	82.95%	76.51%	93.70%
3000	83.96%	77.88%	94.30%
4000	84.57%	78.76%	94.70%
5000	85.06%	79.40%	95.10%
6000	85.24%	79.69%	95.50%
7000	85.57%	80.13%	95.60%
8000	85.71%	80.36%	95.80%
9000	85.98%	80.75%	95.90%
10000	86.13%	80.92%	96.10%
11000	86.25%	81.09%	96.10%
12000	86.52%	81.46%	96.30%
13000	86.55%	81.47%	96.30%
14000	86.72%	81.77%	96.30%
15000	86.69%	81.71%	96.40%

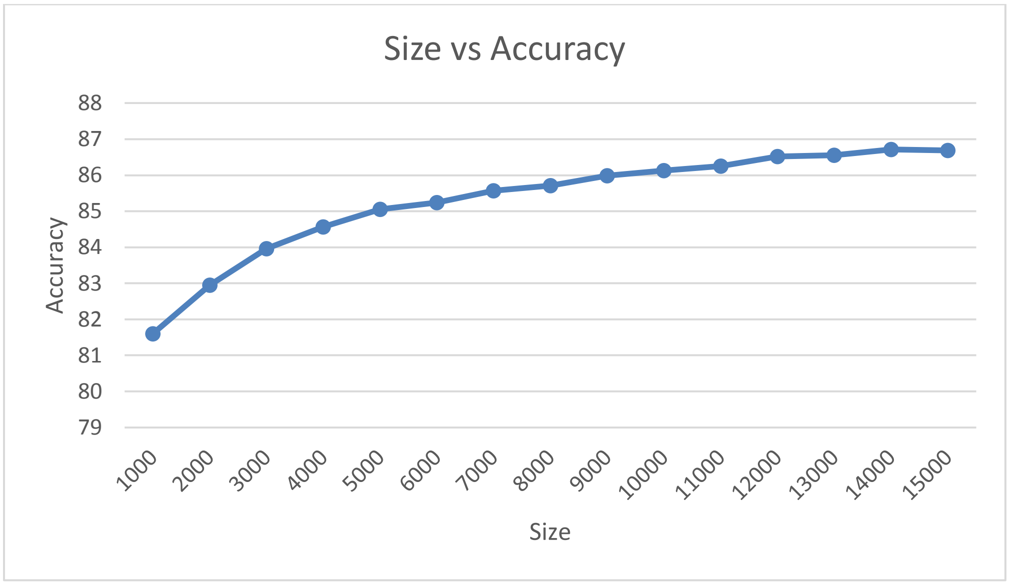


Figure 19 "TASHKEEL-2016" experimental results: Size vs Accuracy

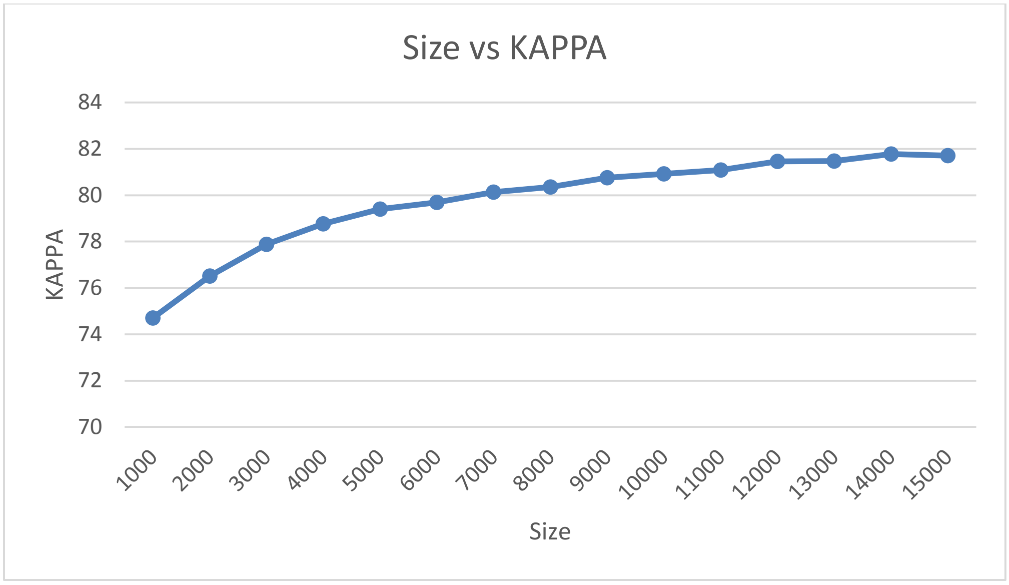


Figure 20 “TASHKEEL-2016” experimental results: Size vs KAPPA

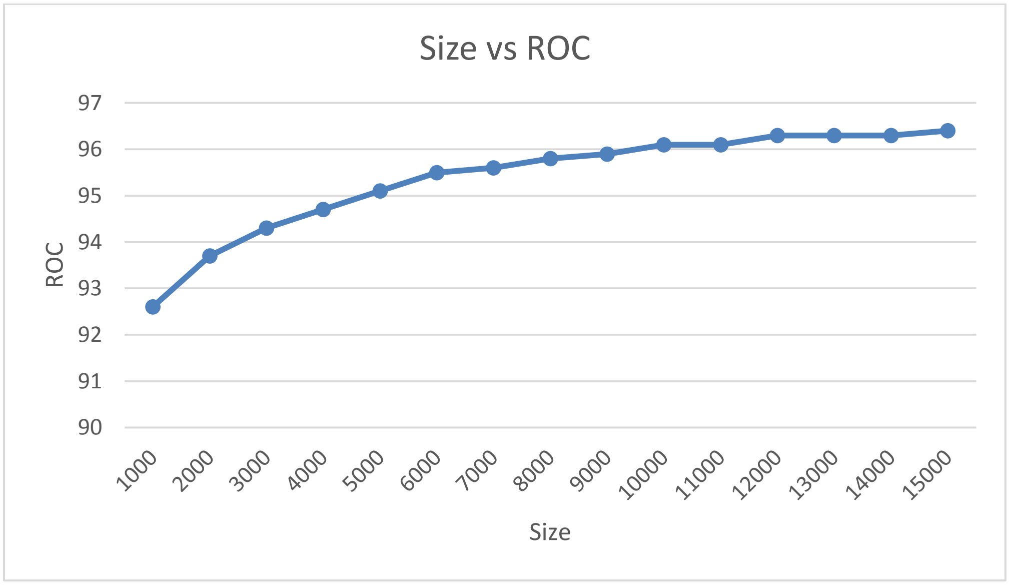


Figure 21 “TASHKEEL-2016” experimental results: Size vs ROC

To understand the results better, we take a closer look at the ROC values generated for each class using the best models of 14000 and 15000 lines. Table 30 shows the ROC values for each class per each dataset. Figure 22 shows a comparison between the resulted ROC values. While the difference between the results is not high, we can see that the class "Shadda-Tanween-Kasr" scored the lowest value while the class "Sukoon" scored the highest value. This denotes that the model cannot distinguish very well when it comes to certain diacritics. The reason behind this is that the diacritics distribution in the data in general and in Arabic in specific is not balanced. In Arabic, the percentage of occurrence of the diacritics "Fatha" and "Kasra" is higher than diacritic "Shadda-Damma". Due to the lack of enough instances of such diacritics, the model could not train very well on those cases.

Table 30: ROC values for each class in 14000 and 15000 lines

#	Class Value	ROC	
-	-	**14000 Lines**	**15000 Lines**
1	Fatha	95.90%	96.00%
2	Kasra	95.60%	95.80%
3	Damma	94.40%	94.10%
4	Sukoon	98.80%	98.80%
5	Tanween-Fath	97.10%	97.10%
6	Tanween-Kasr	90.00%	90.50%
7	Tanween-Damm	88.70%	88.00%
8	Shadda-Fatha	94.40%	95.40%
9	Shadda-Kasra	89.00%	87.60%
10	Shadda-Damma	87.30%	87.30%
11	Shadda-Tanween-Fath	97.70%	94.80%
12	Shadda-Tanween-Kasr	81.00%	82.10%
13	Shadda-Tanween-Damm	87.80%	88.70%

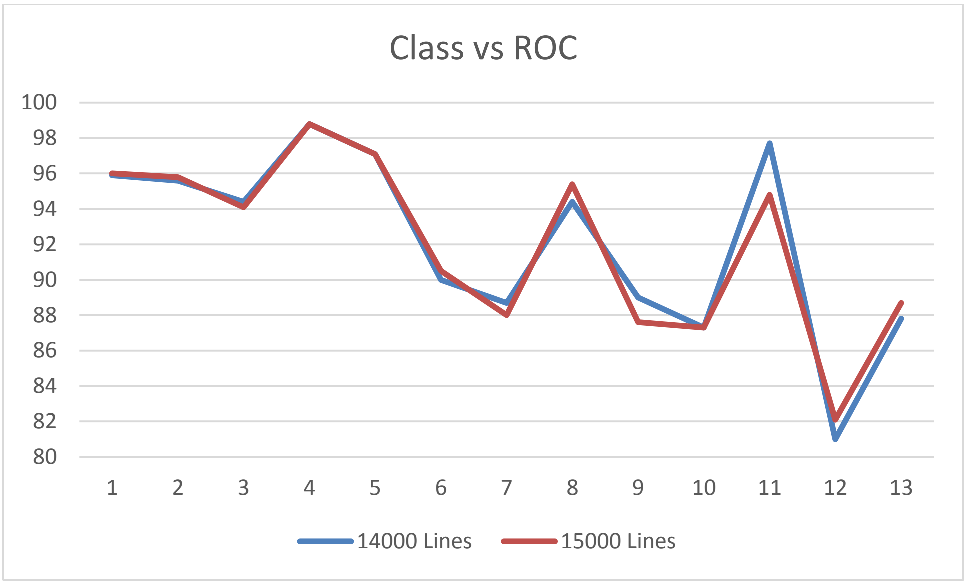


Figure 22 14000 and 15000 lines datasets ROC comparison for each class

5.2.2 Experiments with MUSHAF Corpus

Since the size of the MUSHAF corpus is very small compared to the "TASHKEEL-2016" corpus, we used the whole corpus for training and testing. Note that when extracting features from the text, the POS features were generated through the MUSHAF POS tagger. Table 31 shows the results of the experiments. The ROC result for each class value is presented in Figure 23, while Table 32 shows the detailed results. The MUSHAF model results dominate the results achieved from "TASHKEEL-2016" with significance. Furthermore, we notice that the ROC values are much higher than what have been achieved before, with the exception of the "Shadda-Tanween-Kasr" which is the same class that the previous models suffered from. The lowest ROC value achieved by the highest model for the same class was 82.1% for MUSHAF as opposed to 74.8% for the "TASHKEEL-2016".

Table 31: MUSHAF experimental results

Accuracy	KAPPA	Average ROC
90.72%	87.29%	96.8%

Table 32: MUSHAF ROC value for each class

#	Class Value	ROC
1	Fatha	96.50%
2	Kasra	95.80%
3	Damma	94.80%
4	Sukoon	98.80%
5	Tanween-Fath	98.30%
6	Tanween-Kasr	92.10%
7	Tanween-Damm	92.20%
8	Shadda-Fatha	95.50%
9	Shadda-Kasra	90.60%
10	Shadda-Damma	90.50%
11	Shadda-Tanween-Fath	97.90%
12	Shadda-Tanween-Kasr	74.80%
13	Shadda-Tanween-Damm	93.70%

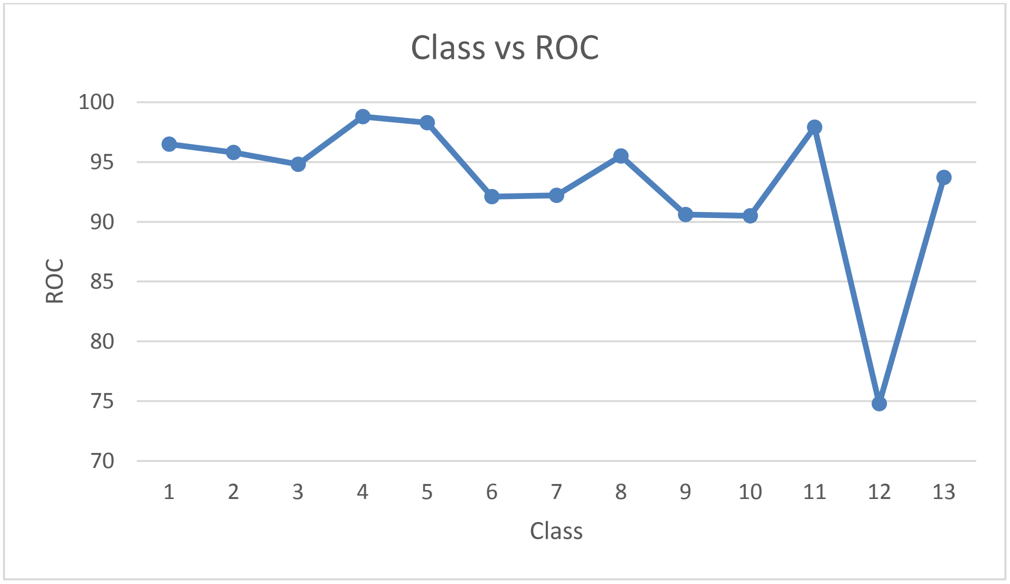


Figure 23 MUSHAF experimental results: Class vs ROC

Trying to achieve better results than what we currently have, we pursued tuning to optimize the J48 classifier parameters. The next subsection describes the tuning process of the classifier.

5.3 Performance Tuning

Tuning the classifier for better performance depends on how the classifier actually works. The J48 decision tree classifier uses a technique called pruning while building the decision trees. Pruning means reducing the size of the resulting decision tree by removing parts that are not contributing much in the classification. This reduces the complexity of the tree and as a consequence may improve the accuracy. While the J48 classifier provides several options for controlling the pruning process, most studies [55] [56] that conducted performance tuning focused throughout their work on the "confidence factor" parameter. The confidence factor controls the level of pruning and allows a range of values from 0 - 1. Choosing a low confidence value close to zero results in an aggressive pruning. While increasing the confidence value to its upper bound results in a minimal pruning.

To tune the classifier, we pursued 10 experiments in which we tested various confidence values. We started at 0.05 with a step size of 0.05 and keep incrementing until we reach 0.5 confidence value. We used the best dataset achieved from "TASHKEEL-2016" and the MUSHAF corpora.

Figure 24, Figure 25 and Figure 26 show the results of performance turning for accuracy, KAPPA and ROC respectively. Figure 27 Figure 28, and Figure 29 show the results for MUSHAF for the same metrics respectively. Table 33 shows the detailed results for both corpora.

From the results, we notice the following:

- An aggressive pruning using a CF of 0.05 hindered the accuracy performance, while the ROC value increased.
- A minimal pruning using a CF of 0.5 hindered both accuracy and ROC for the “TASHKEEL-2016” while the contrary happened for the MUSHAF.
- For the “TASHKEEL-2016”, the highest score achieved was with a CF of 0.3 and 0.35. It scored an 86.72% for both CF values which is an increase of 0.01% over the base result. As for the MUSHAF the highest result was achieved by using a CF of 0.3 to 0.45 with a score of 90.75% compared to 90.72% before tuning.
- We see that the relation between the level of pruning and the ROC value is linear. A high level of pruning achieved the highest score, and while lowering the pruning level, the ROC values decrease gradually.

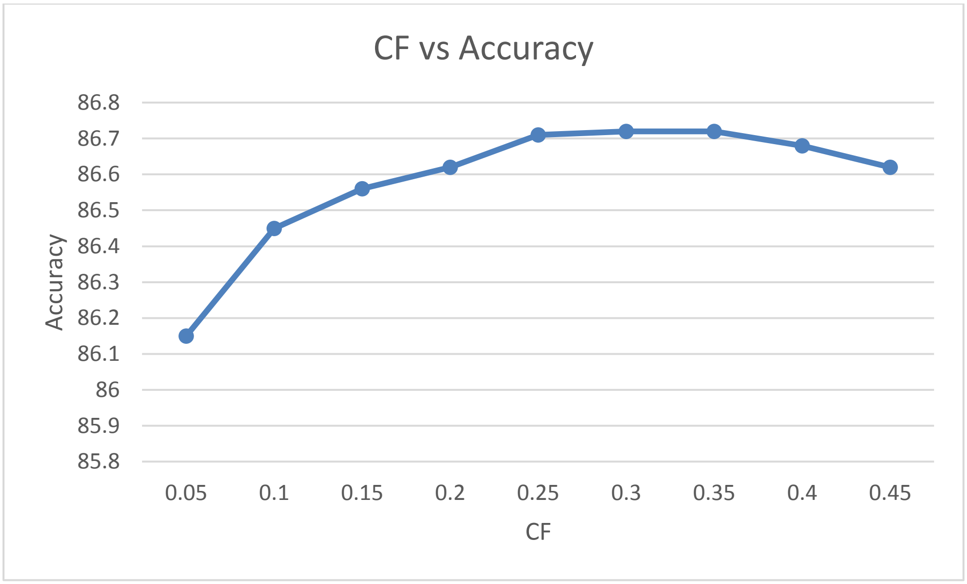


Figure 24 “TASHKEEL-2016” performance tuning results: CF vs Accuracy

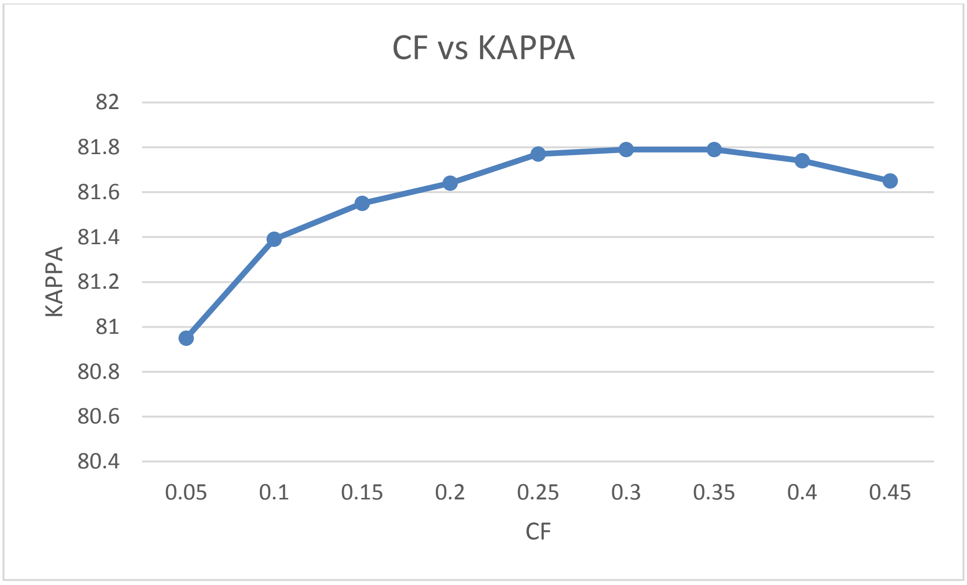


Figure 25 “TASHKEEL-2016” performance tuning results: CF vs KAPPA

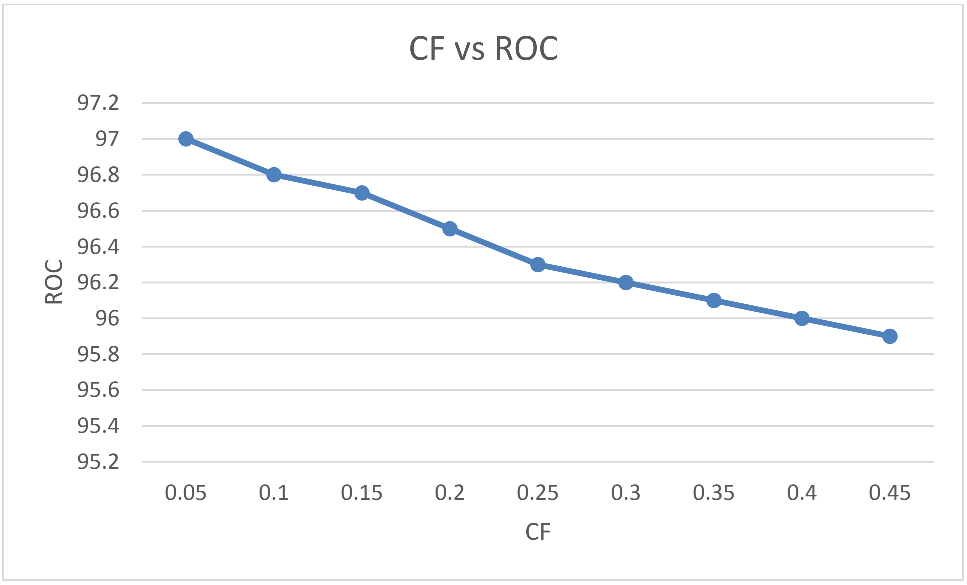


Figure 26 “TASHKEEL-2016” performance tuning results: CF vs ROC

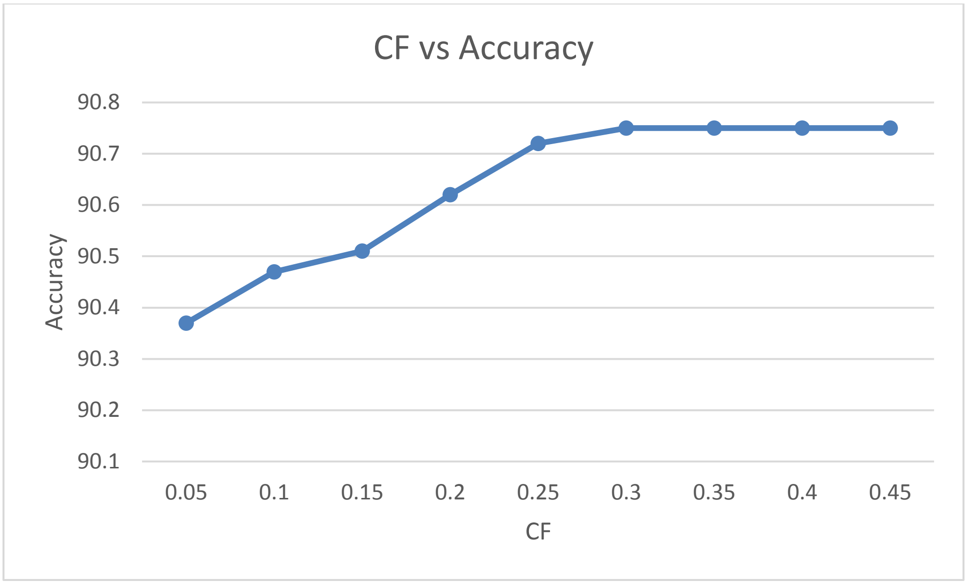


Figure 27 MUSHAF performance tuning results: CF vs Accuracy

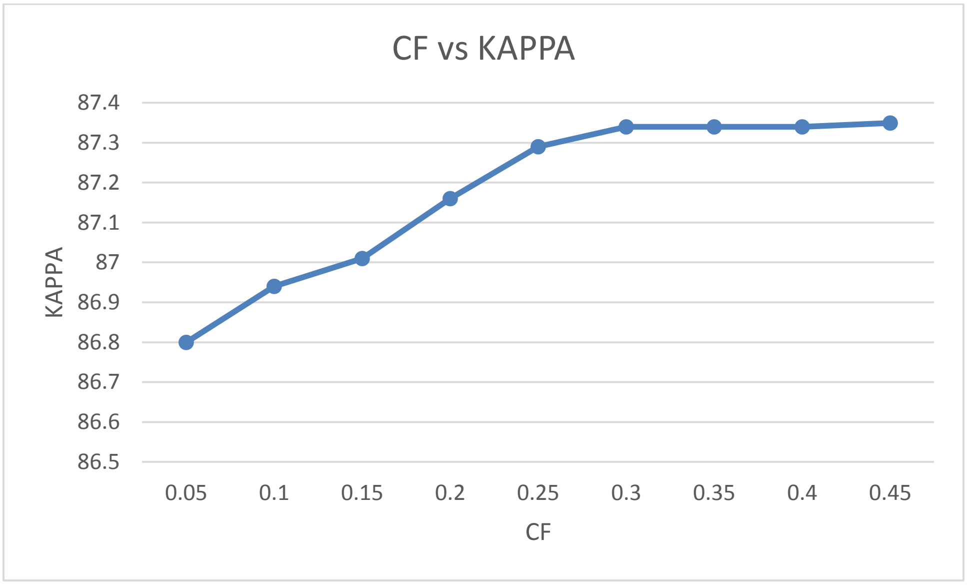


Figure 28 MUSHAF performance tuning results: CF vs KAPPA

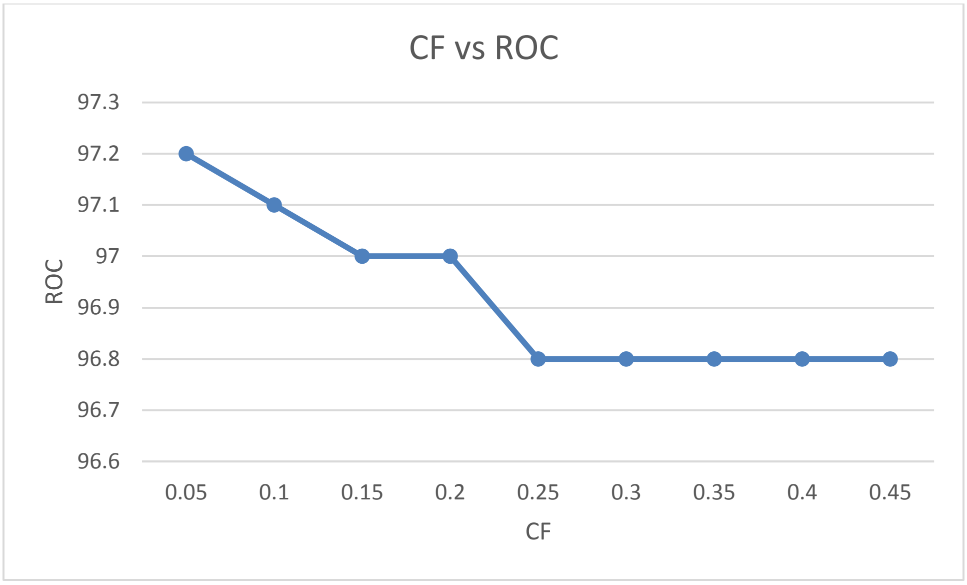


Figure 29 MUSHAF performance tuning results: CF vs ROC

Table 33: J48 classifier performance turning results

	"Tashkeel-2016"				MUSHAF		
CF	**Accuracy**	**KAPPA**	**Avg ROC**		**Accuracy**	**KAPPA**	**Avg ROC**
0.05	86.15%	80.95%	97.00%		90.37%	86.80%	97.20%
0.1	86.45%	81.39%	96.80%		90.47%	86.94%	97.10%
0.15	86.56%	81.55%	96.70%		90.51%	87.01%	97.00%
0.2	86.62%	81.64%	96.50%		90.62%	87.16%	97.00%
0.25	86.71%	81.77%	96.30%		90.72%	87.29%	96.80%
0.3	86.72%	81.79%	96.20%		90.75%	87.34%	96.80%
0.35	86.72%	81.79%	96.10%		90.75%	87.34%	96.80%
0.4	86.68%	81.74%	96.00%		90.75%	87.34%	96.80%
0.45	86.62%	81.65%	95.90%		90.75%	87.35%	96.80%
0.5	86.57%	81.59%	95.90%		90.74%	87.33%	96.70%

To calculate DER and WER, we used the best datasets from the previous experiments along with the best settings from performance tuning. Table 34 shows the results.

Table 34: "TASHKEEL-2016" and MUSHAF DER and WER results

Model	DER without case ending	DER	WER without case ending	WER
"TASHKEEL-2016"	7%	13%	20%	37%
MUSHAF	6%	9%	18%	28%

We see from the results that the MUSHAF dataset performed better than the dataset from the "TASHKEEL-2016". The highest results achieved was a 6% and 9% for DER (without case ending) and DER respectively. Also, a result of 18% and 28% was achieved for both the WER (without case ending) and WER respectively.

5.4 Comparison

To validate our work, we considered comparing our work with other researchers' work. The problem was that the majority of reported related researches were either not available or licensed. We had access to only one research by Shaaban [9]. We used his testing set, and produced a fully vocalized and consistent subset. The test set we used will be made available public.

Table 35 shows the comparison results between both systems. One thing to note about Shabban's system, is that the level of vocalization is around 81%, while in our system it is 100%. Thus to make the comparison fair, we considered each undiacritized letter as a misclassified letter. We can see from the results that our systems performs better.

Table 35: Systems comparison

System	DER1	DER2	WER1	WER2
Shabban	36.28%	36.28%	78.37%	78.37%
Our System	9.8%	9.76%	30.81%	30.81%

5.5 Summary

In this chapter, we introduced the most commonly evaluation metrics that researchers use to test their vocalization systems. A set of experiments were conducted to determine the best classification rate and the best models. Performance tuning was performed to enhance the vocalization results.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

Restoring diacritics of unvocalized text is an active research area. Several studies have been pursued with different languages and different approaches. However using decision trees for vocalization have been not been explored thoroughly. We showed in this thesis that using decision trees classifier (J48) is effective in the area of Arabic text vocalization.

In this thesis we developed a corpus that contains only MSA text, and due to the nature of the domain selected in developing the corpus, we had to vocalize it using our own built vocalizer. We also refined the “TASHKEEL-2016” and the MUSHAF corpora.

Through the development process of the automatic vocalizer, we conducted feature extraction and produced 23 features. These features were later reduced to 13 features using feature selection. Feature selection was used through the wrapper evaluator within WEKA which employed the J48 classifier as its learning scheme. Stanford and MUSHAF POS taggers were used in POS features. Furthermore, to enhance the vocalization, voting and n-gram models were introduced.

For the evaluation we used several metrics such as DER, WER, KAPPA and others. We evaluated our experiments based on these metrics. We tuned the used parameters of the classifier for possible better results. The highest result achieved was with the MUSHAF corpus. Results of 6% and 9% for DER (without case ending) and DER were achieved respectively. Also, Results of 18% and 28% were achieved for both the WER (without case ending) and WER respectively.

Finally, we compared our work with a previous work [9]. The comparison showed that our vocalizer performed better.

6.2 Future work

The features we came up with did not consider partially diacritized text. Thus, a possible alternative approach would be to come up with features selected particularly for partially diacritized text.

As the “AKHBAR-2016” corpus we developed was diacritized using our developed system, the corpus needs to be validated to make sure that the diacritized content is more accurate.

Another future work would be to use the corpus we have to build a minimal corpus. The minimal corpus would cover different linguistic cases, letters with all their possible shapes or letters with all their possible diacritics…etc.

References

[1] J. Owens, The Oxford handbook of Arabic linguistics, Oxford University Press, 2015.

[2] A. D. Rubin, "The Subgrouping of the Semitic Languages," *Language and Linguistics Compass,* vol. 2, no. 1, pp. 79-102, 2008.

[3] K. Kirchhoff, D. Vergyri, J. Bilmes, D. Kevin and A. Stolcke, "Morphology-based Language Modeling for Conversational Arabic Speech Recognition," *Computer Speech & Language,* pp. 589-608, 2006.

[4] D. Vergyri and K. Kirchhoff, "Automatic diacritization of Arabic for acoustic modeling in speech recognition," in *Proceedings of the workshop on computational approaches to Arabic script-based languages. Association for Computational Linguistics*, 2004.

[5] A. Messaoudi, L. Lamel and J.-L. Gauvain, "The LIMSI RT-04 BN Arabic System," in *Proceedings of the EARS RT-04 Workshop*, 2004.

[6] M. Elshafe, H. Al-Muhtaseb and M. Al-Ghamdi, "Techniques for High Quality Arabic Speech Synthesis," *Information sciences,* vol. 140, no. 3, pp. 255-267, 2002.

[7] Y. Hifny, S. Qurany, S. Hamid, M. Rashwan, M. Atiyya, A. Ragheb and G. Khallaaf, "ArabTalk®: An Implementation for Arabic Text To Speech System," in *The proceedings of the 4th Conference on Language Engineering*, 2004.

[8] H. Trost, "Recognitionand Generation Of Word Forms For Natural Language Understanding Systems: Integrating Two-Level Morphology," *Applied Artificial Intelligence an International Journal,* vol. 5, no. 4, pp. 411-457, 1991.

[9] O. Shaaban, "Automatic Diacritic Restoration for Arabic Text," King Fahd University of petroleum and minerals, Al-Dahran, 2012.

[10] H. Zarrabi-Zadeh, "Tanzil Quran Text," [Online]. Available: http://tanzil.net/download/. [Accessed 2 3 2016].

[11] R. F. Mihalcea, "Diacritics restoration: Learning from letters versus learning from words," in *Computational linguistics and intelligent text processing*, 2002.

[12] R. Mihalcea and V. Nastase, "Letter level learning for language independent diacritics restoration," in *Proceedings of The 6th conference on natural language learning*, 2002.

[13] Y. Gal, "An HMM approach to vowel restoration in Arabic and Hebrew," in *Proceedings of the ACL-02 workshop on Computational approaches to semitic languages*, 2002.

[14] D. Crandall, "Automatic accent restoration in Spanish text," Indiana University Bloomington, 2005.

[15] M. Elshafei, H. Al-Muhtaseb and M. Alghamdi, "Statistical methods for automatic diacritization of Arabic text," in *The Saudi 18th National Computer Conference*, Riyadh, 2006.

[16] M. Elshafei, H. Al-Muhtaseb and M. Al-Ghamdi, "Machine Generation of Arabic Diacritical Marks," in *The 2006 International Conference on Machine Learning*, 2006.

[17] A. Harby, M. Shehawey and R. Barogy, "A statistical approach for Quran vowel restoration," *ICGST International Journal on Artificial Intelligence and Machine Learning,* vol. 8, no. 3, pp. 9-16, 2008.

[18] J. A. Mahar, G. Q. Memon and H. Shaikh, "Automatic Diacritics Restoration System for Sindhi," *Sindh University Research Journal,* vol. 43, no. 1, 2011.

[19] M. Khorsheed, "A HMM-Based System To Diacritize Arabic Text," *Journal of Software Engineering and Applications,* vol. 5, pp. 124-127, 2012.

[20] Microsoft, "Hidden Markove Model Toolkit," Microsoft, 29 November 2015. [Online]. Available: http://htk.eng.cam.ac.uk. [Accessed Sunday November 2015].

[21] Y. Hifny, "Higher Order n-gram Language Models for Arabic Diacritics Restoration," in *Proceedings of the 12th Conference on Language Engineering (ESOLEC'12),* Cairo, Egypt, 2012.

[22] Y. Hifny, "Smoothing techniques for Arabic diacritics restoration," in *Proceedings of the 12th Conference on Language Engineering (ESOLEC'12),* Cairo, 2012.

[23] M. Bebah, C. Amine, M. Azzeddine and L. Abdelhak, "Hybrid Approaches for Automatic Vowelization of Arabic Texts," *International Journal on Natural Language Computing (IJNLC),* vol. 3, no. 4, August 2014.

[24] S. Harrat, M. Abbas, K. Meftouh and K. Smaili, "Diacritics restoration for Arabic dialect texts," in *14th Annual Conference of the International Speech Communication Association*, Lyon, France, 2013.

[25] M. Alghamdi and a. Z. Muzaffar, "KACST Arabic diacritizer," in *The First International Symposium on Computers and Arabic Language*, 2007.

[26] K. Shaalan, H. M. Abo Bakr and I. Ziedan, "A hybrid approach for building Arabic diacritizer," in *Proceedings of the EACL 2009 workshop on computational approaches to semitic languages. Association for Computational Linguistics*, 2009.

[27] M. Rashwan, M. Attia, S. Abdou, M. Al Badrashiny and A. Rafea, "Stochastic Arabic hybrid diacritizer," in *Natural Language Processing and Knowledge Engineering*, 2009.

[28] R. A. Haraty, M. M. Allaham and A. El-Homaissi, "Towards Diactritizing Arabic Text," in *International Conference on Computer Applications in Industry and Engineering*, 2013.

[29] B. Rosenfeld, "A Comparison of Letterl-Level Classifiers of Portuguese Diacratic and Capitalization Restoration," Princeton University, 2014.

[30] E. Al-Thwaib, "Text Summarization as Feature Selection for Arabic Text Classification," *World of Computer Science and Information Technology Journal (WCSIT),* vol. 4.7, no. 2221-0741, pp. 101-104, 2014.

[31] A. Almuhareb, W. A. Almutairi, H. Altuwaijri, A. Almubarak and M. Khan, "Recognition of Modern Arabic Poems," *Journal of Software,* vol. 10, no. 4, pp. 454-464, 2015.

[32] K. Gal, "Hebrew vowel restoration using a bigram HMM model," Harvard University, 2011.

[33] "Arabi NLP Website," [Online]. Available: http://www.arabinlp.com/. [Accessed 1 5 2013].

[34] "Mishkal Diacritization Tool," [Online]. Available: http://sourceforge.net/projects/mishkal/. [Accessed 14 6 2016].

[35] M. A. Rashwan, M. A. Al-Badrashiny, M. Attia, S. M. Abdou and A. Rafea, "A Stochastic Arabic Diacritizer Based on a Hybrid of Factorized and Unfactorized Textual Features," *Audio, Speech, and Language Processing, IEEE Transactions on,* vol. 19, no. 1, pp. 166-175, jan. 2011.

[36] "Sakhr Software," [Online]. Available: http:/www.sakhr.com. [Accessed 14 6 2016].

[37] Alexa, "Alexa - Actionable Analytics for the Web," [Online]. Available: http://www.alexa.com. [Accessed 29 November 2015].

[38] "https://www.httrack.com," [Online]. Available: https://www.httrack.com. [Accessed 29 November 2015].

[39] H. Liu, H. Motoda, R. Setiono and Z. Zhao, "Feature Selection: An Ever Evolving Frontier in Data Mining," *Fuzzy Systems and Data Mining,* vol. 10, pp. 4-13, 2010.

[40] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *International Conference on Machine Learning*, 2003.

[41] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research,* vol. 3, no. Mar, pp. 1157-1182, 2003.

[42] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD explorations newsletter,* vol. 11, no. 1, pp. 10-18, 2009.

[43] J. R. Quinlan, C4.5: Programs for Machine Learning, Elsevier, 2014.

[44] J. R. Quinlan, "Induction of decision trees," *Machine learning,* vol. 1, no. 1, pp. 81-106, 1986.

[45] K. Toutanova, D. Klein, C. D. Manning and Y. Singer. [Online]. Available: http://nlp.stanford.edu/software/tagger.shtml. [Accessed 21 3 2016].

[46] K. Dukes, 2009. [Online]. Available: http://corpus.quran.com. [Accessed 2 4 2016].

[47] K. Toutanova, D. Klein, C. D. Manning and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, Stroudsburg, 2003.

[48] J. Frijters. [Online]. Available: http://www.ikvm.net. [Accessed 21 3 2016].

[49] V. Balasubramanian, S. Ho and V. Vovk, Conformal Prediction for Reliable Machine Learning: Theory, Adaptations and Applications, Newnes, 2014.

[50] M. Dash and H. Liu, "Feature selection for classification," *Intelligent data analysis,* vol. 1, no. 3, pp. 131-156, 1997.

[51] G. H. John, R. Kohavi and K. Pfleger, "Irrelevant features and the subset selection problem," in *Machine learning: proceedings of the eleventh international conference*, 1994.

[52] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data Engineering,* vol. 15, no. 6, pp. 1437 - 1447, 2003.

[53] A. G. Karegowda, M. Jayaram and A. Manjunath, "Feature Subset Selection Problem using Wrapper Approach in Supervised Learning," *International journal of Computer applications,* vol. 1, no. 7, pp. 13-17, 2010.

[54] I. Zitouni, J. S. Sorensen and R. Sarikaya, "Maximum entropy based restoration of Arabic diacritics," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. Association for Computational Linguistics*, Stroudsburg, PA, USA, 2006.

[55] E. C. Lemnaru, "Strategies for dealing with real world classification problems," Technical University of Cluj-Napoca, 2012.

[56] O. Villacampa, "Feature Selection and Classification Methods for Decision Making: A Comparative Analysis," Nova Southeastern University, 2015.

Appendix I How to Vocalize a Text File

To vocalize a text file, the following steps needs to be followed:

1. Run the Instant Diacritizer application. Figure 30 shows the Instant diacritizer main screen.
2. Assuming that we do not have any models, we need to generate models for classification. If you already have any model, you can skip to step 8.

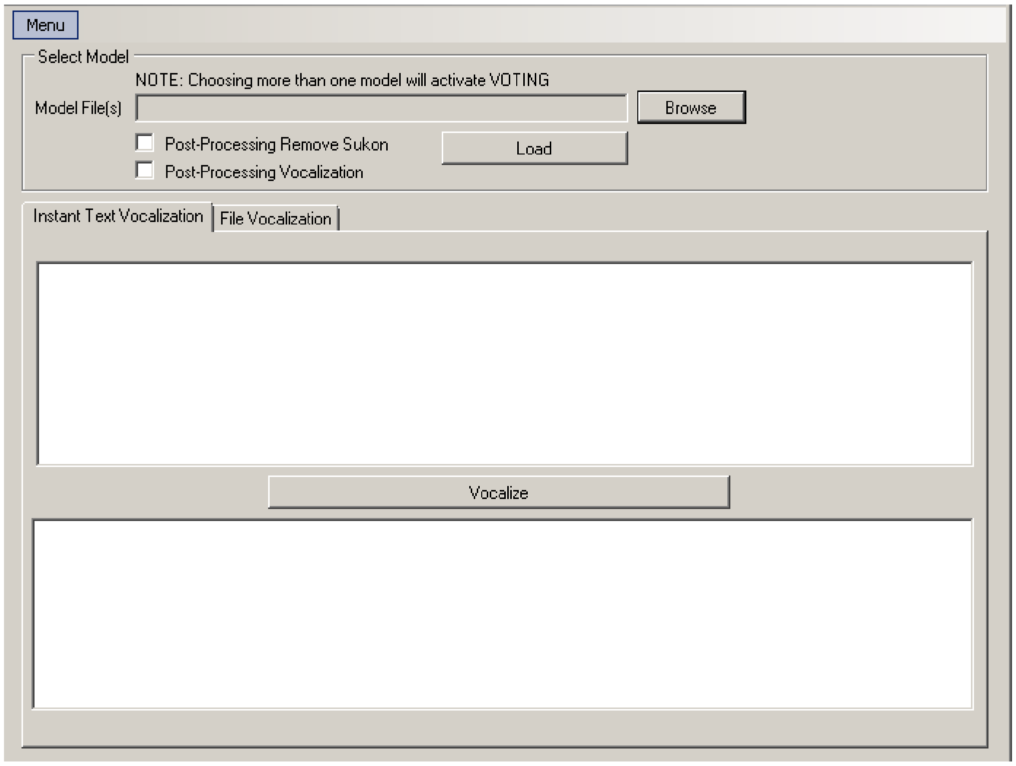
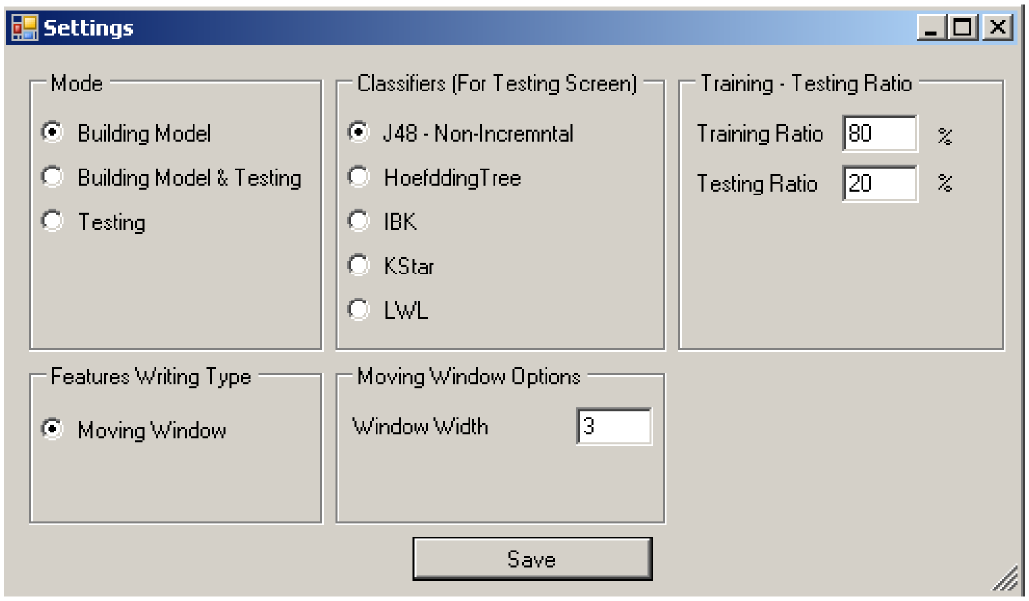

Menu
Select Model
NOTE: Choosing more than one model will activate VOTING
Model File(s)
Browse
Post-Processing Remove Sukon
Load
Post-Processing Vocalization
Instant Text Vocalization
File Vocalization
Vocalize

Figure 30 Instant Diacritizer Main Screen

3. Click on the menu button located on the upper left of the screen. Then click on the “Settings” menu item. This will open up a new window which contains all the settings for building classification models. Figure 31 shows the settings screen.



Mode

[✓ Building Model]
Building Model & Testing
Testing

Classifiers (For Testing Screen)

[✓ J48 - Non-Incremntal]
HoefddingTree
IBK
KStar
LWL

Training - Testing Ratio

Training Ratio: 80 %
Testing Ratio: 20 %

Features Writing Type

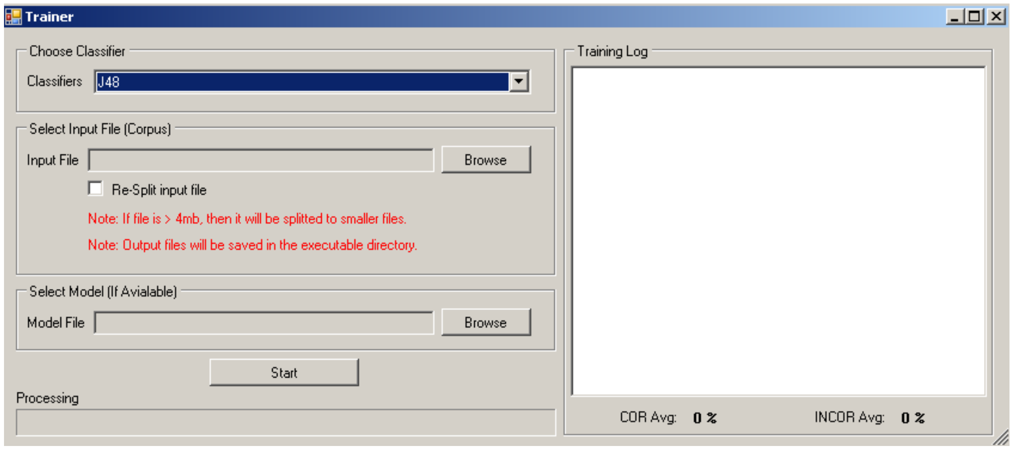
[✓ Moving Window]

Moving Window Options

Window Width: 3

Figure 31 Instant Diacritizer Settings Screen

4. Make sure that the settings are set exactly as shown in Figure 31. After that, click on the "Save" button.
5. In the main screen, click on the menu button again. Then click "Trainer & Tester" menu item. Figure 32 shows the Trainer & Tester screen.



Choose Classifier

Classifiers: J48

Select Input File (Corpus)

Input File

Re-Split input file

Note: If file is > 4mb, then it will be splitted to smaller files.

Note: Output files will be saved in the executable directory.

Select Model (If Avialable)

Model File

Processing

Training Log

COR Avg: 0 %   INCOR Avg: 0 %

Figure 32 Trainer & Tester Screen

6. Make sure that the Classifier “J48” is selected. Click on the browse button for the input file. For building models, the input file must be a vocalized text file so that it can be used for Training. After choosing the appropriate input file, click on the Start button to start the training process.
7. After the training process finishes, a log will be inserted in the “Training Log” and process progress bar will be full. The models built will be located in a folder named “Model” which can be found in application root directory. After that, close the “Trainer” screen.
8. In the main screen, click on the browse button and select all the models built in the previous steps. Then click on the “Load” button. Upon clicking the “Load” button, the screen will be disabled until the loading process is finished.
9. After loading is finished, check both options:
    a. Post-Processing Remove Sukoon (Sukon).
    b. Post-Processing Vocalization.
10. Go to the file vocalization tab. Figure 33 shows the file vocalization tap.

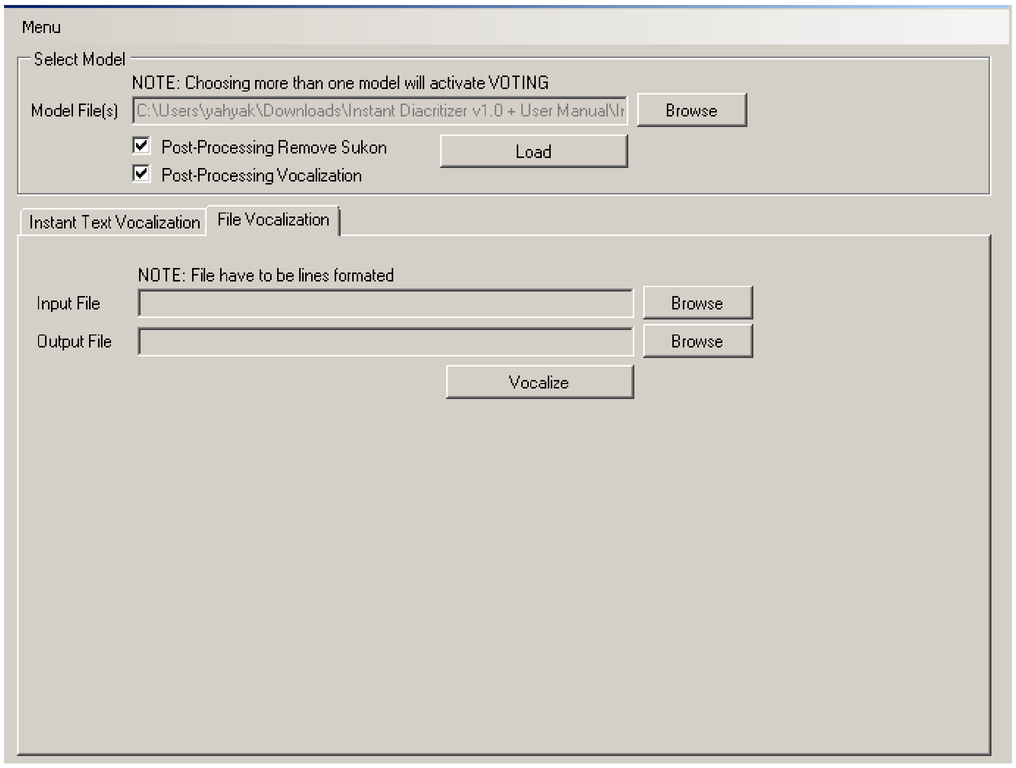

Menu
Select Model
NOTE: Choosing more than one model will activate VOTING
Model File(s)
C:\Users\yahyak\Downloads\Instant Diacritizer v1.0 + User Manual\I
Browse
☑ Post-Processing Remove Sukon
Load
☑ Post-Processing Vocalization
Instant Text Vocalization
File Vocalization
NOTE: File have to be lines formated
Input File
Browse
Output File
Browse
Vocalize

Figure 33 Instant Diacritizer Main Screen - File Vocalization Tap

11. Click on the "Browse" button for the input file, and select the input file you want to vocalize. Also, click on the "Browse" button for the output file name and location.
12. Click on the "Vocalize" button, and wait for the vocalization process to finish.

Vitae

Personal Information

Gender:	Male
Nationality:	Jordanian
Birth Place:	Saudi Arabia
Birth Date:	27/10/1988
E-Mail:	yahya.khraishi@gmail.com
Address:	Amman, Jordan

Education

2016	**King Fahd University of Petroleum & Minerals, Dahran, Saudi Arabia** Master of Science in Computer Science ***(GPA 3.34 out of 4)***
2010	**Al-Zaytoonah University, Amman, Jordan** Bachelor in Computer Science ***(Cumulative Average 83.5%)***

Publications

2016 *Khrishe, Yahya, and Mohammad Alshayeb. "An empirical study on the effect of the order of applying software refactoring." Computer Science and Information Technology (CSIT), 2016 7th International Conference on. IEEE, 2016.*

Experience

2016-now **SSS Process**, Amman, Jordan
"*Senior Software Engineer*"

- Developing ASP.Net applications using MVC technology under C# using Visual Studio 2015.
- Database development using SQL 2015.

3/2016- 6/2016 **OrderMe,** Dahran, Saudi Arabia
"*Senior Software Developer*"

- Integration and customization of an open source Java project "JSprit" for solving vehicle routing problem

6/2012- 7/2013 **Safat Enterprise Solutions,** Kuwait
"*Senior Software Developer*"

- Developing ASP.Net and Windows applications using third party component "DevExpress" under C#.
- Database development using SQL 2008, 2012.
- Design and Development of Crystal Reports.
- Worked in the following projects:
  - MyApp – National Bank of Kuwait (NBK).
  - Operation Document Flow – Kuwait Financial House (KFH).

12/2011- 6/2012 **CrownIT,** Amman, Jordan
"*Software Developer*"

- Developing ASP.Net Applications under C#.
- Worked on Ajax, JQuery, JavaScript, JSON, and XML.
- Tuning of SMPP application.
- Integration and customization of open source projects with existing applications, such as Blogs, Forums, Wikipedia, etc.
- Database development using SQL2008.
- Worked in the following projects:
  - www.icn.com
  - www.fxpulp.com

10/2010- 11/2011 **HyperExecution,** Amman, Jordan
"*Software Developer*"

- Developing ASP.Net and Windows Applications under C# using Visual Studio 2005, 2008.
- Worked on Ajax and Javascript.
- Developing Windows Mobile Application - Windows Mobile 6.0 using C#.
- Database development using SQL2005, SQL2008.
- Design and development of Crystal Report.
- Worked in the following projects:
  - www.hiwash.com
  - www.hiwashportal.com
  - (Hammurabi) Law firm application.

Programming Languages & Computer Skills

Programming Languages

- ASP.Net
- ADO.Net
- C#.Net
- VB.Net
- SQL Server 2005 ,2008, 2012 :-
    - Database design and implementation.
    - Relations and constrains.
    - Queries and procedures.

Web Design Skills

- Scripting: HTML, CSS, JQuery, JavaScript and Json.

Operating System& Maintenance Skills

- Microsoft Windows (98, XP, Vista , 7, 8, 10)
- Experience in troubleshooting all problems in windows
- Microsoft Office 2003, 2007, 2010
  (Very Good at Word, Good PowerPoint, Excel and Access)
- Experience in troubleshooting and maintaining PC Hardware
- Experience in troubleshooting and maintaining windows applications
- Ability to get familiar with applications quickly

Graphic Design Skills

- Adobe Photoshop
- Microsoft Expression Design