International Journal of Innovative Computing, Information and Control
Volume **15**, Number **3**, June **2019**

ICIC International ©2019 ISSN 1349-4198
pp. 1009–1021

PERFORMANCE EVALUATION OF SPHINX AND HTK SPEECH RECOGNIZERS FOR SPOKEN ARABIC LANGUAGE

Fawaz S. Al-Anzi* and Dia AbuZeina

Computer Engineering Department
Kuwait University
P.O. Box 5969 Safat, Kuwait City 13060, Kuwait
*Corresponding author: fawaz.alanzi@ku.edu.kw

Received August 2018; revised December 2018

Abstract. *Automatic speech recognition (ASR) has lately been a focus consideration of researchers with respect to more convenient human-computer interaction. Despite the successful implementation of ASR technology in different languages, employing this technology in Arabic natural language processing (NLP) applications is limited and constrained to a small vocabulary such as digits and control commands or a limited set of words. Therefore, particular attention has been paid to promoting research in this field to automate man-machine communication. We aim to examine the performance of two popular ASR engines for identical Arabic speech collection. The ASR engines include the Carnegie Mellon University (CMU) Sphinx and the Hidden Markov Model Toolkit (HTK). In fact, performing an ASR task using different recognizers will increase researcher knowledge regarding which engine is the best fit for particular target applications, as well as enhancing research in this field. In this paper, an experimental evaluation is presented for both Sphinx and HTK recognizers using a new "in-house" Arabic continuous speech corpus that contains a total of 15.93 hours (12.74 training hours and 3.19 testing hours). The vocabulary contains 30,986 words. In these experiments, we used two text formats, Arabic characters for CMU Sphinx (PocketSphinx decoder) and Roman characters for HTK (HVite decoder) because HTK expects Roman characters. The experimental comparison shows that Sphinx outperforms (even in a shorter time) HTK. In addition, this study demonstrates the intermediate steps followed for models training including acoustic and language models.*

Keywords: Arabic speech recognition, Sphinx, HTK, HVite, Buckwalter, Language model, Pronunciation dictionary

1. Introduction.

Arabic is the most widely spoken Semitic language today that recently has received significant attention of researcher for automatic speech recognition (ASR). ASR is a component of the natural language processing (NLP), which is used to automate the communication process between human and machine, i.e., the man-machine interaction. In this regard, much research has been devoted to introducing innovative technologies in dialogue systems for automation purposes (e.g., banking services, cars, and control machines). However, employing ASR technology in Arabic NLP applications is still limited due to various challenges about within the Arabic language itself. For instance, it is challenging to attain corpora for spoken dialects rather than written, i.e., there is a lake of writing standard as well as difficulty in obtaining a sizable diacritized text since Arabic language allows writing without such diacritics. Therefore, there are enormous numbers of word possible forms due to the morphology richness of Arabic. In fact, one of the most difficult tasks in Arabic ASR is preparing a large diacritized text

DOI: 10.24507/ijicic.15.03.1009

for ASR systems, which is a time-consuming preprocessing stage. In order to promote research on the Arabic ASR, we considered the corpora availability problem by producing a manually diacritized large-vocabulary speaker-independent continuous speech corpus for Modern Standard Arabic (MSA). The contents of the prepared corpus belong to general broadcast news.

In this work, we used the prepared corpus for an experimental evaluation of two off-the-shelf open source speech recognition toolkits, namely the Carnegie Mellon University (CMU) Sphinx [1] and the Hidden Markov Model Toolkit (HTK) [2]. In fact, it is important to find the performance of the popular speech engines using an identical speech collection because it reveals the unique characteristics of each engine for further understanding of their behavior in NLP and ASR applications. With the growing interest in ASR technology, it becomes more important to evaluate recent ASR systems in order to find the best-suited system for the tasks in question. For instance, the study in [3] demonstrated a large-scale evaluation of open-source speech recognition toolkits that include Sphinx, HTK, and Kaldi [4]. The study in [3] indicated that Kaldi is better than Sphinx and HTK in terms of results and training recipes (for the German and English languages).

The work with the Arabic ASR considers one of two textual formats, either Arabic characters or Roman characters. Accordingly, there are two main approaches to address the training with Arabic text. The first approach considers using Arabic alphabet characters such as the Sphinx recognizer; the second approach uses Roman characters such as the HTK recognizer. It is sometimes required to convert Arabic characters into Roman characters, especially when recognizers assume that ASCII is used to write the training textual files rather than Unicode. HTK expects Roman characters, which means that Romanization of the Arabic ASR tasks must be completed. From the ASR point of view, Romanization means that the recognizer is trained on the Romanized transcriptions of the data. In general, choosing a recognizer on the basis of accepting Arabic characters should have no difference in performance since the cores of the algorithms of training and decoding are the same in the different ASR engines. One purpose of this work is to evaluate the performance using the Sphinx and HTK recognizers. Therefore, we choose Buckwalter (BW) transliteration for the Romanization process in the HTK recognizer. There are many character options for Romanization; however, BW transliteration [5] is a good option for Romanization as it has the following two advantages: namely, it is popular, so the data can be easily made available to others as well as making it possible to use the data of other people.

In this paper, we demonstrate an experimental evaluation of two cases: the Arabic character-based recognizer is the CMU Sphinx (PocketSphinx decoder), and the Roman character-based recognizer is the HTK version 3.4.1 (HVite decoder). However, the HTK toolkit has another famous decoder, which is 'HDecode', but in this work, we used HVite. As future work, we will aim at implementing HDecode for the Arabic ASR. The CMU Sphinx toolkit includes the latest available releases as follows: 'Sphinxbase-5Prealpha', 'PocketSphinx-5prealpha', and 'SphinxTrain-5prealpha'. For the experiments, we used a new "in-house" continuous speech corpus that contains 15.93 hours of MSA speech. The training set contains 12.74 hours (1,611 speech files), while the testing set contains 3.19 hours (403 speech files). This study also presents the intermediate steps to train the models as well as the Romanization process. For fairness in the comparison, we emphasized that the training set and the testing set are identical in both systems (i.e., the same wave files, the same diacritized text, and the same phonemes set but with different symbols). Based on our best knowledge, this attempt is the first to experimentally compare the Sphinx and HTK recognizers for continuous Arabic speech.

This paper is structured as follows. Section 2 presents a review of the literature. In Section 3, phonemes and the pronunciation dictionary are presented. The language models are explained in Section 4. The implementation and the results of the Sphinx and HTK recognizers are presented in Section 5, followed by the conclusion in Section 6.

2. **Literature Review.** Textual training data are an essential part of any speech corpus as they represent speech transcription. However, not all open-source recognizers can handle Arabic characters, which may lead to complications in ASR implementation. In fact, most of the current recognizers are designed for the English language, which is sometimes not compatible with the Arabic language due to the character representations. The literature shows that two types of textual data are used according to the employed recognizer. If the used recognizer supports Arabic characters, then it is straightforward to use the Arabic characters. However, if the recognizer cannot handle the Arabic characters, then the choice is to consider Romanization. In general, Arabic ASR researchers use either Sphinx or HTK for the recognition task. However, no single work presents a comparison between both systems, which is the motivation of this work. Based on the literature, most Arabic ASR studies employ the CMU Sphinx engine because it is compatible with Arabic characters. However, few works have used the HTK engine based on Romanization. For instance, previous studies that used the CMU Sphinx are as follows: Hyassat and Zitar in [6] employed Sphinx tools for the Holy Quran speech recognition, Alghamdi et al. in [7] used Sphinx tools for an MSA ASR system, AbuZeina et al. in [8] used the Sphinx tool for crossword Arabic pronunciation variation modeling for ASR, AbuZeina et al. in [9] used the Sphinx tool for within-word Arabic pronunciation variation modeling for ASR, Abushariah et al. in [10] used Sphinx tools for an Arabic ASR system based on a phonetically rich and balanced speech corpus, and Al-Anzi and AbuZeina reported the most recent Arabic ASR work [11] that is based on Arabic characters. They used the CMU Sphinx to evaluate the impact of phonological rules on the Arabic ASR.

The literature shows that employing Roman characters for the Arabic ASR is less than what was observed for the Arabic characters. The following are results from previous studies. Kirchhoff et al. in [12] proposed the use of the Romanization method for the transcription of the speech corpus. Vergyri et al. in [13] used Romanized transcriptions to train the speech recognizer. Kirchhoff et al. in [14] used Romanized transcription in language modeling for large-vocabulary conversational Arabic speech recognition. Satori et al. in [15] introduced an Arabic voice recognition system where both the training and recognizing processes use the Romanized characters. Elmahdy et al. in [16] used Sphinx tools in addition to the SAMPA Romanization method for dialectal Arabic speech recognition. Al-Qatab and Ainon in [17] employed HTK for a small corpus of continuous speech and isolated words; however, they did not indicate anything regarding Romanization. The study in [18] used BW Arabic transliteration for Tunisian dialect dialogue systems. A recent work using HTK was reported by Merad-Boudia et al. in [19], and they employed the HTK engine for a small corpus of isolated and connected words. In summary, the literature shows that no single work has conducted practical research to evaluate the performance using an identical speech corpus, which indicates the importance and originality of this work.

3. **Phonemes Set and Pronunciation Dictionaries.** In order to train the models for ASR tasks, a speech corpus is required. This contains a set of speech files and corresponding textual transcription. The textual transcription is used to find the phonetic transcriptions through the phonemes set. Therefore, defining the phonemes set is the initial step in ASR work. A phoneme is the basic unit of sound that ASR uses for

classification. While it is possible to use words and syllables for ASR classification, the phoneme approach is the most widely used in recent ASRs. On the other hand, the number of phonemes for Arabic is still a debated matter. For instance, Alghamdi et al. in [7] used 46 phonemes. Al-Qatab and Ainon in [17] used 34 phonemes (28 consonants and 6 vowels). Elmahdy et al. in [16] indicated that MSA consists of 38 phonemes, where 28 are original consonants, 4 are foreign and rare consonants, and 6 are vowels. Haraty and Ariss in [20] indicated that Arabic has at least 112 phonemes, as they considered that every letter has four diacritics and therefore four phonemes. Alotaibi in [21] used 37 MSA phonemes as given by the Language Data Consortium (LDC).

Hence, the defined phonemes set in addition to the corpus transcription are used to generate the pronunciation dictionary that contains the phonetic transcription. In this work, the employed ASR recognizers have the following two options to handle the Arabic characters: the Sphinx recognizer allows the use of Arabic characters. Therefore, we do not need to consider Romanization. However, HTK expects Roman characters. That is, if the corpus textual transcription is stored using the standard Arabic characters set, then it needs to be transcribed into something that HTK can handle. For the Arabic ASR, BW transliteration has a distinguished attribute in which it uses one Roman character for each Arabic character, making it reasonable to approximate that each Arabic character corresponds to a single phoneme. For instance, the phonetic transcription of the word "kataba", which means "he wrote", is "k a t a b a". However, BW uses a number of non-alphabetic characters to consider in the conversion process (e.g., BW uses "$" for the Arabic letter 'Sheen' "ش" which is a special character in some recognizers such as HTK). Hence, the conversion of the Arabic text to BW transliteration requires a further step to swap the non-alphabetic characters with some other arbitrary characters. Of course, it is straightforward to convert the Romanized text back to the Arabic transcription in the case of the need to output in Arabic. Table 1 shows the Arabic and Romanized characters using BW transliteration. In the table, we indicate some replacement cases.

In fact, using either Roman or Arabic characters is a non-issue since there is no real difference between both types of transcription. It is just a matter of handling the textual characters. Therefore, the performance of the ASR recognizer based on Arabic characters should have no significant difference from one based on Roman characters. However, Romanization is harder to read for the native Arabic speaker. Of course, it is straightforward to do the transliteration in both directions in case of the need to output in the Arabic transcription.

In addition to the phonemes set shown in Table 1, we use three more phonemes to represent the Fatha that proceeds Alif (ا) ➔ ـَا as a single phoneme that is (aA), the Damma that proceeds Waw (و) ➔ ـُو as a single phoneme that is (uw), and the Kasra that proceeds Ya (ي) ➔ ـِي as (iy). The reason for handling these cases as a single phoneme is that the pronunciation of the short vowels is different when it proceeds the long vowels. Hence, it will be correctly transcribed as single phonemes. Therefore, we used 46 phonemes for the HTK system, 42 phonemes in Table 1 (Shadda (◌ّ<>~) and Sukun (◌ْ<>o) are discarded) in addition to the three phonemes (aA, iy, and uw) and the sil phonemes to indicate silence. In the Sphinx system, decoding fails when the phonemes set contains small letter and capital letter symbols. For instance, it fails when the phonemes set has a phoneme b and another phoneme B. However, this problem has not been observed in the HTK system. Hence, we used a new phonemes set for Sphinx that contains 45 phonemes (◌ّ is not counted since it is duplicated) as shown in Table 2. The phonemes set contains 46 phonemes as the set that is used for the HTK system. Hence, both systems have the same phonemes set.

TABLE 1. Buckwalter (BW) transliteration with some replacements

#	Arabic character	Phoneme (BW)	#	Arabic character	Phoneme (BW)
1	ا	A	23	ك	k
2	أ	> (replaced to) O	24	ل	l
3	ب	B	25	م	m
4	ت	T	26	ن	n
5	ث	v	27	و	w
6	ج	j	28	ى	Y
7	ح	H	29	ه	h
8	خ	x	30	ي	y
9	د	d	31	آ	| (replaced with) U
10	ذ	* (replaced to) J	32	ء	G
11	ر	r	33	ؤ	& (replaced with) W
12	ز	z	34	ئ	} (replaced with) Q
13	س	s	35	ة	p
14	ش	$ (replaced to) C	3 6	◌ً	F
15	ص	S	37	◌ٌ	N
16	ض	D	38	◌ٍ	K
17	ط	T	39	◌َ	a
18	ظ	Z	40	◌ُ	u
19	ع	E	41	◌ِ	i
20	غ	g	42	◌ّ	~ (duplicate previous)
21	ف	f	43	◌ْ	o (not used)
22	ق	q	44	إ	> (replaced with) I

TABLE 2. Proposed phonemes set for the Sphinx recognizer

#	Letter	Phoneme	#	Letter	Phoneme	#	Letter	Phoneme
1	ء	E	17	ر	R	33	ه	H
2	آ	AA	18	ز	Z	34	و	W
3	أ	O	19	س	S	35	ى	AY
4	ؤ	EW	20	ش	SH	36	ي	Y
5	إ	I	21	ص	SS	37	◌ً	AU
6	ئ	EY	22	ض	DD	38	◌ٌ	AWW
7	ا	A	23	ط	TT	39	◌ٍ	AIY
8	ب	B	24	ظ	ZZ	40	◌َ	AU
9	ة	P	25	ع	AE	41	◌ُ	AW
10	ت	T	26	غ	GH	42	◌ِ	AI
11	ث	TH	27	ف	F	43	◌ّ	duplicate
12	ج	J	28	ق	Q	44	ـَا	AUA
13	ح	HH	29	ك	K	45	ـُو	AWW
14	خ	KH	30	ل	L	46	ـِي	AIY
15	د	D	31	م	M			
16	ذ	DH	32	ن	N			

The pronunciation dictionary is an ASR component that contains the phonetic transcription of each word. That is, each word that appears in the training text is listed in the dictionary in terms of phonemes. Nevertheless, generating a pronunciation dictionary is a hard task due to the different acoustic cases of pronunciations. In the case of a large number of words (i.e., a sizable vocabulary), it is quite time-consuming. In this work, we used a Python program to generate the pronunciation dictionaries for both the Sphinx and HTK recognizers. In the literature, there are various studies that describe the rules to generate a pronunciation dictionary for MSA. For instance, Ramsay et al. [22] described a knowledge-based approach to generate the phonetic transcription for MSA. Ali et al. in [23] presented a tool for generating phonetic dictionaries for MSA.

Figure 1 shows entries of the dictionaries that are used in this work. The figure shows the phonemes sequence of each word that appears in the training textual data. The pronunciation of each word is used to model the acoustic model during the training phase. For instance, the word "الوَزَارَة" is mapped to the sequence "A L W AU Z AUA R AU P". Hence, wherever this word appears, it is replaced with the phonemes sequence in order to produce the phonetic transcription of the speech file. Each phoneme is then used to train a part of the acoustic signal that corresponds to a phoneme name. This process (i.e., the training stage) is performed using the Baum-Welch algorithm to create a single hidden Markov Model (HMM) for each phoneme in the phoneme list. The figure also shows two sets of phonemes: the phonemes set for Sphinx (on the left of the figure), which is proposed in this work, and the HTK set (on the right of the figure), which is based on the BW transliteration. Creating a single HMM model for each phoneme is called the 'context-independent' phase (CI). After the CI stage, the training stage continues to perform the untied 'context-dependent' phase (CD) for creating triphones; finally, the tied context-dependent phase is used for tying some HMM states.

Arabic-based dictionary (for Sphinx)	Roman character-based dictionary (for HTK)
...	...
الوَزَارَة A L W AU Z AUA R AU P	AlwazaArap A l w a z aA r a p
الوَزِير A L W AU Z AIY R	Alwaziyr A l w a z iy r
الوَزِيرَ A L W AU Z AIY R AU	Alwaziyra A l w a z iy r a
الوَزِيرَة A L W AU Z AIY R AU P	Alwaziyrap A l w a z iy r a p
الوَزِيرُ A L W AU Z AIY R AW	Alwaziyru A l w a z iy r u
الوَزِيرِ A L W AU Z AIY R AI	Alwaziyri A l w a z iy r i
الوَزِيرِة A L W AU Z AIY R AI P	Alwaziyrip A l w a z iy r i p
...	...

FIGURE 1. Entries of the pronunciation dictionaries

In both implemented systems, we consider the pronunciation case of Shadda as follows: for the Shadda case, we first remove the germination marker "◌ّ" and then duplicate the previous consonant. Another case where an unnecessary symbol is deleted is the 'Sukon' "◌ْ". Ramsay et al. in [22] described pre-processing cases that should be considered for phonetic transcription. For illustration, Figure 1 shows some cases where the semi-vowels y, w and A are written with a preceding short vowel, so it is assumed that iy, uw and aA are two-character names for the relevant vowels. The same is found in the Sphinx dictionary (i.e., AUA, AWW, and AIY).

4. **Language Models.** LMs are considered to be an important addition to the enhancement of the performance of ASR and machine translation and NLP systems in general. LMs have been effectively utilized in various linguistic systems including Part-of-Speech (PoS) tagging, parsing, summarization, spell correction, information retrieval, etc. Indeed, LM is considered as a vital component in linguistic systems that produce an output of sequences of words. Recently, researchers have extensively explored the area of advancement of new techniques for the compilation of LMs and addressing their challenges including the issues of missing n-grams. ASR decoders, in speech recognition, use the information provided by an LM to determine the optimal word sequence when testing speech transcription. Generally, it is really vital for language systems to have the capability of predicting the next word based on previous word(s), or a historical sequence.

There are two types of LMs: probabilistic or non-probabilistic. A probabilistic LM is based on statistical information of n-grams, while a non-probabilistic LM is recognized as 'any-word' grammar. Any-word grammar tends not to use probabilities of words. Since it is an unconstrained grammar, this leads to very humble performance in continuous ASR systems because it relies entirely on an acoustic model. In contrast, a statistical language model is based on computing the probabilities of all possible word sequences in a training set. Statistical LMs are generally expressed using ARPA textual format that includes the statistical values of the desired n-grams, classically up to 3-grams are used. Building a good statistical language model needs a huge set of words from diverse textual resources. Furthermore, the data should not be particular domain specific; otherwise, it will not be considered as general for generic use.

In the ASR decoding phase, a recognizer utilizes the language model that uses transcribe speech files of a pronunciation dictionary and an acoustic model. Hence, the most likely proposition for each testing speech file is produced as an output. Using language models strengthens the accuracy if speech recognition. This means that, the more the system can constrain the utterances range, the more the recognizer accuracy will be. The probability of a word sequence is computed based on the information provided in the n-grams, by means of the next formula [25]:

$$P\left(w_1^n\right) = \prod_{k=1}^{n} P\left(w_k | w_1^{k-1}\right) \tag{1}$$

where 'n' is limited to include the history of the word. Chain Rule is used to compute the joint probability of words of a sentence. Markov assumption is utilized in LMs for data estimation simplification. For example, when $n = 2$, we calculate the bigram for a word sequence as:

$$P(w_1 w_2) = P(w_2|w_1)P(w_1) \tag{2}$$

For speech features, Mel Frequency Cepstral Coefficients (MFCC) are the classical front-end analyses used in speech recognition to produce the sequence of real-valued numbers that represent feature vectors based on the input signal. Since 1980, it has dominated the ASR feature extraction method due to its good performance. The success of MFCC makes it the standard choice in the state-of-the-art speech recognizers such as the CMU Sphinx, HTK, and Kaldi speech recognizers. [24] has some details of MFCC. Given the speech feature vectors, it is estimated that the most likely sequence of words is computed as follows [25]:

$$\hat{w} = \underset{W \in L}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)} = \underset{W \in L}{\operatorname{argmax}} P(O|W)P(W) \tag{3}$$

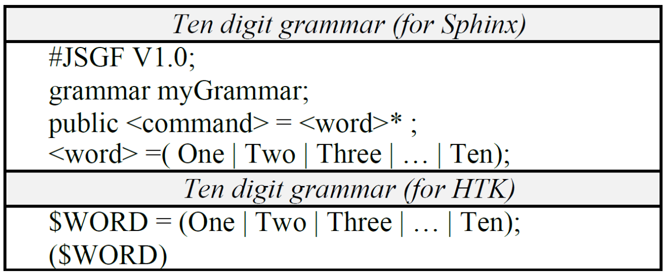
where $\hat{w}$ is the best recognized words, $P(O|W)$ is the feature vectors probability, for a sequence of words in the acoustic model, and $P(W)$ is the words sequence probability that is calculated by the language model. $P(O)$ is the acoustic observation sequence probability that can be safely ignored. Henceforth, we first compute the statistical LM to be able to decode the ASR testing speech files. Tainting the statistical n-grams LM is done by first counting n-grams occurrences in a large transcription corpus and then smoothed and normalized. The n-grams parameter is estimated using following formula [25]:

$$P\left(w_n | w_{n-N+1}^{n-1}\right) = \frac{Count\left(w_{n-N+1}^{n-1} w_n\right)}{Count\left(w_{n-N+1}^{n-1}\right)} \tag{4}$$

The unseen words or n-grams are considered a major problem in LMs that exist in the testing set while not in the vocabulary. Therefore, the items that are not seen in the training data are given a probability of 0.0. That is to say, not all n-grams will be existing in the training data. Assigning non-zero or small probabilities to the unseen n-grams in which all word sequences can occur with some probability is considered a process of smoothing. Therefore, a better way of estimating the probability of zero frequency n-grams, that never occur in order to produce generalized LMs, can be achieved by smoothing. Smoothing is sometimes called discounting. It is more efficient to use log probabilities when creating a language model, rather than actual probabilities to reduce the numerical underflow risk of lengthy strings. It also can be an efficiency issue in ASR decoding algorithms such as its use in Viterbi algorithm.

There are three main steps in creating an n-gram LM entails: first, calculate the word unigram counts; second, transform the word unigram counts to vocabulary list; third, produce bigram and trigram tables based on this vocabulary. It is essential as a pre-processing step, to contain special words such as <s> in order to designate the "start of sentence" and the </s> to specify the "end of the sentence". The CMU Cambridge University toolkits [26] employ the <UNK> token to designate unknown words, whereas HTK toolkits [2] specify !!UNK for the same reason.

Sometimes ASR recognizers employ the grammar of LMs for small applications with limited isolated words. Such grammar is simple and does not have probabilities; they are intended for the information use and its application. Namely, grammar essentially comprises isolated words such as commands, control words, and digits. However, grammar might also allow for sequences of words. Figure 2 shows an example of a grammar consisting of ten digits that are used in continuous speech recognition for selecting one word or more from that list. Using JSGF format, the example grammar utilizes the star (*) to denote for zero or more words. This is usually useful when a grammar is used in continuous speech to be able to recognize a sentence that has many words.



Ten digit grammar (for Sphinx)

```
#JSGF V1.0;
grammar myGrammar;
public <command> = <word>* ;
<word> =( One | Two | Three | … | Ten);
```

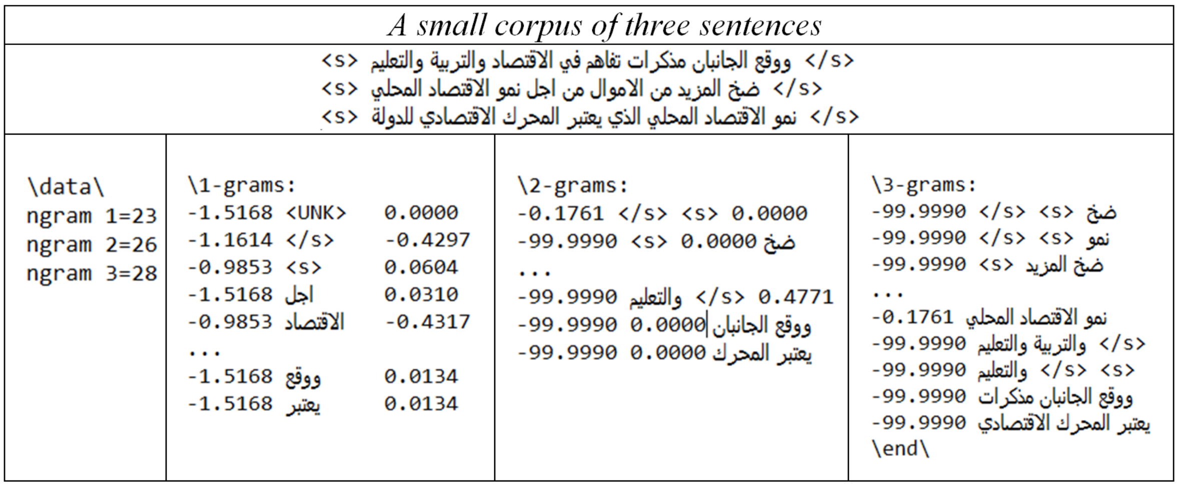
Ten digit grammar (for HTK)

```
$WORD = (One | Two | Three | … | Ten);
($WORD)
```

FIGURE 2. A grammar for any-word of ten digits

Grammar can be either written by hand, or generated using a program. The majority of grammars does not use probabilities; nevertheless, some components can be weighed. Actually, grammar is seldom used in ASR systems because the language probabilistic models are more suitable than the hard models in various languages.

There are a number of toolkits that are used to compile LMs such as the CMU-Cambridge LM toolkit [26], the Cambridge University HTK language modeling tools [2], and the SRI Language Modeling Toolkit (SRILM) [27]. Five commands are used in CMU-Cambridge to produce the LM dump file: text2wfreq, wfreq2vocab, text2idngram, idngram2lm, and lm3g2dmp [28]. On the other hand, two approaches are used in HTK to generate n-grams. The first method utilizes the HLStats function, which is used solely to produce the bigram language model (i.e., 2-grams). The second method uses a series of functions to compute n-grams. In our work, we used the first method (i.e., 2-grams statistical LM). In addition to HLStats, HBuild is used to create the word network that describes the allowable word sequences, of course, with the corresponding probabilities. Figure 3 shows a simple 3-gram LM for three sentences as shown in the figure. It was created using the CMU-Cambridge toolkit.



A small corpus of three sentences

```
<s> ووقع الجانبان مذكرات تفاهم في الاقتصاد والتربية والتعليم </s>
<s> ضخ المزيد من الاموال من اجل نمو الاقتصاد المحلي </s>
<s> نمو الاقتصاد المحلي الذي يعتبر المحرك الاقتصادي للدولة </s>
```

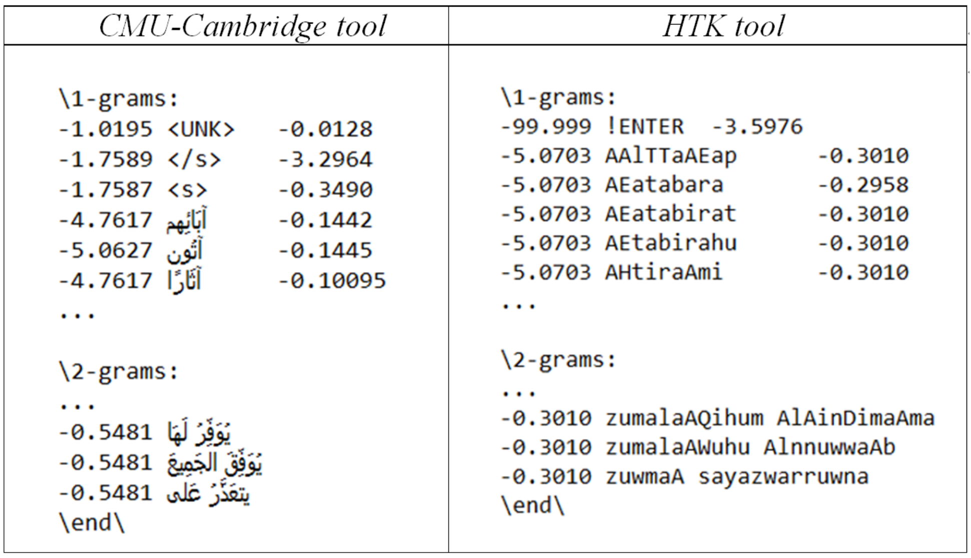
```
\data\
ngram 1=23
ngram 2=26
ngram 3=28
```

```
\1-grams:
-1.5168 <UNK>     0.0000
-1.1614 </s>     -0.4297
-0.9853 <s>       0.0604
-1.5168 اجل       0.0310
-0.9853 الاقتصاد -0.4317
...
-1.5168 ووقع      0.0134
-1.5168 يعتبر     0.0134
```

```
\2-grams:
-0.1761 </s> <s> 0.0000
-99.9990 <s> 0.0000 ضخ
...
-99.9990 والتعليم </s> 0.4771
-99.9990 0.0000 ووقع الجانبان
-99.9990 0.0000 يعتبر المحرك
```

```
\3-grams:
-99.9990 </s> <s> ضخ
-99.9990 </s> <s> نمو
-99.9990 <s> ضخ المزيد
...
-0.1761 نمو الاقتصاد المحلي
-99.9990 والتربية والتعليم </s>
-99.9990 والتعليم </s> <s>
-99.9990 ووقع الجانبان مذكرات
-99.9990 يعتبر المحرك الاقتصادي
\end\
```

Figure 3. Three sentences with the 3-grams LM using the CMU-Cambridge tool



CMU-Cambridge tool

```
\1-grams:
-1.0195 <UNK>   -0.0128
-1.7589 </s>    -3.2964
-1.7587 <s>     -0.3490
-4.7617 آبَائِهِم   -0.1442
-5.0627 آتُون    -0.1445
-4.7617 آثَارًا   -0.10095
...

\2-grams:
...
-0.5481 يُوَفِّرُ لَهَا
-0.5481 يُوَفِّقَ الجَمِيعَ
-0.5481 يتعَذَّرُ عَلى
\end\
```

HTK tool

```
\1-grams:
-99.999 !ENTER  -3.5976
-5.0703 AAlTTaAEap      -0.3010
-5.0703 AEatabara       -0.2958
-5.0703 AEatabirat      -0.3010
-5.0703 AEtabirahu      -0.3010
-5.0703 AHtiraAmi       -0.3010
...

\2-grams:
...
-0.3010 zumalaAQihum AlAinDimaAma
-0.3010 zumalaAWuhu AlnnuwwaAb
-0.3010 zuwmaA sayazwarruwna
\end\
```

Figure 4. Parts of the 2-grams LM that are used in this work

For this work, we used 2-grams statistical language models as shown in Figure 4 for both systems. In fact, the 3-grams language model is more commonly used for ASR tasks; however, we used the 2-grams language model in this work due to the restriction of HVite. HVite uses 2-grams, while HTK HDecode can use 3-grams. Sphinx can use either 2-grams or 3-grams language models.

5. **Implementation of the Sphinx and the HTK Methods.** The Sphinx and HTK methods used Cygwin, which is a Unix-like environment for Windows. However, it is preferred to run the command line in a UNIX-based system rather than Cygwin that is installed for the Windows environment. Implementing Sphinx for Arabic ASRs includes the steps described in [29]. The first step includes creating the directory where the files live. The files include the training and testing speech files, the transcription of the entire speech collection, and other necessary files that are used for training and decoding. In particular, the speech files are stored in the wav directory, while the etc directory has the following files: the pronunciation dictionary, the phonemes file, the list of fillers, the

list of files for training, the transcription for training, the list of files for testing, and the transcription for testing. Of course, the language model also lives in the ***etc*** directory. The following are the main commands (for task1, for instance) that are used in Sphinx:

- $ mkdir task1 (create the main directory)
- $ sphinxtrain -t task1 setup (create the structure of the main directory)
- $ sphinxtrain run (start training, once done, it provides the word error rate (WER))

For HTK implementation, [30] presents comprehensive details for training and decoding as the following steps: Step 1 – the Task Grammar, Step 2 – the Dictionary, Step 3 – Recording the Data, Step 4 – Creating the Transcription Files, Step 5 – Coding the Data, Step 6 – Creating Flat Start Monophones, Step 7 – Fixing the Silence Models, Step 8 – Realigning the Training Data, Step 9 – Making Triphones from Monophones, Step 10 – Making Tied-State Triphones, and finally, Step 11 – Recognizing the Test Data. In the previous sections, we demonstrated some of the steps such as the language model and pronunciation dictionary. To train a model, a further set of files is needed such as the following:

- words.mlf: this is just a rearranged version of the training textual files,
- train.scp: a list of the training speech file names,
- phones0.mlf: the phonetic transcriptions, obtained by substituting the entries in the pronunciation dictionary for the words in the textual transcriptions,
- monophones0: list of the phones that appear in phones0.mlf,
- codetrain.scp: pairs linking .wav files to .mfc files (the MFCC speech features),
- proto.txt: the "flat start" file for the hidden Markov models (HMMs),
- config.txt that contains some parameters related to the speech features.

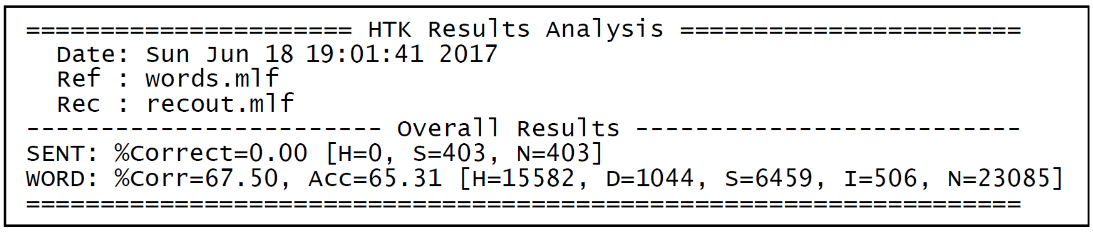
After creating the necessary files in the same directory where the speech files and the speech features (.mfc) reside, it is possible to start training. This produces trained models that will be used for decoding. During training, many functions are executed, such as the HLEd and HERest functions. For decoding, HVite is used.

To investigate the performance, we split the speech corpus (15.93 hours) into two parts including the training set that contains 12.74 hours (1,611 speech files) and the testing set that contains 3.19 hours (403 speech files). That is, the testing set is 20% of the overall speech corpus. The speech files were prepared to have a fixed length between 10-40 seconds, mono, and sampled at 16 kHz. The average length of the textual files is 55 words. The total number of speakers in the corpus is 29 native Arabic speakers (19 males and 10 females). In this work, we used three emitting states of HMMs that corresponded to the subphones at the beginning, middle, and end of the phones. The acoustic models were calculated using context-dependent HMM triphones. Regarding Sphinx recognizers, the acoustic models are trained using the SphinxTrain for the phonetic tied-mixture (PTM). However, other acoustic model types can be used such as semi-continuous and fully continuous models. The performance is measured based on different parameters such as the number of Senones and the number of Gaussian densities. Table 3 shows the WER for different cases.

Table 3 shows that the best (lowest) WER was found to be 20.7% using 16 Gaussian densities and 2000 Senones. We emphasize that these results are based on the language model that contains both the training and testing transcriptions. However, if the language model contains only the training transcription, the results will be less than what we scored in Table 3. The reason for using the training and the testing sets to create the language model is that the language model requires a large amount of data that is not available in our work. During experiments, we considered speeding up the execution time using the Sphinx configuration (i.e., number of parts to run Forward-Backward estimation ➔

Table 3. The performance of the Sphinx recognizer

Experiment	Densities	Senones	WER (%)	Accuracy (%)
1	8	500	22.6	77.4
2	8	1000	22.2	77.8
3	8	2000	21.5	75.5
4	16	500	21.8	78.2
5	16	1000	21.1	78.9
6	16	2000	20.7	79.3
7	32	500	21.8	78.2
8	32	1000	21.3	78.7
9	**32**	**2000**	**21.3**	**78.7**
10	64	500	21.9	78.1
11	64	1000	21.9	78.1
12	64	2000	21.9	78.1
13	128	500	21.7	78.3
14	128	1000	22.6	77.4
15	128	2000	22.1	77.9
16	256	500	21.8	78.2
17	256	1000	22.2	77.8



```
===================== HTK Results Analysis =======================
  Date: Sun Jun 18 19:01:41 2017
  Ref : words.mlf
  Rec : recout.mlf
------------------------ Overall Results --------------------------
SENT: %Correct=0.00 [H=0, S=403, N=403]
WORD: %Corr=67.50, Acc=65.31 [H=15582, D=1044, S=6459, I=506, N=23085]
===================================================================
```

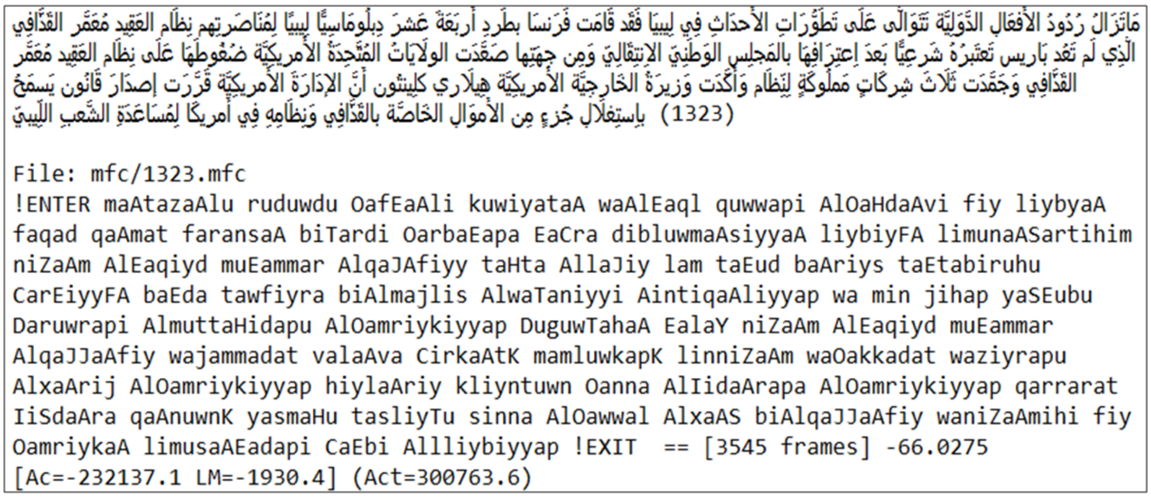
Figure 5. Performance of the HTK recognizer

$CFG_NPART = 10; and how many pieces to split decode in ➔ $DEC_CFG_NPART = 10). The number 10 is just an optional number that can be fixed based on the user preferences. This option is helpful since it clearly reduces the execution time by utilizing a number of processors in multicore machines. For the HTK recognizer, we found the performance to be less than what we achieved using the Sphinx recognizer. The WER is (100% − 65.31% = 34.69%). However, the lowest WER in the Sphinx is 20.7%. Figure 5 shows the HTK overall results.

Regarding the output, Sphinx gives the results using Arabic characters. However, the HTK gives Romanized characters. Figure 6 shows an example of the output of the HTK recognizer. In the figure, Arabic characters (the upper part in Figure 6) represent the transcription of a speech file after recognition, and the lower English characters are the Romanized text as recognized by the HTK HVite decoder.

In this work, we used the default settings of HTK. The default settings include the thresholds for outlier removal (RO = 100) and the tree branch threshold (TB = 350). TB is used for the decision of tree clustering of states. Both RO and TB affect the degree of tying and therefore the number of states output in the clustered system [30].

As a comparison, Sphinx is better in some issues such as handling long speech files, since some of the long speech files were discarded due to failure execution using the HTK (i.e., the training fails using long speech files). Sphinx is also better in terms of execution time as it takes less training and decoding time compared to the HTK. In addition, we have found that it is easier to perform an ASR task with Sphinx than HTK. The only issue with Sphinx is that it fails when the phonemes set has capital and small letters. For instance, if we use the character <B> to indicate a specific phoneme and, at the same time, use the character <b> to indicate another phoneme, then we get an error during



```
مَاتَزَالُ رُدُودُ الأَفعَالِ الدَّوَلِيَّة تَتَوَالَى عَلَى تَطَوُّرَاتِ الأَحدَاثِ فِي لِيبِيَا فَقَد قَامَت فَرَنسَا بطَردِ أَربَعَةَ عَشرَ دِبلُومَاسِيًّا لِيبِيًّا لِمُنَاصَرتِهِم نِظَامِ العَقِيد مُعَمَّر القَذَّافِي
الَّذِي لَم تَعُد بَارِيس تَعتَبِرُهُ شَرعِيًّا بَعدَ إِعتِرَافِهَا بالمَجلِسِ الوَطَنِيِّ الإِنتِقَالِيِّ وَمِن جِهَتِهَا صَعَّدَت الوِلَايَاتُ المُتَّحِدَةُ الأَمرِيكِيَّة ضُغُوطَهَا عَلَى نِظَامِ العَقِيد مُعَمَّر
القَذَّافِي وَجَمَّدَت ثَلَاثَ شِركَاتٍ مَملُوكَةٍ لِنِظَامِ وَأَكَّدَت وَزِيرَةُ الخَارِجِيَّة الأَمرِيكِيَّة هِيلَارِي كلِينتُون أَنَّ الإِدَارَةَ الأَمرِيكِيَّة قَرَّرَت إِصدَارَ قَانُون يَسمَحُ
بِاستِغلَالِ جُزءٍ مِن الأَموَالِ الخَاصَّة بالقَذَّافِي وَنِظَامِهِ فِي أَمرِيكَا لِمُسَاعَدَةِ الشَّعبِ اللِّيبِيِّ (1323)

File: mfc/1323.mfc
!ENTER maAtazaAlu ruduwdu OafEaAli kuwiyataA waAlEaql quwwapi AlOaHdaAvi fiy liybyaA
faqad qaAmat faransaA biTardi OarbaEapa EaCra dibluwmaAsiyyaA liybiyFA limunaASartihim
niZaAm AlEaqiyd muEammar AlqaJAfiyy taHta AllaJiy lam taEud baAriys taEtabiruhu
CarEiyyFA baEda tawfiyra biAlmajlis AlwaTaniyyi AintiqaAliyyap wa min jihap yaSEubu
Daruwrapi AlmuttaHidapu AlOamriykiyyap DuguwTahaA EalaY niZaAm AlEaqiyd muEammar
AlqaJJaAfiy wajammadat valaAva CirkaAtK mamluwkapK linniZaAm waOakkadat waziyrapu
AlxaArij AlOamriykiyyap hiylaAriy kliyntuwn Oanna AlIidaArapa AlOamriykiyyap qarrarat
IiSdaAra qaAnuwnK yasmaHu tasliyTu sinna AlOawwal AlxaAS biAlqaJJaAfiy waniZaAmihi fiy
OamriykaA limusaAEadapi CaEbi Allliybiyyap !EXIT  == [3545 frames] -66.0275
[Ac=-232137.1 LM=-1930.4] (Act=300763.6)
```

FIGURE 6. The recognition output of a speech file using the HTK

training. On the other hand, this error did not appear in the HTK system. We also have found that HTK is better documented than Sphinx. In conclusion, more research is required to understand the reasons for the performance difference between both systems.

6. Conclusion.

This work discusses the implementation of two well-known speech recognizers: the CMU Sphinx, and the HTK. It includes a comparative study of both recognizers using a continuous speech corpus of MSA. The results show that Sphinx outperforms the HTK recognizer. In future work, it is worth evaluating the performance using different values of RO and TB. It is also worth employing the HDecode, which is an HTK extension decoder released on a restricted basis. Finally, implementing and comparing Kaldi ASR with the Sphinx and the HTK systems is a valuable research for Arabic speech recognition.

Acknowledgement. This research was supported by the Research Administration Research at Kuwait University, Project Number EO06/12.

REFERENCES

[1] *https://cmusphinx.github.io/*, retrieved December 2, 2018.

[2] *http://htk.eng.cam.ac.uk/*, retrieved December 2, 2018.

[3] C. Gaida et al., *Comparing Open-Source Speech Recognition Toolkits*, Tech. Rep., DHBW Stuttgart, 2014.

[4] *http://kaldi-asr.org/doc/index.html*, retrieved December 2, 2018.

[5] *http://www.qamus.org/transliteration.htm*, retrieved December 2, 2018.

[6] H. Hyassat and R. A. Zitar, Arabic speech recognition using SPHINX engine, *International Journal of Speech Technology*, vol.9, no.3, pp.133-150, 2006.

[7] M. Alghamdi, M. Elshafei and H. Al-Muhtaseb, Arabic broadcast news transcription system, *International Journal of Speech Technology*, vol.10, no.4, pp.183-195, 2007.

[8] D. AbuZeina et al., Cross-word Arabic pronunciation variation modeling for speech recognition, *International Journal of Speech Technology*, vol.14, no.3, pp.227-236, 2011.

[9] D. AbuZeina et al., Within-word pronunciation variation modeling for Arabic ASRs: A direct data-driven approach, *International Journal of Speech Technology*, vol.15, no.2, pp.65-75, 2012.

[10] M. A. M. Abushariah, R. N. Ainon et al., Arabic speaker-independent continuous automatic speech recognition based on a phonetically rich and balanced speech corpus, *International Arab Journal of Information Technology (IAJIT)*, vol.9, no.1, pp.84-93, 2012.

[11] F. S. Al-Anzi and D. AbuZeina, The impact of phonological rules on Arabic speech recognition, *International Journal of Speech Technology*, pp.1-9, 2017.

[12] K. Kirchhoff et al., Novel approaches to Arabic speech recognition: Report from the 2002 Johns-Hopkins summer workshop, *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol.1, 2003.

[13] D. Vergyri et al., Morphology-based language modeling for Arabic speech recognition, *Interspeech*, vol.4, 2004.

[14] K. Kirchhoff et al., Morphology-based language modeling for conversational Arabic speech recognition, *Computer Speech & Language*, vol.20, no.4, pp.589-608, 2006.

[15] H. Satori, M. Harti and N. Chenfour, Introduction to Arabic speech recognition using CMUSphinx system, *arXiv preprint arXiv:0704.2083*, 2007.

[16] M. Elmahdy et al., Modern standard Arabic based multilingual approach for dialectal Arabic speech recognition, *The 8th International Symposium on Natural Language Processing*, 2009.

[17] B. A. Q. Al-Qatab and R. N. Ainon, Arabic speech recognition using hidden Markov model toolkit (HTK), *International Symposium on Information Technology (ITSim)*, vol.2, 2010.

[18] M. Graja, M. Jaoua and L. HadrichBelguith, Lexical study of a spoken dialogue corpus in Tunisian dialect, *The International Arab Conference on Information Technology (ACIT)*, Benghazi, Libya, 2010.

[19] N. Merad-Boudia, A. Benyettou and A. J. Rubio, Arabic speech recognition for connected words using HTK: Triphones expanded to Gmm based Quran recognition, *International Review on Computers and Software (IRECOS)*, vol.11, no.12, pp.1209-1216, 2016.

[20] R. A. Haraty and O. E. Ariss, CASRA+: A colloquial Arabic speech recognition application, *American Journal of Applied Sciences*, vol.4, no.1, pp.23-32, 2007.

[21] Y. A. Alotaibi, Comparative study of ANN and HMM to Arabic digits recognition systems, *Journal of King Abdulaziz University: Engineering Sciences*, vol.19, no.1, pp.43-59, 2008.

[22] A. Ramsay, I. Alsharhan and H. Ahmed, Generation of a phonetic transcription for modern standard Arabic: A knowledge-based model, *Computer Speech & Language*, vol.28, no.4, pp.959-978, 2014.

[23] M. Ali et al., Arabic phonetic dictionaries for speech recognition, *Journal of Information Technology Research (JITR)*, vol.2, no.4, pp.67-80, 2009.

[24] F. S. Al-Anzi and D. AbuZeina, The capacity of Mel Frequency Cepstral Coefficients for speech recognition, *International Journal of Computer and Information Engineering*, vol.11, no.10, 2017.

[25] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, Pearson, 2014.

[26] *http://www.speech.cs.cmu.edu/SLM/toolkit.html*, retrieved December 2, 2018.

[27] *http://www.speech.sri.com/projects/srilm/*, retrieved December 2, 2018.

[28] *http://www.speech.cs.cmu.edu/SLM/toolkit_documentation.html*, retrieved December 2, 2018.

[29] *https://cmusphinx.github.io/wiki/tutorialam/*, retrieved December 2, 2018.

[30] S. Young et al., *The HTK Book (for HTK Version 3.4)*, Cambridge University Engineering Department, 2006.