The Islamic University-Gaza
Deanery of Higher Studies
Faculty of Engineering
Computer Engineering Department

Arabic Continuous Speech Recognition System using Sphinx-4

Submitted By:

Eman Ziad Elzaza

Supervisor:

Dr. Wesam Ashour

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

(JULY, 2012) 1433 H

ACKNOWLEDGEMENTS

First of all, I would like to thank ALLAH for providing me the power and the will to complete this thesis.

Then , I would like to thank Dr. Wesam Ashour, for his indispensable guidance during the execution of the Master thesis and putting up with my many questions and for helping me with all the technical aspects of this thesis, and for that I will always be grateful.

I would also like to thank all my instructors in the computer engineering department, particularly Prof. Ibrahim Abuhaiba, for everything he has done for us.

I would like to include a personal thanks to my family and my parents, for all the sacrifices they made for me, for the love they showed to me, and for providing me with the inspiration to always do my best.

Most importantly, I would like to thank my friend Rwand for the supporting she has shown to me.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

A/D	Analog to Digital
ASR	Automatic Speech Recognition
BNF	Backus Normal Form
CFG	Context-Free Grammars
CHMM	Continuous Hidden Markov Model
CMN	Cepstral Mean Normalization
CMU	Carnegie Mellon University
CPU	Central Processing Unit
CSR	Command Success Rate
DARPA	Defense Advanced Research Projects Agency
DCT	Discrete Cosine Transform
ELRA	European Language Resource Association
FSG	Finite-State Grammars
FST	Finite-State Transducer
HMMs	Hidden Markov Models
JSGF	JavaTM Speech API Grammar Format
KACST	King Abdulaziz City of Science and Technology
ksps	Kilo [1000] Sample Per Second
LDC	Linguistic Data Consortium
LMGrammar	Language Model Grammar
LPC	Linear Predictive Encoding
MFCC	Mel Frequency Cepstral Coefficients
MSA	Modern Standard Arabic
PCI	Peripheral Component Interconnect
PLP	Prediction Coefficient Extraction
RAM	Random Access Memory
SAPI	Speech API

SCHMM	Semi-Continuous Hidden Markov Model
SWER	Single Word Error Rate
UN	United Nations
WER	Word Error Rate
WRR	Word Recognition Rate

نظام التعرف على الكلام العربي المستمر باستخدام سفنكس-٤

الملخص

الكلام هو النمط الأكثر استخداما في التواصل الإنساني ومعالجة الكلام كان واحدا من أكثر المواضيع المهمة في مجال معالجة الإشارات. ولقد حققت تكنولوجيا التعرف على الكلام إمكانية متابعة الأوامر الصوتية للإنسان وفهم لغات البشر باستخدام جهاز الكمبيوتر، فكان الهدف الرئيسي لمجال التعرف على الكلام تطوير التقنيات والنظم لإدخال الكلام إلى الآلة ومنها جهاز الكمبيوتر ومن ثم معالجته لاستخدامه في مجالات متعددة. وبما أن اللغة العربية هي واحدة من أكثر اللغات انتشارا في العالم، وكما تظهر الإحصائيات أنها اللغة الأولى -اللغة الأم- لـ 206 مليون ناطق أصلي باللغة العربية كما أنها تحتل المرتبة الرابعة بعد الماندرين والإسبانية والإنكليزية . على الرغم من أهميتها إلا أن الجهود التي تبذل في أبحاث التعرف على الكلام في اللغة العربية للأسف ما زال غير كافي.

هذه الأطروحة تقترح وتصف كيفية تصميم وتطوير نظام التعرف على الكلام العربي التلقائي، المعتمد على المتحدث المستقل والخطاب المستمر، وذلك باستخدام قاعدة كلام غنية صوتيا ومتوازنة. ويتم الاستناد في تطوير النظام على أدوات Sphinx من جامعة كارنيجي ميلون.

ولبناء هذا النظام تم إنشاء ثلاث وحدات أساسية وهي القاموس اللفظي وهو يحتوي على النطق الصوتي للكلمات، ووحدة نموذج اللغة وهو يحوي احتمالات ورود الكلمات معا، وأخيرا وحدة المحرك وهو يعمل على مقارنة الصوت المدخل مع المعلومات الصوتية المخزنة للكلام في القاموس اللفظي.

النظام يستخدم قاعدة بيانات غنية ومتوازنة تحتوي على 367 جملة تتضمن ما مجموع 14232 كلمة. أما القاموس اللفظي فيحتوي على حوالي 23841 تعريف يقابل كل منها كلمة في قاعدة البيانات، .بينما نموذج اللغة المستخدم يحتوي على 14233 أحادية المقطع و 32813 ثنائي المقطع و 37771 متعدد. وتم استخدام محرك مبني على نماذج ماركوف المخفية مستندا إلى النماذج الصوتية الثلاثية المقاطع، و تم تدريب النظام على 5 ساعات من خطاب البيانات باللغة العربية.

CONTINUOUS ARABIC SPEECH RECOGNITION SYSTEM USING SPHINX-4

Abstract

Speech is the most natural form of human communication and speech processing has been one of the most exciting areas of the signal processing. Speech recognition technology has made it possible for computer to follow human voice commands and understand human languages. The main goal of speech recognition area is to develop techniques and systems for speech input to machine and treat this speech to be used in many applications. As Arabic is one of the most widely spoken languages in the world. Statistics show that it is the first language (mother-tongue) of 206 million native speakers ranked as fourth after Mandarin, Spanish and English. In spite of its importance, research effort on Arabic Automatic Speech Recognition (ASR) is unfortunately still inadequate[7].

This thesis proposes and describes an efficient and effective framework for designing and developing a speaker-independent continuous automatic Arabic speech recognition system based on a phonetically rich and balanced speech corpus. The developing Arabic speech recognition system is based on the Carnegie Mellon university Sphinx tools.

To build the system, we develop three basic components. The dictionary which contains all possible phonetic pronunciations of any word in the domain vocabulary. The second one is the language model such a model tries to capture the properties of a sequence of words by means of a probability distribution, and to predict the next word in a speech sequence. The last one is the acoustic model which will be created by taking audio recordings of speech, and their text transcriptions, and using software to create statistical representations of the sounds that make up each word. The system use the rich and balanced database that contains 367 sentences, a total of 14232 words. The phonetic dictionary contains about 23,841 definitions corresponding to the database words. And the language model contains14233 mono-gram and 32813 bi-grams and 37771 tri-grams. The engine uses 3-emmiting states Hidden Markov Models (HMMs) for tri-phone-based acoustic models..

Keywords: *Arabic automatic speech recognition, acoustic model, and language Model*

Chapter 1 Introduction

1.1 Automatic Speech Recognition

Automatic Speech Recognition (ASR) is a technology that allows a computer to identify the words that a person speaks into a microphone or telephone. It has a wide area of applications: command recognition (voice user interface with the computer), dictation, interactive voice response, and it can be used to learn a foreign language. ASR can help also, handicapped people to interact with society. It is a technology which makes life easier and very promising [1].

View the importance of ASR too many systems are developed, the most popular are: Dragon Naturally Speaking, IBM via voice, Microsoft SAPI. Many open source speech recognition systems are available too, such as [2 , 3 , 4 , 5 , 6], which is based on Hidden Markov Models (HMMs) [1].

Arabic is one of the six official languages of the United Nations (UN), and is one of the most widely spoken languages in the world. Statistics show that it is the first language (mother-tongue) of 206 million native speakers, and ranked as the fourth after Mandarin, Spanish and English [7]. In spite of its importance, research effort on Arabic Automatic Speech Recognition is unfortunately still inadequate.

Modern Standard Arabic (MSA) is the formal linguistic standard of Arabic language, which is widely taught in schools and universities, and used in the office and the media. It has been the focus and the core interest of many previous and recent researches compared to dialectal Arabic [8, 9, 10, 11]. Lack of spoken and written training data is one of the main issues encountered by Arabic ASR researchers. A list of most popular (from 1986 through 2005) corpora is provided [12] showing only 19 corpora (14 written, 2 spoken, 1 written and spoken, and 2 conversational). These corpora are not readily available to the public and many of them can only be purchased from the Linguistic Data Consortium (LDC) or the European Language Resource Association (ELRA). It is clear that there is a shortage of spoken data as compared to written data resulting in a great need for more speech corpora in order to serve different domains of Arabic ASR. The available spoken corpora were mainly collected from broadcast news (radios and televisions), and telephone conversations. This kind of

spoken data may not necessarily serve quality Arabic ASR research, because of the quality of the spoken data itself in terms of recording attributes and parameters used (e.g., sampling rate). They are also limited to certain applications and domains. The coverage of any corpora cannot contain complete information about all aspects of language lexicon and grammar [13], due to the limited written training data and therefore inadequate spoken training data. In addition, a clear strong relationship between written and spoken forms needs to be clarified.

Writing is claimed to be more structurally complex, elaborate, more explicit, more organized and planned than speech [14]. These differences generally lead to the approach that the written form of the corpora needs to be created carefully before producing and recording the spoken form. Therefore, linguists and phoneticians carefully produce written corpora before handling them to speech recording specialists.

This can also be seen throughout the past few years, where a number of phonetically rich and/or balanced corpora for many languages have been produced. Many ASR researches are now based on phonetically rich and/or balanced corpora, e.g., English [15 - 17], Mandarin [18], Japanese [19], Indonesian, Korean, Cantonese Hindi, Turkish and many others obtaining comparatively competitive results. As far as Arabic language is concerned, automatic speech recognition tasks mainly addressed for Arabic digits, broadcast news, command and control, the Holy Qur'an, and Arabic proverbs researches. They explored various state-of-the-art techniques and tools for Arabic speech recognition[20, 21, 9, 11].

The development of accurate Automatic Speech Recognition (ASR) systems is faced with two major issues. The first problem is related to dicritization where diacritic symbols refer to vowel phonemes in the designated words. Arabic texts are almost never fully diacritized: implying that the short strokes placed above or below the consonant, indicating the vowel following this consonant, are usually absent. This limits the availability of Arabic ASR training material. The lack of this information leads to many similar word forms, and consequently, decreases predictability in the language model. The second problem is related to the morphological complexity since Arabic has a rich potential of word forms which increases the out-vocabulary rate [22, 23].

1.2 ASR Techniques

The HMM-based ASR technique has led to numerous applications requiring large vocabulary, speaker independent and continuous speech recognition.

HMM is a statistical model where the system being modeled with unknown parameters, and the challenge is to determine the hidden parameters, from the observable parameters. The extracted model parameters can then be used to perform further analysis, for example the pattern recognition applications. Its extension into foreign languages (English is the standard) represent a real research challenge area[24].

The HMM-based system essentially consists of recognizing speech by estimating the likelihood of each phoneme at contiguous, small frames of the speech signal [25, 26]. Words in the target vocabulary are modeled into a sequence of phonemes, and then a search procedure is used to find, amongst the words in the vocabulary list, the phoneme sequence that best matches the sequence of phonemes of the spoken words.

Each phoneme is modeled as a sequence of HMM states. In standard HMM-based systems, the likelihoods (also known as the emission probabilities) of a certain frame observation being produced by a state is estimated using traditional Gaussian mixture models. The use of HMM with Gaussian mixtures has several notable advantages such as a rich mathematical framework, efficient learning and decoding algorithms, and an easy integration of multiple knowledge sources. Two notable successes in the academic community in developing high performance large vocabulary, speaker independent, speech recognition systems are the HMM tools, known as the Hidden Markov Model Toolkit (HTK), developed at Cambridge University [27]; and the Sphinx system developed at Carnegie Mellon University [28], over the last two decades. The Sphinx tools can be used for developing wide spectrum of speech recognition tasks.

For example, the Sphinx-II [29] uses the Semi-Continuous Hidden Markov Model (SCHMM) models to reduce the number of parameters and the computer resources required for decoding, but has limited accuracy and complicated training procedure. On the other hand Sphinx-III uses the Continuous Hidden Markov Model (CHMM) with higher performance, but requires substantial computer resources. Sphinx-4, which was developed in Java, can be used for building platform independent speech recognition applications [30, 28].

1.3 Arabic Speech Recognition

Development of an Arabic speech recognition system is a multi-discipline effort, which requires integration of Arabic phonetic [31, 32], Arabic speech processing techniques [33], and Natural language [34, 35]. Development of an Arabic speech recognition system has recently been addressed by a number of researchers. Recognition of Arabic continuous speech was addressed by Al_Otaibi, [36]. He provided a speech dataset for Modern Standard Arabic (MSA). He studied different approaches for building the Arabic speech corpus, and proposed a new technique for labeling Arabic speech. He reported a recognition rate for speaker dependent ASR of 93.78% using his technique.

The ASR was built using the HTK tool kit. Bila et al. [37] addressed the problems of indexing of Arabic news broadcast, and discussed a number of research issues for Arabic speech recognition. There are a number of other attempts to build Arabic ASR (AASR), but they considered either limited vocabulary, or speaker dependant system [4, 8, 9, 23, 26].

The most difficult problems in developing highly accurate ASRs for Arabic are the predominance of non diacritized text material, the enormous dialectal variety, and the morphological complexity[24]. Kirchhoff et al.[10] investigated the recognition of dialectal Arabic and study the discrepancies between dialectal and formal Arabic in the speech recognition point of view. Vergyri et al. [38] investigate the use of morphology-based language model at different stages in a speech recognition system for conversational Arabic; he studied also the automatic diacritizing Arabic text for use in acoustic model training for ASR. Satori et al.[11] introduced an Arabic voice recognition system where both training and recognizing process used romanized characters. Most of previous works on Arabic ASR had been concentrated on developing recognizers using romanized characters. The system in [9] used Carnegie Mellon University's (CMU) Sphinx-IV engine was based on Hidden Markov Models (HMM), which obtained a word recognition rate of 99.21% for about 35 minutes of training speech data and 7 minutes of testing speech data.

The system in [11] was also using CMU Sphinx- IV engine based on HMM for the same task and obtained a word recognition rate of 85.56% for male speakers and 83.34% for female speakers. In [21], a different kind of speech data was presented for

Arabic digits recognition system using telephony Saudi accented Arabic corpus. The system used Cambridge HTK tools based on HMMs and reported correct digit recognition rate of 93.67%. In addition, The Holy Qur'an was also considered for Arabic speech recognition in [9], which used Sphinx-IV engine based on HMMs and obtained a word recognition rate of 70.81% and a word error rate]of 40.18% for corpus of 18.35 hours.

On the other hand, Arabic speech recognition system using broadcast news corpus was developed in [8]. The system was trained using about 7 hours of speech using Sphinx 3 tools based on HMMs and tested using 400 utterances adding to about half an hour of speech. The system obtained a correct word recognition rate of 90.78% and a WER of 10.87% with full diacritical marks, whereas it obtained a correct word recognition rate of 93.04% and a WER of 8.61% without diacritical marks. Other Arabic automatic speech recognition systems were developed for different tasks such as in [39, 9, 40]. A command and control system covering approximately 30 words was developed in [9] using Sphinx-IV engine based on HMMs and obtained a word recognition rate of 98.18%, whereas an Arabic ASR system to recognize 16 sentences of Egyptian proverbs was developed [39] based on HMMs and obtained a word recognition rates of 56.8%, 66.65%, and 81.79% for Mono-phone, Tri-phone, and Syllable based recognition respectively.

The technical report in [41] is one of the earliest works on producing written Arabic training data based on phonetically rich and balanced sentences. This technical report was submitted to King Abdulaziz City of Science and Technology (KACST) in Saudi Arabia as the final deliverable of the project "Database of Arabic Sounds: Sentences". This written training data was created by experts from KACST and consists of 367 sentences written using 663 phonetically rich words. KACST written (text) training data was used as the baseline for creating our phonetically rich and balanced speech corpus. Another 10 written sentences were created for testing purposes, which taken from Arabic news.

1.4 The Contribution

In this thesis, we will construct an Arabic speech recognition system depending on a rich and balance Arabic speech data set, the used data set coverage all Arabic phoneme clustering with minimum words repletion and simple sentences structure. The

first thing to do is to get the phonetic pronunciations of all words in the domain vocabulary, and also the variant pronunciations for the words if their, these all possible phonetic pronunciations will considered as dictionary.

The second step is to capture the properties of a sequence of words by means of a probability distribution, and to predict the next word in a speech sequence through generate the uni-gram , bi-gram and the tri-gram of the words, keeping them in the language model. Then we will generate the corresponding HMM units of the used words, by create statistical representations of the sounds that make up each. Using the three modules together we can construct the Arabic speech recognition system for large vocabulary and recognize continuous Arabic speech. The next phase is to test the system . the testing include recording a speech audio from news channels and recognize it through the system, and using the microphone to directly enter the sound to system to be recognized.

1.5 Thesis Structure

In the next chapter we present a brief description of the speech recognition, it contains the topics: speech technology, basic concepts, types of speech, speech uses and applications, hardware needed to manipulate with speech, how the recognizer work, illustrate the Arabic language, and a deep view of the feature extraction technique. Chapter 3 gives a detailed view of the Sphinx_4 (open source framework), the view covers its components, the way it used to construct the recognizer and the technique used to recognize the speech.

In Chapter 4, we offer our proposed system, the chapter contains two main parts: how to train the system and how to test the system. In Chapter 5, we present experimental results in details. Finally, in Chapter 6, we provide our conclusion and future works.

Chapter 2 Speech Recognition

2.1 Speech Technology

Not all developers are familiar with speech technology, as an emerging technology. There are subtle and powerful capabilities -with the basic function of both speech synthesis and speech recognition- which are provided by computerized speech, that developers will want to understand and utilize. Speech synthesis and speech recognition technologies still have significant limitations despite very substantial investment in speech technology research over the last 40 years. Most importantly, speech technology does not always meet the high expectations of users familiar with natural human-to-human speech communication. It s important for effective use of speech input and output in a user interface to understand the limitations — as well as the strengths. An understanding of the capabilities, strengths and limitations of speech technology is also important for developers in making decisions about whether a particular application will benefit from the use of speech input and output or not.

Speech Synthesis (TTS)

A speech synthesizer's task is converting a given written text into some spoken language. Speech synthesis is also known as text-to-speech (TTS) conversion. Producing speech from text has many steps, the major steps in producing speech from text are:

Structure analysis and,

Text pre-processing.

We will illustrate the algorithm implementation steps in more details:

- *Structure analysis*: this step is for determining the paragraphs, sentences and other structures start and end after processing the input text. Punctuation and formatting data are used in this stage for most languages.
- *Text pre-processing*: in this step the speech synthesizer analyze the given input text for special constructs of the language.

Speech Recognition

The speech recognition can be defined as the process of converting spoken language to written text or some similar form[44].

2.2. Speech Recognition Basics

Here in this section we will define some important terms which will be used a lot in the coming sections and these are the basics needed for understanding speech recognition technology.

Utterance

Utterances can be a single word, a few words, a sentence, or even multiple sentences, it is the vocalization (speaking) of a word (words) or sentences that represent a single meaning to the computer..

Speaker Dependence

There are two types of Speaker Dependence. The first type are the Speaker independent systems are designed for a variety of speakers. Adaptive systems usually start as speaker independent systems and utilize training techniques to adapt to the speaker to increase their recognition accuracy. The second type are the speaker dependent systems which designed around a specific speaker. They assume the speaker will speak in a consistent voice and tempo. Thus**,** they generally are more accurate for the correct speaker, but much less accurate for other speakers.

Vocabularies

Vocabularies (or dictionaries) can be defines as a lists of words or utterances that can be recognized by the SR system. There is a difference between the words large and small in computer recognition. For example, to explain more, smaller vocabularies are easier for a computer to recognize, while larger vocabularies are more difficult. Unlike normal dictionaries, each entry doesn't have to be a single word. They can be as long as a sentence or two. Smaller vocabularies can have as few as 1 or 2 recognized utterances (e.g. "Wake Up"), while very large vocabularies can have a hundred thousand or more!

Accurate

We can examine the ability of a recognizer by measuring its accuracy – or how well that recognizer recognizes utterances. This includes not only correctly identifying an utterance but also identifying if the spoken utterance is not in its vocabulary. The acceptable accuracy of a system really depends on the application. Good ASR systems have an accuracy of 98% or more!

Training

Sometimes the system has a recognizers that have the ability to adapt to a speaker. And when the system has this ability, it may allow training to take place. Training a recognizer usually improves its accuracy. Training can be used by speakers that have difficulty speaking, or pronouncing certain words. As long as the speaker can consistently repeat an utterance, ASR systems with training should be able to adapt. An ASR system is trained by having the speaker repeat standard or common phrases and adjusting its comparison algorithms to match that particular speaker.

2.3. Types of Speech Recognition

Speech recognition systems can be separated in several different classes by describing what types of utterances they have the ability to recognize. These classes are based on the fact that one of the difficulties of ASR is the ability to determine when a speaker starts and finishes an utterance. Most packages can fit into more than one class, depending on which mode they're using.

Isolated Words

Isolated word recognizers usually require each utterance to have quiet (lack of an audio signal) on both sides of the sample window. It doesn't mean that it accepts single words, but does require a single utterance at a time. Often, these systems have "Listen/Not-Listen" states, where they require the speaker to wait between utterances (usually doing processing during the pauses). Isolated utterance might be a better name for this class.

Connected Words

Connected word systems (or more correctly 'connected utterances') are similar to Isolated words, but allow separate utterances to be 'run-together' with a minimal pause between them.

Continuous Speech

The next step is the Continuous recognition. It is the most difficult to create the recognizers with continuous speech capabilities, this is because they must utilize special methods to determine utterance boundaries. Continuous speech recognizers allow users to speak almost naturally, while the computer determines the content. Basically, it is computer dictation.

Spontaneous Speech

Spontaneous speech has a large number of definitions. At a basic level, it can be thought of as speech that is natural sounding and not rehearsed. The speaker starts an utterance, and reaches a point where he can not find the right word or thinks better of a word, and needs time to find a suitable alternative, they repeat a word as a sort of 'run up' to the second attempt. An ASR system with spontaneous speech ability should be able to handle a variety of non-standard grammar speech features such as words being run together, "اممم" and "ايييه", and even slight stutters.

Voice Verification/Identification

Some ASR systems have the ability to identify specific users.

Automatic Speech Recognition system classification:

The following tree structure emphasizes the speech processing applications. Depending on the chosen criterion, Automatic Speech Recognition systems can be classified as shown in Figure 2.1[45] for many classes, the speech mode that defined the type of the utterance that will be used as isolated or continuous utterance.

The speaking mode comprises three kinds of speakers mode used in systems**,** the speaker dependent, independent and the adaptive speakers. Every speech recognition system must deal with input corpus, the size of the corpus can be small or medium or large. The last classification is the speaking style that determine if the

system will considered as dictation system or spontaneous one. In our system we cover the continuous speech and speaker independent using a large size vocabulary, the speaking style is dictation.

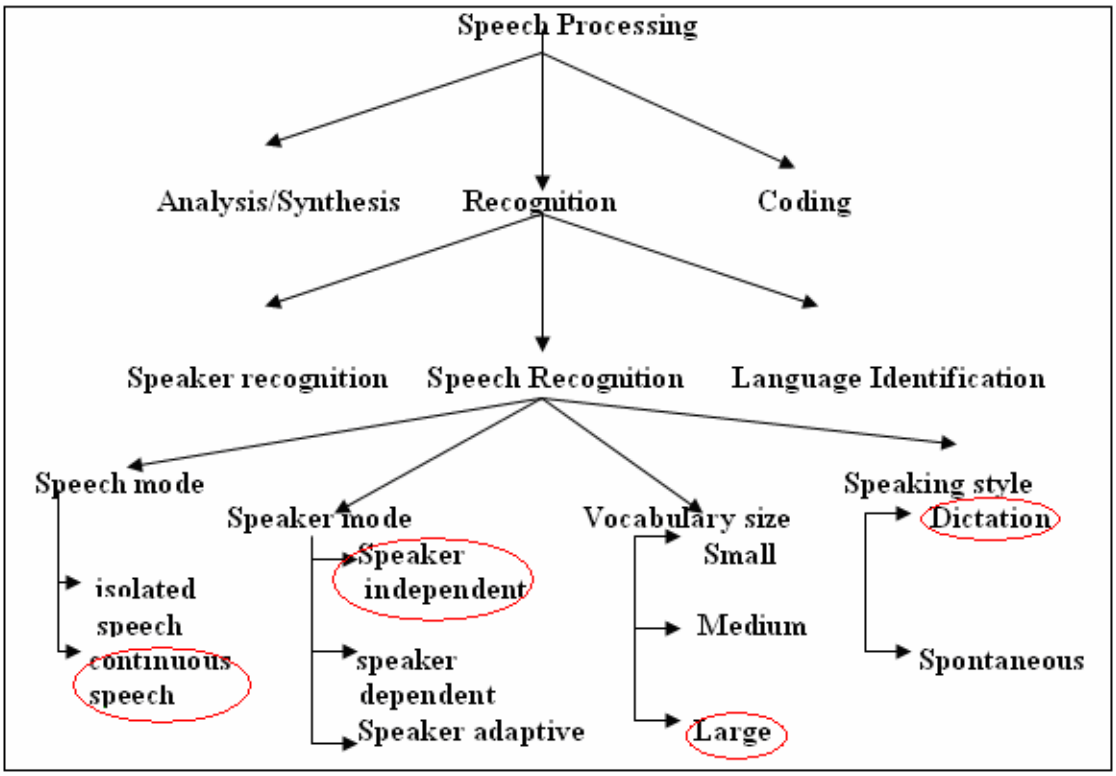


Figure 2.1: the speech processing classification

2.4. Uses and Applications

Although any task that involves interfacing with a computer can potentially use ASR, the following applications are the most common right now.

Dictation

Today, dictation is the most common use for ASR systems. Medical transcriptions, legal and business dictation, as well as general word processing are Of the most important examples of dictation uses . In some cases special vocabularies are used to increase the accuracy of the system.

Command and Control

ASR systems that are designed to perform functions and actions on the system are defined as Command and Control systems. Utterances like "Open Netscape" will do just that.

Telephony

Some voice systems allow callers to speak commands instead of pressing buttons to send specific tones

Wearable

Because inputs are limited for wearable devices, speaking is a natural possibility.

Medical/Disabilities

Many people have difficulty typing due to physical limitations such as repetitive strain injuries, muscular dystrophy, and many others. For example, people with difficulty hearing could use a system connected to their telephone to convert the caller's speech to text.

Embedded Applications

Some newer cellular phones include speech recognition that allow utterances such as "Call Home". This could be a major factor in the future of ASR and Linux[43].

2.5. Hardware

2.5.1. Sound Cards

You must have sound enabled in your kernel, and you must have correct drivers installed. Because speech requires a relatively low bandwidth, just about any medium-high quality 16 bit sound card will get the job done. Sound card quality often starts a heated discussion about their impact on accuracy and noise.

Sound cards with the 'cleanest' A/D (analog to digital) conversions are recommended, but most often the clarity of the digital sample is more dependent on the microphone quality and even more dependent on the environmental noise. Electrical "noise" from monitors, pci slots, hard-drives, etc. are usually nothing compared to audible noise from the computer fans, squeaking chairs, or heavy breathing.

You will have to weigh the benefits and costs if you are considering packages that require specific hardware to function properly. Some ASR software packages may

require a specific sound card. It's usually a good idea to stay away from specific hardware requirements, because it limits many of your possible future options and decisions.

2.5.2. Microphones

The quality of microphone is very important. It is the key when utilizing ASR. A desktop microphone just will not do the job in most cases. Because they tend to pick up more ambient noise that gives ASR programs a hard time.

Hand held microphones are also not the best choice as they can be cumbersome to pick up all the time. While they do limit the amount of ambient noise, they are most useful in applications that require changing speakers often, or when speaking to the recognizer isn't done frequently (when wearing a headset isn't an option).

The headset style is the best choice and by far it is the most common. It minimized the ambient noise, while allowing you to have the microphone at the tip of your tongue all the time. Headsets are available without earphones and with earphones (mono or stereo). We recommend the stereo headphones, but it is just a matter of personal taste[43].

2.5.3. Computers/Processors

processing speed is very important in all applications in general. ASR applications are heavily dependent on processing speed. And this is because a large amount of digital filtering and signal processing can take place in ASR. It is the faster the better about any CPU intensive software. The more memory the better also. Most software packages list their minimum requirements, because of the processing required. For fast processing (large dictionaries, complex recognition schemes, or high sample rates), you should shoot for a minimum of a 400MHz and 128M RAM.

2.6. Inside Speech Recognition

2.6.1. How Recognizers Work

Recognition systems can be broken down into two main types. Pattern recognition systems compare patterns to known/trained patterns to determine a match. Acoustic Phonetic systems use knowledge of the human body (speech production, and hearing) to compare speech features (phonetics such as vowel sounds). Most modern systems focus on the pattern recognition approach because it combines nicely with current computing techniques and tends to have higher accuracy.

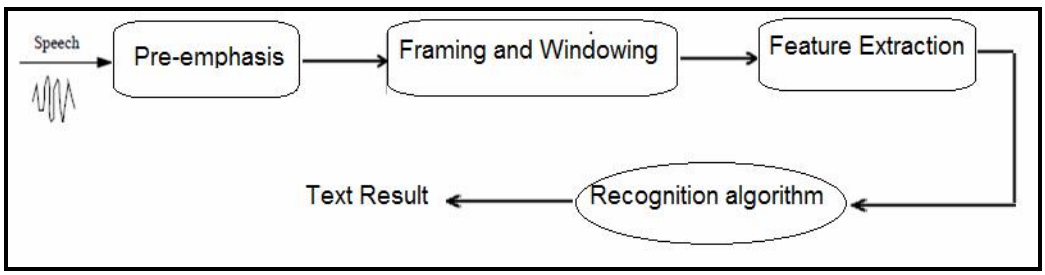


Figure 2.2: General Recognition Process

Most recognizers can be broken down into the following step, as Figure 2.2:

1. Audio recording and Utterance detection
2. Pre-Filtering (pre-emphasis, normalization, banding, etc.)
3. Framing and Windowing (chopping the data into a usable format)
4. Filtering (further filtering of each window/frame/freq. band)
5. Comparison and Matching (recognizing the utterance)
6. Action (Perform function associated with the recognized pattern)

Although each step seems simple, each one can involve a multitude of different (and sometimes completely opposite) techniques.

(1) Audio/Utterance Recording: can be accomplished in a number of ways. Starting points can be found by comparing ambient audio levels (acoustic energy in some cases) with the sample just recorded. Endpoint detection is harder because speakers tend to leave "artifacts" including breathing/sighing, teeth chatters, and echoes.

(2) Pre-Filtering: is accomplished in a variety of ways, depending on other features of the recognition system. The most common methods are the "Bank-of-Filters" method which utilizes a series of audio filters to prepare the sample, and the Linear Predictive Coding method which uses a prediction function to calculate differences (errors). Different forms of spectral analysis are also used.

(3) Framing/Windowing involves separating the sample data into specific sizes. This is often rolled into step 2 or step 4. This step also involves preparing the sample boundaries for analysis (removing edge clicks, etc.)

(4) Additional Filtering is not always present. It is the final preparation for each window before comparison and matching. Often this consists of time alignment and normalization.

(5) There are a huge number of techniques available for Comparison and Matching. Most involve comparing the current window with known samples. There are methods that use Hidden Markov Models (HMM), frequency analysis, differential analysis, linear algebra techniques/shortcuts, spectral distortion, and time distortion methods. All these methods are used to generate a probability and accuracy match.

(6) Actions can be just about anything the developer wants.

2.6.2. Digital Audio Basics

Audio is inherently an analog phenomenon. Recording a digital sample is done by converting the analog signal from the microphone to a digital signal through the A/D converter in the sound card. When a microphone is operating, sound waves vibrate the magnetic element in the microphone, causing an electrical current to the sound card (think of a speaker working in reverse). Basically, the A/D converter records the value of the electrical voltage at specific intervals.

There are two important factors during this process. First is the "sample rate", or how often to record the voltage values. Second, is the "bits per sample", or how accurate the value is recorded. A third item is the number of channels (mono or stereo), but for most ASR applications mono is sufficient. Most applications use pre-set values for these parameters and user's shouldn't change them unless the documentation suggests it.

Developers should experiment with different values to determine what works best with their algorithms.

So what is a good sample rate for ASR? Because speech is relatively low bandwidth (mostly between 100Hz-8kHz), 8000 samples/sec (8kHz) is sufficient for most basic ASR. But, some people prefer 16000 samples/sec (16kHz) because it provides more accurate high frequency information. If you have the processing power, use 16kHz. For most ASR applications, sampling rates higher than about 22kHz is a waste.

And what is a good value for "bits per sample"? 8 bits per sample will record values between 0 and 255, which means that the position of the microphone element is in one of 256 positions. 16 bits per sample divides the element position into 65536 possible values. Similar to sample rate, if you have enough processing power and memory, go with 16 bits per sample. For comparison, an audio Compact Disc is encoded with 16 bits per sample at about 44kHz.

2.7 Arabic Language

Arabic is a Semitic language, it is one of the oldest languages in the world. Currently, it is the second most spoken language in terms of number of speakers [47]. Arabic is the first language in the Arab world, *i.e.*, 25 countries. Arabic alphabets are used in other languages besides Arabic, such as Persian and Urdu [24]. The estimated number of Arabic speakers is about 300 million. However, a greater number of speakers have a passive knowledge of Arabic since it is the language of instruction in Islam. Recent approaches in language and speech processing categorize the Arabic language as Modern Standard Arabic (MSA) and Modern Colloquial Arabic (MCA). Modern Standard Arabic is the form of Arabic that is used in education, media, and formal talks. Colloquial Arabic is what is spoken in everyday conversation and varies considerably not only across countries, but also within the same country. It has many differences when compared with Indo-European languages. Some of the differences include unique phonemes and phonetic features, and a complex morphological word structure[49].

Arabic language is comparatively much less researched compared to other languages such as English and Japanese. Most of the reported studies to-date have been conducted on Arabic language and speech digital processing in general, a limited

number of research studies have been carried out on MSA, classical and Quraanic (Islamic Holy Scripture based) versions of Arabic.

Modern Standard Arabic (MSA) has 36 phonemes, of which six are vowels, two diphthongs, and 28 are consonants. In addition to the two diphthongs, the six vowels are /a, i, u, a: , i:, u:/ where the first three ones are short vowels and the last three are their corresponding longer versions (that is, the three short vowels are /a, i, u /, and their three long counterparts are /a:, i:, u:/) [48]. The Arabic language has fewer vowels than the English language as the American English has twelve vowels.

A phoneme is the smallest element of speech units that makes a difference in the meaning of a word, or a sentence. Arabic phonemes contain two distinctive classes, which are named *pharyngeal* and *emphatic* phonemes. These two classes can be found only in other Semitic languages such as Hebrew [34]. The allowed syllables in the Arabic language are: CV*, CVC, and CVCC where V indicates a (long or short) vowel while C indicates a consonant.*

Table 2.1: Arabic Syllables Pattern

	Open	Example	Close	Example
Short	CV	با		
Long			CVC	جيم
			CVCC	عين

All Arabic syllables must contain a vowel. In addition, Arabic vowels cannot be word initial and must occur either between two consonants or at word-final position. Arabic syllables can be classified as short or long. The CV syllable type is a short one while all others are long. Syllables can also be classified as open or closed. An open syllable ends with a vowel while a closed syllable ends with a consonant. For Arabic, a vowel always forms a syllable nucleus, and there are as many syllables in a word as vowels in it [50]. With very few exceptions, alphabet-to-sound conversion for Arabic usually has simple one-to-one mapping between graphemes to phonemes for a given correct dicritized text [51].

2.8 Arabic Phoneme Set

In order to produce a robust speaker-independent continuous automatic Arabic speech recognizer, a set of speech recordings that are rich and balanced is required. The rich characteristic is in the sense that it must contain all the phonemes of Arabic language. It must be balanced in preserving the phonetics distribution of Arabic language too. This set of speech recordings must be based on a proper written set of sentences and phrases created by experts. Therefore, it is crucial to create a high quality written (text) set of the sentences and phrases before recording them.

Creating phonetically rich and balanced text corpus requires selecting a set of phonetically rich words, which are combined together to produce sentences and phrases. These sentences and phrases are verified and checked for balanced phonetic distribution. [64]. In 1997, KACST created a database for Arabic language sounds. The purpose of this work was to create the least number of phonetically rich Arabic words. As a result, a list of 663 phonetically rich words containing all Arabic phonemes, which are subject to all Arabic phonotactic rules was produced. This work is the backbone for creating individual sentences and phrases, which can be used for Arabic ASR and text-to-speech synthesis applications. The list of 663 phonetically rich words was created based on the following characteristics and guidelines :

1. Coverage of all Arabic phonemes which must be balanced so as to be close in frequency as possible.
2. Coverage of all Arabic phoneme clusters.
3. The presence of the least possible number of words so that the list does not contain a single word whose goal of existence is achieved by another word in the same list.

In 2003, KACST produced a technical report of the project “Database of Arabic Sounds: Sentences.”, Arabic independent sentences have been written using the said 663 phonetically rich words. The database consists of 367 sentences; 2 to 9 words per sentence. Therefore, we have an Arabic phrases and sentences that are phonetically rich and balanced based on the previously created list of 663 phonetically rich words, which were put in phrases and sentences while taking into consideration the following goals [41]:

• To have the minimum word repetitions as far as possible.
• To have structurally simple sentences in order to ease readability and pronunciation.
• To have the minimum number of sentences.

An average of 2 phonetically rich words and 5 other words were used in each single sentence. Statistical analysis shows that 1333 words were repeated once only and 99 words were repeated more than once in the entire 367 sentences, whereas 17 words were repeated 5 times and more. KACST 367 phonetically rich and balanced sentences are used for training purposes in our system, Table 4.1 shows more technical details of our speech corpus[42].

Table 2.2: Data Base Criteria

Criteria	**Data**
No. of Sentences	367 sentences
Number of Words	1435 words
Average No. of Words/Sentence	5 words
Min. and Max. No. of Words/Sentence	Min. of 2 and max. of 9
No. of Speakers	40 speakers
Speakers Age	18 to 66 years
Speakers Gender	20 males and 20 females
Average no. of sound Files/sentence	100 sound files/sentence
Sampling Rate(Hz)	44.1 KHz
No. of Bits	16 bits

2.9 Feature Extraction

Feature extraction stage is the most important one in the entire process, since it is responsible for extracting relevant information from the speech frames, as feature parameters or vectors. Common parameters used in speech recognition are Linear Predictive Coding (LPC) coefficients, and Mel Frequency Cepstral Coefficients (MFCC). These parameters have been widely used in recognition system partly to the following reasons:

- The calculation of these parameter leads to a source-filter separation.
- The parameters have an analytically tractable model.
- Experience proves that these parameters work well in recognition applications.

Due to their significance, they will be described in two different subsections.

2.9.1. Pre-emphasis

In order to flatten speech spectrum, a pre-emphasis filter is used before spectral analysis. Its aim is to compensate the high-frequency part of the speech signal that was suppressed during the human sound production mechanism. The most used filter is a high-pass filter described in Equation(1), and whose transfer function corresponds to Figure 2.3.

$$H_{preem}(z) = 1 - a_{preem} z^{-1} \quad \text{.....(1)}$$

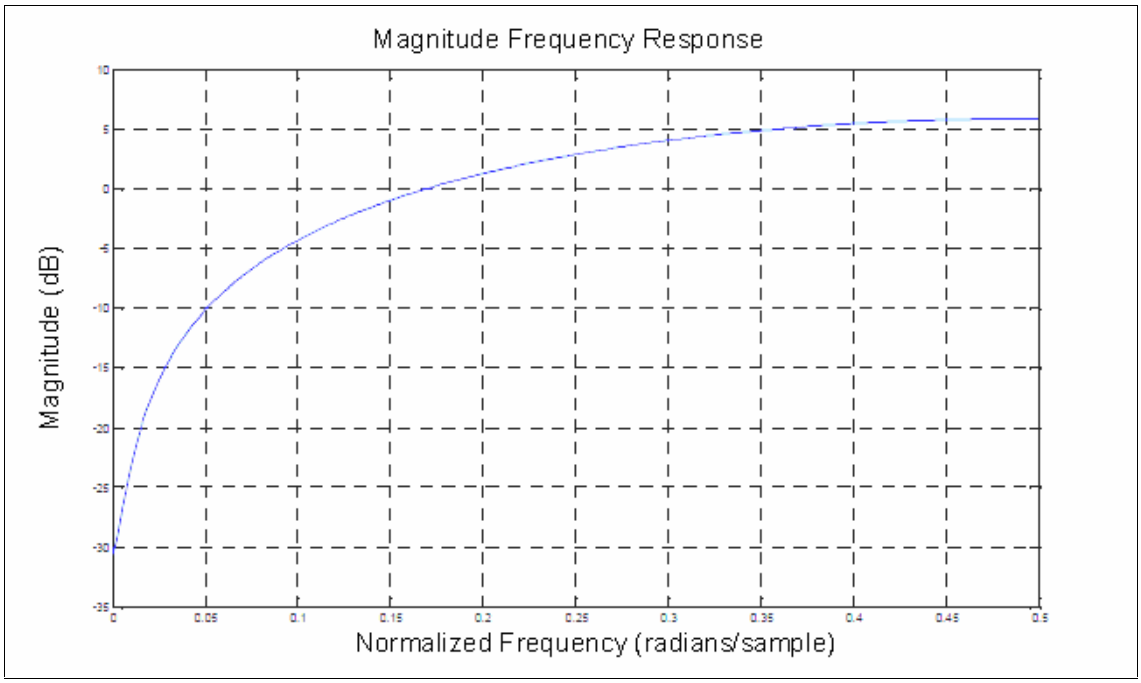


Figure 2.3: Pre-emphasis Filter, a=0.97

2.9.2. Frame Blocking and Windowing

The speech signal is divided into a sequence of frames where each frame can be analyzed independently and represented by a single feature vector. Since each frame is supposed to have stationary behavior, a compromise, in order to make the frame blocking, is to use a 20-25 ms window applied at 10 ms intervals (frame rate of 100 frames/s and overlap between adjacent windows of about 50%), as Holmes & Holmes exposed in 2001. One can see this in Figure 2.4.

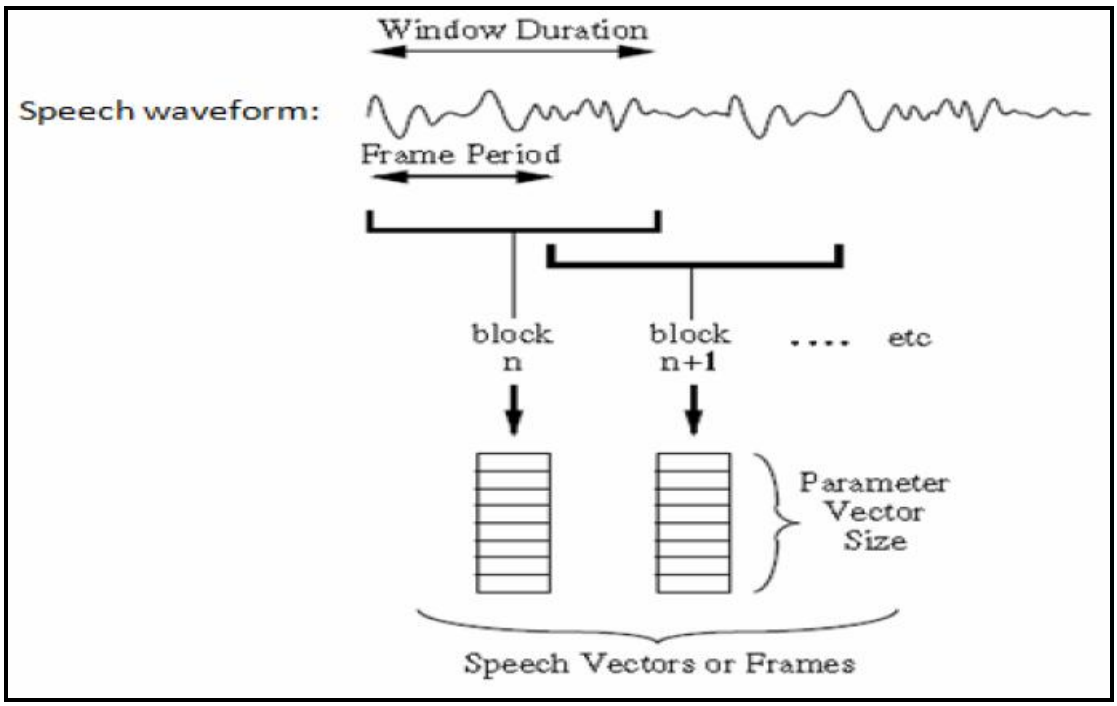


Figure 2.4: Frame Blocking

In order to reduce the discontinuities of the speech signal at the edges of each frame, a tapered window is applied to each one. The most common used window is Hamming window, described in Equation(2) and shown in Figure 2.5.

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\Pi(n-1)}{N-1}\right) \quad \text{.....(2)}$$

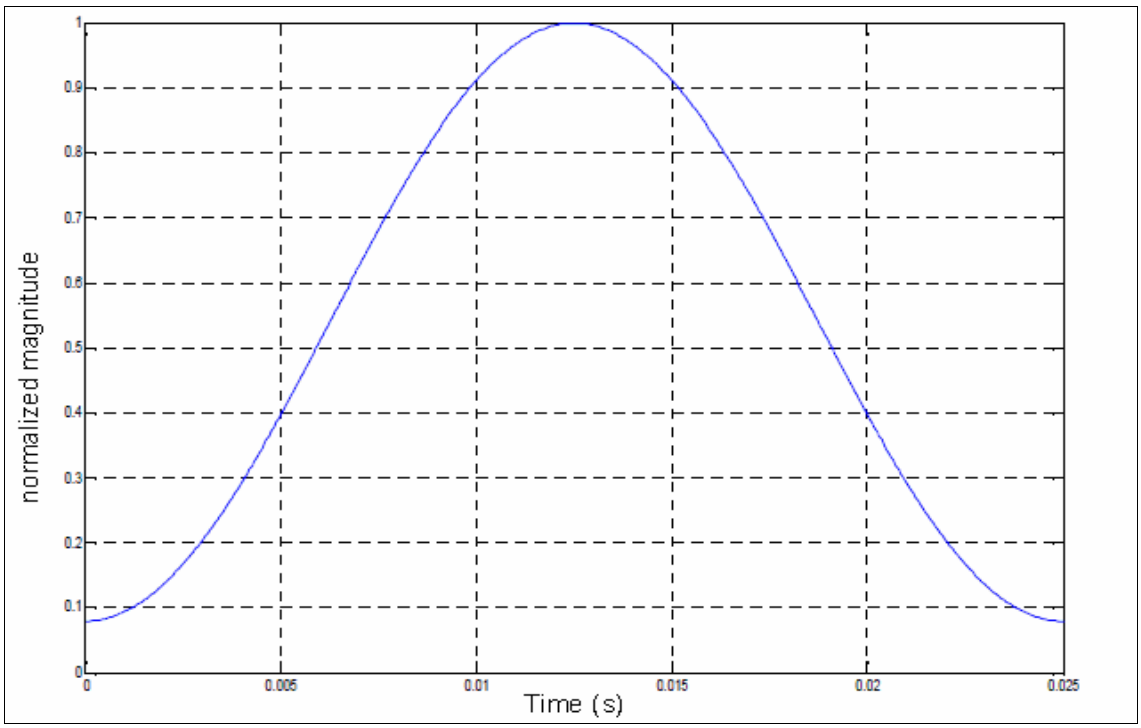


Figure 2.5: An 25ms Hamming Window (fs=16KHz)

2.9.3. Mel- Cepstrum

The Mel Frequency Cepstrum6 Coefficients (MFCC) representation as a beneficial approach for speech recognition. The MFCC is a representation of the speech signal defined as the real cepstrum of a windowed short-time signal derived from the Fast Fourier Transform (FFT) of that signal which, is first subjected to a log-based transform of the frequency axis (mel-frequency scale), and then de-correlated using a modified Discrete Cosine Transform (DCT-II).

Figure 2.6 illustrates the complete process to extract the MFFC vectors from the speech signal. It is to be emphasized that the process of MFCC extraction is applied over each frame of speech signal independently.

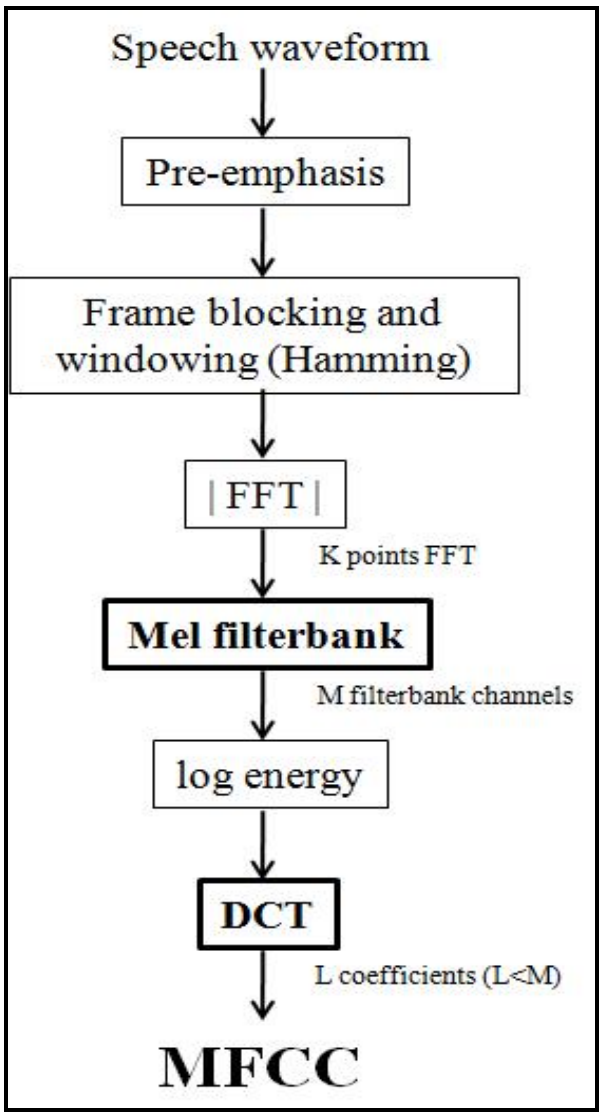


Figure 2.6:The MFCC extraction process

After the pre-emphasis and the frame blocking and windowing stage, the MFCC vectors will be obtained from each speech frame. The process of MFFC extraction will be described below considering in any instant that all the stages are being applied over speech frames. The first step of MFCC extraction process is to compute the Fast Fourier Transform (FFT) of each frame and obtain its magnitude.

The FFT is a computationally efficient algorithm of the Discrete Fourier Transform (DFT). If the length of the FFT, is a power of two (K=2^n), a faster algorithm can be used, so a zero-padding to the nearest power of two within speech frame length is performed. The next step will be to adapt the frequency resolution to a perceptual frequency scale which satisfies the properties of the human ears, such as a perceptually mel-frequency scale. This issue corresponds to Mel filter bank stage. The filter-bank analysis consists of a set of band pass filter whose bandwidths and spacings are roughly equal to those of critical bands and whose range of the centre frequencies covers the most important frequencies for speech perception[52].

Chapter 3 Sphinx-4: Open Source Framework

3.1. Introduction

In the speech recognition approach, when researchers need to explore one problem of it, they are often forced with the needing of core speech recognition, to be used in research, so they often need to develop an entire system from scratch.

Many open source speech recognition systems are available, such as HTK [53], ISIP [54], AVCSR [55] and earlier versions of the Sphinx systems [56], [57] these open source systems are typically optimized for a single approach to speech system design. So they considered as barriers to future research that departs from the original purpose of this systems.

3.2. Speech Recognition Systems

Many research systems such as Dragon [58], Harpy [59], Sphinx and others, are created by using the traditional approach to speech recognition system design, which using an optimized around a particular methodology to create the entire system.
This approach has proved to be quite valuable in speech recognition, as a result these systems have provided foundational methods for speech recognition research

3.3. Sphnix-4 Framework

Sphinx-4 is a speech recognition system written entirely in the Java programming language. It is a flexible, modular and pluggable framework to help support new innovations in the research of hidden Markov model (HMM) recognition systems. It is a very flexible system capable of performing many different types of recognition tasks. Figure 3.1 shows the overall architecture of the system. Each component in Figure 3.1 represents a module that can be easily replaced, allowing researchers to experiment with different module implementations without needing to modify other portions of the system.

The Sphinx-4 framework consists of three main modules that are the Front-End, the Decoder, and the Linguist. One or more input signals are enter the Front-End

module, it parameterizes them into a sequence of Features. The Linguist translates any type of standard language model, along with pronunciation information from the Dictionary and structural information from one or more sets of AcousticModels function, into a SearchGraph. The decoder has a SearchManager, that uses the Features from the Front-End and the SearchGraph from the Linguist to perform the actual decoding, and then generating the results. At any time prior to or during the recognition process, the application can issue Controls to each of the modules, effectively becoming a partner in the recognition process. Like most speech recognition systems, Sphinx-4 system has a large number of configurable parameters, such as search beam size, that controls the system performance. These parameters configured using the Sphinx-4 configuration manager. Unlike other systems, this configuration manager also gives Sphinx-4 the ability to dynamically load and configure modules at run time, yielding a flexible and pluggable system. number of Tools provided by Sphinx-4 are used to give applications and developers the ability to track decoder statistics such as word error rate [60], run time speed, and memory usage. This Tools are highly configurable, allowing users to perform a wide range of system analysis. And also provide an interactive run time environment that allows users to modify the parameters of the system while the system in the run time, and allow rapid experimentation with various parameters settings.

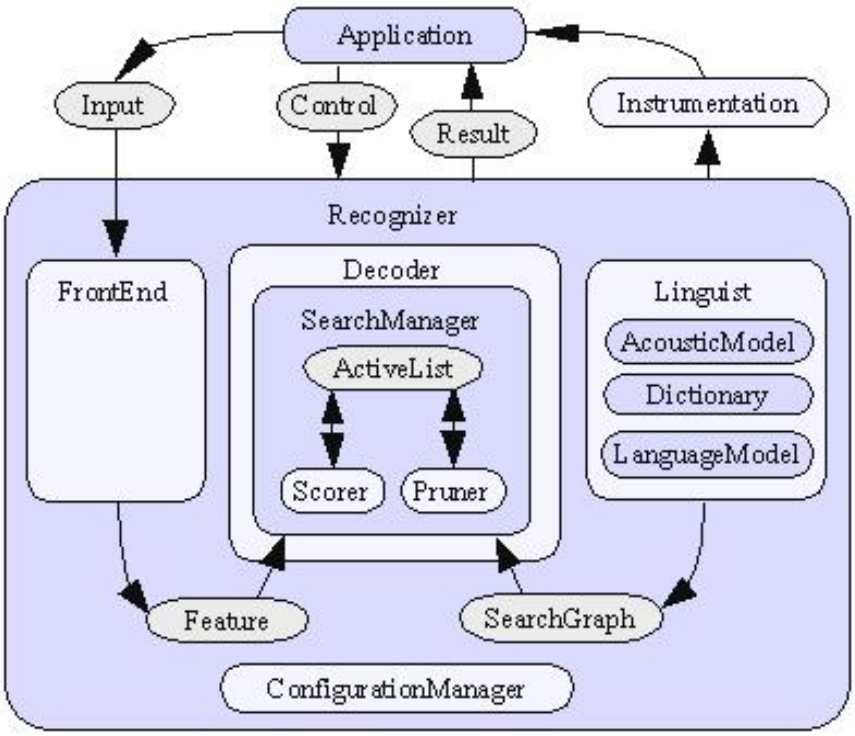


Figure 3.1: Sphinx 4 components

3.3.1 Front-End

The front-end module receive the input signal [audio] then parameterize it into a sequence of output features. The Front-End consists of one or more parallel chains of replaceable communicating signal processing modules called DataProcessors. These DataProcessors are responsible for producing a data object composed of parameterized signals, called features, to be used by the decoder.

Figure 3.2 illustrates the entire structure of the front-end module, many types of DataProcessors each implement common signal processing techniques. These implementations include support for the following: reading from a variety of input formats for batch mode operation, reading from the system audio input device for live mode operation, preemphasis, windowing with a raised cosine transform (e.g., Hamming and Hanning windows), discrete Fourier transform (via FFT), mel frequency filtering, bark frequency warping, discrete cosine transform (DCT), linear predictive encoding (LPC), end pointing, cepstral mean normalization (CMN), mel cepstra frequency coefficient extraction (MFCC), and perceptual linear prediction coefficient extraction (PLP).

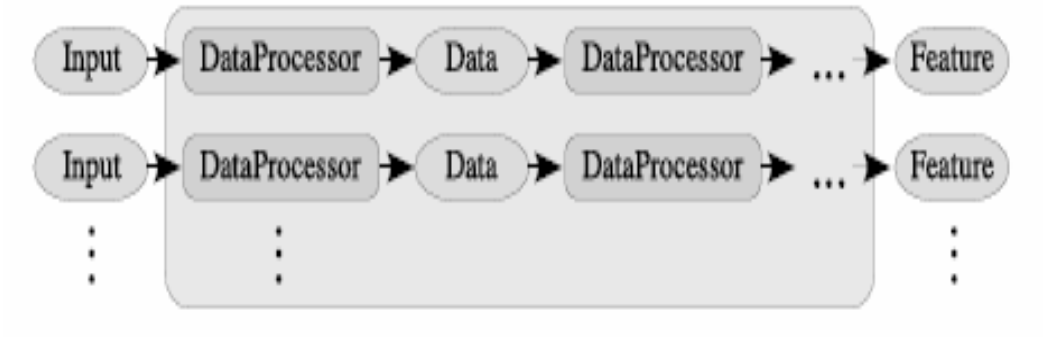


Figure 3.2 : Front-End Framework

Within the generic Front-End framework, supporting multiple chains permits simultaneous computation of different types of parameters from the same or different input signals. This enables the creation of systems that can simultaneously decode using different parameter types, such as MFCC and PLP.

3.3.2 Linguist

The Linguist module consists of three pluggable components: the Language Model, the Dictionary, and the Acoustic Model, these components used to generate the SearchGraph that is used by the decoder during the search, while at the same time hiding the complexities involved in generating this graph. This SearchGraph constructed using the language structure as represented by a given Language Model and the topological structure of the Acoustic Model (HMMs for the basic sound units used by the system). and also use a Dictionary (typically a pronunciation of the word) to map words from the Language Model into sequences of Acoustic Model elements.

The size of the linguist is very important in the performance of the recognizer, for example when using a simple numerical digits recognition application might use a simple Linguist that keeps the search space entirely in memory. On the other hand, a dictation application with a 100K word vocabulary might use a sophisticated Linguist that keeps only a small portion of the potential search space in memory at a time. Now we will produce the basic components of the linguist in more details.

Language Model

The first component of the linguist is the Language Model, this component provides word-level language structure, which can be represented by any number of pluggable implementations as following:

- SimpleWordListGrammar: defines a grammar based upon a list of words. An optional parameter defines whether the grammar “loops” or not. If the grammar does not loop, then the grammar will be used for isolated word recognition. If the grammar loops, then it will be used to support trivial connected word recognition that is the equivalent of a unigram grammar with equal probabilities.
- JSGFGrammar: supports the Java™Speech API Grammar Format (JSGF) [60], which defines a BNF-style, platform-independent, and vendor-independent Unicode representation of grammars.
- LMGrammar: defines a grammar based upon a statistical language model. LMGrammar generates one grammar node per word and works well with smaller unigram and bigram grammars of up to approximately 1000 words.

- FSTGrammar: supports a finite-state transducer (FST) [61] in the ARPA FST grammar format.
- SimpleNGramModel: provides support for ASCII N-Gram models in the ARPA format. The SimpleNGramModel makes no attempt to optimize memory usage, so it works best with small language models.
- LargeTrigramModel: provides support for true N-Gram models generated by the CMUCambridge Statistical Language Modeling Toolkit [62]. The Large Trigram Model optimizes memory storage, allowing it to work with very large files of 100MB or more.

Dictionary

The second component is the Dictionary, which contains of the words and their pronunciation phoneme. The pronunciations break words into sequences of sub-word units found in the Acoustic Model. The Dictionary interface also supports the classification of words and allows for a single word to be in multiple classes.

Sphinx-4 currently provides implementations of the Dictionary interface to support the CMU Pronouncing Dictionary [63]. The various implementations optimize for usage patterns based on the size of the active vocabulary. For example, one implementation will load the entire vocabulary at system initialization time, whereas another implementation will only obtain pronunciations on demand.

Acoustic Model

The third component of the linguist is the Acoustic Model, which describes sounds of the language. It provides a mapping between a unit of speech and an HMM that can be scored against incoming features provided by the Front-End. This mapping take into account the contextual and word position information. For example, in the case of triphones, the context represents the single phonemes to the left and right of the given phoneme, and the word position represents whether the triphone is at the beginning, middle, or end of a word (or is a word itself).

SearchGraph

As mentioned before the linguist use its three components to generate the SearchGraph, which will be used in the decoding process. It is a data structure and the manner in which it is constructed affects the memory footprint, speed, and recognition accuracy. As in Figure 3.3 it is a directed graph in which each node, called a SearchState, represents either an emitting or a non-emitting state. Emitting states can be scored against incoming acoustic features while non-emitting states are generally used to represent higher-level linguistic constructs such as words and phonemes that are not directly scored against the incoming features.

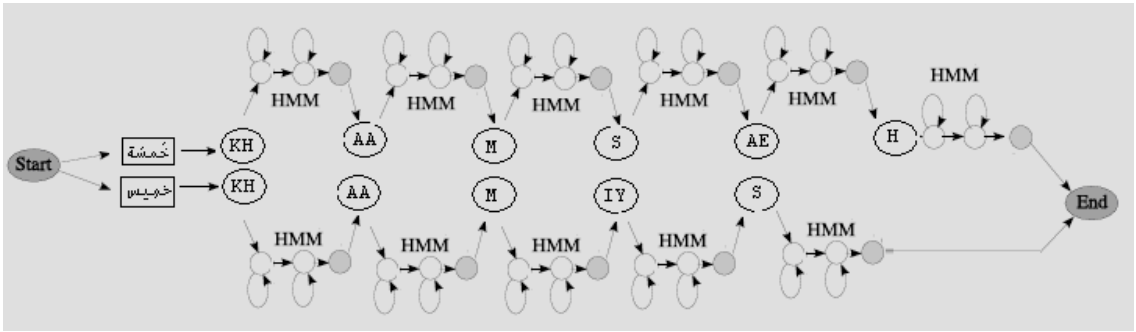


Figure 3.3: SearchGraph Example

Implementations

As with the Front-End, Sphinx-4 provides several implementations of the Linguist to support different tasks.

-The FlatLinguist is appropriate for recognition tasks that use context-free grammars (CFG), finite-state grammars (FSG), finite-state transducers (FST) and small N-Gram language models.

-The DynamicFlatLinguist is similar to the FlatLinguist in that is appropriate for similar recognition tasks. The main difference is that the DynamicFlatLinguist dynamically creates the SearchGraph on demand, giving it the capability to handle far more perplex grammars. With this capability, however, comes a cost of a modest decrease in run time performance.

-The LexTreeLinguist is appropriate for large vocabulary recognition tasks that use large N-Gram language models. The LexTreeLinguist supports ASCII and binary

language models generated by the CMU-Cambridge Statistical Language Modeling Toolkit [62].

3.3.3 Decoder

The main part in the recognizer is the decoder, as it is responsible of generating the resulting text using the features extracted by the Front-End and the SearchGraph that constructed in the Linguist module. The primary component of the Decoder block is the SearchManager, this SearchManager creates a Result object that contains all the paths that have reached a final non-emitting state. The SearchManager's primary role is to execute the search for a given number of frames. The SearchManager will return interim results as the recognition proceeds and when recognition completes a final result will be returned. Sphinx-4 provides a sub-framework to support SearchManagers composed of an ActiveList, a Pruner and a Scorer. The implementations of a SearchManager may construct a set of active tokens in the form of an ActiveList at each time step.

Applications can configure the Sphinx-4 implementations of the Pruner to perform both relative and absolute beam pruning. The implementation of the Pruner is greatly simplified by the garbage collector of the Java platform. With garbage collection, the Pruner can prune a complete path by merely removing the terminal token of the path from the ActiveList. The act of removing the terminal token identifies the token and any unshared tokens for that path as unused, allowing the garbage collector to reclaim the associated memory. The third part is the Scorer, a pluggable state probability estimation module that provides state output density values on demand. When the SearchManager requests a score for a given state at a given time, the Scorer accesses the feature vector for that time and performs the mathematical operations to compute the score, the Scorer retains all information pertaining to the state output densities[46].

Chapter 4 The Arabic Speech Recognition

4.1 The Proposed Work

This section describes how to create and develop an Arabic speech recognition system using the open source framework Sphinx-4. Both training and recognizing process use Arabic characters.

4.1.1 System Overview

A complete ASR system based on CMUSphinx4 system, which is HMM-based, is built. The system is speaker-independent and continuous recognition. It is capable of handling large vocabularies. Our approach for modeling Arabic sounds in the CMU Sphinx system consist of construct and train the acoustic and language models with Arabic speech data and generate the dictionary with Arabic characters.

Figure 4.1 shows a mathematical representation of speech recognition system in simple equations which contain front end unit, acoustic model unit, language model unit, and search unit.

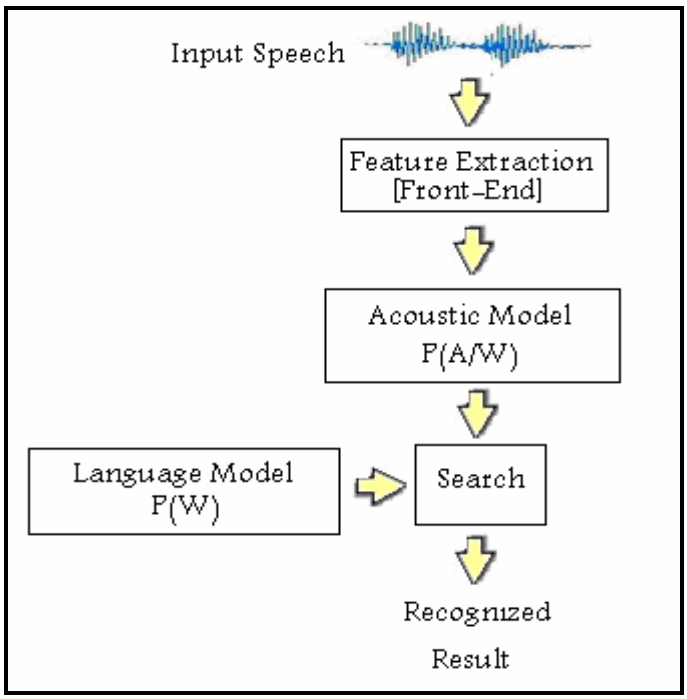


Figure 4.1 :Basic model of speech recognition

The standard approach to large vocabulary continuous speech recognition is to assume a simple probabilistic model of speech production where by a specified word sequence, W, produces an acoustic observation sequence Y, with probability P(W,Y). The goal is then to decode the word string, based on the acoustic observation sequence, so that the decoded string has the maximum a posteriori (MAP) probability.

$$\hat{P}(W/A) = \arg\max_{W} P(W/A) \quad \ldots\ldots(3)$$

Using Baye's rule, the equation can be written as

$$P(W/A) = \frac{P(A/W)P(W)}{P(A)} \quad \ldots\ldots(4)$$

Since P(A) is independent of W, the MAP decoding rule of the equation is

$$\hat{W} = \hat{\arg\max}_{w} P(A/W)P(W) \quad \ldots\ldots(5)$$

The first term in the last equation P(A/W), is generally called the acoustic model, as it estimates the probability of a sequence of acoustic observations, conditioned on the word string. Hence P(A/W) is computed. For large vocabulary speech recognition systems, it is necessary to build statistical models for sub word speech units, build up word models from these sub word speech unit models (using a lexicon to describe the composition of words), and then postulate word sequences and evaluate the acoustic model probabilities via standard concatenation methods. The second term in equation P(W), is called the language model. It describes the probability associated with a postulated sequence of words. Then we illustrate how to employing our system in an application and test it.

4.2 Training Phase

To train a new model we must prepare our system by installing a group of software, which are:

-- SphinxTrain: We can download the SphinxTrain CMU, it is a free training package.

The execution of SphinxTrain requires additional software:

-- Active Perl: To edit scripts provided by SphinxTrain.

4.2.1 Feature Extraction

The recorded speech is sampled at a rate of 16 ksps. The basic feature vector uses the Mel Frequency Cepstral Coefficients MFCC. The mel-frequency scale is linear frequency spacing below 1000 Hz and a logarithmic spacing above 1000 Hz. The MFCCs are obtained by taking the Discrete Cosine Transform (DCT) of the log power spectrum from Mel spaced filter banks [17]. 13 Mel frequency cepstra are computed, x(0), x(1),… x(12), for each window of 25 ms, with adjacent windows overlapped by 15 ms. x(0) represents the log mel spectrum energy, and is used to derive other feature parameters, see Section 2.8 for more information. The system uses the rest 12 coefficients as a basic feature vector. The basic feature vector is usually normalized by subtracting the mean over the sentence utterance. Table 4.2 show all parameters used in feature extraction.

Table 4.2 : Parameters used in feature extraction

Parameter	**Value**
Sampling Rate	16Khz
Pre-emphases Coefficient	0.97
Window Size	25ms
Overlap Duration	15ms
Hamming Window	True
Zero Mean	True
Cepstral lifter	22
Number of Cepstral Coefficient	12

4.2.2 Linguist model

This sub-system contains the details that describe the recognized language itself. This sub-system is where most of the adjustments are made in order to support the Arabic Language recognition. It consists of three main modules:

The Acoustic Model: This module provides the HMMs of the Arabic triphones to be used to recognize speech.

The Language Model: This module provides the grammar that is used by the system (Usually the grammar of a natural language or a subset of it).

The Dictionary: This module serves as an intermediary between the Acoustic Model and the Language Model. It contains the words available in the language and the pronunciation of each in terms of the phonemes available in the acoustic model.

It will use the knowledge from these three components to construct a search graph that is appropriate for recognizing task.

So we can construct the acoustic model, the dictionary, and the language model as follow:

The Dictionary

In this step we mapped each word in the vocabulary to a sequence of sound units representing pronunciation; that it contains all words with all possible variants of their pronunciation.

To take into account pronunciation variability, caused by various speaking manners and the specificity of Arabic. Careful preparation of phonetic dictionary prevents the incorrect association to a phoneme with audio parameters, which would effect in decreasing the model's accuracy[65].

Table 4.3 shows the listing of the phoneme set used in the training stage and the corresponding symbols. The table also shows illustrative examples of the vowel usage.

We use the training wav files to construct the dictionary, the file result is ar.dict that contain all the words pronunciations. For example:

الأرض	E L E AE R DD
الأرض-2	L E AE R DD
إدَارَة	E IH D AE: R AA H
إدَارَة-2	E IH D AE: R AA T

Table 4.3: The phoneme list used in the training

الحرف	الرمز الصوتي	الحرف	الرمز الصوتي
د	/D/	ـَ ◄ بَ	/AE/
ذ	/DH/	ـَا ◄ بَاب	/AE:/
ر	/R/	ـَ ◄ خَ	/AA/
ز	/Z/	ـُ ◄ بُ	/UH/
س	/S/	ـُو ◄ دُون	/UW/
ص	/SS/	ـُ ◄ غُصن	/UX/
ض	/DD/	ـِ ◄ بِنت	/IH/
ط	/TT/	ـِي ◄ فِيل	/IY/
ظ	/DH2/	ـَو ◄ لَوم	/AW/
ع	/AI/	ـَي ◄ ضَيف	/AY/
غ	/GH/	ـُ ◄ نُنجي	/UN/
ف	/F/	ء	/E/
ف◄فيزا	/V/	ب	/B/
ق	/Q/	ت	/T/
ك	/K/	ث	/TH/
ل	/L/	جيم فصحة	/JH/
م	/M/	جيم مصرية	/G/
ن	/N/	جيم معطشة	/ZH/
هـ	/H/	ح	/HH/
و	/W/	خ	/KH/
ي	/Y/		

Language Model

There are two types of models that describe language - grammars and statistical language models. Grammars describe very simple types of languages for command and control, and they are usually written by hand or generated automatically with plain code.

Language Model is another important requirement for any ASR system. Creation of a language model consists of computing the word uni-gram counts, which are then converted into a task vocabulary with word frequencies, generating the bi-grams and trigrams from the training text based on this vocabulary, and finally converting the n-grams into a binary format language model and standard ARPA format.

There are many ways to build the statistical language models. When a model is small, you can use an online quick web service. When your data set is large, there is sense to use CMU language modeling toolkit(CMU SLM toolkit), which we used here.

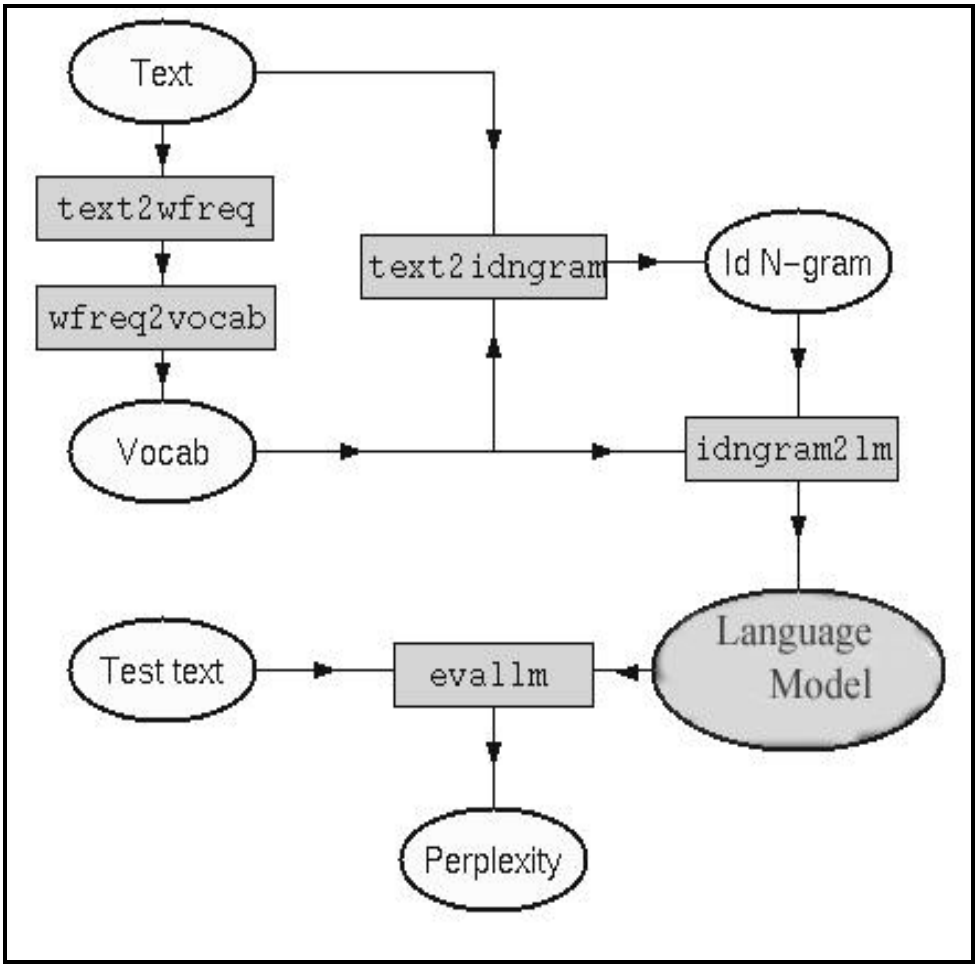


Figure 4.2: Text to Language Model Mapping Instructions[63]

The steps for creating and testing the language model are shown in Figure 4.2. The creation of a language model from a training text consists of the following steps:

1) Prepare a reference text that will be used to generate the language model. The language model toolkit expects its input to be in the form of normalized text files, with utterances delimited by <s> and </s> tags. The output file called a.txt.

2) Compute the word unigram counts

```
cat a.txt | text2wfreq > a.wfreq
```

3) Convert the word unigram counts into a vocabulary consisting of the 20,000 most common words

```
cat a.wfreq | wfreq2vocab -top 20000 > a.vocab
```

4) Generate a binary id 3-gram of the training text, based on this vocabulary

```
cat a.text | text2idngram -vocab a.vocab > a.idngram
```

5) Convert the idngram into a binary format language model

```
idngram2lm -idngram a.idngram -vocab a.vocab -binary a.binlm
```

See appendix A for more details.

Converting model into DMP format

To quickly load large models you probably would like to convert them to binary format that will save your decoder initialization time. That's not necessary with small models. Sphinx4 requires you to submit DMP model into TrigramModel component. DMP format is mutually convert able. You can produce other file with sphinx_lm_convert command from sphinxbase:

```
sphinx_lm_convert -i model.lm -o model.dmp
```

Testing your language model with PocketSphinx

In PocketSphinx, we have a program called pocketsphinx_continuous which can be run from the command-line to recognize speech. We try running the following command:

```
pocketsphinx_continuous -lm a.lm -dict dict.dic
```

Acoustic Model

The trainer learns the parameters of the models of the sound units using a set of sample speech signals. This is called a training database. The database contains information required to extract statistics from the speech in form of the acoustic model. You have to design database prompts and post process the results to ensure that audio actually corresponds to prompts. The folder structure is:

- AR
    - ar.dic - Phonetic dictionary
    - ar.phone - Phoneset file
    - ar.lm.DMP - Language model
    - ar.filler - List of fillers
    - ar_train.fileids - List of files for training
    - ar_train.transcription - Transcription for training
    - ar_test.fileids - List of files for testing
    - ar_test.transcription - Transcription for testing
- wav
    - speaker_1
        - file_1.wav - Recording of speech utterance
    - speaker_2
        - file_2.wav

Let's go through the files and describe their format and the way to prepare them:

** *Fileids (ar_train.fileids and ar_test.fileids)* file is a text file listing the names of the recordings (utterance ids) one by line

```
speaker_1/file_1
```

** *Transcription file (ar_train.transcription and ar_test.transcription)* is a text file listing the transcription for each audio file

<s> السلام عليكم </s> (file_1)

** *Phoneset file (ar.phone)* should have one phone per line. The number of phones should match the phones used in the dictionary plus the special SIL phone for silence:

AH
AX

** *Filler dictionary (ar.filler)* contains filler phones (not-covered by language model non-linguistic sounds like breath, hmm or laugh). It can contain just silences:

<s> SIL

</s> SIL

<sil> SIL

To start the training change to the database folder and run the following commands:

For SphinxTrain

../SphinxTrain/scripts_pl/setup_SphinxTrain.pl -task an4

../PocketSphinx/scripts/setup_sphinx.pl -task an4

Using the model

After training, the acoustic model is located in *model_parameters/a.cd_cont_1000* You need only that folder. The model should have the following files:

- ✓ license.term
- ✓ mdef [model definition]
- ✓ feat.params
- ✓ means
- ✓ mixture_weights
- ✓ transition_matrices
- ✓ variancesic

Now we have a complete linguistic model components: Arabic dictionary, Arabic language model, and Arabic acoustic model. How to use these components is the topic of the next section.

4.3 Testing Phase

For testing the Arabic speech recognition we need at first to have this software installed on our processor:

1) *Sphinx-4:* Sphinx-4 can be downloaded either in binary format or in source codes [66] It was compiled and tested on several versions of Linux and on Windows operating systems.

   ***sphinx4-{version}-bin.zip*:** provides the jar files, documentation, and demos

sphinx4-{version}-src.zip: provides the sources, documentation, demos, unit tests and regression tests.

2) Running, building and testing sphinx-4 requires additional software:

- Java 2 SDK, Standard Edition 6.0 [67].
- Java Runtime Environment (JRE)
- Ant: the tool to facilitate compilation and the implementation of sphinx-4 system [68].
- Subversion: install cygwin, which will give a Linux like environment in a command prompt window.
- Get an IDE as Eclipse

4.3.1 Test System Overview

To test the model in an application, the project must contain these files:

- Project Name
  - Src … source folder
    - Java file
    - Config.xml file
  - JRE library … contain the general library used in java
  - External Library … contain the sphinx files and our model files

4.3.2 Test System Creation

Now at first, we must create a new java project using the Eclipse, then do the following :

• Insert the APIs Sphinx-4 into the new project.

You can start to link in the jars that you will need to do simple speech recognition. By open the project java build path and add a new library then add a new jars by expand the lib folder of the sphinx-4 and you'll see the following jar files in it:

js.jar

jsapi.jar

sphinx4.jar

WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.jar

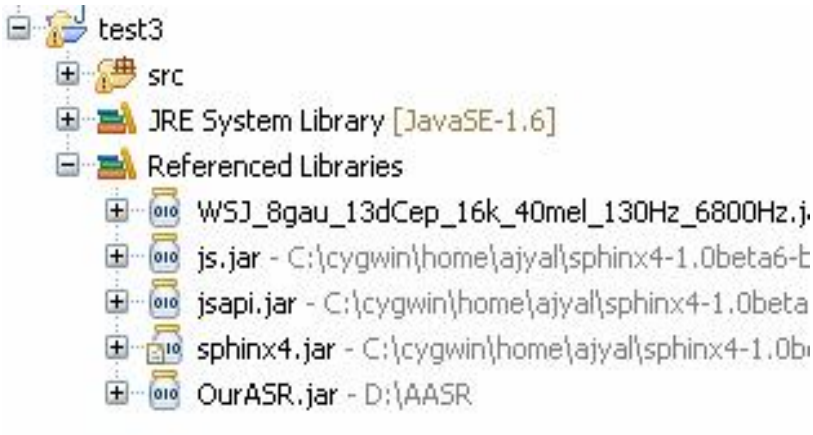
This adds (links) the jar to your build path allows the IDE to use code from the jar for your project.

• Create a jar file that contain the dictionary and the models:

Open the folder that contains the models, then use the command:

```
C:\jdk1.6.0_30\bin\Jar –cvf  OurASR.jar  *
```

A new jar file will be created with the name OurASR.jar, then insert it the project with the same steps: My Project->build path->add the jar to build path. When that is done, your folder structure should look something like this:



Notice when using the sphinx systems the important file here is the configure xml file. this file determine the configuration of sphinx 4 system. This configuration file defines the following:

- The names and types of all of the components of the system
- The connectivity of these components - that is, which components talk to each other
- The detailed configuration for each of these components.

Using new models is easy, you just need to configure the recognizer properly. It usually includes three steps:

- Defining a dictionary and a language model
- Defining a model and a model loader
- Configure a front-end

4.3.3 Used Data

For testing phase : used the microphone to record the speaker sound And to allow the system to listen to the microphone, we must added the property of it as follow in the configure file:

```
<component name="microphone"
           type="edu.cmu.sphinx.frontend.util.Microphone">
    <property name="closeBetweenUtterances" value="false"/>
</component>
```

4.3.4 Feature Extraction:

Sphinx-4 used Front-End, which is a wrapper class for the chain of front end processors. It provides methods for manipulating and navigating the processors. The front end is modeled as a series of data processors see Figure 3.1, each of which performs a specific signal processing function. For example, a processor performs Fast-Fourier Transform (FFT) on input data, another processor performs high-pass filtering.
The input data to the front end is typically audio data, but this front end allows any input type. Similarly, the output data is typically features, but this front end allows any output type. You can configure the front end to accept any input type and return any output type. The front end must be configured through the Sphinx properties file.
Current front ends generate features that contain MFCC. To specify such a front end (called a 'pipeline') in Sphinx-4, we insert the following lines in the Sphinx-4 configuration file:

```
<!-- ********************************************************** -->
<!-- The live frontend configuration                            -->
<!-- ********************************************************** -->
<component name="epFrontEnd" type="edu.cmu.sphinx.frontend.FrontEnd">
    <propertylist name="pipeline">
        <item>microphone </item>
        <item>dataBlocker </item>
        <item>speechClassifier </item>
        <item>speechMarker </item>
        <item>nonSpeechDataFilter </item>
        <item>preemphasizer </item>
        <item>windower </item>
        <item>fft </item>
        <item>melFilterBank </item>
        <item>dct </item>
        <item>liveCMN </item>
        <item>featureExtraction </item>

        <item>wavWriter </item>
    </propertylist>
</component>
```

The signal processing filters applied on the recording audio signal are mentioned as follow:

1- preemphasizer >> Preemphasize filter

2- windower >> Raised Cosine Windower

3- fft >> Discrete Fourier Transform

4- melFilterBank >> MelFrequency Filter Bank

5- Dct >> Discrete Cosine Transform

6- liveCMN >> liveCMN feature

7- featureExtractor >> Deltas Feature Extractor

Then in sphinx 4 there are many data processing could be used as:

1- SpeechClassifier - classifies chunks of audio into speech and non-speech. It has the property 'threshold' to controls how sensitive the endpointer is. It is empirically determined that the value of 13 is optimal for most environments. A lower threshold will make the endpointer more sensitive, that is, mark more

audio as speech. A higher threshold will make the endpointer less sensitive, that is, mark less audio as speech.

2- SpeechMarker - marks the audio stream into speech and non-speech regions, giving some 'cushion areas' around these regions.

3- NonSpeechDataFilter - removes the non-speech regions from the audio.

4- LiveCMN: Subtracts the mean of all the input so far from the Data objects.

Obtaining a Front End

In order to obtain a front end, The Sphinx-4 front end is connected to the rest of the system via the scorer. We will show how the scorer will obtain the front end. In the configuration file, the scorer should be specified as follows:

```
<!-- ******************************************************* -->
<!-- TheScorer                                               -->
<!-- ******************************************************* -->
<component name="threadedScorer"
           type="edu.cmu.sphinx.decoder.scorer.ThreadedAcousticScorer">
    <property name="frontend" value="${frontend}"/>
</component>
```

4.3.5 Using Our Model

-- The Dictionary

To use our new dictionary we must update the configure file with the path of our dictionary and the path of the filler file also in the dictionary component.

-- The Language Model

In the training phase we build the Arabic language model, here we want to use it. So the link where to load the model must be changed to the path of the building one, as our system cover a large vocabulary so the trigram model is used here.

-- The Acoustic Model

The acoustic model used is the tiedstate Acoustic model and the sphinx3 loader, so to inform the sphinx system the new acoustic model location we must change the

sphinx3 loader manager location to our new location. See Appendix B for the complete configure file.

Then we can add the simple java file that contain our code, that test the model we build a java file that listen to the microphone and then convert the audio to text using the command prompt, the file must determine its configure file in the first.

The pseudo code of test file :

We have **cm** is an instant of the configuration manger, **A** is the configure xml file, **reg** is an instant of the recognizer class, and **mic** is an instant of the microphone class

we begin with…

- Set **cm** ← get source from (**A** xml)
- Set **reg** ← look up from **cm**
- Allocate the **reg**
- Test the **mic** for connectivity
    - If **mic** not start
        - Then ::
            - print "can not start microphone"
            - Deallocate the **reg**
            - Exit
        - Else ::
            - Loop recognition until the program exit
            - Print "Start Speaking. Press Ctrl-C to quit "
            - Result← get the recognized data
            - If Result not null
                - Then txt ← get best final result
                  Print "you said : " + txt
                - Else Print "I can not hear what you said.. "
            - End

See appendix c for source code.

Chapter 5 Experiments Results

5.1 Performance of Speech Recognition Systems

The performance of speech recognition systems is usually specified in terms of accuracy and speed. Accuracy may be measured in terms of performance accuracy which is usually rated with word error rate (WER), whereas speed is measured with the real time factor. Other measures of accuracy include Single Word Error Rate (SWER) and Command Success Rate (CSR).

WER is a common metric of the performance of a speech recognition or machine translation system. The general difficulty of measuring performance lies in the fact that the recognized word sequence can have a different length from the reference word sequence (supposedly the correct one). The WER is working at the word level instead of the phoneme level. This problem is solved by first aligning the recognized word sequence with the reference (spoken) word sequence using dynamic string alignment. Word error rate can then be computed as:

$$WER = \frac{S + D + I}{N}$$

where

S is the number of substitutions,

D is the number of the deletions,

I is the number of the insertions,

N is the number of words in the reference.

When reporting the performance of a speech recognition system, sometimes **word recognition rate (WRR)** is used instead:

$$WRR = 1 - WER = \frac{N - S - D - I}{N} = \frac{H - I}{N}$$

Where *H* is N-(S+D), the number of correctly recognized words.

5.2 Our Experiments

In this section, we will discuss the experiments that done on the proposed system.

5.2.1 Experiment one

In this experiment we test the system using 144 sentences, divided to 10 groups, each group for one speaker, so we have ten speakers [4 female and 6 male]. The speech data recorded from Aljazeera news channel. In order to use a wave file, the code of testing application must be changed to be able to read from existing file. And must process that wave file to be compatible with our system, so every file must pass the checking channel to test its format, if it suitable to one used in system then the wave file just enter the system to be recognized, and if the file has different format type, then it must be enter convert channel to be convert to the used format then enter the system to be recognized Figure 5.1 shows the process of checking the wave file.

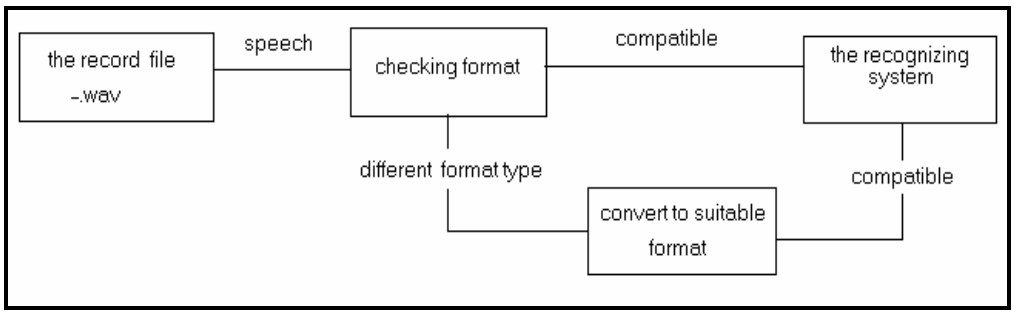


Figure 5.1 The process of checking the wave file

The compatible formats of the speech file are mentioned in Table 5.1, see appendix C for source code of using existing speech file.

Table 5.1 : Proper file format

File Type	.wav
Sample Rate	16000Hz
Sample Size in bits	16
Recording Type	Mono
Signed	True
Character type	Little-Endian

The detailed of this experiment speech information are summarized in Table 5.2.

Table 5.2 : The speech information in experiment one

# of Speakers	**# of Sentences**	**Duration in sec**	**Ratio of Test. Data (%)**
Speaker 1	3	12	2.14
Speaker2	34	169	30.12
Speaker3	8	28	4.99
Speaker4	6	29	5.17
Speaker5	22	71	12.65
Speaker6	23	76	13.55
Speaker7	26	100	17.83
Speaker8	7	30	5.35
Speaker9	3	15	2.67
Speaker10	10	31	5.53

5.2.2 Experiment Two

In this experiment we test the system using 10 sentences, with two speakers [male]. Here we use the microphone to enter the audio data directly to the recognizer, the configuration needed to do that was mentioned in the testing phase Section 4.2.4 When using the microphone in testing we must take care of the parameters that listed in Table 5.1. The details of the speech information used in this experiment are summarized in Table 5.3:

Table 5.3 : The speech information in experiment two

# of Speakers	**# of Sentences**	**Duration in sec**
Speaker 1	10	75
Speaker2	10	75
Total	10 each one	150≈2.5min

5.3 The Discussion

Using the WER from Section 5.1, the first experiment we get the result listed in Table 5.4.

Table 5.4 : The result of experiment one

# of Speakers	**Accuracy (%)**	**WER (%)**
Speaker 1	90.8	9.2
Speaker2	92.3	7.7
Speaker3	92.4	7.6
Speaker4	91.4	8.6
Speaker5	91.7	8.3
Speaker6	90.2	9.8
Speaker7	91.1	8.9
Speaker8	92.2	7.8
Speaker9	91.5	8.5
Speaker10	91.8	8.2
Total	91.6	8.4

The second experiment gives us this result that mentioned in Table 5.5.

Table 5.5 : The result of experiment two

# of Speakers	**Accuracy (%)**	**WER (%)**
Speaker 1	90.4	9.6
Speaker2	90	10
Total	90.2	9.8

At last we can summarized that our proposed Arabic speech recognition system used the rich and balanced text corpus was testing with sphinx-4 framework with the parameters absolute word beam width equal 20 and word insertion probability equal 0.7 and the language weight set to 0.7, the system gives an accuracy rate near to 90.2 % and the WER rate in general got equals to 9.8% using the microphone and an accuracy rate

near to 91.6 % and the WER rate in general got equals to 8.4% using previous recording audio file. For systems that depend on medium and small size vocabulary the system performance obtained WER ranged between 15% to 0% dedicated to standard Arabic, as we used a large size vocabulary, our system performance obtained a good WER [8.4%, 9.8%]. In Table 5.6 we summarize the accuracy rate and the WER rate of previous Arabic systems cover a large size vocabulary and our system values.

Table 5.6 : Comparison our system to others

The compared systems	**Accuracy Rate (%)**	**WER Rate (%)**
System-1	90.5	9.5
System-2	92.8	7.2
Our System	91.6 [recorded] 90.2 [mic]	8.4 9.8

System-1 is from "Investigation Arabic Speech Recognition Using CMU Sphinx System" paper[24], which depended on an in-house corpus was created from all 10 Arabic digits. A number of 6 Moroccan speakers (6 males) were asked to utter all digits 5 times. In order to evaluate the performances of the application, they performed some experiments on different individuals (three men) each one of them was asked to utter 10 Arabic digits. they recorded the number of words that were correctly recognized, and then a mean recognition ratio for all tester was calculated as 90.5%.

System-2 is from the paper with the title " speaker-independent natural Arabic speech recognition system " for Moustafa Elshafei, Husni Al-Muhtaseb, and Mansour Al-Ghamdi they built a system with in housing audio files recorded from several TV news channels at a sampling rate of 16 ksps. A total of 249 news stories, summing up to 5.4 hours of speech, and they got the correctly recognized words was 92.8%, and WER came down to 7.2 %.

Comparing it with our result, we got a good accuracy as we used the rich and balanced corpus. To increase the accuracy of the system, we need more training.

Chapter 6 Conclusion and Future Work

6.1 Conclusion

To conclude, a spoken Arabic recognition system was designed to investigate the process of automatic speech recognition using Arabic environment. Both training and recognizing process use Arabic characters. Our experiment demonstrates the possible adaptability of the CMU Sphinx4 to Arabic language with large vocabulary size. The system consisted of three basic components, which were an Arabic phonetic dictionary contained all phonetic sound of the words used in training, and the Arabic statistical language model that gave us the probability of the sequence of words. The last component was the acoustic model that generated the HMM represented unit for each phoneme. We used a phonetically rich and balanced speech corpus for training the recognizer.

The developed system providing a good accuracy with speaker independent, and natural Arabic continuous speech as it got 91.6% accuracy rate with recorded files and 90.2% accuracy rate using the microphone. The recognition results produced by our system showed to be satisfactory and when compared, they can match with the results of other ASR systems.

6.2 Future Work

The work reported in this thesis may be extended in a number of ways, some of which are discussed below:

- extending the corpus to 10 hours of Arabic speech.
- enhancing the rule based phonetic dictionary, and parameterization of the acoustic model.
- Build a Arabic Dictation Notepad Software that can be used.
- Using the Arabic dictionary in the mobile systems.

References

[1] Haton M., Cerisara C., Fohr D., Laprie Y., and Smaili K., *Reconnaissance Automatique de la Parole du Signal a Son Interpretation*, Monographies and Books, Oxford, 2006.

[2] Young S., "The HTK Hidden Markov Model Toolkit: Design and Philosophy," Technical Report TR 152, Department of Engineering, Cambridge University, Cambridge, 1994

[3] Deshmukh N., Ganapathiraju A., Hamaker J., Picone J., and Ordowski M., "A Public Domain Speech to Text System," *in Proceedings of 6th European Conferences on Speech Communication and Technology*, Hungary, pp. 2127-2130, 1999.

[4] Muhammad A., "Alaswaat Alaghawaiyah," *in Proceedings of International Conference on Signal Processing*, Jordan, pp. 646-651, 1990.

[5] Li X., Zhao Y., Pi X., Liang H., and Nefian V., "Audio Visual Continuous Speech Recognition Using a Coupled Hidden Markov Model," *in Proceedings of 7th International Conferences on Spoken Language Processing*, Denver, pp. 213-216, 2002.

[6] Huang X., Acero A., and Hon H., "Spoken Language Processing: A Guide to Theory", *Algorithm and System Design*, Prentice Hall, 2001.

[7] Gordon R., *Ethnologue: Languages of the World*, Texas: Dallas, SIL International, 2005.

[8] Alghamdi M., Elshafei M., and Al-Muhtaseb H., "Arabic Broadcast News Transcription System," *International Computer Journal of Speech Technology*, vol. 10, no. 4, pp. 183-195, 2009

[9] Hyassat H. and Abu Zitar R., "Arabic Speech Recognition Using SPHINX Engine," *International Computer Journal of Speech Technology*, vol. 9, no. 3-4, pp. 133-150, 2008.

[10] Kirchhoff K., Bilmes J., Das S., Duta N., Egan M., Ji G., He F., Henderson J., Liu D., Noamany M., Schone P., Schwartz R., and Vergyri D., "Novel

Approaches to Arabic Speech Recognition: Report from the 2002 Johns- Hopkins Summer Workshop," *in Proceedings of IEEE International Conference on Acoustics, Speech, and Single processing*, Hong Kong, vol. 1, pp. 344-347, 2003

[11] H., Harti M., and Chenfour N., "Arabic Speech Recognition System Based on CMUSphinx," *in Proceedings of IEEE International Symposium on Computational Intelligence and Intelligent Informatics*, Morocco, pp. 31-35, 2007.

[12] Alsulaiti L. and Atwell E., "The Design of a Corpus of Contemporary Arabic," *International Computer Journal of Corpus Linguistics*, John Benjamins Publishing Company, pp. 1-36, 2006.

[13] Alansary S., Nagi M., and Adly N., "Building an International Corpus of Arabic Progress of Compilation Stage," *in Proceedings of 8th International Conference on Language Engineering*, Egypt, pp. 337-344, 2007.

[14] Parkinson D. and Farwaneh S., *Perspectives on Arabic Linguistics XV*, John Benjamins Publishing Company, Philadelphia, 2003.

[15] Black A. and Tokuda K., "The Blizzard Challenge Evaluating Corpus-Based Speech Synthesis on Common Datasets," *in Proceeding of Interspeech*, Portugal, pp. 77-80, 2005.

[16] D'Arcy S. and Russell M., "Experiments with the ABI (Accents of the British Isles) Speech Corpus," *in Proceeding of Interspeech 08*, Australia, pp. 293-296, 2008.

[17] Garofolo J., Lamel L., Fisher W., Fiscus J., Pallett D., Dahlgren N., and Zue V., "TIMIT Acoustic-Phonetic Continuous Speech Corpus," *Technical Document*, Philadelphia, 1993.

[18] Chou F. and Tseng C., "The Design of Prosodically Oriented Mandarin Speech Database," *in Proceedings of International Congress of Phonetics Sciences*, San Francisco, pp. 2375-2377, 1999.

[19] Sagisaka Y., Takeda K., Abel M., Katagiri S., Umeda T., and Kuwabara H., "A Large-Scale Japanese Speech Database," *in Proceedings of International Conference on Spoken Language Processing*, 1990.

[20] Alotaibi Y., "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.

[21] Alotaibi Y., Alghamdi M., and Alotaiby F., “Using a Telephony Saudi Accented Arabic Corpus in Automatic Recognition of Spoken Arabic Digits,” *in Proceedings of 4th International Symposium on Image/Video Communications over Fixed and Mobile Networks*, Spain, pp. 43-60, 2008.

[22] Ahmed Omar, “Study of Linguistic phonetics,” *Aalam Alkutob*, Eygpt 1991. (in Arabic)

[23] El-Imam, “An Unrestricted Vocabulary Arabic Speech Synthesis System”, *IEEE Transactions on Acoustic, Speech, and Signal Processing*, Vol. 37,No. 12, Dec. 1989, pp.1829-45.

[24] Hassan S., Hussein H., Mostafa H. and Noureddine C., " Investigation Arabic Speech Recognition Using CMU Sphinx System", *The International Arab Journal of Information Technology*, Vol. 6, No. 2, April 2009

[25] Rabiner, and Juang, "Fundamentals of Speech Recognition." Prentice Hall, 1993.

[26] Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[27] HTK speech recognition tool kit. http://htk.eng.cam.ac.uk/ (accessed July, 2012).

[28] Sphinx-4 Java-based Speech Recognition Engine, http://cmusphinx.sourceforge.net/sphinx4/ (accessed July, 2012).

[29] Huang, Alleva, Hon, Hwang, and Rosenfeld, “The SPHINX-II speech recognition system: an overview,” *Computer Speech and Language, vol*. 7, no. 2, pp. 137–148, 1993

[30] Lamere, Kwok, Walker, Gouvea, Singh, Raj, and Wolf, “Design of the CMU Sphinx-4 decoder,” in *Proceedings of the 8th European Conference on Speech Communication and Technology*, Geneve, Switzerland, Sept. 2003, pp. 1181–1184

[31] Algamdi M., *Arabic Phonetics*, Attaoobah, Riyadh, 2000.

[32] Algamdi M., "KACST Arabic Phonetics Database", *The Fifteenth International Congress of Phonetics Science*, Barcelona, 3109-3112, 2003.

[33] Elshafei M., Al-Muhtaseb H. and Alghamdi M., "Speech Units for Arabic Text-to-speech", *The Fourth Workshop on Computer and Inforamtion Sciences*, 199-212, 2002.

[34] Elshafei M., Al-Muhtaseb H. and Alghamdi M., "Techniques for High Quality Text-to-speech", *Information Science*, 140 (3-4) 255-267, 2002

[35] Elshafei M., Al-Muhtaseb H. and Alghamdi M., "Statistical Methods for Automatic Diacritization of Arabic text", *Proceedings 18th National computer Conference NCC'18*, Riyadh, March 26-29, 2006.

[36] Al-Otaibi F., Speaker-Dependant Continuous Arabic Speech Recognition, M.Sc. Thesis, *King Saud University, 2001.*

[37] Billa, J.; Noamany, M.; Srivastava, A.; Liu, D.; Stone, R.; Xu, J.; Makhoul, J.; Kubala, F.," Audio indexing of Arabic broadcast news*", Proceedings. (ICASSP '02). IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2002. Volume 1, 2002 Page(s):I- 5 - I-8 vol.1

[38] Vergyri D. and Kirchhoff K., *Automatic Diacritization of Arabic for Acoustic Modelling in Speech Recognition,* Editors, Coling, Geneva, 2004.

[39] Azmi M. and Tolba H., "Syllable-Based Automatic Arabic Speech Recognition in Different Conditions of Noise," *IEEE Proceedings of the 9th International Conference on Signal Processing*, China, pp. 601-604, 2008.

[40] Nofal M., Abdel-Raheem E., El Henawy H., and Abdel Kader N., "Acoustic Training System for Speaker Independent Continuous Arabic Speech Recognition System," *in Proceedings of the 4th IEEE International Symposium on Signal Processing and Information Technology*, Italy, pp. 200-203, 2004.

[41] Alghamdi M., Alhamid A., and Aldasuqi M., "Database of Arabic Sounds: Sentences," Technical Report, *King Abdulaziz City of Science and Technology*, Saudi Arabia, 2003.

[42] Raja A., Roziati Z., Moustafa E., and Othman Kh., " Arabic Speaker-Independent Continuous Automatic Speech Recognition Based on a Phonetically Rich and Balanced Speech Corpus", *International Arab Journal Information Technology*, vol. 9(1): 84-93 (2012)

[43] "The Speech Recognition Information Source", http://www.sayican.com, (accessed July, 2012).

[44] Sun Microsystems, " Java™ Speech API Programmer's Guide "Version 1.0— October 26, 1998.

[45] M.A.Anusuya, S.K.Katti, "Speech Recognition by Machine:A Review", *(IJCSIS) International Journal of Computer Science and Information Security*, Vol. 6, No. 3, 2009

[46] Willie W.r, Paul L., Philip K., Bhiksha R.," Sphinx-4: A Flexible Open Source

Framework for Speech Recognition", Sun Microsystems, Inc., November 2004

[47] Top Internet Languages - Internet World Stats, http://www.internetworldstats.com/stats7.htm , 2010. (accessed July, 2012).

[48] Alotaibi1 Y., Hussain A.," Comparative Analysis of Arabic Vowels using Formants and an Automatic Speech Recognition System", *International Journal of Signal Processing, Image Processing and Pattern Recognition* Vol. 3, No. 2, June, 2010

[49] Ghania Droua-Hamdani, " Algerian Arabic Speech Database: Corpus Design and Automatic Speech Recognition Application", *The Arabian Journal for Science and Engineering*, Volume 35, Number 2C 157, 2010

[50] El-Imam Y., "An Unrestricted Vocabulary Arabic Speech Synthesis System", *IEEE Transactions on Acoustic, Speech, and Signal Processing*, 37(12)(1989), pp. 1829–1845.

[51] Youssef and Emam, "An Arabic TTS System on the IBM Trainable Synthesizer", *Le Traitement Automatique de l'Arabe*, JEP-TALN 2004, Fes, 19–21 avril 2004.

[52] Tsuhan Ch., "Audiovisual speech processing", *IEEE Signal Processing Magazine*, 18(1):9-21, 2001.

[53] Young, "The HTK hidden Markov model toolkit: Design and philosophy," *Cambridge University Engineering Department*, UK, Tech. Rep. CUED/F-INFENG/TR152, Sept. 1994.

[54] Deshmukh, Ganapathiraju, Hamaker, Picone, and Ordowski, "A public domain speech-to-text system," in *Proceedings of the 6th European Conference on Speech Communication and Technology*, vol. 5, Budapest, Hungary, Sept. 1999, pp. 2127–2130.

[55] Li, Zhao, Pi, Liang, and Nefian, "Audio-visual continuous speech recognition using a coupled hidden Markov model," in *Proceedings of the 7th International Conference on Spoken Language Processing*, Denver, CO, Sept. 2002, pp. 213–216.

[56] K. F. Lee, H. W. Hon, and R. Reddy, "An overview of the SPHINX speech recognition system," IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 38, no. 1, pp. 35–45, Jan. 1990.

[57] Ravishankar, "Efficient algorithms for speech recognition," PhD Thesis (CMU

Technical Report CS-96-143), Carnegie Mellon University, Pittsburgh, PA, 1996.

[58] Baker, "The Dragon system - an overview," in *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. 23, no. 1, Feb. 1975, pp. 24–29.

[59] Lowerre, "The Harpy speech recognition system," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, 1976.

[61] NIST. Speech recognition scoring package (score). [Online]. Available: http://www.nist.gov/speech/tools for Developers/JSGF/ (accessed July, 2012).

[62] Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 2, pp. 269–311, 1997.

[63] Clarkson and Rosenfeld, "Statistical language modeling using the CMU-Cambridge toolkit," in *Proceedings of the 5th European Conference on Speech Communication and Technology*, Rhodes, Greece, Sept. 1997.

[64] Carnegie Mellon University. CMU pronouncing dictionary. [Online]. Available: http://www.speech.cs.cmu.edu/cgi-bin/cmudict (accessed July, 2012).

[65] Pineda L., G.mez M., Vaufreydaz D., and Serignat J., "Experiments on the Construction of a Phonetically Balanced Corpus from the Web," *in Proceedings of 5th International Conference on Computational Linguistics and Intelligent Text Processing, Lecture Notes in Computer Science*, Korea, pp. 416-419, 2004.

[66] Yacine Y., Yekhlef M., Belkacem K., " Towards Quranic reader controlled by speech", *(IJACSA) International Journal of Advanced Computer Science and Applications*, Vol. 2, No. 11, 2011

[67] http://cmusphinx.sourceforge.net/sphinx4. (accessed July, 2012).

[68] Sun Microsystems. Available: http://java.sun.com. (accessed July, 2012).

[69] http://ant.apache.org/ . (accessed July, 2012).

[70] Zavaliagkos, 1998. The BBN byblos 1997 large vocabulary conversational speech recognition system. Proc. ICASSP.

[70] Vergyri D., Katrin K., Kevin D. and Stolcke A., "Morphology-based language modeling for Arabic speech recognition". Proc. *ICSLP*, Jeju, South Korea, 2004.

Appendix A

Building The Language Model

Installing the Toolkit

At first the variable *BYTESWAP_FLAG* will need to be set in the Makefile. This can be done by editing src/Makefile directly, so that the line

#BYTESWAP_FLAG = -DSLM_SWAP_BYTES

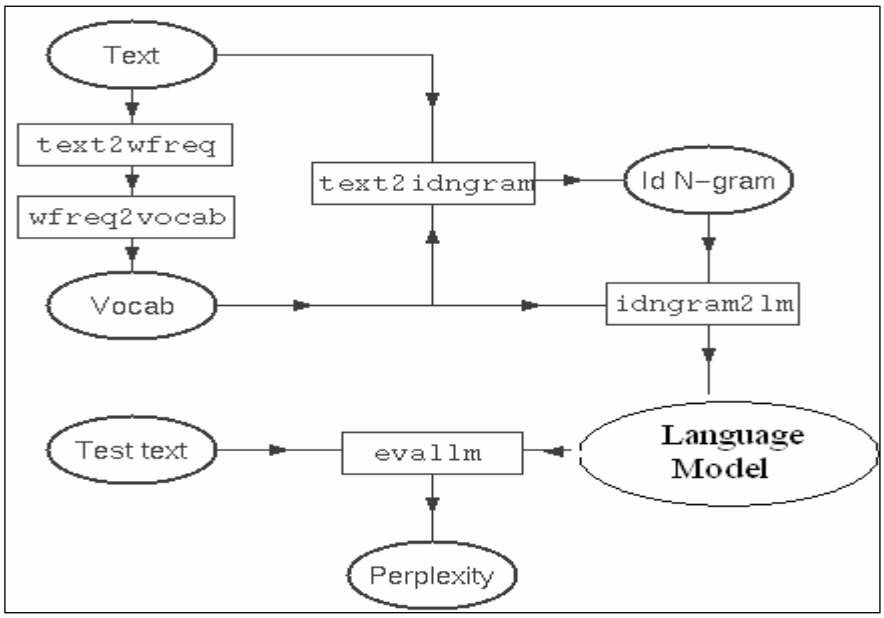
is changed to

BYTESWAP_FLAG = -DSLM_SWAP_BYTES

Then the installation procedure is simply to change into the src/ directory and type:

make install The executables will then be copied into the bin/ directory, and the library file SLM2.a will be copied into the lib/ directory.

Typical Usage



The Tools

text2wfreq

Input : Text stream

Output : List of every word which occurred in the text, along with its number of occurrences.

Command Line Syntax:

text2wfreq [-hash 1000000]
[-verbosity 2]
< .text > .wfreq

Higher values for the -hash parameter require more memory, but can reduce computation time.

wfreq2vocab

Input : A word unigram file, as produced by text2wfreq

Output : A vocabulary file.

Command Line Syntax:

wfreq2vocab [-top 20000 | -gt 10]
[-records 1000000]
[-verbosity 2]
< .wfreq > .vocab

The -top parameter allows the user to specify the size of the vocabulary; if the program is called with the command -top 20000, then the vocabulary will consist of the most common 20,000 words.

The -gt parameter allows the user to specify the number of times that a word must occur to be included in the vocabulary; if the program is called with the command -gt 10, then the vocabulary will consist of all the words which occurred more than 10 times.

The -records parameter allows the user to specify how many of the word and count records to allocate memory for. If the number of words in the input exceeds this number, then the program will fail, but a high number will obviously result in a higher memory requirement.

text2wngram

Input : Text stream

Output : List of every word n-gram which occurred in the text, along with its number of occurrences.

Command Line Syntax:

text2wngram [-n 3]
[-temp /usr/tmp/]
[-chars n]
[-words m]
[-gzip | -compress]
[-verbosity 2]
< .text > .wngram

The maximum numbers of charactors and words that can be stored in the buffer are given by the -chars and -words options. The default number of characters and words are chosen so that the memory requirement of the program is approximately that of STD_MEM, and the number of charactors is seven times greater than the number of words.

The -temp option allows the user to specify where the program should store its temporary files.

text2idngram

Input : Text stream, plus a vocabulary file.

Output : List of every id n-gram which occurred in the text, along with its number of occurrences.

Notes : Maps each word in the text stream to a short integer as soon as it has been read, thus enabling more n-grams to be stored and sorted in memory.

Command Line Syntax:

text2idngram -vocab .vocab
[-buffer 100]
[-temp /usr/tmp/]
[-files 20]
[-gzip | -compress]
[-n 3]
[-write_ascii]
[-fof_size 10]
[-verbosity 2]
< .text > .idngram

By default, the id n-gram file is written out as binary file, unless the -write_ascii switch is used.

The size of the buffer which is used to store the n-grams can be specified using the -buffer parameter. This value is in megabytes, and the default value can be changed from 100 by changing the value of STD_MEM in the file src/toolkit.h before compiling the toolkit.

The program will also report the frequency of frequency of n-grams, and the corresponding recommended value for the -spec_num parameters of idngram2lm. The -fof_size parameter allows the user to specify the length of this list. A value of 0 will result in no list being displayed.

The -temp option allows the user to specify where the program should store its temporary files.

In the case of really huge quantities of data, it may be the case that more temporary files are generated than can be opened at one time by the filing system. In this case, the temporary files will be merged in chunks, and the -files parameter can be used to specify how many files are allowed to be open at one time.

idngram2lm

Input : An id n-gram file (in either binary (by default) or ASCII (if specified) format), a vocabulary file, and (optionally) a context cues file. Additional command line parameters will specify the cutoffs, the discounting strategy and parameters, etc.

Output : A language model, in either binary format (to be read by evallm), or in ARPA format.

Command Line Syntax:

idngram2lm -idngram .idngram
-vocab .vocab
-arpa .arpa | -binary .binlm
[-context .ccs]
[-calc_mem | -buffer 100 | -spec_num y ... z]
[-vocab_type 1]
[-oov_fraction 0.5]
[-linear | -absolute | -good_turing | -witten_bell]
[-disc_ranges 1 7 7]
[-cutoffs 0 ... 0]
[-min_unicount 0]
[-zeroton_fraction 1.0]
[-ascii_input | -bin_input]
[-n 3]
[-verbosity 2]
[-four_byte_counts]

[-two_byte_bo_weights
[-min_bo_weight -3.2] [-max_bo_weight 2.5]
[-out_of_range_bo_weights 10000]]

The -context parameter allows the user to specify a file containing a list of words within the vocabulary which will serve as context cues (for example, markers which indicate the beginnings of sentences and paragraphs).

-calc_mem, -buffer and -spec_num x y ... z are options to dictate how it is decided how much memory should be allocated for the n-gram counts data structure. -calc_mem demands that the id n-gram file should be read twice, so that we can accurately calculate the amount of memory required. -buffer allows the user to specify an amount of memory to grab, and divides this memory equally between the 2,3, ..., n-gram tables. -spec_num allows the user to specify exactly how many 2-grams, 3-grams, ... , and n-grams will need to be stored. The default is -buffer STD_MEM.

The toolkit provides for three types of vocabulary, which each handle out-of-vocabulary (OOV) words in different ways, and which are specified using the -vocab_type flag.

A *closed vocabulary* (-vocab_type 0) model does not make any provision for OOVs. Any such words which appear in either the training or test data will cause an error. This type of model might be used in a command/control environment where the vocabulary is restricted to the number of commands that the system understands, and we can therefore guarantee that no OOVs will occur in the training or test data.

An *open vocabulary* model allows for OOVs to occur; out of vocabulary words are all mapped to the same symbol. Two types of open vocabulary model are implemented in the toolkit. The first type (-vocab_type 1) treats this symbol the same way as any other word in the vocabulary. The second type (-vocab_type 2) of open vocabulary model is to cover situations where no OOVs occurred in the training data, but we wish to allow for the situation where they could occur in the test data. This situation could occur, for example, if we have a limited amount of training data, and we choose a vocabulary which provides 100% coverage of the training set. In this case, an arbitrary proportion of the discount probability mass (specified by the -oov_fraction option) is reserved for OOV words.

The discounting strategy and its parameters are specified by the -linear, -absolute, -good_turing and -witten_bell options. With Good Turing discounting, one can also specify the range over which discounting occurs, using the -disc_ranges option.

The user can specify the cutoffs for the 2-grams, 3-grams, ..., n-grams by using the -cutoffs parameter. A cutoff of *K* means that > n-grams occurring *K* or fewer times are discarded. If the parameter is omitted, then all the cutoffs are set to zero.

The -zeroton_fraction option specifies that P(zeroton) (the unigram probability assigned to a vocabulary word that did not occurred at all in the training data) will be at least that fraction of P(singleton) (the probability assigned to a vocabulary word that occurred exactly once in the training data).

By default, the n-gram counts are stored in two bytes by use of a count table (this allows the counts to exceed 65535, while keeping the data structures used to store the model compact). However, if more than 65535 **distinct** counts need to be stored (very unlikely, unless constructing 4-gram or higher language models using Good-Turing discounting), the -four_byte_counts option will need to be used.

The floating point values of the back-off weights may be stored as two-byte integers, by using the -two_byte_alphas switch. This will introduce slight rounding errors, and so should only be used if memory is short. The -min_alpha, -max_alpha and -out_of_range_alphas are parameters used by the functions for using two-byte alphas. Their values should only be altered if the program instructs it. For further details, see the comments in the source file src/two_byte_alphas.c.

binlm2arpa

Input : A binary format language model, as generated by idngram2lm.

Output : An ARPA format language model.

Command Line Syntax:

binlm2arpa -binary .binlm
    -arpa .arpa

[-verbosity 2]

evallm

Input : A binary or ARPA format language model, as generated by idngram2lm. In addition, one may also specify a text stream to be used to compute the perplexity of the language model. The ARPA format language model does not contain information as to which words are context cues, so if an ARPA format lanaguage model is used, then a context cues file may be specified as well.

Output : The program can run in one of two modes.

- compute-PP - Output is the perplexity of the language model with respect to the input text stream.
- validate - Output is confirmation or denial that the sum of the probabilities of each of the words in the context supplied by the user sums to one.

Command Line Syntax:

evallm [-binary .binlm |
-arpa .arpa [-context .ccs]]

Notes: evallm can receive and process commands interactively. When it is run, it loads the language model specified at the command line, and waits for instructions from the user. The user may specify one of the following commands:

Appendix B

The XML Configuration File

The Configure Xml file, which used in the test application.

```
<?xml version="1.0" encoding="UTF-8"?>


<!--
   Sphinx-4 Configuration file
-->

<!-- ********************************************************* -->
<!--  an4 configuration file                                   -->
<!-- ********************************************************* -->

<config>

    <!-- ********************************************************* -->
    <!-- frequently tuned properties                               -->
    <!-- ********************************************************* -->

    <property name="logLevel" value="WARNING"/>

    <property name="sampleRate" value="16000"/>
     <property name="relativeBeamWidth" value="1E-60"/>
     <property name="absoluteWordBeamWidth" value="22"/>
     <property name="relativeWordBeamWidth" value="1E-30"/>
     <property name="wordInsertionProbability" value=".2"/>
     <property name="languageWeight" value="10.5"/>
     <property name="silenceInsertionProbability" value=".1"/>
     <property name="acousticLookahead" value="1.7"/>
     <property name="frontend" value="epFrontEnd"/>
     <property name="recognizer" value="recognizer"/>
     <property name="showCreations" value="false"/>

     <property name="sampleRate" value="16000"/>
    <!-- ********************************************************* -->
    <!-- word recognizer configuration                             -->
    <!-- ********************************************************* -->

    <component name="recognizer"
type="edu.cmu.sphinx.recognizer.Recognizer">
        <property name="decoder" value="decoder"/>
        <propertylist name="monitors">
            <item>accuracyTracker </item>
            <item>speedTracker </item>
            <item>memoryTracker </item>
        </propertylist>
    </component>

    <!-- ********************************************************* -->
    <!-- The Decoder   configuration                               -->
    <!-- ********************************************************* -->
```

```
    <component name="decoder" type="edu.cmu.sphinx.decoder.Decoder">
        <property name="searchManager"
value="wordPruningSearchManager"/>
        <property name="featureBlockSize" value="50"/>
    </component>

    <!-- ******************************************************** -->
    <!-- The Search Manager                                       -->
    <!-- ******************************************************** -->

    <component name="wordPruningSearchManager"

type="edu.cmu.sphinx.decoder.search.WordPruningBreadthFirstSearchManag
er">
        <property name="logMath" value="logMath"/>
        <property name="linguist" value="lexTreeLinguist"/>
        <property name="pruner" value="trivialPruner"/>
        <property name="scorer" value="threadedScorer"/>
        <property name="activeListManager" value="activeListManager"/>
        <property name="growSkipInterval" value="0"/>
        <property name="checkStateOrder" value="false"/>
        <property name="buildWordLattice" value="false"/>
        <property name="acousticLookaheadFrames" value="1.7"/>
        <property name="relativeBeamWidth"
value="${relativeBeamWidth}"/>
    </component>



    <component name="activeListManager"

type="edu.cmu.sphinx.decoder.search.SimpleActiveListManager">
        <propertylist name="activeListFactories">
          <item>standardActiveListFactory</item>
          <item>wordActiveListFactory</item>
          <item>wordActiveListFactory</item>
          <item>standardActiveListFactory</item>
          <item>standardActiveListFactory</item>
          <item>standardActiveListFactory</item>
      </propertylist>
    </component>
<component name="standardActiveListFactory"

type="edu.cmu.sphinx.decoder.search.PartitionActiveListFactory">
        <property name="logMath" value="logMath"/>
        <property name="absoluteBeamWidth"
value="${absoluteBeamWidth}"/>
        <property name="relativeBeamWidth"
value="${relativeBeamWidth}"/>
    </component>

    <component name="wordActiveListFactory"

type="edu.cmu.sphinx.decoder.search.PartitionActiveListFactory">
        <property name="logMath" value="logMath"/>
        <property name="absoluteBeamWidth"
value="${absoluteWordBeamWidth}"/>
        <property name="relativeBeamWidth"
value="${relativeWordBeamWidth}"/>
    </component>
```

```
    <!-- ********************************************************** -->
    <!-- The Pruner                                                 -->
    <!-- ********************************************************** -->
    <component name="trivialPruner"
               type="edu.cmu.sphinx.decoder.pruner.SimplePruner"/>

    <!-- ********************************************************** -->
    <!-- TheScorer                                                  -->
    <!-- ********************************************************** -->
    <component name="threadedScorer"

type="edu.cmu.sphinx.decoder.scorer.ThreadedAcousticScorer">
        <property name="frontend" value="${frontend}"/>
    </component>
<!-- ********************************************************** -->
    <!-- The linguist  configuration                                -->
    <!-- ********************************************************** -->

    <component name="lexTreeLinguist"

type="edu.cmu.sphinx.linguist.lextree.LexTreeLinguist">
        <property name="logMath" value="logMath"/>
        <property name="acousticModel" value="wsj"/>
        <property name="languageModel" value="trigramModel"/>
        <property name="dictionary" value="dictionary"/>
        <property name="addFillerWords" value="false"/>
        <property name="fillerInsertionProbability" value="1E-10"/>
        <property name="generateUnitStates" value="false"/>
        <property name="wantUnigramSmear" value="true"/>
        <property name="unigramSmearWeight" value="1"/>
        <property name="wordInsertionProbability"
                value="${wordInsertionProbability}"/>
        <property name="silenceInsertionProbability"
                value="${silenceInsertionProbability}"/>
        <property name="languageWeight" value="${languageWeight}"/>
        <property name="unitManager" value="unitManager"/>
    </component>


    <!-- ********************************************************** -->
    <!-- The Dictionary configuration                             -->
    <!-- ********************************************************** -->
    <component name="dictionary"
        type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
        <property name="dictionaryPath"
                  value="file:///D:/OurASR/dict.6d"/>
        <property name="fillerPath"
                value="file:///D:/AASR/OurASR.filler"/>
        <property name="addSilEndingPronunciation" value="false"/>
        <property name="wordReplacement" value="&lt;sil&gt;"/>
        <property name="allowMissingWords" value="false"/>
        <property name="unitManager" value="unitManager"/>
    </component>


    <!-- ********************************************************** -->
    <!-- The Language Model configuration                           -->
    <!-- ********************************************************** -->
    <component name="trigramModel"

type="edu.cmu.sphinx.linguist.language.ngram.SimpleNGramModel">
```

```
        <property name="location"
            value="file:///D:/AASR/OurASR.lm"/>
        <property name="logMath" value="logMath"/>
        <property name="dictionary" value="dictionary"/>
        <property name="maxDepth" value="3"/>
        <property name="unigramWeight" value=".7"/>
    </component>


    <!-- ********************************************************** -->
    <!-- The acoustic model configuration                           -->
    <!-- ********************************************************** -->
    <component name="wsj"

type="edu.cmu.sphinx.linguist.acoustic.tiedstate.TiedStateAcousticMode
l">
        <property name="loader" value="wsjLoader"/>
        <property name="unitManager" value="unitManager"/>
    </component>

    <component name="wsjLoader"
type="edu.cmu.sphinx.linguist.acoustic.tiedstate.Sphinx3Loader">
        <property name="logMath" value="logMath"/>
        <property name="unitManager" value="unitManager"/>
        <property name="location" value="file:///D:/AASR/OurASR-
model"/>

<property name="properties_file" value="am.props"/>
    <property name="FeatureVectorLength" value="39"/>

    </component>

    <!-- ********************************************************** -->
    <!-- The unit manager configuration                             -->
    <!-- ********************************************************** -->

    <component name="unitManager"
        type="edu.cmu.sphinx.linguist.acoustic.UnitManager"/>

    <!-- ********************************************************** -->
    <!-- The frontend configuration                                 -->
    <!-- ********************************************************** -->

    <component name="frontEnd"
type="edu.cmu.sphinx.frontend.FrontEnd">
        <propertylist name="pipeline">
            <item>microphone </item>
            <item>preemphasizer </item>
            <item>windower </item>
            <item>fft </item>
            <item>melFilterBank </item>
            <item>dct </item>
            <item>liveCMN </item>
            <item>featureExtraction </item>
        </propertylist>
    </component>

    <!-- ********************************************************** -->
    <!-- The live frontend configuration                            -->
    <!-- ********************************************************** -->
```

```
    <component name="epFrontEnd"
type="edu.cmu.sphinx.frontend.FrontEnd">
        <propertylist name="pipeline">
            <item>microphone </item>
            <item>dataBlocker </item>
            <item>speechClassifier </item>
            <item>speechMarker </item>
            <item>nonSpeechDataFilter </item>
            <item>preemphasizer </item>
            <item>windower </item>
            <item>fft </item>
            <item>melFilterBank </item>
            <item>dct </item>
            <item>liveCMN </item>
            <item>featureExtraction </item>

            <item>wavWriter </item>
        </propertylist>
    </component>

    <!-- ******************************************************* -->
    <!-- The frontend pipelines                                  -->
    <!-- ******************************************************* -->
    <component name="featureTransform"
type="edu.cmu.sphinx.frontend.feature.FeatureTransform">
      <property name="loader" value="wsjLoader"/>
  </component>

 <component name="audioFileDataSource"
type="edu.cmu.sphinx.frontend.util.AudioFileDataSource"/>
    <component name="dataBlocker"
type="edu.cmu.sphinx.frontend.DataBlocker">
        <!--<property name="blockSizeMs" value="10"/>-->
    </component>

    <component name="speechClassifier"

type="edu.cmu.sphinx.frontend.endpoint.SpeechClassifier">
        <property name="threshold" value="15"/>
    </component>

    <component name="nonSpeechDataFilter"

type="edu.cmu.sphinx.frontend.endpoint.NonSpeechDataFilter"/>

    <component name="speechMarker"
               type="edu.cmu.sphinx.frontend.endpoint.SpeechMarker" >
        <property name="speechTrailer" value="50"/>
        <property name="startSpeech"   value="200">
        <property name="speechLeader" value="50"/>
        <property name="endSilence" value="500"/>
</property>
    </component>


    <component name="preemphasizer"
               type="edu.cmu.sphinx.frontend.filter.Preemphasizer"/>

    <component name="windower"

type="edu.cmu.sphinx.frontend.window.RaisedCosineWindower">
```

```
    </component>

    <component name="fft"

type="edu.cmu.sphinx.frontend.transform.DiscreteFourierTransform">
    </component>

    <component name="melFilterBank"

type="edu.cmu.sphinx.frontend.frequencywarp.MelFrequencyFilterBank">
    <property name="numberFilters" value="31"/>
    <property name="minimumFrequency" value="200"/>
    <property name="maximumFrequency" value="3500"/>
    </component>

    <component name="dct"

type="edu.cmu.sphinx.frontend.transform.DiscreteCosineTransform"/>

    <component name="liveCMN"
               type="edu.cmu.sphinx.frontend.feature.LiveCMN"/>

    <component name="featureExtraction"

type="edu.cmu.sphinx.frontend.feature.DeltasFeatureExtractor"/>

    <component name="microphone"
               type="edu.cmu.sphinx.frontend.util.Microphone">
        <property name="closeBetweenUtterances" value="false"/>
    </component>


      <component name="wavWriter"
type="edu.cmu.sphinx.frontend.util.WavWriter">
                  <property name="outFilePattern" value="test-
seg000000"/>
                  <property name="captureUtterances" value="true"/>
      </component>
    <!-- ******************************************************* -->
    <!--  monitors                                               -->
    <!-- ******************************************************* -->

    <component name="accuracyTracker"

type="edu.cmu.sphinx.instrumentation.BestPathAccuracyTracker">
        <property name="recognizer" value="${recognizer}"/>
        <property name="showAlignedResults" value="false"/>
        <property name="showRawResults" value="false"/>
    </component>

    <component name="memoryTracker"
               type="edu.cmu.sphinx.instrumentation.MemoryTracker">
        <property name="recognizer" value="${recognizer}"/>
      <property name="showSummary" value="false"/>
      <property name="showDetails" value="false"/>
    </component>

    <component name="speedTracker"
               type="edu.cmu.sphinx.instrumentation.SpeedTracker">
        <property name="recognizer" value="${recognizer}"/>
        <property name="frontend" value="${frontend}"/>
```

```
        <property name="showSummary" value="true"/>
        <property name="showDetails" value="false"/>
    </component>


    <!-- ********************************************************* -->
    <!--  Miscellaneous components                                 -->
    <!-- ********************************************************* -->

    <component name="logMath" type="edu.cmu.sphinx.util.LogMath">
        <property name="logBase" value="1.0001"/>
        <property name="useAddTable" value="true"/>
    </component>

</config>
```

Appendix C

The Java Source Code

The java source code for listening to the microphone, then convert the listening speech to its corresponding text in Arabic language using our Arabic model and sphinx-4 APIs.

```
import java.io.File;
import java.io.IOException;
import java.net.URL;
import edu.cmu.sphinx.frontend.util.Microphone;
import edu.cmu.sphinx.recognizer.Recognizer;
import edu.cmu.sphinx.result.Result;
import edu.cmu.sphinx.util.props.ConfigurationManager;

public class AA {

      public static void main(String[] args)
      {
            ConfigurationManager cm;

        if (args.length > 0) {
            cm = new ConfigurationManager(args[0]);
        } else {
            cm = new
ConfigurationManager(AA.class.getResource("AA.config.xml"));
        }

        Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
        recognizer.allocate();
  // start the microphone or exit if the program if this is not
possible
        Microphone microphone = (Microphone) cm.lookup("microphone");
        if (!microphone.startRecording()) {
            System.out.println("Cannot start microphone.");
            recognizer.deallocate();
            System.exit(1);
        }

        // loop the recognition until the program exits.
     while (true) {
       System.out.println("Start speaking. Press Ctrl-C to quit.\n");

            Result result = recognizer.recognize();

            if (result != null) {
                String resultText =
result.getBestFinalResultNoFiller();
                System.out.println("You said: " + resultText + '\n');
            } else {
                System.out.println("I can't hear what you said.\n");
            }

    }    }}
```

The java source code for enter the speech from existing audio file, then convert the file to adapting format then using our Arabic model and sphinx-4 APIs to display its corresponding text in Arabic language.

```
public boolean convertedFile = false;
public static void main(String[] args) {
try {
URL audioFileURL;
//if (args.length > 0) {
audioFileURL = new File(args[0]).toURI().toURL();
System.out.println(audioFileURL);
//} else {
//if the ${file_prompt} isn't in the program arguments, it'll go with
this:
//audioFileURL = test.class.getResource("1.wav");
//}
URL configURL = test.class.getResource("config.xml");
System.out.println("URL:"+ configURL);
System.out.println("Loading Recognizer…\n");
ConfigurationManager cm = new ConfigurationManager(configURL);
System.out.println("cm:"+cm);
Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
System.out.println("Recognise: "+recognizer);
/* allocate the resource necessary for the recognizer */
recognizer.allocate();
System.out.println("Recognizer : "+recognizer);
 System.out.println("Decoding " + audioFileURL.getFile());
System.out.println(AudioSystem.getAudioFileFormat(audioFileURL));
System.out.println("begin..... ");
StreamDataSource reader = (StreamDataSource)
cm.lookup("streamDataSource");
System.out.println("reader: "+reader);
AudioInputStream ais = AudioSystem.getAudioInputStream(audioFileURL);
System.out.println("ais: "+ais);
test wavFile = new test();
System.out.println(wavFile);
// Convert it to the proper format
AudioFormat targetFormat =
new AudioFormat(16000f,
16, // sample size in bits
1, // mono
true, // signed
false);
System.out.println(targetFormat);
//new AudioFormat(AudioFormat.Encoding.PCM_SIGNED, 16000, 16, 1, 2,
16000, false);
AudioInputStream convertedAis = wavFile.convertAudioInputStream(ais,
targetFormat);
File newFile = null;
if (wavFile.convertedFile)
{
newFile = wavFile.writeConvertedFile(convertedAis,
audioFileURL.toString());
audioFileURL = newFile.toURI().toURL();
ais = AudioSystem.getAudioInputStream(audioFileURL);
} /* set the stream data source to read from the audio file */
reader.setInputStream(ais, audioFileURL.getFile());
/* decode the audio file */
Result result = recognizer.recognize();
/* print out the results */
if (result != null) {
```

```
System.out.println("\nRESULT: " +
result.getBestFinalResultNoFiller() + "\n");
} else {
System.out.println("Result: null\n");
}
if (newFile != null)
newFile.delete();
} catch (IOException e) {
System.err.println("Problem when loading WavFile: " + e);
e.printStackTrace();
} catch (PropertyException e) {
System.err.println("Problem configuring WavFile: " + e);
e.printStackTrace();
}
// catch (InstantiationException e) {System.err.println("Problem
creating WavFile: ” + e); e.printStackTrace();}
catch (UnsupportedAudioFileException e) {
System.err.println("Audio file format not supported: " + e);
e.printStackTrace();
}
}
private AudioInputStream convertAudioInputStream(AudioInputStream
sourceAis, AudioFormat targetFormat) {
AudioFormat baseFormat = sourceAis.getFormat();
AudioFormat intermediateFormat;
AudioInputStream convertedAis = sourceAis;
// First convert the encoding, if necessary
if (!baseFormat.getEncoding().equals(targetFormat.getEncoding())) {
intermediateFormat = new AudioFormat(
targetFormat.getEncoding(),
baseFormat.getSampleRate(), baseFormat.getSampleSizeInBits(),
baseFormat.getChannels(),
baseFormat.getChannels() * 2, baseFormat.getSampleRate(),
false);
convertedAis = AudioSystem.getAudioInputStream(intermediateFormat,
sourceAis);
//this.writeConvertedFile(convertedAis, "C:\\encoding.wav”);
baseFormat = intermediateFormat;
sourceAis = convertedAis;
convertedFile = true;
}
// Then convert the sample rate
if (baseFormat.getSampleRate() != targetFormat.getSampleRate()) {
intermediateFormat = new AudioFormat(
baseFormat.getEncoding(),
targetFormat.getSampleRate(), baseFormat.getSampleSizeInBits(),
baseFormat.getChannels(),
baseFormat.getChannels() * 2, targetFormat.getSampleRate(),
false);
convertedAis = AudioSystem.getAudioInputStream(intermediateFormat,
sourceAis);
//this.writeConvertedFile(convertedAis, "C:\\sample.wav”);
baseFormat = intermediateFormat;
sourceAis = convertedAis;
convertedFile = true;
}
// Then convert the number of channels
if (baseFormat.getChannels() > targetFormat.getChannels()) {
intermediateFormat = new AudioFormat(
baseFormat.getEncoding(),
```

```
baseFormat.getSampleRate(), baseFormat.getSampleSizeInBits(),
targetFormat.getChannels(),
targetFormat.getChannels() * 2, baseFormat.getSampleRate(),
false);
convertedAis = AudioSystem.getAudioInputStream(intermediateFormat,
sourceAis);
//this.writeConvertedFile(convertedAis, "C:\\channels.wav”);
baseFormat = intermediateFormat;
sourceAis = convertedAis;
convertedFile = true;
}
return convertedAis;
}
private File writeConvertedFile(AudioInputStream sourceAis, String
fileName)
{
File tempfile = null;
fileName = "tempwavfile.wav";
//fileName = fileName.substring(6, fileName.length()-4) + "_new.wav";
try
{
//This just takes an audio stream, writes it to disk, then plays it
the way TALL usually does.
//it’s a test to see if the input stream is readable by the Java audio
providers like Tritonus
//System.out.println(fileName);
tempfile = new File(fileName);
AudioSystem.write(sourceAis, AudioFileFormat.Type.WAVE, tempfile);
}
catch (Exception e)
{
System.out.println(e);
}
return tempfile;
}
}
```