Digital Automatic Speech Recognition using Kaldi

By
Sarah Habeeb Alyousefi

Bachelor of Science
Computer and software Engineering
Al-Mustansiriya University
College of Engineering

A thesis submitted to the College of Engineering at
Florida Institute of Technology

in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

Melbourne, Florida
May 2018
We the undersigned committee hereby approve the attached thesis, “Digital Automatic Speech Recognition using Kaldi”

By
Sarah H Alyousefi

Veton Z. Këpuska, Ph.D.
Associate Professor
Electrical and Computer Engineering

Samuel P. Kozaitis, Ph.D.
Professor and Department Head
Electrical and Computer Engineering

Josko Zec, Ph.D.
Associate Professor
Electrical and Computer Engineering

Marius Silaghi, Ph.D.
Associate Professor
Computer Sciences and Cybersecurity
Abstract

Title: Digital Automatic Speech Recognition using Kaldi
Author: Sarah Habeeb Alyousefi
Advisor: Veton Z. Këpuska, Ph.D.

The Automatic speech recognition (ASR) system is one of most important technologies that used for human-machine interaction. The main goal of the ASR system is to recognize different natural languages that are spoken by humans. The difficulties of these recognition systems depend on many factors, such as noises, variability of the speakers, and problems of continuous speech. For that reason, many researchers and foundations have designed different kinds of licensed toolkits and software that are specialized in building speech recognition systems, including, Julius, Sphinx-4, RWTH ASR, and HTK toolkits.

In this thesis, Kaldi toolkit, which is one of the most notable speech recognition tools that is written in C++ and released under the Apache License v2.0, is used to build, train, and evaluate a digital ASR system. First, the speech recognition system has been explained in detail and built using the TIDIGITS corpus. Second, different training approaches including discriminative training methods) have been studied and applied to improve the accuracy of the speech recognition system.

The ASR system accuracy has been evaluated using two evolution metrics: the word error rate (WER) and the sentence error rate (SER). The overall obtained system performance is ranged from 99.05% to 99.55% depending on the training methods that have been applied.
# Table of Contents

List of Figures .............................................................................................................................. vii

List of Tables ................................................................................................................................. viii

Chapter One Introduction and Thesis Outline ........................................................................ 1
  1.1 Introduction ....................................................................................................................... 2
  1.2 Thesis Goal ....................................................................................................................... 3
  1.3 Thesis Outline ................................................................................................................... 3

Chapter Two Overview of Automatic Speech Recognition .................................................... 4
  2.1 Introduction ....................................................................................................................... 5
  2.2 The History of Automatic Speech Recognition Systems ............................................ 5
  2.3 The Architecture of Automatic Speaker Recognition System ............................... 8
    2.3.1 Preprocessing ........................................................................................................ 10
    2.3.2 Feature Extraction ................................................................................................. 11
    2.3.3 Decoding[16,18,19] .............................................................................................. 19
    2.3.4 Acoustic Modeling ................................................................................................. 20
    2.3.5 Language Modeling ............................................................................................... 21
    2.3.6 Post-Processing ..................................................................................................... 23
  2.4 Hidden Markov Model(HMM) ....................................................................................... 23
  2.5 Speech Recognition Discriminative Training ............................................................ 29
  2.6 Speech Recognition Adaptation Techniques ............................................................... 31
    2.6.1 Maximum-Likelihood Linear Regression (MLLR) ............................................. 32
  2.7 Evaluation Measures of the ASR Systems ..................................................................... 33

Chapter Three Kaldi toolkit .......................................................................................................... 34
  3.1 Introduction ....................................................................................................................... 35
  3.2 Kaldi Toolkit Overview and Structure ........................................................................... 35
  3.3 ASR System Component Using Kaldi Toolkit ............................................................... 37
3.4 Kaldi directories structure and required libraries.................................38

Chapter Four Work Procedure of Building ASR System ..........................44
4.1 Introduction .............................................................................................45
4.2 TIDIGITS Corpus ...................................................................................45
4.3 Tools preparation ....................................................................................47
4.4 Development of digits ASR system using Kaldi .................................48
  4.4.1 Data Preparation ...............................................................................48
  4.4.2 Language preparation .......................................................................52
  4.4.3 Preparing the lang Directory ...........................................................54
  4.4.4 Preparing the Configuration Files ..................................................58
  4.4.5 Feature Extraction .........................................................................59
  4.4.6 Training and alignment the acoustic modeling .........................60

Chapter Five: Results and Discussion .........................................................69
5.1 Introduction .............................................................................................70
5.2 Results of decoding the monophone model ........................................70
5.3 Results of decoding the delta-based triphone model ..........................71
5.4 Result of decoding triphone model (tri2a) .........................................73
5.5 Result of decoding LDA-MLLT triphone model .................................74
5.6 Results of Discriminative Training .......................................................75
  5.6.1 Results of Decoding Maximum Mutual Information ....................75
  5.6.2 Results of Decoding Boosting Maximum Mutual Information ....77
  5.6.3 Results of decoding Minimum Phone Error .................................78
5.7 Results of decoding Speaker Adaptive Training(SAT) .......................80

Chapter Six: Conclusion and Future Work ................................................83
6.1 Conclusion .............................................................................................84
6.2 Future Work ..........................................................................................85

Chapter Seven References ...........................................................................86

Appendices .........................................................................................................91
List of Figures

Figure 1. Milestones in speech recognition technology research [1] ...................7
Figure 2. Speech Recognition System Mechanism ...............................................8
Figure 3. The Architecture of Typical Speech Recognition System [16] ..........9
Figure 4. Time frames of feature extraction process .................................12
Figure 5. MFCC Features Extraction Process ..............................................15
Figure 6. Block Diagram of the PLP Modeling Process .................................16
Figure 7. The HMM for the activity that will be done by Sarah depending on
the weather (rainy or sunny) .................................................................................25
Figure 8. Hidden Markov Model for the Word Nine .....................................26
Figure 9. Hidden Markov Model representation for word "nine" using three
emitted states for each phone ...........................................................................27
Figure 10. forward-backward training algorithm (the Baum-Welch) [19] ......28
Figure 11. The Schematic Architecture of the Kaldi Toolkit ..........................36
Figure 12. Overview of Kaldi Directory Structure ........................................40
Figure 13. Simple Weighted Finite State Transducer (WFST) .......................42
Figure 14. The Final Representation of the Lexicon FST ..............................55
Figure 15. Representation of left-to-right HMM Topology with Three Emitting
States and One non-emitting State .................................................................57
Figure 16. Relationships among some types of training methods in the ASR
system ..............................................................................................................68
# List of Tables

Table 1. Hidden Markov model main parameters .............................................................. 24  
Table 2. example of Speech recognition semiring [20] ..................................................... 41  
Table 3. speakers distribution over train and test files [40] .............................................. 46  
Table 4. classification of United states dialects and Distribution of Speakers 46  
Table 5. The WER% and SER% results of decoding the monophone model ................. 71  
Table 6. The WER% and SER % results of decoding the delta triphone model (tri1) ....... 72  
Table 7. WER% and SER% results of decoding the delta+delta-delta triphone model (tri2a) ......................................................................................................................... 73  
Table 8. WERs% and SERs% results of decoding the LDA-MLLT triphone model (tri2b) ............................................................................................................................ 74  
Table 9. WERs% and SERs% results of decoding the MMI model with (--iter 3) .............. 76  
Table 10. WERs% and SERs% results of decoding the MMI model with (--iter 4) .......... 76  
Table 11. WERs% and SERs% results of decoding the bMMI model with (--iter 3) ......... 77  
Table 12. WERs% and SERs% results of decoding the bMMI model with (--iter 4) .......... 78  
Table 13. WERs% and SERs% results of decoding the MPE model with (--iter 3) ............ 79  
Table 14. WERs% and SERs% results of decoding the MPE model with (--iter 4) .......... 79  
Table 15. WERs% and SERs% results of the first-pass of speaker independent decoding (SAT model) .............................................................................................................. 80  
Table 16. WERs% and SERs% results of decoding SAT model ..................................... 81  
Table 17. The summarized Results of different models ................................................ 82
Acknowledgement

Through this work, I have the opportunity to express my sincere regards and appreciation to all the people who assisted and guided me to make this thesis a reality.

First of all, I want to thank my GOD Almighty for giving me the wisdom, the health, and strength to reach this stage of knowledge.

Second, I would like to express my deep gratitude to my advisor, Prof. Veton Këpuska, for his consistent support and motivation during my Master's degree studies. He provided me with guidance and knowledge, and led me in the right direction during the time of researching and writing of this thesis.

Third, I would like to thank my parents for their help and support during each step of my life. They taught me the value of hard work and an education. Without them, I may never have gotten to where I am today. Also, I would like to thank my beloved husband and my mother-in law for their financial support and endless trust and encouragement during the time of my Master's degree studies.

Finally, special thanks to my committee members: Prof. Samuel Kozaitis, Josko Zec, and Prof. Marius Silaghi, who kindly agreed to serve on my committee.
Dedication

To my father, Mr. Habeeb Alyousefi, my first love and my first hero.

To my mother, a piece of heaven, Mrs. Jinan Alrubaye, my whole life.

To my husband, Mr. Hayder Khzaali, my husband, my love, my friend, and my everything.

To my lovely daughters, Saba and Lara, the apples of my eye.

To my brothers, Mr. Yousif Alyousefi and Mr. Ibrahim Alyousefi, the source of my strength.

To my lovely sisters, Mrs Zina Alyousefi, Shaimma Alyousefi, and Rawah Alyousefi, the source of my inspiration.

To all my relatives and friend.
Chapter One Introduction and Thesis Outline
1.1 Introduction

Speech is the main way of communication that people use to express their thoughts and feelings to each other. Due to the huge developments in technology and the human need for machines and applications that could mimic humans behaviors, such as the ability to speak and respond to humans' languages, scientists and engineers have become much more interested in researching automatic speech recognition[1].

Speech recognition, or as it is known as Automatic Speech Recognition (ASR), is a technology that has been used to recognize and translate human spoken language by converting the speech recorded signals to text words[2]. People use their production mechanisms (the lungs, vocal cords, and articulators) to produce the spoken words, while the ASR systems work on the signals of these recorded words and try to recognize them.

Recognizing natural languages that are spoken by humans is the main goal of the ASR systems[3]. In addition, ASR systems have many advantages, such as reducing costs by using machines with speech recognition intelligence instead of humans[4]. This replacement helps people with disabilities to enable the “hands-free” control while working with other electronic devices [5]. Today, ASR technology has widespread applications that are used in real life, including, foreign language applications, automated telephone systems, and customers service calls, and medical dictation software.

There are different types of methods and algorithms that are used to build the ASR systems, such as Neural networks, deep feed forward neural network (DNN), Hidden Markov Models(HMM), and End-to-End Automatic Speech Recognition. However, Hidden Markov Model (HMM) is considered one of the most important techniques that is used for modeling the acoustics of speech in the most recent ASR systems[3]. Furthermore, language and acoustic modeling are considered some of the most important parts of ASR algorithms.
1.2 Thesis Goal

The main goal of this thesis is to build a complete ASR system by utilizing Kaldi toolkits. To achieve this goal, it is necessary to study and review all the processes involved with speech recognition technology and the toolkits in detail to gain a better understanding for creating this system. Also, this thesis will work on applying different types of training methods to evaluate the system performance by examining the Word Error Rate (WER), and Sentence Error rate (SER).

Moreover, this project aims to use the TIDIGITS data, which is one of the most well-known corpus that is licensed under the Linguistic Data Consortium (LDC).

1.3 Thesis Outline

This thesis is divided into seven chapters. Chapter One, above, gives a brief introduction about speech technologies and the goal of the project. Chapter Two, gives a brief overview of Automatic Speech Recognition systems, including: a brief history of ASR and its progress, architecture of the ASR systems, and the evaluation metrics. Also, this chapter discusses the different stages of ASR such as feature extraction, acoustic and language modeling, and the decoding method.

This research aims to use the Kaldi toolkit, so Chapter Three will explain this toolkit in detail. This part will give a brief overview about the design of this toolkits and will list all its important features and libraries.

Chapter Four will provide all the details about the methodology and feasible work that is done in this thesis supported with codes and figures. Results and system evolution will be discussed on Chapter Five. Chapter six, will include the conclusion of this thesis and the future work that could be done. References and appendices will be listed in chapter seven.
Chapter Two Overview of Automatic Speech Recognition
2.1 Introduction

Chapter Two gives a brief overview of Automatic Speech Recognition history and its developmental stages over the years. Also, this chapter explains in detail the architecture of the typical speech recognition system, such as, feature extraction and decoding methods and discusses the evaluation matrices of this system. Furthermore, the chapter discusses the most important statistical model, which is HMM.

2.2 The History of Automatic Speech Recognition Systems

Automatic speech recognition technology went through several stages of development to reach its present state. Over the past one hundred years, scientists and researchers have been very interested in inventing an instrument that could mimic human behavior, a machine that could interact with human languages, including speaking, recording, and responding[1].

In 1881, Alexander Graham Bell, and Charles Sumner Tainter were the first who created a recording machine that could respond to the pressure of the receiving voice[6]. They used a rotating wax cylinder that had multiples grooves, to allow a special needle to move through it easily[6]. Later In 1930, Homer Dudley was the first engineer who designed an electronic sound system analyzer, which was used during the second world war by sending secure voice messages[7,8]. After this invention, the scientists and researchers become more interested in the speech recognition field, and that led them to create more sophisticated speech machines.

In 1952, three Bell Laboratories scientists, Balashek, Davis, and Biddulph designed a simple speech recognition system which was used for separated digital recognition for one speech producer[9,10]. Four years later, Olson and Belar designed a system that could distinguish 10 syllables inserted among 10 single words, for one speaker at a time[9,11]. In 1959s, several ASR systems were introduced and designed. For
example, the MIT Lincoln Laboratories, by Forgie and Forgie, was able to create a vowel recognition system that could identify 10 vowels of single speaker [9,12]. Throughout the 1960s, many hardware research and designs, relative to ASR systems, were demonstrated and produced. Examples of the most well-known works were the vowel recognition system by Suzuki and Naka [5,9], the phonemes recognition system by Sakai and Doshita, and the digital recognition system by the NEC Laboratories [5].

The field of speech recognition systems witnessed significant progress during the 1970s. One of the most important developments of this period was the introduction of the statistical methods of the Hidden Markov Model (HMM) in the late 1960s and the beginning of the 1970s [13]. Moreover, some of the studies in Russia and Japan, made the technology that used the separated vocabulary recognition or isolated utterance recognition a very common way to build a simple ASR system [9]. Also, some scientists started using a large vocabulary instead of a small set of words to build their ASR system, such as the International Business Machine Corporation (IBM) [9,13]. Also, AT & T Bell laboratories started working on building a “speaker independent speech recognition systems” which is required collecting a huge amount of vocabularies [9]. Also, at this time, the Defense Advanced Research Projects Agency (DARPA) financially sponsored some aspirant proposals about building speech understanding systems, such as CMU’s Hearsay(-II) and BBN’s HWIM [1,9].

In the 1980s, scientists focused on using more strict methodologies instead of the template matching methodologies which is considered the basis of the ASR systems nowadays, such as the statistical modeling HMM. Also, during this period, scientists successfully re-presented the technology of artificial neural networks (ANNS) which is discussed and failed in 1950 [1]. Also, in 1986 the Carnegie Mellon University (CMU) designed and improved the Sphinx system which is one of the most well-known speech recognition systems that could utilize both of the Hidden Markov Model and acoustic models.

During the 1990s, ASR systems faced many improvements especially in the pattern recognition field by using the minimum classification error rate techniques. This
progress led to the introduction of some of the important methods, like discriminative training and kernel based techniques [1,14]. Moreover, DARPA continued their work using a large set of vocabulary and improving the ASR evaluation techniques. They used the word error rate (WER) and sentence error rate (SER) as an evaluation metric for the ASR system performance. Furthermore, many software tools were designed and presented during this time. For example, the Hidden Markov Model Tool Kit (HTK) was developed at the University of Cambridge and it is considered one of the most viable toolkits for ASR systems [1].

Speech recognition systems went through rapid improvements throughout the 2000s, due to the sophisticated software innovations and Internet improvements. Today, ASR systems are considered some of the most important technologies that are used every day. For example, the ASR system is used in telephone communication, in car systems, military vehicles and equipment, robotic design, health care devices, and in aerospace. Figure 1 illustrates the progress of the ASR system throughout the past century.

Figure 1. Milestones in speech recognition technology research [1]
2.3 The Architecture of Automatic Speaker Recognition System

Speech is one of the most important means of communication among humans and consequently speech processing became an interesting topic for researchers and engineers[15]. Speech recognition is a technology that used to convert the speech signal to understandable words or sequences of understandable words. ASR is the field of studying speech signals and the ways to process these signals. To get this signal, humans use their vocal cords to produce an amount of sound or speech and then they record this sound by using a high-quality microphone. After that, the speech goes through a speech recognizer system to recognize and convert this signal into series of words (text). The block diagram in Figure 2 illustrates the general speech processing mechanism.

![Figure 2. Speech Recognition System Mechanism](image)

After humans convert their sound to speech signal, the speech recognition system starts processing this signal through different stages to produce the most likely text for it. Figure 3 shows the general stages of processing the speech recognition system.
During the pre-processing stage, the speech signal goes through preprocessing operations to improve it, such as applying some pre-emphasis filters and noise removal or reduction. After that, the speech signal goes through feature extraction stage to get some parsimonious features which are useful for the next stages. The decoding stage aims to find the most likely matching between all sequence of words and the signal of feature vectors by applying the acoustic and language models which are both consider as the heart of the speech recognition systems [16,17]. The goal of the last stage, the post processing stage, is to get the perfect hypothesis among the n-best hypothesis[16].

Before explaining the ASR process in detail, it is very important to give a brief explanation about the ASR probability theory. Speech recognition systems aim to hypothesize the ideal discrete character among the whole given sequence for the acoustic input, O, where O can be represented as [17] :

\[ O = o_1, o_2, o_3, ..., o_t \]  \hspace{1cm} (1)

Also, the selected symbol sequence can be represented as:
\[ \mathbf{W} = w_1, w_2, w_3, ..., w_n \]  

(2)

So the basic goal of the speech recognition systems can be represented as:

\[ \hat{\mathbf{W}} = \arg \max P(\mathbf{W}|\mathbf{O}) \quad \text{for} \quad \mathbf{W} \in \mathcal{L} \]  

(3)

to calculate the probability of \( P(\mathbf{W}|\mathbf{O}) \) in the third equation, Bayes’ law was used to get the equation below:

\[ P(\mathbf{W}|\mathbf{O}) = \frac{P(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{O})} \]  

(4)

From the fourth equation, the probability \( P(\mathbf{W}|\mathbf{O}) \) on the left-hand side is more difficult to calculate than the right-hand side[17]. So, substituting the probability \( P(\mathbf{W}|\mathbf{O}) \) in Equation 3 by Equation 4, can be produce the following equation:

\[ \hat{\mathbf{W}} = \arg \max \frac{P(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{O})} = \arg \max P(\mathbf{O}|\mathbf{W})P(\mathbf{W}) \quad \text{for} \quad \mathbf{W} \in \mathcal{L} \]  

(5)

Equation 5 is considered as the primary formula of the statistical speech recognition system which is represented in the acoustic and language modeling. It is noted that, the quantity of \( P(\mathbf{O}|\mathbf{W}) \), which is also known by the observation likelihood, can be computed by applying the HMM[17](will explain in this chapter).

2.3.1 Preprocessing

The main goal of the preprocessing phase is to minimize the negative effects of the environment on the speech signal and to work on improving it to perform a better recognition for this signal in the upcoming stages. There are several techniques and
operations could be performed during this stage[19]. Converting the speech signal from analog to digital is considered one of the most important steps in this stage. This can be done by sampling the signal using a proper sampling frequency, such as 16kHz or 8 kHz, and applying the quantization process[19]. Also, one of the first improvements that could be applied to the signal at this phase is the pre-emphasis processing. The goal of pre-emphasis filter is to increase the amplitude of the signals that have high frequencies and reduce the amplitude of the signals that have low frequencies (e.g. 100Hz) [16,17].

In general, the normal ASR system works to assign each sound in the speech to some amount of probability[16]. So, any kind of noises that could happen during the recording operation could lead to inserting word/words to the output hypotheses. The preprocessing stage tries to avoid this by applying the speech/non-speech segmentation operation. This operation works to delete some part of the recording such as removing the part that is located between the beginning of the recording and the time when the actual speech starts, and the end of the recording after the actual speech finishes[16]. This segmentation method is known by the end point detection.

2.3.2 Feature Extraction

After the preprocessing operation, the speech signal goes through the feature extraction phase. The goal of this stage is to extract sequences of acoustic observations that contain all the useful features and information for recognition operation[17,19]. These features are extracted by using an overlapped windows of time(frames) that have an equal length of 25 ms [16,20]. Theses frames are shifted by fixed time equal to 10 ms. Figure 4 shows an example of these overlapped shifted frames in MFCC.

There are several methods used in the feature extraction operation, such as Mel-frequency cepstral coefficients (MFCC), Feature space transformations, perceptual linear prediction (PLP), Cepstral mean and variance normalization (CMVN), Kernal Based Feature Extraction, and Dynamic Feature Extraction. However, the MFCC and PLP methods are considered the most public processes that have been used lately [18,20].
1. **Mel-Frequency Cepstral Coefficients (MFCC):** The MFCC extraction methods are used with a sampled and quantized speech signal. It is very important to explain this process because it will be used in this thesis to extract the features for the speech signals. First, the signal should be sampled and pre-emphasised, as explained in the pre-processing stage. Second, the signal should go through the windowing stages which divides the sampled signal into sections by using time frames of equal length of $n$ milliseconds, and equal shift phase of $m$ milliseconds\[18]. The hamming window is considered as the most popular window that its used by the MFCC, which aims to reduce the spectral distortion by shrinking the beginning and end of each frame of the signal to zero. The hamming window can be computed as:

$$w[n] = \begin{cases} 
0.54 - 0.46 \cos\left(\frac{2\pi n}{L}\right) & 0 \leq n \leq L - 1 \\
0 & Otherwise
\end{cases}$$  \hspace{1cm} (6)
Where L is the frame size.

After windowing the speech signal, the discrete Fourier transform (DFT) should be applied to each frame to calculate the energy spectrum of the speech signal\[17,18\]. The DFT calculation can be defined as the following:

\[
X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}} \quad 0 \leq k \leq N - 1
\]  

(7)

The \(x(n)\) represents the framed signal, the \(X(k)\) defines the DFT, and \(N\) refers to the window length. However, the energy spectrum can be calculated by squaring the DFT value.

\[|X(k)|^2 = X(k)X^*(k)\]  

(8)

Where the \(|X(k)|^2\) is the energy spectrum.

After the DFT processes, each frequency of the speech signal will be loaded with information. However, the sensitivity of the human hearing system is less when the frequency is above 1000Hz, and that will affect the ASR system performance. To avoid this problem, the Mel-frequency scale should be used to warp the energy spectrum results of the DFT, and that could be done by using triangular filters. The following equation defines the Mel-frequency scale:

\[mel(f) = 1127 \ln(1 + \frac{f}{700})\]  

(9)

Furthermore, after getting the result of the Mel-Filter Bank process, the logarithm of this result should be computed to get the co-efficients of the Mel spectrum.
The last step of the MFCC feature extraction process is to apply the discrete cosine transform (DCT). The DCT should be applied to the co-efficients of the Mel spectrum to obtain 12th order of the coefficients of the Mel-cepstral [18,21]. Also, applying the DCT will allow getting the maximum features decorrelation which leads to enhance the classification operation. Furthermore, it is very important to compute the first and second derivative of the last results to detect any modifications that could happen to the signal in each frame[22]. Figure 5 illustrates the MFCC feature extraction process.
Figure 5. MFCC Features Extraction Process

1 Energy Feature

Pre-emphasis

Windowing process

DFT

Mel-filter Bank

log ( )

DCT(IDFT)

12 MFCC Coefficients

Deltas

12 MFCC + 1 Energy

12 Δ MFCC + 1Δ Energy

12 ΔΔ MFCC + 1 ΔΔ Energy
2. **Perceptual Linear prediction (PLP):** This is another modeling technique that is presented by H. Hermansky to model the speech signal depended on the hearing psychophysics principles[23,24]. This analyzation method works on enhancing the recognition result by rejecting any unnecessary information and focusing only on the data that is related to the speech signal. The PLP technique is similar to the Linear Predictive Codes (LPC) modeling method, but it employs three essential concepts. As shown in Figure 6, these concepts are the critical band analysis, the curve of equal loudness, and the relation of intensity-loudness power-law. For more information about the PLP, please review[24].

![Figure 6. Block Diagram of the PLP Modeling Process](image)

3. **Feature space transformations:** This extraction method is applied on each frame in addition to the MFCCs method. However, when the Feature space transformation is applied, it takes into consideration the left and right context[20]. This method can be represented by matrix multiplications $A_x$ and $A_x^+$, where the $A$
performs the transformation, \( x \) represents the input vectors, and \( A_x \) perform the transformed features. In general, these different transformations can be trained and estimated discriminatively or generatively (as explained in Chapter 4). Kaldi toolkit supports different types of these transformations, such as Heteroscedastic Linear Discriminate Analysis (HLDA), Linear Discriminate Analysis (LDA), Maximum Likelihood Linear Transform (MLLT), Exponential Transform (ET), and Cepstral Mean and Variance Normalization (CMVN).

- **Linear Discriminate Analysis (LDA):** This is a feature extraction method that is used to produce a linear features transformation of \( p \)-dimensional space from \( n \)-dimensional feature vector, where the \( p \)-dimensional space is smaller than \( n \)-dimensional feature vectors \((p<n)\). Linear Discriminate Analysis (LDA) applies to spliced MFCC features with left and right context. Features that are close to each other are classified as the same class, while features far away from each other are classified in different classes. LDA tries to increase the ratio of between class variance and within class variance. Consider \( \mathbf{T} \) is the total variance matrix which can be defined as the following [41]:

\[
\mathbf{T} = \sum_{j=1}^{J} \sum_{g_i=j} (\mathbf{x}_i - \bar{x})(\mathbf{x}_i - \bar{x})^T
\]  

(10)

The within class variance \((W_j)\) can be represented as:

\[
W_j = \sum_{g_i=j} (\mathbf{x}_i - \bar{x})(\mathbf{x}_i - \bar{x})^T
\]  

(11)

Consider \( W \) equal:
\[ W = \sum_{j=1}^{j} W_j \]  
(12)

The total between variance is:

\[ B = T - W \]  
(13)

From questions 10 through 13, the ratio of between group to within group variance with projection can be represented as:

\[ y = \hat{\theta}^T X \]  
(14)

\[ \hat{\theta} = \arg\max_{\theta} \frac{\theta^T B \theta}{\theta^T W \theta} \]  
(15)

The solution for this equation is the eigenvectors of $W^{-1}B$ that contains the maximize eigenvectors:

\[ y_p = \hat{\theta}_p^T X \]  
(16)

Where $\hat{\theta}_p$ is a matrix of eigenvectors ($n \times p$)

However, by applying the above calculation, that will produce the wrong $p$ features. Therefore, features are achieved from smaller dimensionality ($p < n$) in order to reduce the model parameters of the speech recognizer [41].

Sometimes, ASR uses Linear Discriminate Analysis (LDA) and Maximum Likelihood Linear Transform (MLLT) transformed features on top of MFCC in order to perform speaker independent training. According to [42], MLLT is a matrix of square feature space transformation and its
function aims to find the average per-frame log-likelihood of the transformed features and transform log determinant.

- **Cepstral mean and variance normalization (CMVN):** The CMVN is a noise compensation technique that is commonly used in the speech recognition field in addition to the MFCCs[17]. This technique works on subtracting the cepstral mean from each feature component in the utterance to achieve a zero mean feature vectors (like the CMN method). After that the unity variance should be estimated by scaling each of these features separately [25]. note that the mean and variance of these transformed utterances should be matched with the mean and the variance of clean data. This technique has many advantages, such as extracting more robust features that enhance the performance of the recognition process, allowing the phones to have fixed size and location among the features, and minimizing the mismatching that could happen between the testing and training utterances by compensating them with zero mean and unit variance [17].

### 2.3.3 Decoding[16,18,19]

The decoding stage is considered one of the most important parts of the ASR system. The main goal of this stage is to find the best sequences of words (Ŵ) that could match the acoustic signal that is represented by the observed features components. The decoding operation employs two types of modeling which are the acoustic and language models to find the optimal matching. There are three essential types of information that must be available for the decoding process:

- An acoustic model and the Hidden Markov Model (HMM) for each utterance
- A language model and word sequence likelihoods.
• A pronunciation dictionary, which contains a list of words and their related phonemes.

The decoding process works to finding the list of words that can be said. These words could be found in the dictionary part, which is a list of all the possible words plus their phonemes. Another important part which the acoustic model which has probability-function, that is represented by the Gaussian mixture and the likelihood of each observed component \( P(O|W) \). However, the language modeling is not so important in comparison to The acoustic modeling, but it is used because it works to raise the words' accuracy and to correct the words' grammar.

For a better understanding, according to Equation 5, the decoding mechanism tries to find the sequence of words (\( \hat{W} \)) that is most likely to match the observation vector (\( O \)). Furthermore, the value of the \( P(W) \) is achieved through the language model, and the value of \( P(O|W) \) can be computed through the word phonemes that are available in the pronunciation dictionary.

It is not possible to compute the probabilities of the entire available path through the state network if the space of the states are very large. Therefore, some of the algorithms are used to find the hidden states, such as the Viterbi search algorithm that is depend on the dynamic programming method [2].

2.3.4 Acoustic Modeling

The acoustic model is one of the most essential processes for the ASR system, and it could be considered as the heart of the recognition operation. The main goal for the acoustic model is to enhance the speech recognition accuracy by specifying the modeling units and computing the likelihood of the acoustic features components for the phonetic units that need to be recognized[18]. It's very important to select the linguistic units, which
can be phones, words, or subparts of phones, because using a variety of units can affect the ASR system performance.

The acoustic modeling operation can be defined as the operation of computing the statistical representation of the observed features vectors of the speech signal. According to [18], the acoustic model is like a file that collects all the statistical representations of each distinct phoneme of the speech. Also, these statistical representations should be assigned to a special label that is named by the “phoneme acoustic model” and that is obtained by using the speech corpus and special training algorithms.

There are several methods that are used to build the acoustic model, such as Hidden Markov model (HMM), conditional random fields, segmental models, and maximum entropy models. However, the Hidden Markov Model HMM (explained in this chapter) is considered one of the most powerful statistical models that is intensively used in the speech recognition field.

In ASR systems, most of the acoustic model stage uses the Hidden Markov Model with the Gaussian Mixture Model (GMM) to extract the Acoustic Features Vectors[32]. The GMM works on calculating the probabilities of the observation vector $P(o|q)$ for each HMM state (q), depending on the phone (o). For more information about the Gaussian Mixture Model (GMM), please review[32].

### 2.3.5 Language Modeling

According to the American language, there are many sequences of words that have very similar phonemes, such as “wreck a nice beach” and “recognize speech”. These two sentences have almost the same pronunciation, but with different meanings. In general, humans do not have any problem with recognizing these words because they already know what the possible words are that can be used in the context. However, the computer will not have the human ability to recognize the matching sounds, and for that purpose the statistical language model is used for the ASR system.
The language model work on estimating the probabilities distribution on a sequence of words. In other words, if there is a sequence of words, \( w=(w_1,w_2,...,w_K) \), the language model uses probabilities distribution of the \( P(w) \) over this sequence. According to [20], the main goal of language modeling is minimizing the hypothesis of the acoustic model and prioritizing it at the same time. These probability results are gathered with the probability results of the acoustic model to calculate the final probability of the entire transcription.

There are different kinds of language modeling that are used in the ASR field, but the n-gram is considered one of the most common methods. The n-gram uses the previous words (n-1) to estimate the following word. This can be define as the following:

\[
P (w) = \prod_{k=1}^{K} P(w_k \mid w_{k-1}, w_{k-2}, ..., w_1)
\]

(17)

This estimation process is named a Markov, after the mathematician Andrey Markov, who invented this process which indicates that the probability of each word relies on the previous word.

The bigram and trigram are a common types of the n-gram language modeling that can be used in speech recognition. The bigram can be obtained when the \( n \) of the n-gram is equal to two \( (n=2) \), and the trigram can be achieved when the \( n \) of n-gram is equal to three \( (n=3) \). Therefore, the n-gram can be achieved be looking at a \( (n-1) \) word in the past. The following equation can be used to compute the n-gram for the large vocabularies recognition system,

\[
P (w) = \prod_{k=1}^{K} P(w_k \mid w_{k-1}, w_{k-2}, ..., w_{k-n+1})
\]

(18)

Where \( n \) ranges from 2-4.
2.3.6 Post-Processing

According to [16], most of the decoding stage algorithms, such as the Viterbi search algorithm, are produced a set of all possible sequences of words (five to ten limited hypotheses) that are arranged by their total score. This output list is called the n-best list because its contains the best hypotheses among the others. The goal of the post-processing stage is to arrange this list and choose the best hypotheses among the n-best hypotheses. So the words with a higher score will be the recognition output. One of the most common ways that is used to re-score the list is utilizing a different source of information, such as a higher-order language model.

2.4 Hidden Markov Model (HMM)

The Hidden Markov Model is one of the most popular statistical modeling methods that was presented and researched between 1960s and 1970s. This modeling technique is considered the heart of most of the applications, such as the ASR systems for two main reasons[29]:

- HMM models have a very strong mathematical structure and that allows it to form most of the basic theories that can be used in many applications.
- HMM models always give accurate results when it is applied correctly for many important applications.

Before explaining the HMM in detail, it's very important to give a brief introduction about the Markov Chain. According to [30], the Markov Chain is a weighted finite automaton that representing a sequence of states and transitions that connect them. each of these arcs are assigned to a weight value (probability). The probability of any state among the set is dependent only on the previous state. This chain is mostly used to calculate the probability for an observed events but it is not able to observe the hidden event. However, the Hidden Markov Model became more popular because it can compute the probabilities of hidden events, such as the part of speech tags, and the observed event,
such as the normal words that we observed at the input[30,19]. The HMM components are defined in the following Table:

<table>
<thead>
<tr>
<th>HMM components</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q = q_1 q_2 q_3 \ldots q_N$</td>
<td>$Q$ is a set of $N$ state</td>
</tr>
<tr>
<td>$A = a_{11} a_{12} a_{13} \ldots a_{n1} \ldots a_{nn}$</td>
<td>$A$ is a transition probability matrix; each $a_{ij}$ performs the probability of moving from $i$ to $j$: $\sum_{j=1}^{n} a_{ij} = 1 \ \forall \ i$</td>
</tr>
<tr>
<td>$O = o_1 o_2 o_3 \ldots o_T$</td>
<td>A set of $T$ observations, each one obtained from a vocabulary $V = v_1, v_2, \ldots, v_V$</td>
</tr>
<tr>
<td>$B = b_{i}(o_t)$</td>
<td>It is a set of observation probability, each expressing the probability of an observation $o_t$ that is generated from a state $i$.</td>
</tr>
<tr>
<td>$q_0, q_F$</td>
<td>The start state and final state that are not related with observations.</td>
</tr>
</tbody>
</table>

The Hidden Markov Model has two main assumptions that could be illustrated as the following[30]:

- The probability of the current event, depends on the previous event only. This assumption can be defined as:

$$P(q_t | q_1 \ldots q_{t-1}) = P(q_t | q_{t-1}) \quad (19)$$

- The probability of the outcome observation ($o_t$) depends on the event that created the observation of ($q_t$) only.

$$P(o_t | q_1, \ldots, q_T, o_1, \ldots, o_t, \ldots, o_T) = P(o_t | q_t) \quad (20)$$
In the Hidden Markov Model, the hidden state is the event that cannot observed immediately. Figure 7 gives a simple example of the Hidden Markov Model about the activities (which include going to the park, going to the shopping center, or cleaning the house) that Sarah will do, depending on whatever the weather is sunny or rainy. This example has two hidden unobserved states which are represented by sunny weather (S) and rainy weather (R) and the observations are represented by activities that Sarah will do (going to the park, going to the shopping center, or cleaning the house).

Figure 7. The HMM for the activity that will be done by Sarah depending on the weather (rainy or sunny)
In speech, the HMM states represent each signal phones of the words, and these states are concatenated between each other. For instance, the HMM for the word "nine" can be seen in Figure 8. This HMM has three emitted states which represent the phones of word "nine" (n, ay, n) and two not emitted states (Start, End).

![Figure 8. Hidden Markov Model for the Word Nine](image)

However, when there is a limited amount of data, it is insufficient to use the HMM to represent each single phone by a single state. Therefore, the HMM used another representation method that is called phone model where each phone is represented by three emitted states: the beginning (b), middle (m), and final (f) states[19]. Thus, to represent the word "nine", 11 states are needed, three for representing the phone (n), three for representing the phone(ay), three for representing the last phone (n), and two non-emitting states for the start and end state. Figure 9 shows the HMM representation for the word "nine" using phone modeling.
Figure 9. Hidden Markov Model representation for word "nine" using three emitted states for each phone.

When all the HMM states are completely connected and there is no zero probability transitions among these states, the HMMs are called ergodic HMM[30]. However, there are other kinds of HMMs that have a zero probability, such as the Bakis HMM (left to right). In Bakis HMM, the transitions among the states are drawn from left to right, or as a self loop to the same state, and transitions from higher order states to lower order states are not allowed (have zero probability transitions)[19]. The best example for that is shown in Figure 8.

According to [21,29] the HMM should be characterized based on the following points:

- likelihood: Given an HMM $\lambda = (A, B)$ and an observation sequence $O$, determine the likelihood $P(O|\lambda)$. 

S0 is the Start state

E10 is the end state
decoding: Given an observation sequence O and an HMM \( \lambda = (A, B) \), discover the best hidden state sequence Q.

Learning: Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B.

Re-assessing the parameters of \( \lambda \) to increase \( P(O|\lambda) \).

In the ASR system, Hidden Markov model can be trained using different training algorithms, such as Viterbi training, maximum likelihood estimation (MLE), and the Expectation Maximization (EM). However, the most common method that is used is the forward-backward algorithm (the Baum-Welch) which is shown in Figure 10. This training method initializes some HMM probabilities, \( \lambda = (A, B) \), and then applies two steps: the expectation step (E-step), and the maximization step (M-step)[30]. The E-step aims to estimate the expected state occupancy (\( \gamma \)) and the expected state transition (\( \xi \)) from the initiated probabilities A, and B while M-step aims to re-calculate a new A and B parameters from the \( \gamma \) and \( \xi \) count[30].

Figure 10. forward-backward training algorithm (the Baum-Welch) [19]
2.5 Speech Recognition Discriminative Training

Most of the speech recognition system accuracy depends on the HMM and its trained parameters. Previously, the maximum likelihood estimation (MLE) was the most usable training technique because it generates accurate systems that are easy to train with the Baum-Welch algorithm [42]. Also, the MLE produces numbers of assumptions, such as unlimited training data, identity of the true language model, and observations which always belong to the Gaussian distribution family. However, most of these features and assumptions do not apply to the speech and sometimes lead to non-accurate results. Therefore, researchers started applying other techniques, such as the discriminative training [31].

Discriminative training formulates an objective function to be used for discriminating probable hypotheses that tend to confuse correct and incorrect answers [20,31]. Speech recognition systems have different types of discriminative training objective functions, but this thesis will focus on the ones that are used in training our digits ASR system:

1. Maximum Mutual Information (MMI): the goal of this discriminative training objective function is to minimize the conditional entropy in order to maximize the probability of the right word sequence given the models:

\[
\mathcal{F}_{\text{MMI}}(\lambda) = \sum_{r=1}^{R} \log \frac{p_{\lambda}(X_r|M_{sr})^{k} \cdot P(s_r)}{\sum_{s} p_{\lambda}(X_r|M_{s})^{k} \cdot P(s)}
\]

where \(\lambda\) refers to the parameters of the acoustic model, \(X_r\) are the training words, sequences \(M_s\) represent the HMM sequence for \(s\) and transcriptions, \(k\) refer to the acoustic scale, and \(p(s)\) represents the language model[31,42].

The MMI objective function and other types of it can be optimized by using the extended Baum-Welch algorithm (which is different than the traditional
Baum-Welch algorithm, and the steepest gradient descent. Also, this objective function can make statistics accumulation by creating lattices and performing forward-backward on the lattices on each iteration. This process will not only optimize Equation 21, but also it will minimize the calculations and its expenses. In addition, MMI can enhance the generalization process to the unseen dataset by using different techniques, such as 1-smoothing, H-criterion, Acoustic Model Scaling, and frame discrimination (FD) [31].

2. **Minimum Phone Error (MPE):** This is an alternative objective function that aims to minimize the number of phone errors to increase the phone level accuracy [42]. MPE proved that it provides better test set performance. MPE can be determined as the following:

\[
F_{\text{MPE}}(\lambda) = \sum_{r=1}^{R} \sum_{s} \frac{p_{\lambda}(X_{r} | M_{s})^k P(s) A(s, s_{r})}{\sum_{s} p_{\lambda}(X_{r} | M_{s})^k P(s)}
\]

where \(A(s, s_{r})\) represents raw phone accuracy, and \(k\) represents the acoustic model scaling factor.

3. **Minimum Word Error (MWE):** This is an alternative objective function that aims to minimize the amount of word errors to increase the word accuracy. MWE determines this by using the same MPE Equation 8, except \(A(s, s_{r})\) should represent the raw word accuracy [31].

4. **Boosted Maximum Mutual Information (bMMI):** In this objective vector, a booting vector is used in order to boost the likelihood of the sentences that contain high errors. bMMI can be calculated as the following [42]:

\[
F_{\text{bMMI}}(\lambda) = \sum_{r=1}^{R} \log \frac{p_{\lambda}(X_{r} | M_{s_{r}})^k P(s_{r})}{\sum_{s} p_{\lambda}(X_{r} | M_{s})^k P(s) \exp(-bA(s, s_{r}))}
\]  

In the above equation, \(b\) represents the boosting vector which is set to (0.05) in this thesis (see Chapter four). However, the \(A(s, s_{r})\), which represents the
raw sentence accuracy, can be calculated using different methods. Therefore, the calculation method needs to be specified with the boosting factor $b$.

Extra calculations on bMMI is very small compared to the IMM objective function. When the forward-backward algorithm is applied on the denominator lattice, its work on subtracting from the acoustic log likelihood ($b$) times the contribution to the $A(s, s_r)$ arising from the arc in question[42].

### 2.6 Speech Recognition Adaptation Techniques

To build a speech recognition system, it is necessary to process the speech signal through four main stages: preprocessing, feature extraction, decoding (with acoustic and language model), and post-processing. However, sometimes this system may have a low recognition performance due to the variation among speakers, the surrounding noise, or the different channels[27]. Many speaker adaptation techniques have been presented to solve this problem. The main goal of the speaker adaptation is to modify the acoustic model parameters to match the features of the actual acoustic. The following equation defines the process of the speaker adaptation:

$$\hat{\theta} = f(\theta_1, ..., \theta_n)$$  \hspace{1cm} (24)

where the $\hat{\theta}$ is the target model that needed to be obtained, the $f$ is the adaptation model, and $n$ is the number of initial models provided.

Moreover, each adaptation technique should fulfill the following goals[27]:

1. It should be able to enhance the accuracy of the recognition process, even if that enhancement was on a small amount of adaptation data.
2. When the amount of adaptation data is raised, it should be able to make the recognition accuracy asymptotically reach the accuracy of a matched model.
There are many types of adaptation techniques, such as, the Maximum-likelihood linear regression (MLLR), Maximum a posteriori (MAP), and semi-tied covariance (STC). However, the Maximum-likelihood linear regression (MLLR) is considered one of the most common techniques that is used widely in the ASR field [17,19,27].

### 2.6.1 Maximum-Likelihood Linear Regression (MLLR)

Maximum-likelihood linear regression (MLLR) is one of the most common adaptation techniques that is used to calculate the linear transformations of the model parameters of the Gaussian mixture to adapt the speaker by increasing the likelihood between the actual model and the adaptation model[19,28]. This can be defined as the following:

\[
\hat{\mu} = W\hat{\mu}
\]

Where \( W \) is the transformation matrix that could be represented as \( W \in \mathbb{R}^{n \times n+1} \), \( n \) refers to the amount of means that are used for adapting the speaker, and \( \hat{\mu} \) represents the extended mean vector which can be computed as the following:

\[
\hat{\mu} = [\mu_1 \mu_2 \ldots \mu_n]^T
\]

However, there are different kinds of approaches that could apply to implementing different kinds of MLLR methods. For example, the mean-only MLLR can be achieved by applying Equation 15; the standard MLLR can be approached by estimating the mean and variance transform, while the constrained MLLR (fMLLR) can be obtained by calculating linear transformations of the observation features rather than model parameters. For more information about Maximum-likelihood linear regression (MLLR), please refer to [28].
2.7 Evaluation Measures of the ASR Systems

The performance of the ASR systems can be evaluated in different measures, such as error, accuracy, and correctness[16,18]. In common, the ASR systems have three common types of error: substitution, insertion, and deletion. The substitution occurs when the system recognizes a word that is different than the spoken word. The insertion occurs when the output hypotheses contains a word that is not spoken by the speaker. The deletion happens when the recognized results have missed some spoken words[16,17]. Usually, the ASR system used the Word Error Rate (WER) as metric of accuracy, which is defined as:

\[
WER(\%) = \frac{\text{Insertion}(I) + \text{Substitution}(S) + \text{Deletion}(D)}{\text{No. of Reference Word}(N)} \times 100
\]  

(27)

Also, the term of accuracy could be measured with the Word Recognition Rate (WRR) metric, and Sentence Error Rate (SER), but in general most of the speech recognition systems use the WER as a main metric[18,17]. The WRR can be represented as:

\[
\text{Word Recognition Rate (WRR)} = 1 - WER = \frac{N - S - D - 1}{N}
\]  

(28)

Furthermore, the correctness metric can be computed as:

\[
Corr = \frac{N - D - S}{N}
\]  

(29)
Chapter Three  Kaldi toolkit
3.1 Introduction

This chapter will explain the Kaldi toolkits, the main toolkit that is used in this thesis, by giving a brief introduction about it. Also, each component of the speech recognition system that is supported by this toolkit will be illustrated, such as the feature extraction, acoustic modeling, and decoding. Furthermore, this chapter will present overview of the structure of the toolkit, its features, and the main library.

3.2 Kaldi Toolkit Overview and Structure

Kaldi toolkit is open source software that is designed for the speech recognition system. The code of this toolkit is written in C++ language and is freely released by the Apache License v2.0[33]. This ASR toolkit has all the necessary speech recognition algorithms, including ready templates that can be used to train different acoustic models by using a recorded speech corpora, such as TIDIGITS and TIMIT. The main goal of Kaldi is to build a software that is modern, easy to use, flexible enough to extend and update, and powerful at training the acoustic model[34]. Mainly, Kaldi toolkit depends on the finite-state transducers (FST) and Shell script files to create a speech recognition system.

The architecture of Kaldi toolkit can be described as a sequence of models that consist of libraries and training scripts that are used to access these libraries. Models on the lower levels depend on one or more of models on the higher levels. As seen in Figure 11, Kaldi depends on two main external libraries: The OpenFst, that is designed for finite-state transducers (FST) and both of the Basic Linear Algebra Subroutines (BLAS) and Linear Algebra Package(LAPACK) that are designed for linear algebra purposes. Both libraries are available at no cost. This ASR toolkit supports some library models which are divided into two groups; each group is supported by only one external library, while the decodable model is considered as a link that works on connecting these two groups of models. In order to reach the library functionalities, Kaldi provides a C++ command line tools that can be called using (Shell) Scripts language. In addition, Kaldi uses pipes system for reading and writing these tools and that makes it easy to chain different tools.
Moreover, Kaldi toolkit has an executable that is used for loading the input from, and storing the output to, the files. Also, this toolkit is designed in a way that could accept the new features by adding them as a new code and command line without the need to change the original algorithms.

![The Schematic Architecture of the Kaldi Toolkit](image)

In general, the ASR system deals with different kind of open source toolkits that are designed for speech recognition purposes. Examples of these toolkits are RWTH ASR, HTK(which is similar in concept and goal to Kaldi), and Sphinx4. However, Kaldi toolkit can be characterized among the other ASR toolkits by the following features[33,34]:

1. Kaldi code integrated with the Finite State Transducers (FST) by using the library of OpenFst as a main library.
2. Highly supportive for linear algebraic operations because it implements the matrix library, which includes both slandered BLAS and LAPACK libraries.

3. Code design that is easy to extend and modify due to the algorithms that are provided in the most generic way possible.

4. Free license by the Apache software foundation.

5. Providing complete recipes that are used for creating ASR systems by utilizing some databases, such as TIMIT, TIDIGITS or any recorded data.

6. Kaldi code is inclusively tested so it can provide precise results.

3.3 ASR System Component Using Kaldi Toolkit

As explained, in Chapter Two, regarding the components of the speech recognition system, it is very important to illustrate how Kaldi toolkit supports each part of these components as well[33]:

- **Feature Extraction**: commonly, during feature extraction phase, Kaldi aims to extract the standard MFCC and PLP features. However, Kaldi supports other types of feature extraction methods, such as cepstral mean and variance normalization, HLDA, STC/MLLT, VTLN, and LDA.

- **Acoustic Modeling**: the main goal of Kaldi toolkit is to support the traditional Gaussian Mixture models (GMM) and Subspace Gaussian Mixture model (SGMM). For GMM, Kaldi supports the diagonal and full covariance design by using a `DiagGmm` class for diagonal GMM and an `AmDiagGmm` class for GMM-based acoustic model. For SGMM, Kaldi uses a special class called `AmSgmm`. Moreover, Kaldi supports the Hidden Markov model by designing a special class called `HmmTopology` which permits to use of the non-emitting states and reconnecting the pdf's of the states.

- **Speaker Adaptation**: Kaldi toolkit supports the model space adaptation by utilizing maximum likelihood linear regression (MLLR) and Feature space Maximum Likelihood Linear Regression (fMLLR). Furthermore, speaker
normalizations are supported by Kaldi by employing a more generic way named by the exponential transform.

- **Language Modeling:** Kaldi toolkit supports any kind of language model by using specific tools which are designed to convert the language model to FST, such as the SRILM and IRSTLM toolkits,

- **Building Decoding Graph:** Kaldi utilizes the Weighted Finite State Transducers (WFSTs) for decoding-graph construction.

- **Decoding:** Kaldi tool kit supports different kinds of decoding such as SimpleDecoder, FastDecoder, and BiglmDecoder with their lattice-generating version. The Simple Decoder usually works as a debugger for highly advanced decoder, while the FastDecoder is considered as a more advanced decoder that can handle the highest numbers of states that are active at the same time. However, the BiglmDecoder is used to handle the huge language model that sometimes exceeds a million arcs.

In addition, Kaldi provides various utilities, such as logging and error-reporting, command-line parsing, and math and STL functions.

### 3.4 Kaldi directories structure and required libraries

As mentioned above, Kaldi toolkit collects different kinds of libraries and algorithms that could be use to design different kinds of ASR systems. However, there are some important prerequisites that should be installed to help with installation and running this toolkit:

- **ATLAS:** Automatically Tuned Linear Algebra Software (ATLAS)
- **Autoconf:** used for configuration operations
- **git:** version control system that is used for distributing codes for the user after updating it by the developers.
- **Automake:** used for creating Makefiles
• **Subversion (svn):** another version control system that is important for Kaldi downloading and installation,
• **wget:** used for retrieving data by using HTTP, HTTPS and FTP protocols
• **awk:** a programming language that is used for data processing and extraction.
• **SRILM and IRSTLM language model toolkits.**

After installing all these requirements, Kaldi toolkit can be downloaded from [35], and installed by following all the instructions in the "INSTALL" file inside the Kaldi Directory. Make sure that SRILM and SRILM are installed properly in case users want to use the language model.

The Kaldi main directory can be defined as a network of directories, subdirectories, and documentation files. As shown in Figure 12, the high level directory consists of [34]:

• **egs:** which stands for example, and is considered as one of the most important directories that collects templates scripts with different versions for some common speech corpora that help users build their ASR system easily. Inside each of these template directories there are standard subdirectories structures which include: conf (stands for configuration), data, exp (stands for experiments), utils (stands for utilities), steps, and local. Data subdirectories keep all the information about the data that is used in the ASR system, while the exp contains results of the training and decoding processes. Utils and steps directories contain all the required scripts for building the ASR systems. In addition to these subdirectories, templates directories have scripts files, such as run.sh, cmd.sh, and path.sh, which are used for running the speech recognition system. For a better understanding, see Figure 12

• **src:** stands for source and contains all source codes of Kaldi toolkit.
• **tools:** contains all the external tools
• **misc:** contains extra tools and information such as HTK conversion and Kaldi logo
• **window:** contain all the required tools and files for running Kaldi on Windows.
• **documentation files**, such as INSTALL and README.
Figure 12. Overview of Kaldi Directory Structure
3.5 Finite-State Transducers

The finite state transducer (FST) is a finite state automaton that labels their states with both input and output symbols to translate from the input sequence to the output sequence. Sometimes, the finite state transducer is represented labeling the transition with weight, and the input and output symbols are called by the Weighted Finite State Transducer (WFST). These weights could represent a duration, probability, penalty, etc. According to [36], WFST can be defined depending on the semiring algebraic concept which is similar to ring structure. The semiring is a set of \( K \) equipped with two binary operations: cumulative operation \( \oplus \), and associative operation \( \otimes \). Both of these operations have identity elements \( 0 \) and \( 1 \). The multiplication operation is distributive over the addition operation from both directions, and \( a \otimes 0 = 0 \otimes a = 0 \) [36]. Some of the semiring in speech recognition are listed in Table 2.

<table>
<thead>
<tr>
<th>Name</th>
<th>( K )</th>
<th>( \oplus )</th>
<th>( \otimes )</th>
<th>( 0 )</th>
<th>( 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>([0, \infty))</td>
<td>+</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Log</td>
<td>((-\infty, \infty))</td>
<td>-log((e^{-x} + e^{-y}))</td>
<td>+</td>
<td>(\infty)</td>
<td>0</td>
</tr>
<tr>
<td>Tropical</td>
<td>((-\infty, \infty))</td>
<td>min</td>
<td>+</td>
<td>(\infty)</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 13 shows a simple diagram of the Weighted finite state transducer where each state is performed by circles and labeled with different numbers. The bold circle represents the beginning state while the double circles represent the final state. Also, each transition arc's is mapped with weight, as well as both input and output. For example, the self loop of "state 1" weighted by 0.5 and is labeled by "r" as input and "y" as output.
In general, Kaldi toolkit uses the WFST in almost all the training and decoding algorithms. The decoding graph in Kaldi toolkit can be created by constructing the HCLG graph, which can be defined in Equation 30:

$$HCLG = H \circ C \circ L \circ G$$  \hspace{1cm} (30)

The functionality of each component in this equation can be described as the following:

- **G**: represents the encoding of the language model.
- **L**: represents the lexicon, where its input symbols represent phones and its output symbols represent words.
- **C**: represents context-dependency, where its input symbols refer to the context-dependent phones and its output symbols refer to the phones.
- **H**: represents the HMM definitions. Its input symbols represent the transition-ids and output symbols represent context-dependent phones.
- **o**: represents the associative binary operations on the FST.
In addition, Kaldi developers work on inserting the disambiguation symbol to minimize and determine the HCLG output. Also, they work on preserving the stochasticity of the HCLG and testing it. The HCLG approach can be summarized as[34]:

\[
HCLG = asl(min(rds(det(H' o min(det(C o min(det(L o G))))))))
\]

(31)

Where \(asl\) represents the add-self-loops, \(rds\) refers to the remove-disambiguation-symbols, and \(H'\) is \(H\) without the self-loop.
Chapter Four Work Procedure of Building ASR System
4.1 Introduction

This thesis aims to create a complete automatic speech recognition system using Kaldi toolkits. This chapter will briefly explain each step that is required to build this system, supported with figures and snapshots. Also, TIDIGITS data which is used in this thesis will be introduced in this chapter.

4.2 TIDIGITS Corpus

One of the most important requirements, that a user needs to build the ASR system, is to prepare some amount of data for testing and training purposes. This data could be a self-recorded (user sound) or common speech corpus. TIDIGITS is one of the most popular speech corpus that. This data is used for designing and evaluating algorithms for speaker-independent recognition of connected digit sequences [37]. In 1993, TIDIGITS corpus became a member of the Linguistic Data Consortium (LDC), and now it is available for downloading in [38].

TIDIGITS corpus contains more than 25102 digit sequences(recordings) spoken by almost 326 speakers with different genders (111 men, 114 women, 50 boys, and 51 girls) and different ages. Ages ranges from (21-70) for men, (17-59) for women, (8-15) for girls, and (6 -14) for boys. Each speaker worked on producing seventy-seven digit sequences. These digits sequences are included only eleven digits: oh, zero, one, two, three, four, five, six, seven, eight, and nine. These digit sequences have been collected by letting each speaker speak each isolated digit twice(22 isolated digits for all the 11 digits), such as saying “one” two times in separate recording file. In addition, each speaker should speak 11 two-digit sequences, 11 three-digit sequences, 11 four-digit sequences, 11 five-digit sequences, and 11 seven-digit sequences to get 77 digit sequences in total.

The TIDIGITS directory consists of two main sections, the adult and the children sections. The adult section contains the men and women speakers while the
children's section contains the boys and girls speakers. Each of these speaker types are divided into test and train subsets which include different numbers of speakers. Each of these speakers' sub-folders hold 77 audio files that are produced by the same speaker. Table 3 shows the distribution of speakers over the train and test:

Table 3. Speakers Distribution Over Train and Test Files [40]

<table>
<thead>
<tr>
<th>Speakers gender</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Man</td>
<td>55</td>
<td>56</td>
</tr>
<tr>
<td>Woman</td>
<td>57</td>
<td>57</td>
</tr>
<tr>
<td>Boy</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Girl</td>
<td>26</td>
<td>25</td>
</tr>
</tbody>
</table>

This corpus collects different types of dialects by selecting at least five adult speakers for both genders, male and female, from each area in the United States. Furthermore, the database recorded speech data for six adult black female and five adult black males. However, the child speakers were selected randomly, regardless of their genders. Table 4 shows 22 types of different dialects with their regions and the number of the selected speakers.

Table 4. classification of United States Dialects and Distribution of Speakers

<table>
<thead>
<tr>
<th>City</th>
<th>Dialect</th>
<th>M</th>
<th>W</th>
<th>B</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boston, MA</td>
<td>Eastern new England</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Richmond, VA</td>
<td>Virginia Piedmont</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Lubbock, TX</td>
<td>Southwest</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Los Angeles, CA</td>
<td>Southern California</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Knoxville, TN</td>
<td>South Midland</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rochester, NY</td>
<td>Central New York</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>City</td>
<td>Region</td>
<td>Speakers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
<td>----------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Denver, CO</td>
<td>Rocky Mountains</td>
<td>5 5 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Milwaukee, WS</td>
<td>North Central</td>
<td>5 5 2 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Philadelphia, PA</td>
<td>Delaware Valley</td>
<td>5 6 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kansas City, KS</td>
<td>Midland</td>
<td>5 5 4 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chicago, IL</td>
<td>North Central</td>
<td>5 5 1 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charleston, SC</td>
<td>South Carolina</td>
<td>5 5 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Orleans, LA</td>
<td>Gulf South</td>
<td>5 5 2 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dayton, OH</td>
<td>South Midland</td>
<td>5 5 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atlanta, GA</td>
<td>Gulf South</td>
<td>5 5 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miami, FL</td>
<td>Spanish American</td>
<td>5 5 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dallas, TX</td>
<td>Southwest</td>
<td>5 5 34 36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New York, NY</td>
<td>New York City</td>
<td>5 5 2 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Little Rock, AR</td>
<td>South Midland</td>
<td>5 6 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portland, OR</td>
<td>Pacific Northwest</td>
<td>5 5 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pittsburgh, PA</td>
<td>Upper Ohio Valley</td>
<td>5 5 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Black</td>
<td>5 6 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Speakers</td>
<td></td>
<td>111 114 50 51</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.3 Tools preparation

Before starting to build the ASR system, users need to prepare some important tools and prerequisites, such as:

- Download Kaldi toolkits, all its required libraries and software packages, and check Kaldi directories to ensure it is installed correctly (as described in Chapter Three).

- Make sure to download SRI Language Modeling Toolkit (SRILM) and IRST Language Modeling Toolkit (IRSTLM).

- Download the TIDIGITS corpus which is used as the data source for the ASR system.

- use Linux neither Windows because it produces fewer errors. This researcher used “ubuntu 16.04 LTS” operation system.

- Use the Unix shell and script programming language (bash).
4.4 Development of digits ASR system using Kaldi

After downloading and preparing all the ASR requirements, there are four main steps to be done by users to create the ASR system and apply Kaldi: data preparation, language preparation, training, and decoding. Each of these step will be explained in the next sections in detail.

4.4.1 Data Preparation

Data preparation is the first important step to be done before building the ASR system. This step aims to split and prepare data in the right manner before using it. Data preparation can be done as follows:

1. Specify the data that will be used during building the speech recognition system. In this thesis, TIDIGITS corpus is used. As mentioned above, this corpus has two parts: adults and children. This ASR system will be based on adults data only. Adults have 111 male speakers and 114 female speakers and each of these have almost 77 recording utterances for different digits (from zero to nine). In this case, the total speakers will be 225, and the total recordings will be almost 17325.

2. Make sure the used data is in “.wav” format. Usually, TIDIGITS data is in Sphere format, if this is the case, users can use the sph2pipe_v2.5 software to convert it. For example, the audio file “ahm_1a” which records utterances for the speaker “ahm” can be converted to the .wav file as the following:

```
ahm_1a sph2pipe -f wav ../TIDIGITS/adult/test/ahm/1a.wav
```

3. In general, TIDIGITS corpus is split in two parts: train corpus and test corpus for both adults and children. For the adults part, there are 113 speakers inside the test
folder and 112 speakers inside the train folder. In case the corpus is not divided, the user needs to split it before starting to work with it.

4. Inside "Kaldi/egs" create a folder called “digits” (or anything else). This folder will represent the new digit ASR system. Inside this folder create another folder and name it “s5” which represents the newest version.

5. Now it's time to split the data inside the new digit folder. In “kaldi/digits/s5”, create a folder called “digits_audio”. Inside “kaldi/digits/s5/digits_audio”, create two folders, “train” and “test” data folders. Copy all the data inside “../TIDIGITS/adults/test” into “kaldi/digits/s5/digits_audio/test”. Also, copy all the audio data inside “../TIDIGITS/data/adults/train” into “kaldi/digits/s5/digits_audio/train”. Therefore, the test folder will have 113 speakers and train folder will have 112 speakers.

6. Go back to “kaldi/egs/digits” directory and create a folder called “data”; inside this folder, create two sub-folders “test” and “train”. The test sub-folder is related to the test data set, and the train sub-folder is related to the train data set. Inside each of these folders create the same following files(note that these files have the same names, but they are related to a different data sets):

- **wav.scp:** This file maps each utterance said by speakers during the recording process to its related audio file. Wav.scp format can be defined as follows:

  ```
  <utterance_ID> <full_path_to_audio_file>
  ``

An example of wav.scp file:

```
ahm_zb /home/sarah/kaldi/egs/digits/s5/digits_audio/test/ahm/zb.wav
ahm_21 /home/sarah/kaldi/egs/digits/s5/digits_audio/test/ahm/zza.wav
akw_1b /home/sarah/kaldi/egs/digits/s5/digits_audio/test/akw/1b.wav
akw_87 /home/sarah/kaldi/egs/digits/s5/digits_audio/test/akw/87.wav
bnh_zz /home/sarah/kaldi/egs/digits/s5/digits_audio/test/bnh/zz.wav
# and so on...........
```
• **Speaker to Gender File (spk2gender):** This file defines the speaker's gender, which could be male (m) or female (f). In TIDIGITS corpus each speaker has a unique name called the “speaker ID”. This thesis includes 111 males and 114 females. Spk2gender file format can be defined as the following:

```
<speaker_ID> <speaker gender>
```

An example of Spk2gender file:

```
ahm m
akw f
apw f
arm m
atm m
# and so on......
```

• **Text Transcription File (text):** This file includes the text transcription for each utterance. This file can be defined as follows:

```
<utterance_ID> <text_transcription>
```

An example of Text file :

```
rpf_25a  z z z 5
rrf_12a  1 2
rsf_zz713a  z z 7 1 3
sjr_14o1a  1 4 o 1
# and so on ............
```

Also, this file can be represented as the following:

```
rpf_25a  zero zero zero five
rrf_12a  one two
rsf_zz713a  zero zero seven one three
sjr_14o1a one four oh one
# and so on ..........
• **Utterance to Speaker Mappings (utt2spk):** This file work on mapping each utterance to its speaker by applying the following format:

\(<\text{utterance\_ID}>\ <\text{speaker\_ID}>\)

An example of utt2spk file:

```
brf_zza brf
caf_12z5z14a caf
rsf_zz713a rsf
sir_14o1a sir
# and so on ........
```

• **Speaker to Utterance Mappings (spk2utt):** This file is the opposite of utt2spk file. Its works on mapping each speaker to all its utterances. The spk2utt file can be created from the utt2spk by using the “utils/utt2spk_to_spk2utt.pl” script.

• **Segmentation File (segments):** This file works on mapping the recording_ID and the start and end times of the segments to the utterance_ID. In case the "segments" file is not available, Kaldi will assign utterance_ID to recording_ID.

\(<\text{utterance\_id}>\ <\text{recording\_id}>\ <\text{segment\_begin}>\ <\text{segment\_end}>\)

7. It is very important to mention that data inside these files need to be sorted in the right way. Kaldi provides two useful scripts that could be used for checking and fixing sorting problems. These scripts are:

```
utils/fix_data_dir.sh   (fix the sorting issue)
utils/validate_data_dir.sh (check for any sorting issues)
```

8. data preparation can be done by hand or by using a file written in script (bash) language, calling it the “run.sh” file.
4.4.2 Language preparation

This section will create all the language modeling files that are required to build the digit ASR system. First, the user needs to create a special directory able to hold all language modeling files. In “kaldi/egs/digits/data” directory, create a new folder called “local”, and inside it create another folder called “dict”. Inside “kaldi/egs/digits/data/local/dict” directory, create the following files:

1. **lexicon.txt**: This file lists every word that is in the TIDIGITS dictionary with its phoneme transcriptions. In lexicon, each word should be mapped in one signal line. Also, this file should represent the silence words by “SIL” and speech noise by “<UNK>“. In general, the line format of the lexicon file is defined as:

```
<word> <phoneme 1> <phoneme 2> <phoneme 3>
```

Notice that there is a single space between the word and the word phonemes. The complete lexicon file of TIDIGITS data is shown below:

```
!SIL sil
<UNK> spn
z z iy r ow
z z ih r ow
o ow
1 w ah n
1 hh w ah n
2 t uw
3 th r iy
4 f ao r
5 f ay v
6 s ih k s
7 s eh v ah n
8 ey t
9 n ay n
```
2. **nonsilence_phones.txt**: This file list all the phones that are not silenced. The TIDIGITS non-silence phone file is defined as the following:

```
iy
th
ow
ah
ao
uw
eh
f
hh
ay
k
n
ih
r
ey
s
t
v
w
```

3. **silence_phones.txt**: This file contains all the salient phones that might be included in the corpus.

```
sil
spn
```

4. **optional_silence.txt**: This file contains only the representation of silence words “SIL”

5. **extra_questions.txt**: This file contains additional equations about a phone’s contextual information. Normally this file is created by Kaldi script.

6. In addition to all these file a “corpus .txt” file must be created, too. This file lists all the possible utterance transcriptions that could happen in this ASR system. Corpus file should be located at “kaldi/egs/digits/data/local”. The following snapshot is a sample of the file:

```
iy
th
ow
ah
ao
uw
eh
f
hh
ay
k
n
ih
r
ey
s
t
v
w
sil
spn
```
4.4.3 Preparing the lang Directory

This step is considered as the second part of the language-data preparation. After creating the “local/dict” directory, the user needs to create another required directory, called “lang”. This directory can be generated by using “prepare_lang.sh” script that is provided by Kaldi toolkit. This directory can be prepared as follows:

```bash
# the code for preparing lang directory
utils/prepare_lang.sh data/local/dict "<UNK>" data/local/lang data/lang
```

Where “dict” folder is considered as an input for “land” directory, “local/lang” is a temporary directory for running operations, and “kaldi/egs/digits/s5/data/lang” is the output for it (the location of “lang” directory). The “<UNK>” phrase is represented “spoken noise-word” that will map "oov" words to it. In general, “lang” directory consists of the files that are listed below:
1. **Phones.txt and words.txt**: These two files are considered symbol table files that used by Kaldi to map between the text and integer. Usually, Kaldi uses both “utils/int2sym.pl” and “utils/sym2int.pl” scripts to access these files.

2. **L.fst**: This file is the lexicon in the form of Finite State Transducer that transduces phones symbols to words symbols. Figure 14 represents the final lexicon, FST.

![Figure 14. The Final Representation of the Lexicon FST](image-url)
3. **L_disambig.fst**: This is the lexicon in the form of Finite State Transducer with the disambiguation symbols and self loop with (#0). However, in TIDIGITS data the “L.fst” is copied to “L_disambig.fst” because the lexicon does not have any homophones, and the Language model does not have a (#0) symbol.

4. **Topo file**: This file represents the Hidden Markov Model topology that is used to build this ASR system. In this thesis, the Bakis model method (left-to-right HMM) has been used to build this digit ASR system. Topo file can be created by using the “gen_topo.pl” Perl script which is provided by Kaldi toolkit. In the end, topo file will look like the following:

```xml
<Topology>
<TopologyEntry>
<ForPhones>
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
86 87 88 89 90
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.75 <Transition> 1 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.75 <Transition> 2 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.75 <Transition> 3 0.25 </State>
<State> 3 </State>
</TopologyEntry>
<TopologyEntry>
<ForPhones>
1 2 3 4 5 6 7 8 9 10
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.25 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.25 <Transition> 3 0.25 <Transition> 4 0.25 </State>
<State> 3 <PdfClass> 3 <Transition> 3 0.25 <Transition> 4 0.25 </State>
<State> 4 <PdfClass> 4 <Transition> 4 0.25 <Transition> 5 0.25 </State>
<State> 5 </State>
</TopologyEntry>
</Topology>
```
From the snapshot above, notice that phones have three emitting states with the standard three states (left-to-right topology). Also, there is a non-emitting final state that is used to connect the other states. For example, the value (0.75) in state zero represents the transition weight from state zero to itself, and the value (0.25) represents the transition weight from state zero to one. However, the silent phones have a different topology, which consists of one initial emitting state, one final emitting state, and three emitting states in the middle. Figure 15 represents the left-to-right HMM topology for this topo file.

Figure 15. Representation of left-to-right HMM Topology with Three Emitting States and One non-emitting State

5. **oov.txt**: This file has only one line filled out with a “<UNK>” phrase that represents any noise that happens in speech. During the training operation, all out-of-vocabulary words will be mapped to this phrase. However, in the “lexicon.txt” file, make sure that the “<UNK>” phrase is mapped to “SPN” phone, which represents a garbage phone.

6. **oov.int**: This file is the integer form of “oov.txt” file.

7. **G.fst**: This file represents the finite state transducer form of the language mode. G.fst file can be created in different methods, and in this thesis, the process of creating “G.fst” file depends on two main steps. First, create the n-gram language model in ARPA format by using SRILM language modeling tool. This can be done by using an executable file provided by SRILM tool called “ngram-count”. During
this step, two main files will be created: “lm.arpa”, which contains the n-gram language model in ARPA format, and “vocab-full.txt”. These new two files are located inside the “data/local/tmp” directory. The second step, is to build the “G.fst” by converting this ARPA language model to an FST format by using an executable file provided by Kaldi toolkit called “arpa2fst” as follows:

```
echo
echo "===== MAKING G.fst ====="
lang=data/lang
arpa2fst --disambig-symbol=#0 --read-symbol-table=$lang/words.txt $local/tmp/lm.arpa $lang/G.fst
```

8. **Phones**: In addition to all files were that mention above, lang directory has a phones folder which has different files about the phone set. Most of these files contain the same information but with different versions, such as, "txt", "int", and "cs". Other examples, are “disambig.cs”, “disambig.txt”, and “disambig.txt”.

### 4.4.4 Preparing the Configuration Files

It is a good habit to create configuration files related to the speech recognition system. Before preparing these files, the user needs to create a special sub-folder called “conf” inside “kaldi/egs/digits/s5”. Now, inside “conf” folder creates two files as follows:

- **mfcc.conf**: This text file includes some information about the MFCC feature extraction process, such as the sampling frequency that is used in the audio data. A snapshot of the “mfcc.conf” file is shown below:

```
--use-energy=false
--sample-frequency=20000
```
- **decode.config**: This file deals with the decoding operation which look like the following:

```
first_beam=10.0
beam=13.0
lattice_beam=6.0
```

### 4.4.5 Feature Extraction

After preparing all important database files, it is very important to extract the Mel-frequency cepstral coefficient (MFCCs) acoustic features and Cepstral Mean and Variance Normalization statistics (CMVN). MFCC features can be extracted by using “make_mfcc.sh” script file, while CMVN statistics can be calculated by utilizing “compute_cmvn_stats.sh” script file. Both of these Kaldi scripts are located at “steps” sub-directory and can be called inside “run.sh” scripts. The below code is a snapshot of “run.sh” file:

```
echo
echo "====== FEATURES EXTRACTION ======
echo
# specify number of jobs (divides the data in to 4 sections )
  nj=4

# Making feats.scp files
mfccdir=mfcc
steps/make_mfcc.sh --nj $nj --cmd "$train_cmd" data/train exp/make_mfcc/train $mfccdir
# Making cmvn.scp files
steps/compute_cmvn_stats.sh data/train exp/make_mfcc/train $mfccdir
```
4.4.6 Training and alignment the acoustic modeling

After building the “digits” directory and preparing the acoustic and language data, it is time to train the acoustic model and align the audio with this acoustic model. In this thesis, different training methods have been applied in order to get more reasonable results and more refined models. Each training process is followed by an alignment process because most of the advance training methods depend on alignment values of the previous trained acoustic models. In other words, aligning the audio to the reference transcript with the most current acoustic model allow the advance training algorithms to use these initial values to enhance the parameters of the model.

Most of the training is done by using special scripts provided by Kaldi toolkits. However, there are some required arguments that need to be defined in addition to these scripts to start the training process. These arguments include:

- Training data directory: “data/train” directory
- Language data directory: “data/lang” which represents the directory that has all the language model files
- Source directory: represents the directory of the previous trained model, “exp/previous-model”
- Destination directory: which contains the result of the current model “exp/current-model”
- Alignment operation required the same arguments definition except the last two steps where the source directory represents “exp/current-model” and the destination directory represents “exp/current-model-ali”.

The training methods that are applied in this thesis are explained in detail below:

1. Monophone training(mono): This is the first training that is used to train the Monophone model. This Monophone model does not use any contextual information from previous or future phone. This model is trained from a flat start by using MFCC, delta, and acceleration (delta+delta) features to be used later as
an initial block for the triphone models. Training the mono-phones can be done by using “train_mono.sh” script as described in the following snapshot:

```bash
#steps/train_mono.sh [options] <training-data-dir> <lang-dir> <exp-dir>
echo
echo "===== MONO TRAINING ====="

steps/train_mono.sh --nj $nj --cmd "$train_cmd" data/train data/lang exp/mono || exit 1
```

As shown in the above snapshot, there is no source directory definition, and that is because the mono-train is the first training pass that does not depend on any other model.

The alignment process can be done by applying the following script:

```bash
echo
echo "===== MONO ALIGNMENT ====="

steps/align_si.sh --nj $nj --cmd "$train_cmd" data/train data/lang exp/mono || exit 1
```

2. **Triphone model training** (tri1): The triphone models represent a phoneme variant in the context of two other (left and right) phonemes. Training the triphone model can be done by using the “train_deltas.sh” script. The source directory for this training method represents “exp/mono_ali” directory of the mono-alignment. In addition, triphone training models require defining more arguments, such as the number of HMM states on the decision tree and the number of Gaussians.

Triphone training can be done as follows:
Where the value “300” represents the number of HMM states, and “3000” represents the number of Gaussians.

The triphone alignment can be done as follows:

```bash
echo
echo "=============TRI1 (first triphone pass) TRAINING ==========
echo

steps/train_deltas.sh --cmd "$train_cmd" 300 3000 data/train data/lang exp/mono_ali exp/tri1 || exit 1

echo
echo "================align for the tri1=================
echo

steps/align_si.sh --nj $nj --cmd "$train_cmd"
--use-graphs true data/train data/lang exp/tri1 exp/tri1_ali
```

3. **Delta and delta-delta triphone training(tri2a):** This is the second triphone pass which is calculated by computing the delta and delta- delta features to support the features of MFCCs. In general, delta and delta-delta fractures represent the first order derivatives ($\Delta$) and the second order derivatives ($\Delta\Delta$). The features of delta ($\Delta$) are calculated on the window of the original features, while the features of delta-delta ($\Delta\Delta$) are calculated on the window of the delta features($\Delta$). This training method can be done by applying the following script code:

```bash
echo
echo "================ train tri2a [delta+delta-deltas]=============
echo

steps/train_deltas.sh --cmd "$train_cmd" 300 3000\ 
data/train data/lang exp/tri1_ali exp/tri2a
```
As shown in the code above, the (Δ+ΔΔ) training process depends on alignment results of the first triphone model “exp/tri1.ali”. However, this model does not require any aligning process because the next training method does not require its alignment results.

4. **LDA-MLLT triphone training(tri2b):** The LDA stands for the Linear Discriminate Analysis, and MLLT stands for Maximum Likelihood Linear Transform. The LDA-MLLT training method has a better performance in most of the operations in comparison to the MFCC and delta and delta-delta training methods. Linear Discriminate Analysis (LDA) applies to spliced MFCC features with left and right context. Both LDA and MLLT process the feature transformation in two steps. First, the LDA uses the feature components and reduces the feature dimension (to 40 by default) for all the data to create the HMM state. Second, the MLLT receives the reduced feature dimension from the LDA and applies the linear simple transformation to get a significant transformation for each speaker[20]. This training method can be done by using the “train_lda_mllt.sh” script as follows:

```bash

```

In the above code, I spliced the seven frames of MFCC features (three to the left and three to the right). Moreover, the LDA-MTTL alignment can be done as follows:
5. **MMI training:** As mentioned in Chapter Two, the MMI stands for the maximum mutual information which is one of the discriminative training methods that is used for speech recognition systems. In this thesis, MMI will be trained on top of the LDA-MLLT triphone model. Kaldi provides a special script called “train_mmi.sh” for MMI training. However, MMI requires creating denominator lattices which can be obtained by using the script “make_denlats.sh”. The following snapshot shows how to train the MMI on top of the LDA-MLLT, using four and three iterations:

```bash
steps/align_si.sh --nj $nj --cmd "$train_cmd" --use-graphs true \
data/train data/lang exp/tri2b exp/tri2b.ali

echo

echo "============Do MMI on top of LDA+MLLT=========="
echo

steps/make_denlats.sh --nj $nj --cmd "$train_cmd" \
data/train data/lang exp/tri2b exp/tri2b_denlats

steps/train_mmi.sh data/train data/lang\exp/tri2b.ali exp/tri2b_denlats exp/tri2b_mmi

steps/decode.sh --config conf/decode.config --iter 4 --nj $nj --cmd "$decode_cmd" \
exp/tri2b/graph data/test exp/tri2b_mmi/decode_it4

steps/decode.sh --config conf/decode.config --iter 3 --nj $nj --cmd "$decode_cmd" \
exp/tri2b/graph data/test exp/tri2b_mmi/decode_it3
```

6. **Boosted MMI training:** Re-training the previous MMI model but with boosting value equal to (0.05) as the follows:
7. **MPE training (tri2b_mpe)**: This is another discriminative training that is used for the ASR system as mentioned in Chapter Two. MPE stands for Minimum Phone Error which is considered as an alternative to the MMI. In this work, MPE is trained on top of LDA-MLLT triphone model by using the “train_mpe.sh” script that is provided by Kaldi toolkits. The code below shows how to train and evaluate MPE with four and three iterations:

```bash
echo 
echo "================ Do the same with boosting vector (b=0.05) ================
echo 
steps/train_mmi.sh --boost 0.05 data/train data/lang \
   exp/tri2b_ali exp/tri2b_denlats exp/tri2b_mmi_b0.05 
steps/decode.sh --config conf/decode.config --iter 4 --nj $nj --cmd "$decode_cmd" \
   exp/tri2b/graph data/test/exp/tri2b_mmi_b0.05/decode_it4 
steps/decode.sh --config conf/decode.config --iter 3 --nj $nj --cmd "$decode_cmd" \
   exp/tri2b/graph data/test/exp/tri2b_mmi_b0.05/decode_it3
```

8. **SAT triphone training**: SAT stands for Speaker Adaptive Training which is applied to the speech recognizer of the Hidden Markov Model (HHM) that adopts Gaussian Mixture Models (GMMs)[39]. SAT is a training technique that performs speaker and noise normalization by adapting to each single speaker with a particular data transform in order to obtain speech recognition systems with high
performance and recognition accuracy[39]. Kaldi provides a script called “train_sat.sh” to train SAT models, and it can be run as follows:

```bash
    echo
    echo "=============Do LDA+MLLT+SAT, and decode============"
    echo
    steps/train_sat.sh 300 3000 data/train data/lang exp/tri2b_ali exp/tri3b
    utils/mkgraph.sh data/lang exp/tri3b exp/tri3b/graph
    steps/decode_fmllr.sh --config conf/decode.config --nj $nj --cmd
    "$decode_cmd" \  
    exp/tri3b/graph data/test exp/tri3b/decode
```

However, after the training SAT model is finished, the acoustic model will be trained on the speaker-normalized features and not on the original features. Therefore, it is very important to remove the speaker identity from the feature before using it with the alignment process. The removing process can be done by evaluating the speaker identity using the inverse of the Feature Space Maximum Likelihood Linear Regression (fMLLR) matrix and removing it by multiplying the inverse matrix with the feature vector. The following code illustrates how to align the SAT triphone model with fMLLR:

```bash
    echo
    echo "====== Align all data with LDA+MLLT+SAT system (tri3b)====="
    echo
    steps/align_fmllr.sh --nj $nj --cmd "$train_cmd" --use-graphs true \   
    data/train data/lang exp/tri3b exp/tri3b_ali
```

In addition to all steps were that mention above, the “digits” folder required some extra folders to allow the digit ASR system to work correctly. First, the “digit” folder needs to attach two important folders, utils and steps, which both contain most of the training, decoding, and aligning scripts that are used in this work. These folders are available on
most of the Kaldi recipes, such as the voxforge recipe. Second, a script file “cmd.sh” is created for implementing the parallelization wrapper, which tends to split the data set into small parts to execute them in parallel. These parts (or jobs as it is called sometimes) are specified during the training and decoding phases. Kaldi toolkit implements different scripts and executable files for this purpose, such as “slurm.pl”, “run.pl”, “queue.pl”. However, in this thesis, the perl scrip, the “run.pl”, has been used because it is specially created for users how using their personal computers.

Please note the “path.sh” and “cmd.sh” files in Appendix B. Also, the process of creating this digital ASR system are linked and explained in “run.sh” script that is provided in the same appendix. Also take a look at Figure 15 which is illustrating the relationship among the training methods.

The results of this work will be presented on the next chapter
Figure 16. Relationships among some types of training methods in the ASR system
Chapter Five: Results and Discussion
5.1 Introduction

This chapter will evaluate all the different models that are applied to this ASR system (which is explained in Chapter Four). The decoding process for each model will also be explained in detail in this chapter, supported by snapshots and scripts code. Furthermore, this chapter will present the evaluation results, after the decoding operation, using the word error rate and sentence error rate as evaluation metrics.

5.2 Results of decoding the monophone model

After finishing the training monophone model, using “train_mono.sh” scripts, one can decode the utterance to evaluate the ASR system. The decoding process requires the decoding graph which can be created using “mkgraph.sh” script. After the decoding graph results become available, the decoding process can be done using “decode.sh” script, which can be called from the “utils” directory. However, the decoding process requires identifying the configuration file, “decode.config,” that was created in Chapter Four. The decoding operation can be done by applying the following code:

```
echo
echo "===== MONO DECODING ====="

utils/mkgraph.sh --mono data/lang exp/mono exp/mono/graph || exit 1

steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd"
exp/mono/graph data/test \ exp/mono/decode
```

As mentioned above, this ASR system will use the WER and SER rate as evaluation metrics. During the decoding process, the “decode.sh” script will call another script called “score.sh” which is available on “egs/local” to record and display results. This score script uses a program written in C++ called “compute-wer.cc” to compute both WER and SER and store it in side files called “wer_<LMW>” where the “LMW” refer to language model weight. For this thesis, the language model weight is set to 7 as the
minimum weight, and 20 as the maximum weight. These steps apply for almost all the models that are discussed in this thesis.

Table 5 shows the evaluation results for the monophone model using the WER and SER percentage. Also, this table lists the numbers of error that the system failed to recognize, which include: the number of insertions, number of deletions, and number of substitutions. From this table, the user can notice that the system has a higher WER when the LMW is less than 11 and higher than 17. However, the best WER is when the LWM is set to 12 which is equal to 0.95

Table 5. The WER% and SER% results of decoding the monophone model

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words/ no. of references words</th>
<th>WER%</th>
<th>No. of error sentences/ no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer_7</td>
<td>188</td>
<td>58</td>
<td>65</td>
<td>311 / 28196</td>
<td>1.10</td>
<td>274 / 8575</td>
<td>3.20</td>
</tr>
<tr>
<td>wer_8</td>
<td>167</td>
<td>64</td>
<td>64</td>
<td>295 / 28196</td>
<td>1.05</td>
<td>263 / 8575</td>
<td>3.07</td>
</tr>
<tr>
<td>wer_9</td>
<td>147</td>
<td>73</td>
<td>65</td>
<td>285 / 28196</td>
<td>1.01</td>
<td>254 / 8575</td>
<td>2.96</td>
</tr>
<tr>
<td>wer_10</td>
<td>137</td>
<td>76</td>
<td>65</td>
<td>278 / 28196</td>
<td>0.99</td>
<td>247 / 8575</td>
<td>2.88</td>
</tr>
<tr>
<td>wer_11</td>
<td>129</td>
<td>78</td>
<td>64</td>
<td>271 / 28196</td>
<td>0.96</td>
<td>240 / 8575</td>
<td>2.80</td>
</tr>
<tr>
<td>wer_12</td>
<td>117</td>
<td>85</td>
<td>65</td>
<td>267 / 28196</td>
<td>0.95</td>
<td>236 / 8575</td>
<td>2.75</td>
</tr>
<tr>
<td>wer_13</td>
<td>107</td>
<td>96</td>
<td>65</td>
<td>268 / 28196</td>
<td>0.95</td>
<td>238 / 8575</td>
<td>2.78</td>
</tr>
<tr>
<td>wer_14</td>
<td>100</td>
<td>104</td>
<td>65</td>
<td>269 / 28196</td>
<td>0.95</td>
<td>238 / 8575</td>
<td>2.78</td>
</tr>
<tr>
<td>wer_15</td>
<td>91</td>
<td>115</td>
<td>66</td>
<td>272 / 28196</td>
<td>0.96</td>
<td>241 / 8575</td>
<td>2.81</td>
</tr>
<tr>
<td>wer_16</td>
<td>86</td>
<td>121</td>
<td>67</td>
<td>274 / 28196</td>
<td>0.97</td>
<td>242 / 8575</td>
<td>2.82</td>
</tr>
<tr>
<td>wer_17</td>
<td>82</td>
<td>129</td>
<td>67</td>
<td>278 / 28196</td>
<td>0.99</td>
<td>245 / 8575</td>
<td>2.86</td>
</tr>
<tr>
<td>wer_18</td>
<td>79</td>
<td>139</td>
<td>68</td>
<td>283 / 28196</td>
<td>1.00</td>
<td>250 / 8575</td>
<td>2.92</td>
</tr>
<tr>
<td>wer_19</td>
<td>73</td>
<td>144</td>
<td>68</td>
<td>285 / 28196</td>
<td>1.01</td>
<td>252 / 8575</td>
<td>2.94</td>
</tr>
<tr>
<td>wer_20</td>
<td>67</td>
<td>149</td>
<td>68</td>
<td>284 / 28196</td>
<td>1.01</td>
<td>253 / 8575</td>
<td>2.95</td>
</tr>
</tbody>
</table>

5.3 Results of decoding the delta-based triphone model

For the triphone model, the “decode.sh” script will be used to decode the model and evaluate the ASR system. Also, for this model, a new decoding
graph was created using the same script “mkgraph.sh” but depending on the triphone training results. The decoding process can be done as the follows:

```
echo
echo "====== TRI1 (first triphone pass) DECODING ======

utils/mkgraph.sh data/lang exp/tri1 exp/tri1/graph || exit 1
steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd"
exp/tri1/graph data/test exp/tri1/decode
```

Evaluation results of the triphone model are shown in Table 6. The WER and SER percentages in this table show that the system was able to recognize 98.62% to 99.34% of the speech, and failed to recognize 1.38% to 0.66% of the speech depending on the weight of the language model. When the LMW is set to 20 the system was able to recognize 28009 words over the total references word which is equal to 28196 words. These results show that the triphone model has a slightly better performance than the triphone model.

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words/ no. of references words</th>
<th>WER%</th>
<th>No. of error sentences/ no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer_7</td>
<td>329</td>
<td>27</td>
<td>33</td>
<td>389/28196</td>
<td>1.38</td>
<td>335 / 8575</td>
<td>3.91</td>
</tr>
<tr>
<td>wer_8</td>
<td>299</td>
<td>30</td>
<td>32</td>
<td>361/28196</td>
<td>1.28</td>
<td>308 / 8575</td>
<td>3.59</td>
</tr>
<tr>
<td>wer_9</td>
<td>263</td>
<td>31</td>
<td>32</td>
<td>326 / 28196</td>
<td>1.16</td>
<td>279/8575</td>
<td>3.25</td>
</tr>
<tr>
<td>wer_10</td>
<td>239</td>
<td>31</td>
<td>34</td>
<td>304/28196</td>
<td>1.08</td>
<td>304/8575</td>
<td>3.04</td>
</tr>
<tr>
<td>wer_11</td>
<td>217</td>
<td>32</td>
<td>34</td>
<td>283/28196</td>
<td>1.00</td>
<td>242/8575</td>
<td>2.82</td>
</tr>
<tr>
<td>wer_12</td>
<td>193</td>
<td>33</td>
<td>35</td>
<td>261/28196</td>
<td>0.93</td>
<td>226/8575</td>
<td>2.64</td>
</tr>
<tr>
<td>wer_13</td>
<td>172</td>
<td>34</td>
<td>36</td>
<td>242/28196</td>
<td>0.86</td>
<td>212/8575</td>
<td>2.47</td>
</tr>
<tr>
<td>wer_14</td>
<td>162</td>
<td>37</td>
<td>36</td>
<td>235/28196</td>
<td>0.83</td>
<td>209/8575</td>
<td>2.44</td>
</tr>
<tr>
<td>wer_15</td>
<td>153</td>
<td>39</td>
<td>36</td>
<td>228/28196</td>
<td>0.81</td>
<td>202/8575</td>
<td>2.36</td>
</tr>
<tr>
<td>wer_16</td>
<td>136</td>
<td>42</td>
<td>36</td>
<td>214/28196</td>
<td>0.76</td>
<td>188/8575</td>
<td>2.19</td>
</tr>
<tr>
<td>wer_17</td>
<td>125</td>
<td>42</td>
<td>36</td>
<td>203/28196</td>
<td>0.72</td>
<td>177/8575</td>
<td>2.06</td>
</tr>
<tr>
<td>wer_18</td>
<td>116</td>
<td>42</td>
<td>36</td>
<td>194/28196</td>
<td>0.69</td>
<td>170/8575</td>
<td>1.98</td>
</tr>
<tr>
<td>wer_19</td>
<td>108</td>
<td>44</td>
<td>36</td>
<td>188/28196</td>
<td>0.67</td>
<td>165/8575</td>
<td>1.92</td>
</tr>
<tr>
<td>wer_20</td>
<td>101</td>
<td>50</td>
<td>36</td>
<td>187/28196</td>
<td>0.66</td>
<td>164/8575</td>
<td>1.91</td>
</tr>
</tbody>
</table>
5.4 Result of decoding triphone model (tri2a)

The scripts “mkgraph.sh” and “decode.sh” are both used to decode the delta and
delta-delta triphone model and to score the final result. The decoding process for this
model is shown in the following snap-code:

```
  echo "======decoding delta and delta-delta triphones(tri2a)======="
  utils/mkgraph.sh data/lang exp/tri2a exp/tri2a/graph
  steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd"
  exp/tri2a/graph data/test exp/tri2a/decode
```

Evaluation results for the second triphone model are listed in Table 7. This table
shows that the system was able to recognize 98.61% to 99.33% of the speech data and
failed to recognize only 1.39% to 0.67% of the speech data depending on the language
model weight. These results show that the first triphone(delta) and second triphone
(delta+delta-delta) model have almost equal WER percentages.

Table 7. WER% and SER% results of decoding the delta+delta-delta triphone model (tri2a)

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words/ no. of references words</th>
<th>WER%</th>
<th>No. of error sentences/ no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer_7</td>
<td>331</td>
<td>28</td>
<td>34</td>
<td>393 / 28196</td>
<td>1.39</td>
<td>338 / 8575</td>
<td>3.94</td>
</tr>
<tr>
<td>wer_8</td>
<td>294</td>
<td>30</td>
<td>33</td>
<td>357 / 28196</td>
<td>1.27</td>
<td>310 / 8575</td>
<td>3.62</td>
</tr>
<tr>
<td>wer_9</td>
<td>270</td>
<td>30</td>
<td>33</td>
<td>333 / 28196</td>
<td>1.18</td>
<td>290 / 8575</td>
<td>3.38</td>
</tr>
<tr>
<td>wer_10</td>
<td>237</td>
<td>32</td>
<td>33</td>
<td>302 / 28196</td>
<td>1.07</td>
<td>262 / 8575</td>
<td>3.06</td>
</tr>
<tr>
<td>wer_11</td>
<td>213</td>
<td>33</td>
<td>33</td>
<td>279 / 28196</td>
<td>0.99</td>
<td>242 / 8575</td>
<td>2.82</td>
</tr>
<tr>
<td>wer_12</td>
<td>193</td>
<td>33</td>
<td>34</td>
<td>260 / 28196</td>
<td>0.92</td>
<td>227 / 8575</td>
<td>2.65</td>
</tr>
<tr>
<td>wer_13</td>
<td>174</td>
<td>33</td>
<td>36</td>
<td>243 / 28196</td>
<td>0.86</td>
<td>215 / 8575</td>
<td>2.51</td>
</tr>
<tr>
<td>wer_14</td>
<td>161</td>
<td>35</td>
<td>36</td>
<td>232 / 28196</td>
<td>0.82</td>
<td>204 / 8575</td>
<td>2.38</td>
</tr>
<tr>
<td>wer_15</td>
<td>153</td>
<td>36</td>
<td>36</td>
<td>225 / 28196</td>
<td>0.80</td>
<td>199 / 8575</td>
<td>2.32</td>
</tr>
<tr>
<td>wer_16</td>
<td>140</td>
<td>39</td>
<td>35</td>
<td>214 / 28196</td>
<td>0.76</td>
<td>189 / 8575</td>
<td>2.20</td>
</tr>
<tr>
<td>wer_17</td>
<td>131</td>
<td>40</td>
<td>36</td>
<td>207 / 28196</td>
<td>0.73</td>
<td>182 / 8575</td>
<td>2.12</td>
</tr>
<tr>
<td>wer_18</td>
<td>119</td>
<td>42</td>
<td>37</td>
<td>198 / 28196</td>
<td>0.70</td>
<td>174 / 8575</td>
<td>2.03</td>
</tr>
<tr>
<td>wer_19</td>
<td>114</td>
<td>43</td>
<td>37</td>
<td>194 / 28196</td>
<td>0.69</td>
<td>170 / 8575</td>
<td>1.98</td>
</tr>
<tr>
<td>wer_20</td>
<td>108</td>
<td>44</td>
<td>36</td>
<td>188 / 28196</td>
<td>0.67</td>
<td>164 / 8575</td>
<td>1.91</td>
</tr>
</tbody>
</table>
5.5 Result of decoding LDA-MLLT triphone model

Decoding the LDA-MLLT triphone model using the same scripts that are mentioned earlier, show that the performance of the system has improved by reducing the number of insertions, deletions and substitutions. Also, in this model, the system has been able to recognize up to 99.48% of the total speech data, and failed to recognize only 0.52% of it. Table 8 shows the evolution results of decoding the LDA-MLLT triphone model. Also, the decoding script is shown as below:

```bash
echo
echo "==========decode tri2b [LDA+MLLT]=========="
echo

utils/mkgraph.sh data/lang exp/tri2b exp/tri2b/graph

steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd" \
    exp/tri2b/graph data/test exp/tri2b/decode
```

Table 8. WERs% and SERs% results of decoding the LDA-MLLT triphone model (tri2b)

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words / no. of references words</th>
<th>WER%</th>
<th>No. of error sentences / no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer_7</td>
<td>239</td>
<td>23</td>
<td>26</td>
<td>288 / 28196</td>
<td>1.02</td>
<td>258 / 8575</td>
<td>3.01</td>
</tr>
<tr>
<td>wer_8</td>
<td>216</td>
<td>23</td>
<td>25</td>
<td>264 / 28196</td>
<td>0.94</td>
<td>236 / 8575</td>
<td>2.75</td>
</tr>
<tr>
<td>wer_9</td>
<td>194</td>
<td>24</td>
<td>25</td>
<td>243 / 28196</td>
<td>0.86</td>
<td>217 / 8575</td>
<td>2.53</td>
</tr>
<tr>
<td>wer_10</td>
<td>178</td>
<td>25</td>
<td>25</td>
<td>228 / 28196</td>
<td>0.81</td>
<td>204 / 8575</td>
<td>2.38</td>
</tr>
<tr>
<td>wer_11</td>
<td>161</td>
<td>25</td>
<td>25</td>
<td>211 / 28196</td>
<td>0.75</td>
<td>187 / 8575</td>
<td>2.18</td>
</tr>
<tr>
<td>wer_12</td>
<td>148</td>
<td>27</td>
<td>25</td>
<td>200 / 28196</td>
<td>0.71</td>
<td>176 / 8575</td>
<td>2.05</td>
</tr>
<tr>
<td>wer_13</td>
<td>136</td>
<td>27</td>
<td>25</td>
<td>188 / 28196</td>
<td>0.67</td>
<td>165 / 8575</td>
<td>1.92</td>
</tr>
<tr>
<td>wer_14</td>
<td>128</td>
<td>27</td>
<td>25</td>
<td>180 / 28196</td>
<td>0.65</td>
<td>157 / 8575</td>
<td>1.83</td>
</tr>
<tr>
<td>wer_15</td>
<td>122</td>
<td>29</td>
<td>25</td>
<td>176 / 28196</td>
<td>0.62</td>
<td>154 / 8575</td>
<td>1.80</td>
</tr>
<tr>
<td>wer_16</td>
<td>116</td>
<td>30</td>
<td>25</td>
<td>171 / 28196</td>
<td>0.61</td>
<td>150 / 8575</td>
<td>1.75</td>
</tr>
<tr>
<td>wer_17</td>
<td>111</td>
<td>31</td>
<td>25</td>
<td>167 / 28196</td>
<td>0.59</td>
<td>146 / 8575</td>
<td>1.70</td>
</tr>
<tr>
<td>wer_18</td>
<td>101</td>
<td>34</td>
<td>25</td>
<td>160 / 28196</td>
<td>0.57</td>
<td>140 / 8575</td>
<td>1.63</td>
</tr>
<tr>
<td>wer_19</td>
<td>93</td>
<td>37</td>
<td>26</td>
<td>156 / 28196</td>
<td>0.55</td>
<td>136 / 8575</td>
<td>1.59</td>
</tr>
<tr>
<td>wer_20</td>
<td>85</td>
<td>37</td>
<td>26</td>
<td>148 / 28196</td>
<td>0.52</td>
<td>129 / 8575</td>
<td>1.50</td>
</tr>
</tbody>
</table>
5.6 Results of Discriminative Training.

As explained in Chapter Four, this thesis will explain three types of discriminative training which are, maximum mutual information (MMI), boosted maximum mutual Information (bMMI), and minimum phone error (MPE). This section will present the decoding results and the system performance of these training methods as the following:

5.6.1 Results of Decoding Maximum Mutual Information

In this thesis, the MMI on top of the LDA-MLLT model is decoded twice. During the first decoding process, the MMI model is decoded in four loops by setting the iteration option of the decoding script to four (--iter 4), while in the second decoding process the MMI will be decoded in three loops (--iter 3). Table 9 and Table 10 list the evaluation results for decoding the MMI model using iteration values equal to four and three. From these results it is seen that the system with MMI model has better performance by reducing the number of insertions to almost 55 in MMI with three loops, and to 57 insertions during decoding of the MMI with four loops. However, the system was able to recognize up to 99.54% - 99.55% of the speech data.

The following snap-code shows how to decode with three loops (iteration = 3) and four loops (iteration = 4)

```
echo "========Decoding MMI with two iterations ========="

steps/decode.sh --config conf/decode.config --iter 4 --nj $nj --cmd "$decode_cmd" \
exp/tri2b/graph data/test exp/tri2b_mmi/decode_it4

steps/decode.sh --config conf/decode.config --iter 3 --nj $nj --cmd "$decode_cmd" \
exp/tri2b/graph data/test exp/tri2b_mmi/decode_it3
```
Table 9. WERs% and SERs% results of decoding the MMI model with (--iter 3)

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words / no. of references words</th>
<th>WER%</th>
<th>No. of error sentences / no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer_7</td>
<td>166</td>
<td>23</td>
<td>36</td>
<td>225 / 28196</td>
<td>0.80</td>
<td>201 / 8575</td>
<td>2.34</td>
</tr>
<tr>
<td>wer_8</td>
<td>144</td>
<td>23</td>
<td>37</td>
<td>204 / 28196</td>
<td>0.72</td>
<td>181 / 8575</td>
<td>2.11</td>
</tr>
<tr>
<td>wer_9</td>
<td>133</td>
<td>23</td>
<td>37</td>
<td>193 / 28196</td>
<td>0.68</td>
<td>170 / 8575</td>
<td>1.98</td>
</tr>
<tr>
<td>wer_10</td>
<td>122</td>
<td>24</td>
<td>38</td>
<td>184 / 28196</td>
<td>0.65</td>
<td>164 / 8575</td>
<td>1.91</td>
</tr>
<tr>
<td>wer_11</td>
<td>114</td>
<td>24</td>
<td>38</td>
<td>176 / 28196</td>
<td>0.62</td>
<td>156 / 8575</td>
<td>1.82</td>
</tr>
<tr>
<td>wer_12</td>
<td>105</td>
<td>24</td>
<td>28</td>
<td>167 / 28196</td>
<td>0.59</td>
<td>147 / 8575</td>
<td>1.71</td>
</tr>
<tr>
<td>wer_13</td>
<td>92</td>
<td>24</td>
<td>36</td>
<td>152 / 28196</td>
<td>0.54</td>
<td>134 / 8575</td>
<td>1.56</td>
</tr>
<tr>
<td>wer_14</td>
<td>83</td>
<td>25</td>
<td>36</td>
<td>144 / 28196</td>
<td>0.51</td>
<td>126 / 8575</td>
<td>1.47</td>
</tr>
<tr>
<td>wer_15</td>
<td>77</td>
<td>27</td>
<td>36</td>
<td>140 / 28196</td>
<td>0.50</td>
<td>121 / 8575</td>
<td>1.41</td>
</tr>
<tr>
<td>wer_16</td>
<td>69</td>
<td>29</td>
<td>36</td>
<td>134 / 28196</td>
<td>0.48</td>
<td>116 / 8575</td>
<td>1.35</td>
</tr>
<tr>
<td>wer_17</td>
<td>65</td>
<td>31</td>
<td>36</td>
<td>132 / 28196</td>
<td>0.47</td>
<td>114 / 8575</td>
<td>1.33</td>
</tr>
<tr>
<td>wer_18</td>
<td>62</td>
<td>35</td>
<td>36</td>
<td>133 / 28196</td>
<td>0.47</td>
<td>115 / 8575</td>
<td>1.34</td>
</tr>
<tr>
<td>wer_19</td>
<td>59</td>
<td>36</td>
<td>36</td>
<td>131 / 28196</td>
<td>0.46</td>
<td>113 / 8575</td>
<td>1.32</td>
</tr>
<tr>
<td>wer_20</td>
<td>55</td>
<td>36</td>
<td>36</td>
<td>127 / 28196</td>
<td>0.45</td>
<td>110 / 8575</td>
<td>1.28</td>
</tr>
</tbody>
</table>

Table 10. WERs% and SERs% results of decoding the MMI model with (--iter 4)

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words / no. of references words</th>
<th>WER%</th>
<th>No. of error sentences / no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer_7</td>
<td>159</td>
<td>23</td>
<td>38</td>
<td>220 / 28196</td>
<td>0.78</td>
<td>196 / 8575</td>
<td>2.29</td>
</tr>
<tr>
<td>wer_8</td>
<td>140</td>
<td>23</td>
<td>39</td>
<td>202 / 28196</td>
<td>0.72</td>
<td>180 / 8575</td>
<td>2.10</td>
</tr>
<tr>
<td>wer_9</td>
<td>131</td>
<td>24</td>
<td>39</td>
<td>194 / 28196</td>
<td>0.69</td>
<td>174 / 8575</td>
<td>2.03</td>
</tr>
<tr>
<td>wer_10</td>
<td>114</td>
<td>24</td>
<td>39</td>
<td>177 / 28196</td>
<td>0.63</td>
<td>159 / 8575</td>
<td>1.85</td>
</tr>
<tr>
<td>wer_11</td>
<td>105</td>
<td>24</td>
<td>39</td>
<td>168 / 28196</td>
<td>0.60</td>
<td>150 / 8575</td>
<td>1.75</td>
</tr>
<tr>
<td>wer_12</td>
<td>98</td>
<td>24</td>
<td>39</td>
<td>161 / 28196</td>
<td>0.57</td>
<td>143 / 8575</td>
<td>1.67</td>
</tr>
<tr>
<td>wer_13</td>
<td>90</td>
<td>25</td>
<td>39</td>
<td>154 / 28196</td>
<td>0.55</td>
<td>136 / 8575</td>
<td>1.59</td>
</tr>
<tr>
<td>wer_14</td>
<td>78</td>
<td>27</td>
<td>39</td>
<td>144 / 28196</td>
<td>0.51</td>
<td>126 / 8575</td>
<td>1.47</td>
</tr>
<tr>
<td>wer_15</td>
<td>72</td>
<td>27</td>
<td>39</td>
<td>138 / 28196</td>
<td>0.49</td>
<td>120 / 8575</td>
<td>1.40</td>
</tr>
<tr>
<td>wer_16</td>
<td>71</td>
<td>29</td>
<td>38</td>
<td>138 / 28196</td>
<td>0.49</td>
<td>121 / 8575</td>
<td>1.41</td>
</tr>
<tr>
<td>wer_17</td>
<td>68</td>
<td>30</td>
<td>38</td>
<td>136 / 28196</td>
<td>0.48</td>
<td>119 / 8575</td>
<td>1.39</td>
</tr>
<tr>
<td>wer_18</td>
<td>66</td>
<td>33</td>
<td>37</td>
<td>136 / 28196</td>
<td>0.48</td>
<td>119 / 8575</td>
<td>1.39</td>
</tr>
<tr>
<td>wer_19</td>
<td>59</td>
<td>34</td>
<td>38</td>
<td>131 / 28196</td>
<td>0.46</td>
<td>114 / 8575</td>
<td>1.33</td>
</tr>
<tr>
<td>wer_20</td>
<td>57</td>
<td>37</td>
<td>38</td>
<td>132 / 28196</td>
<td>0.47</td>
<td>115 / 8575</td>
<td>1.34</td>
</tr>
</tbody>
</table>
5.6.2 Results of Decoding Boosting Maximum Mutual Information

The boosting MMI model can be decoded by repeating the same steps of decoding the MMI model, but with using a boosting vector equal to \( b = 0.05 \). Also, the bMMI model will be decoded twice. The first one with three loops (3 iteration) and second one with four loops (4 iteration). Table 11 and Table 12 list the evaluation results of decoding the bMMI model using different decoding iteration values. These tables indicate that the system, during this model, has high performance by reducing the numbers of errors and increasing numbers of recognized words. However, The ASR system has the same performance during the MMI and bMMI models and that can be seen from the WERs values which are almost identical.

```
echo "========Decoding bMMI with two iterations ========="
steps/decode.sh --config conf/decode.config --iter 4 --nj \$nj --cmd "$decode_cmd" \ exp/tri2b/graph data/test exp/tri2b_mmi_b0.05/decode_it4
steps/decode.sh --config conf/decode.config --iter 3 --nj \$nj --cmd "$decode_cmd" \ exp/tri2b/graph data/test exp/tri2b_mmi_b0.05/decode_it3
```

Table 11. WERs% and SERs% results of decoding the bMMI model with (--iter 3)

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words / no. of references words</th>
<th>WER%</th>
<th>No. of error sentences/ no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer_7</td>
<td>157</td>
<td>23</td>
<td>37</td>
<td>217 / 28196</td>
<td>0.77</td>
<td>194 / 8575</td>
<td>2.26</td>
</tr>
<tr>
<td>wer_8</td>
<td>142</td>
<td>23</td>
<td>37</td>
<td>202 / 28196</td>
<td>0.72</td>
<td>179 / 8575</td>
<td>2.09</td>
</tr>
<tr>
<td>wer_9</td>
<td>130</td>
<td>23</td>
<td>37</td>
<td>190 / 28196</td>
<td>0.67</td>
<td>168 / 8575</td>
<td>1.96</td>
</tr>
<tr>
<td>wer_10</td>
<td>119</td>
<td>23</td>
<td>37</td>
<td>179 / 28196</td>
<td>0.63</td>
<td>160 / 8575</td>
<td>1.87</td>
</tr>
<tr>
<td>wer_11</td>
<td>106</td>
<td>24</td>
<td>37</td>
<td>167 / 28196</td>
<td>0.59</td>
<td>148 / 8575</td>
<td>1.73</td>
</tr>
<tr>
<td>wer_12</td>
<td>94</td>
<td>26</td>
<td>37</td>
<td>157 / 28196</td>
<td>0.56</td>
<td>138 / 8575</td>
<td>1.61</td>
</tr>
<tr>
<td>wer_13</td>
<td>87</td>
<td>26</td>
<td>37</td>
<td>150 / 28196</td>
<td>0.53</td>
<td>131 / 8575</td>
<td>1.53</td>
</tr>
<tr>
<td>wer_14</td>
<td>82</td>
<td>26</td>
<td>37</td>
<td>145 / 28196</td>
<td>0.51</td>
<td>126 / 8575</td>
<td>1.47</td>
</tr>
<tr>
<td>wer_15</td>
<td>72</td>
<td>28</td>
<td>36</td>
<td>136 / 28196</td>
<td>0.48</td>
<td>118 / 8575</td>
<td>1.38</td>
</tr>
<tr>
<td>wer_16</td>
<td>68</td>
<td>31</td>
<td>36</td>
<td>135 / 28196</td>
<td>0.48</td>
<td>117 / 8575</td>
<td>1.36</td>
</tr>
<tr>
<td>wer_17</td>
<td>65</td>
<td>31</td>
<td>36</td>
<td>132 / 28196</td>
<td>0.47</td>
<td>114 / 8575</td>
<td>1.33</td>
</tr>
<tr>
<td>wer_18</td>
<td>60</td>
<td>34</td>
<td>36</td>
<td>130 / 28196</td>
<td>0.46</td>
<td>112 / 8575</td>
<td>1.31</td>
</tr>
<tr>
<td>wer_19</td>
<td>56</td>
<td>37</td>
<td>36</td>
<td>129 / 28196</td>
<td>0.46</td>
<td>111 / 8575</td>
<td>1.29</td>
</tr>
<tr>
<td>wer_20</td>
<td>54</td>
<td>37</td>
<td>36</td>
<td>127 / 28196</td>
<td>0.45</td>
<td>110 / 8575</td>
<td>1.28</td>
</tr>
</tbody>
</table>
Table 12. WERs% and SERs% results of decoding the bMMI model with (--iter 4)

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words / no. of references words</th>
<th>WER%</th>
<th>No. of error sentences / no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer_7</td>
<td>157</td>
<td>23</td>
<td>38</td>
<td>218 / 28196</td>
<td>0.77</td>
<td>195 / 8575</td>
<td>2.27</td>
</tr>
<tr>
<td>wer_8</td>
<td>140</td>
<td>23</td>
<td>38</td>
<td>201 / 28196</td>
<td>0.71</td>
<td>182 / 8575</td>
<td>2.12</td>
</tr>
<tr>
<td>wer_9</td>
<td>127</td>
<td>23</td>
<td>39</td>
<td>189 / 28196</td>
<td>0.67</td>
<td>170 / 8575</td>
<td>1.98</td>
</tr>
<tr>
<td>wer_10</td>
<td>114</td>
<td>24</td>
<td>39</td>
<td>177 / 28196</td>
<td>0.63</td>
<td>159 / 8575</td>
<td>1.85</td>
</tr>
<tr>
<td>wer_11</td>
<td>103</td>
<td>24</td>
<td>39</td>
<td>166 / 28196</td>
<td>0.59</td>
<td>148 / 8575</td>
<td>1.73</td>
</tr>
<tr>
<td>wer_12</td>
<td>94</td>
<td>24</td>
<td>39</td>
<td>157 / 28196</td>
<td>0.56</td>
<td>139 / 8575</td>
<td>1.62</td>
</tr>
<tr>
<td>wer_13</td>
<td>87</td>
<td>27</td>
<td>39</td>
<td>153 / 28196</td>
<td>0.54</td>
<td>135 / 8575</td>
<td>1.57</td>
</tr>
<tr>
<td>wer_14</td>
<td>77</td>
<td>27</td>
<td>38</td>
<td>142 / 28196</td>
<td>0.50</td>
<td>124 / 8575</td>
<td>1.45</td>
</tr>
<tr>
<td>wer_15</td>
<td>73</td>
<td>27</td>
<td>38</td>
<td>138 / 28196</td>
<td>0.49</td>
<td>120 / 8575</td>
<td>1.40</td>
</tr>
<tr>
<td>wer_16</td>
<td>72</td>
<td>29</td>
<td>37</td>
<td>138 / 28196</td>
<td>0.49</td>
<td>121 / 8575</td>
<td>1.41</td>
</tr>
<tr>
<td>wer_17</td>
<td>68</td>
<td>29</td>
<td>37</td>
<td>134 / 28196</td>
<td>0.48</td>
<td>117 / 8575</td>
<td>1.36</td>
</tr>
<tr>
<td>wer_18</td>
<td>66</td>
<td>32</td>
<td>37</td>
<td>135 / 28196</td>
<td>0.48</td>
<td>118 / 8575</td>
<td>1.38</td>
</tr>
<tr>
<td>wer_19</td>
<td>59</td>
<td>32</td>
<td>38</td>
<td>129 / 28196</td>
<td>0.46</td>
<td>112 / 8575</td>
<td>1.31</td>
</tr>
<tr>
<td>wer_20</td>
<td>57</td>
<td>38</td>
<td>38</td>
<td>133 / 28196</td>
<td>0.47</td>
<td>116 / 8575</td>
<td>1.35</td>
</tr>
</tbody>
</table>

5.6.3 Results of decoding Minimum Phone Error

The Minimum Phone Error model will also be decoded twice in order to evaluate the system and score the results. First, the MPE will be decoded with three loops(--iter 3) and then re-decoded with four loops(--iter 4). The evaluation results of this model are shown in Table 13 and Table 14. Compared to the previous model, the results from decoding this MPE model show that the performance of the system has slightly decreased due to increase in the number of error words. However, the overall system performance is considered excellent because the system was able to recognize up to 99.48% of the speech data. In other words the system failed to recognize only 146 words out of 28196 words. The snap-code below shows how to decode the MPE model using the “decode.sh” script:

```bash
# Decoding MPE with two iterations

steps/decode.sh --config conf/decode.config --iter 4 --nj $nj --cmd "$decode_cmd" \
exp/tri2b/graph data/test exp/tri2b_mpe/decode_it4
steps/decode.sh --config conf/decode.config --iter 3 --nj $nj --cmd "$decode_cmd" \
exp/tri2b/graph data/test exp/tri2b_mpe/decode_it3
```
Table 13. WERs% and SERs% results of decoding the MPE model with (--iter 3)

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words / no. of references words</th>
<th>WER%</th>
<th>No. of error sentences / no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer 7</td>
<td>230</td>
<td>24</td>
<td>30</td>
<td>284 / 28196</td>
<td>1.01</td>
<td>251 / 8575</td>
<td>2.93</td>
</tr>
<tr>
<td>wer 8</td>
<td>213</td>
<td>24</td>
<td>31</td>
<td>268 / 28196</td>
<td>0.95</td>
<td>236 / 8575</td>
<td>2.75</td>
</tr>
<tr>
<td>wer 9</td>
<td>189</td>
<td>27</td>
<td>31</td>
<td>247 / 28196</td>
<td>0.88</td>
<td>219 / 8575</td>
<td>2.55</td>
</tr>
<tr>
<td>wer 10</td>
<td>170</td>
<td>30</td>
<td>32</td>
<td>232 / 28196</td>
<td>0.82</td>
<td>206 / 8575</td>
<td>2.40</td>
</tr>
<tr>
<td>wer 11</td>
<td>159</td>
<td>31</td>
<td>32</td>
<td>222 / 28196</td>
<td>0.79</td>
<td>196 / 8575</td>
<td>2.29</td>
</tr>
<tr>
<td>wer 12</td>
<td>139</td>
<td>33</td>
<td>32</td>
<td>204 / 28196</td>
<td>0.72</td>
<td>179 / 8575</td>
<td>2.09</td>
</tr>
<tr>
<td>wer 13</td>
<td>127</td>
<td>36</td>
<td>32</td>
<td>195 / 28196</td>
<td>0.69</td>
<td>172 / 8575</td>
<td>2.01</td>
</tr>
<tr>
<td>wer 14</td>
<td>117</td>
<td>37</td>
<td>32</td>
<td>186 / 28196</td>
<td>0.66</td>
<td>163 / 8575</td>
<td>1.90</td>
</tr>
<tr>
<td>wer 15</td>
<td>102</td>
<td>41</td>
<td>31</td>
<td>174 / 28196</td>
<td>0.62</td>
<td>152 / 8575</td>
<td>1.77</td>
</tr>
<tr>
<td>wer 16</td>
<td>92</td>
<td>41</td>
<td>32</td>
<td>165 / 28196</td>
<td>0.59</td>
<td>143 / 8575</td>
<td>1.67</td>
</tr>
<tr>
<td>wer 17</td>
<td>86</td>
<td>43</td>
<td>32</td>
<td>161 / 28196</td>
<td>0.57</td>
<td>139 / 8575</td>
<td>1.62</td>
</tr>
<tr>
<td>wer 18</td>
<td>80</td>
<td>43</td>
<td>32</td>
<td>155 / 28196</td>
<td>0.55</td>
<td>135 / 8575</td>
<td>1.57</td>
</tr>
<tr>
<td>wer 19</td>
<td>76</td>
<td>45</td>
<td>31</td>
<td>152 / 28196</td>
<td>0.54</td>
<td>131 / 8575</td>
<td>1.53</td>
</tr>
<tr>
<td>wer 20</td>
<td>70</td>
<td>45</td>
<td>31</td>
<td>146 / 28196</td>
<td>0.52</td>
<td>126 / 8575</td>
<td>1.47</td>
</tr>
</tbody>
</table>

Table 14. WERs% and SERs% results of decoding the MPE model with (--iter 4)

<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words / no. of references words</th>
<th>WER%</th>
<th>No. of error sentences / no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer 7</td>
<td>233</td>
<td>24</td>
<td>30</td>
<td>287 / 28196</td>
<td>1.02</td>
<td>254 / 8575</td>
<td>2.96</td>
</tr>
<tr>
<td>wer 8</td>
<td>213</td>
<td>24</td>
<td>31</td>
<td>268 / 28196</td>
<td>0.95</td>
<td>236 / 8575</td>
<td>2.75</td>
</tr>
<tr>
<td>wer 9</td>
<td>189</td>
<td>27</td>
<td>31</td>
<td>247 / 28196</td>
<td>0.88</td>
<td>219 / 8575</td>
<td>2.55</td>
</tr>
<tr>
<td>wer 10</td>
<td>170</td>
<td>30</td>
<td>32</td>
<td>232 / 28196</td>
<td>0.82</td>
<td>206 / 8575</td>
<td>2.40</td>
</tr>
<tr>
<td>wer 11</td>
<td>159</td>
<td>31</td>
<td>32</td>
<td>222 / 28196</td>
<td>0.79</td>
<td>196 / 8575</td>
<td>2.29</td>
</tr>
<tr>
<td>wer 12</td>
<td>139</td>
<td>33</td>
<td>32</td>
<td>204 / 28196</td>
<td>0.72</td>
<td>179 / 8575</td>
<td>2.09</td>
</tr>
<tr>
<td>wer 13</td>
<td>127</td>
<td>36</td>
<td>32</td>
<td>195 / 28196</td>
<td>0.69</td>
<td>172 / 8575</td>
<td>2.01</td>
</tr>
<tr>
<td>wer 14</td>
<td>117</td>
<td>37</td>
<td>32</td>
<td>186 / 28196</td>
<td>0.66</td>
<td>163 / 8575</td>
<td>1.90</td>
</tr>
<tr>
<td>wer 15</td>
<td>102</td>
<td>41</td>
<td>31</td>
<td>174 / 28196</td>
<td>0.62</td>
<td>152 / 8575</td>
<td>1.77</td>
</tr>
<tr>
<td>wer 16</td>
<td>92</td>
<td>41</td>
<td>32</td>
<td>165 / 28196</td>
<td>0.59</td>
<td>143 / 8575</td>
<td>1.67</td>
</tr>
<tr>
<td>wer 17</td>
<td>86</td>
<td>43</td>
<td>32</td>
<td>161 / 28196</td>
<td>0.57</td>
<td>139 / 8575</td>
<td>1.62</td>
</tr>
<tr>
<td>wer 18</td>
<td>80</td>
<td>43</td>
<td>32</td>
<td>155 / 28196</td>
<td>0.55</td>
<td>135 / 8575</td>
<td>1.57</td>
</tr>
<tr>
<td>wer 19</td>
<td>76</td>
<td>45</td>
<td>31</td>
<td>152 / 28196</td>
<td>0.54</td>
<td>131 / 8575</td>
<td>1.53</td>
</tr>
<tr>
<td>wer 20</td>
<td>70</td>
<td>45</td>
<td>31</td>
<td>146 / 28196</td>
<td>0.52</td>
<td>126 / 8575</td>
<td>1.47</td>
</tr>
</tbody>
</table>
5.7 Results of decoding Speaker Adaptive Training (SAT)

In general, the Speaker Adaptive Training (SAT) model decoding process is different than the other models. The SAT model will be decoding with fMLLR using a special decoding script provided by Kaldi toolkit called “decode_fmllr.sh”. This script will result in two different results files, which are “decode” and “decode.si”. Both of these results are presented in Table 15 and Table 16. The “decode.si” file includes all the results of speaker-independent decoding pass, which is required when the script “decode_fmllr.sh” does not active the “--si-dir” option, which is used for skipping the first decoding pass. The results of speaker independent decoding pass are considered important because they will be used later to calculate the first-pass of the fMLLR transforms. Also, the SAT model need to obtain the decoding graph before applying the decoding process. For better understanding, see the following script code which shows the full script code for decoding operation:

```
<table>
<thead>
<tr>
<th>wer_&lt;LMW&gt;</th>
<th>No. of Insertion</th>
<th>No. of Deletion</th>
<th>No. of Substitution</th>
<th>No. of error words / no. of references words</th>
<th>WER%</th>
<th>No. of error sentences / no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wer_7</td>
<td>248</td>
<td>23</td>
<td>28</td>
<td>299 / 28196</td>
<td>1.06</td>
<td>261 / 8575</td>
<td>3.04</td>
</tr>
<tr>
<td>wer_8</td>
<td>228</td>
<td>23</td>
<td>28</td>
<td>279 / 28196</td>
<td>0.99</td>
<td>241 / 8575</td>
<td>2.81</td>
</tr>
<tr>
<td>wer_9</td>
<td>196</td>
<td>25</td>
<td>27</td>
<td>248 / 28196</td>
<td>0.88</td>
<td>215 / 8575</td>
<td>2.51</td>
</tr>
<tr>
<td>wer_10</td>
<td>172</td>
<td>25</td>
<td>26</td>
<td>223 / 28196</td>
<td>0.79</td>
<td>194 / 8575</td>
<td>2.26</td>
</tr>
<tr>
<td>wer_11</td>
<td>149</td>
<td>26</td>
<td>26</td>
<td>201 / 28196</td>
<td>0.71</td>
<td>174 / 8575</td>
<td>2.03</td>
</tr>
</tbody>
</table>
```

Table 15. WERs% and SERs% results of the first-pass of speaker independent decoding (SAT model)
These tables show that the system has very good performance because it reduced the number of error words to almost 127, and it made the system able to recognize up to 28069 words over 28196 reference words. In other words, the system performance during this model was up to 99.55%.

Furthermore, in order to summarize the results for each model, Kaldi toolkit provides a special script called “best_wer.sh” that is used for that purpose. This script
works on selecting the best WER% and SER% among the others for each single model. Table 17 summarizes the results of all the models that have been mented in this thesis. Also, the script code is shown in the following snap-code:

```
#!/bin/bash
for x in exp/*/decode*; do 
  [ -d $x ] && [[ $x =~ "$1" ]] && grep WER $x/wer_* | 
  utils/best_wer.sh; done
exit 0
```

Table 17. The summarized Results of different models

<table>
<thead>
<tr>
<th>Model Method</th>
<th>wer_&lt;LMW&gt;</th>
<th>No. of error words / no. of references words</th>
<th>WER%</th>
<th>No. of error sentences / no. of references sentences</th>
<th>SER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monophone</td>
<td>wer_12</td>
<td>267 / 28196</td>
<td>0.95</td>
<td>236 / 8575</td>
<td>2.75</td>
</tr>
<tr>
<td>Triphone</td>
<td>wer_20</td>
<td>187 / 28196</td>
<td>0.66</td>
<td>164 / 8575</td>
<td>1.91</td>
</tr>
<tr>
<td>∆ + ∆∆ triphone</td>
<td>wer_20</td>
<td>188 / 28196</td>
<td>0.67</td>
<td>164 / 8575</td>
<td>1.91</td>
</tr>
<tr>
<td>LDA+MLLT</td>
<td>wer_20</td>
<td>148 / 28196</td>
<td>0.52</td>
<td>129 / 8575</td>
<td>1.50</td>
</tr>
<tr>
<td>LDA+MLLT+MMI (--iter 3)</td>
<td>wer_20</td>
<td>127 / 28196</td>
<td>0.45</td>
<td>110 / 8575</td>
<td>1.28</td>
</tr>
<tr>
<td>LDA+MLLT+MMI (--iter 4)</td>
<td>wer_19</td>
<td>131 / 28196</td>
<td>0.46</td>
<td>114 / 8575</td>
<td>1.33</td>
</tr>
<tr>
<td>LDA+MLLT+bMMI (--iter 3)</td>
<td>wer_20</td>
<td>127 / 28196</td>
<td>0.45</td>
<td>110 / 8575</td>
<td>1.28</td>
</tr>
<tr>
<td>LDA+MLLT+bMMI (--iter 4)</td>
<td>wer_19</td>
<td>129 / 28196</td>
<td>0.46</td>
<td>112 / 8575</td>
<td>1.31</td>
</tr>
<tr>
<td>LDA+MLLT+MPE (--iter 3)</td>
<td>wer_20</td>
<td>146 / 28196</td>
<td>0.52</td>
<td>126 / 8575</td>
<td>1.47</td>
</tr>
<tr>
<td>LDA+MLLT+MPE (--iter 4)</td>
<td>wer_20</td>
<td>147 / 28196</td>
<td>0.52</td>
<td>127 / 8575</td>
<td>1.48</td>
</tr>
<tr>
<td>LDA+MLLT+SAT (speaker independent decoding)</td>
<td>wer_20</td>
<td>138 / 28196</td>
<td>0.49</td>
<td>119 / 8575</td>
<td>1.39</td>
</tr>
<tr>
<td>LDA+MLLT+SAT</td>
<td>wer_20</td>
<td>127 / 28196</td>
<td>0.45</td>
<td>109 / 8575</td>
<td>1.27</td>
</tr>
</tbody>
</table>
Chapter Six: Conclusion and Future Work
6.1 Conclusion

This research has been conducted to design an automatic speech recognition system using Kaldi toolkit, which is one of the most recent speech tools. This toolkit is freely available and licensed under Apache License v2.0. The adults part of the TIDIGITS corpus was used to evaluate the Kaldi Speech Recognition System. This corpus contains 17325 recordings with 111 males and 114 females, all with different accents and from different regions of the United States. This ASR system was evaluated using the Word Error Rate (WER) and Sentence Error Rate (SER).

Evaluation consists of various training methods to investigate performance on digits, namely TIDIGITS. These training models were generated: monophone, delta-base triphone (first pass), delta and delta-delta triphone (second pass), Linear Discriminate Analysis and Maximum Likelihood Linear Transform (LDA+MLLT) triphone and speaker adapting training (SAT). In addition to that, this thesis applied three discriminative training methods on top of the LDA+MLLT triphone model, such as the maximum mutual information (MMI), the boosting maximum mutual information (bMMI), and the minimum phone error (MPE). All these models have been generated and applied to TIDIGITS data by creating a scripts file written in shell (bash).

The overall system performance was in the range between 99.05% and 99.55%, depending on the training method. The best WER for the monophone model was 0.95%, and for the first and second triphone models it was 0.66% and 0.67%, respectively; for the LDA-MLLT model it was 0.52% and 0.45% for the SAT model. On the other hand, the discriminative training model generated very close WERs results. Both MMI and bMMI training methods have WERs equal to 0.45% - 0.46%, while the MPE training method has a bit higher WER, which is equal to 0.52%.
6.2 Future Work

This section will list some of the future work to be performed:

1. Build the same ASR system that was created in this work, but with different corpus, such as the children data section that is provided by TIDIGITS corpus.

2. Extending evaluation by applying the Kaldi System to other corpora, such as The New York Times Annotated Corpus, TIMIT Acoustic-Phonetic Continuous Speech Corpus, CELEX2, and English Gig-word Fifth Edition.

3. Evaluation of additional training methods, such as MMI in addition to SAT model, and fMMI+MMI in addition to SAT model.

4. The results of this work can provide useful information for the researchers to build the Deep Neural Network (nnet1), nnet2, and nnet3.
Chapter Seven References


[34] [http://kaldi-asr.org/doc/index.html](http://kaldi-asr.org/doc/index.html).

[35] [https://github.com/kaldi-asr/kaldi](https://github.com/kaldi-asr/kaldi).


[38] [https://catalog.ldc.upenn.edu/ldc93s10](https://catalog.ldc.upenn.edu/ldc93s10).


Appendices
### Appendix A  Abbreviation

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR</td>
<td>Automatic Speech Recognition</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Feed Forward Neural Network</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>WER</td>
<td>Word Error Rate</td>
</tr>
<tr>
<td>SER</td>
<td>Sentence Error Rate</td>
</tr>
<tr>
<td>LDC</td>
<td>Linguistic Data Consortium</td>
</tr>
<tr>
<td>DARPA</td>
<td>Defense Advanced Research Projects Agency</td>
</tr>
<tr>
<td>CMU</td>
<td>Carnegie Mellon University</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machine Corporation</td>
</tr>
<tr>
<td>ANNs</td>
<td>Artificial Neural Networks</td>
</tr>
<tr>
<td>HTK</td>
<td>Hidden Markov Model Toolkit</td>
</tr>
<tr>
<td>CMS</td>
<td>Cepstral Mean Subtraction</td>
</tr>
<tr>
<td>MFCC</td>
<td>Mel-Frequency Cepstral Coefficients</td>
</tr>
<tr>
<td>PLP</td>
<td>Perceptual Linear Prediction</td>
</tr>
</tbody>
</table>
CMVN  Cepstral Mean and Variance Normalization
DFT  Discrete Fourier Transform
fMLLR  Feature Space Maximum Likelihood Linear Regression
DCT  Discrete Cosine Transform
LPC  Linear Predictive Codes
GMM  Gaussian Mixture Model
MLLR  Maximum-likelihood linear regression
MAP  Maximum a Posteriori Probability Estimate
STC  Semi-tied Covariance
CMLLR  Constrained MLLR
WRR  Word Recognition Rate
FST  Finite-State Transducers
BLAS  Basic Linear Algebra Subroutines
LAPACK  Linear Algebra Package
SGMM  Subspace Gaussian Mixture model
SRILM  Stanford Research Institute Language Modeling Toolkit
IRSTLM The IRST Language Modeling Toolkit
utt2spk Utterance to Speaker
spk2utt Speaker to Utterance
spk2gender Speaker to Gender
MPE Minimum Phone Error
SAT Speaker Adaptive Training
MMI Maximum Mutual Information
MLLT Maximum Likelihood Linear Transform
LDA Linear Discriminated Analysis
WFSTs Weighted Finite State Transducers
ATLAS Automatically Tuned Linear Algebra Software
LM Language model
AM Acoustic model
MWE Minimum Word Error
Appendix B  Script Code for Building ASR system

**** cmd.sh script code****
# according to what kind of queue you have you can change this file.
export train_cmd=run.pl
export decode_cmd=run.pl
export mkgraph_cmd=run.pl

**** path.sh script code ****

# Define Kaldi directory
export KALDI_ROOT=/home/sarah/kaldi

# Define the path for the important tools
export PATH=$PWD/./utils:$KALDI_ROOT/src/bin:$KALDI_ROOT/tools/openfst/bin:
$KALDI_ROOT/src/fstbin:$KALDI_ROOT/src/gmmbin:$KALDI_ROOT/src/featbin:$KALDI_ROOT/src/lmbin:$KALDI_ROOT/src/latbin:
$KALDI_ROOT/src/fgmmbin/:

# Define audio data directory
export DATA_ROOT="/home/{user}/kaldi/egs/digits/digits_audio"

# Enable SRILM tool
. $KALDI_ROOT/tools/env.sh

# This variable is important for sorting the data
export LC_ALL=C

**** run.sh script file ****

#!/bin/bash

./path.sh || exit 1
./cmd.sh || exit 1

# 4 parallel jobs
nj=4
# language model order is equal to 1 (n-gram quantity)

```
ln_order=1
```

```
utils/parse_options.sh || exit 1
[[ $# -ge 1 ]] && { echo "Wrong arguments!"; exit 1; }
```

# this use to remove the data results of last execution
```
rm -rf exp mfcc data/train/spk2utt data/train/cmvn.scp data/train/split1
data/test/spk2utt data/test/cmvn.scp data/test/feats.scp data/test/split1 data/local/lang
data/lang/data/local/tmp data/local/dict/lexiconp.txt
echo
```

```
echo "===== PREPARING ACOUSTIC DATA ====="
echo
```

```
echo " Needs to be created as the following:"
spk2gender [speaker-id] [gender]
wav.scp [utteranceID] [full_path_to_audio_file]
text [utteranceID] [text_transcription]
utt2spk [utteranceID] [speakerID]
corpus.txt [text_transcription]"
echo
```

```
echo "wait until creating spk2utt files........"
echo
```

# creating the spk2utt files for train and test
```
utils/utt2spk_to_spk2utt.pl data/train/utt2spk > data/train/spk2utt
utils/utt2spk_to_spk2utt.pl data/test/utt2spk > data/test/spk2utt
```

```
echo
```

```
echo "spk2utt files has been created sucessfully!!"
echo
```

```
echo
```

```
echo "===== FEATURES EXTRACTION ====="
echo
```

# creating feats.scp
```
mfccdir=mfcc
```

# script for checking prepared data (train)
```
utils/validate_data_dir.sh data/train
```
utils/fix_data_dir.sh data/train
steps/make_mfcc.sh --nj $nj --cmd "$train_cmd" data/train exp/make_mfcc/ train $mfccdir

# script for checking prepared data (test)
utils/validate_data_dir.sh data/test
utils/fix_data_dir.sh data/test
steps/make_mfcc.sh --nj $nj --cmd "$train_cmd" data/test exp/make_mfcc/test $mfccdir

# creating cmvn.scp
steps/compute_cmvn_stats.sh data/train exp/make_mfcc/train $mfccdir
steps/compute_cmvn_stats.sh data/test exp/make_mfcc/test $mfccdir

echo
echo "======== PREPARING LANGUAGE DATA ======"
echo

# It should be created as the following:
#
# lexicon.txt           [<word> <phone 1> <phone 2> ...]
# nonsilence_phones.txt  [<phone>]
# silence_phones.txt    [<phone>]
# optional_silence.txt  [<phone>]
utils/prepare_lang.sh data/local/lang data/local/lang data/lang

echo
echo "======== CREATING lm.arpa ======"
echo

loc=`which ngram-count`; if [ -z $loc ]; then
  if uname -a | grep 64 >/dev/null; then
    sdir=$KALDI_ROOT/tools/srilm/bin/i686-m64
  else
    sdir=$KALDI_ROOT/tools/srilm/bin/i686
  fi
else
  echo "Using SRILM language modelling tool from $sdir"
  export PATH=$PATH:$sdir
fi
if [ -f $sdir/ngram-count ]; then
  echo "SRILM toolkit is probably not installed.
  Instructions: tools/install_srilm.sh"
else
  exit 1
fi
local=$local
mkdir $local/tmp
ngram-count -order $lm_order -write-vocab $local/tmp/vocab-full.txt -wbdiscount -text $local/corpus.txt -lm $local/tmp/lm.arpa

echo "===== CREATING G.fst ====="
echo

echo "===== MONO TRAINING ====="
echo

steps/train_mono.sh --nj $nj --cmd "$train_cmd" data/train data/lang exp/mono || exit 1

echo "===== MONO DECODING ====="
echo

utils/mkgraph.sh --mono data/lang exp/mono exp/graph || exit 1
steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd" exp/mono/graph data/test exp/mono/decode

echo "===== MONO ALIGNMENT ====="
echo

steps/align_si.sh --nj $nj --cmd "$train_cmd" data/train data/lang exp/mono exp/mono.ali || exit 1

echo "===== TRI1 (first triphone pass) TRAINING ====="
echo

steps/train_deltas.sh --cmd "$train_cmd" 300 3000 data/train data/lang exp/mono.ali exp/tri1 || exit 1

echo
echo "====== TRI1 (first triphone pass) DECODING ======"

```
utils/mkgraph.sh data/lang exp/tri1 exp1/graph || exit 1
steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd" exp/tri1/graph
data/test exp/tri1/decode
```

echo

echo"==========THE TREE PDF FILE=========="

echo
draw-tree data/lang/phones.txt exp/tri1/tree | dot -Tps -Gsize=100,100 | ps2pdf - tree.pdf

echo

echo "========align for the tri1==========="

echo

```
steps/align_si.sh --nj $nj --cmd "$train_cmd"
    --use-graphs true data/train data/lang exp/tri1/decode/tri1_ali
```

echo

echo "======= train tri2a [delta+delta-deltas]=========

echo

```
steps/train_deltas.sh --cmd "$train_cmd"
    300 3000
    data/train data/lang exp/tri1_ali/decode/tri2a
```

echo

echo " ==================decode tri2a ==================

echo

```
utils/mkgraph.sh data/lang exp/tri2a exp2a/graph
steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd"
    exp/tri2a/graph/data/test exp/tri2a/decode
```

echo

echo " =======train and decode tri2b [LDA+MLLT]=========

echo

```
steps/train_lda_mllt.sh --cmd "$train_cmd"
    --splice-opts "--left-context=3 --right-context=3"
    300 3000 data/train data/lang exp/tri1_ali exp/tri2b
```
utils/mkgraph.sh data/lang exp/tri2b exp/tri2b/graph

steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd" \
  exp/tri2b/graph data/test exp/tri2b/decode

echo

echo "=====Align all data with LDA+MLLT system (tri2b) ======"

echo

steps/align_si.sh --nj $nj --cmd "$train_cmd" --use-graphs true \
  data/train data/lang exp/tri2b exp/tri2b.ali

echo

echo "=============Do MMI on top of LDA+MLLT=============="

echo

steps/make_denlats.sh --nj $nj --cmd "$train_cmd" \
  data/train data/lang exp/tri2b exp/tri2b_denlats
steps/train_mmi.sh data/train data/lang exp/tri2b ali exp/tri2b_denlats exp/tri2b_mmi
steps/decode.sh --config conf/decode.config --iter 4 --nj $nj --cmd "$decode_cmd" \
  exp/tri2b/graph data/test exp/tri2b_mmi/decode_it4
steps/decode.sh --config conf/decode.config --iter 3 --nj $nj --cmd "$decode_cmd" \
  exp/tri2b/graph data/test exp/tri2b_mmi/decode_it3

echo

echo "============== Do the same with boosting=============="

echo

steps/train_mmi.sh --boost 0.05 data/train data/lang \n  exp/tri2b.ali exp/tri2b_denlats exp/tri2b_mmi_b0.05
steps/decode.sh --config conf/decode.config --iter 4 --nj $nj --cmd "$decode_cmd" \n  exp/tri2b/graph data/test exp/tri2b_mmi_b0.05/decode_it4
steps/decode.sh --config conf/decode.config --iter 3 --nj $nj --cmd "$decode_cmd" \n  exp/tri2b/graph data/test exp/tri2b_mmi_b0.05/decode_it3

echo

echo "===================Do MPE=======================

echo

steps/train_mpe.sh data/train data/lang exp/tri2b ali exp/tri2b_denlats exp/tri2b_mpe
steps/decode.sh --config conf/decode.config --iter 4 --nj $nj --cmd "$decode_cmd" \n  exp/tri2b/graph data/test exp/tri2b_mpe/decode_it4
```
steps/decode.sh --config conf/decode.config --iter 3 --nj $nj --cmd "$decode_cmd"
   exp/tri2b/graph data/test exp/tri2b_mpe/decode_it3

echo
echo "=============Do LDA+MLLT+SAT, and decode=============
Mellon University (CMU)
steps/train_sat.sh 300 3000 data/train data/lang exp/tri2b_ali exp/tri3b
util/mkgraph.sh data/lang exp/tri3b exp/tri3b/graph
steps/decode_fmllr.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd"
   exp/tri3b/graph data/test exp/tri3b/decode

echo
echo "===== Align all data with LDA+MLLT+SAT system (tri3b)=====
steps/align_fmllr.sh --nj $nj --cmd "$strain_cmd" --use-graphs true
   data/train data/lang exp/tri3b exp/tri3b_ali

echo
echo "=====summerizing the final results=====
for x in exp/*/decode*; do [ -d $x ] & & [[ $x =~ "$1" ]] & & grep WER $x/wer_* | 
util/best_wer.sh; done
exit 0
echo
echo "======run.sh is finished successfully ======
```