Speech Recognition in Hindi

Thesis submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Computer Science and Engineering

by

Abhisek Paul (107CS058)
Satyabrata Chayani (107CS060)

under the guidance of

Prof. Banshidhar Majhi

Department of Computer Science and Engineering
National Institute of Technology, Rourkela
Rourkela-769 008, Orissa, India

Department of Computer Science and Engineering
National Institute of Technology, Rourkela
Rourkela-769 008, Orissa, India

Certificate

This is to certify that the work in this thesis Report entitled ***"Speech Recognition in Hindi"*** submitted by ***Abhisek Paul(107CS058)*** and ***Satyabrata Chayani(107CS060)*** has been carried out under my supervision in partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science during session 2010-2011 in the department of Computer Science and Engineering, National Institute of Technology, Rourkela, and this work has not been submitted for any degree or academic award elsewhere.

Place: NIT, Rourkela
Date: 10/05/11

Prof. Banshidhar Majhi
Professor
Department of CSE
NIT Rourkela

ACKNOWLEDGEMENT

First of all we would like to express our deep sense of respect and gratitude towards our advisor **Prof. Banshidhar Majhi**, *Professor, Department of Computer Science and Engineering*, for his guidance, support, motivation and encouragement throughout the period this work was carried out. His readiness for consultation at all times, his educative comments, his concern and assistance even with practical things have been invaluable.

We would also like to convey our sincerest gratitude and indebtedness to our entire faculty members and staff of the Department of Computer Science and Engineering, NIT Rourkela, who bestowed their efforts and guidance at appropriate times without which it would have been very difficult on our part to finish the project work. A vote of thanks to our fellow students for their friendly co-operation and suggestions.

Abhisek Paul(107CS058)

Satyabrata Chayani(107CS060)

Abstract

This project is an attempt towards reducing the gap between the computer and the people of rural India, by allowing them to use Hindi language, the most common language being used by the people in rural areas. Speech recognition will, indeed, play a very significant role in promoting the technology in the rural areas. Although many speech interfaces are already available, the need is for speech interfaces in local Indian languages, hence we attempt to build a speech recognition system in Hindi, in this project.

The project report explains in brief about the basic model of a speech recognition engine and its different modules. It also briefs about the construction of the Hindi language dictionary and training the model for recognition of speech and finally testing the model for accuracy. The results of the tests have been provided and finally the report ends with the derived conclusion and recommended future work.

Contents

List of Figures

1 Introduction

In todays world, computers have become absolutely indispensable especially for the people of urban India. But for the development of the country, where most of the people live in rural areas as a whole, the technology has to reach them as well. The various computer accessories, like the keyboard and the mouse, require a certain level of expertise from the user, which cannot be expected from rural people, or for that matter, the physically challenged or blind people. Besides, they also require people to be proficient in English as well, to be able to use the computer. In a country like India, where the literacy rate is quite low, the above-mentioned constraints have to be discarded, for the technology to be able to reach the grass-root level. This is where speech recognition becomes useful.

The two main components of speech interfaces are: speech synthesiser and speech recogniser. The speech synthesiser transforms the written text into speech where as the speech recogniser understands the spoken words and changes them into text.

1.1 Existing Systems

Although speech recognition systems are already available, most of them have been built for English language. The languages model and the language dictionary that have been used in these systems are in English, and hence, they are not of much use to the rural people. However, there has been some significant work towards developing a speech recognition system in Hindi off late. [1] explains how an acoustic model for English can be used to develop an acoustic model for Hindi.

Many speech recognition software and language processing tools have been developed for this purpose. ISIP and Sphinx are the most commonly used speech recognition software in open source.

1.2 Speech Recognition - Definition and Issues

Speech recognition is the process of converting an input acoustic signal (input in audio format in the form of spoken words) and recognises the various words contained in the speech. These recognised words can be the final results, which may serve as commands and control, or they may serve as input to further language processing. In simple words, speech recognition can be put together as the ability to take the audio format as input and then generate the text format from it as output.

Speech recognition involves different steps:

1. Voice recording
2. Word boundary detection
3. Feature extraction
4. Recognition with the help of language models

1.3 Design of the System

The basic block diagram of the speech recognition system as explained in [5] has been given below:

The different components of the recognition system [5] have been de-briefed below:

1. Sound recording and Word detection Component This component takes the input from the audio recorder, preferably microphone, and identifies the words in the input signal. Word detection is usually done by using the energy and the zero crossing rate of the signal. The output of this component is then sent to the feature extractor module.

2. Feature Extractor component This component is generally responsible for generating the feature vectors for the audio signals input to it from the word detection component. It generates the MFCC (Mel Frequency Cepstrum Coefficients) which is used later to identify the audio signal uniquely.

3. Recognition System It is the most important part of the design model. It is a continuous, multi-dimensional HMM (Hidden Markov Model-based) component which takes as

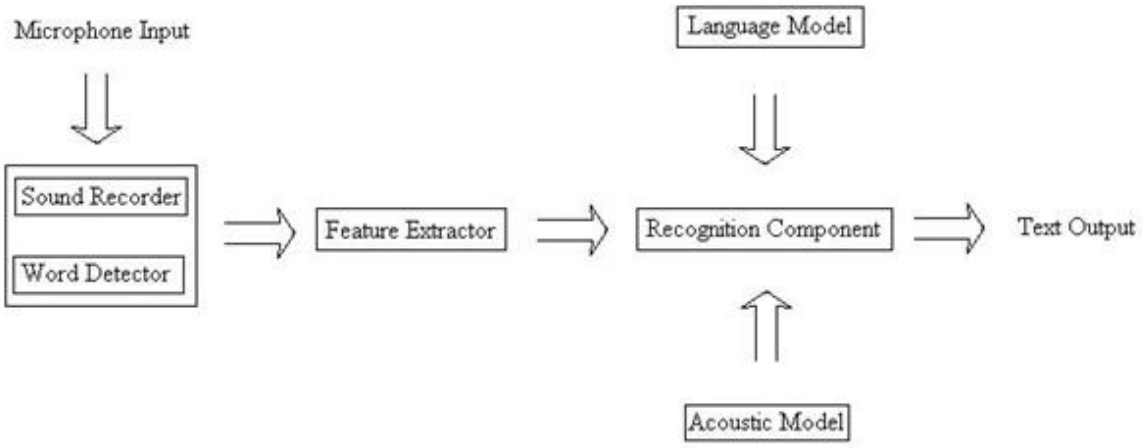

Figure 1: Block diagram of Speech Recognition system

input, the feature vectors generated from the feature extractor component and then finds the best or most suitable match from the knowledge model

4. Knowledge Model This component, along with the language dictionary, is used for identifying the sound signal.

1.4 Structure of Speech

Speech is a form of continuous audio stream where stable states get mixed with dynamically changed states. In this case, more or less similar classes of sounds called phones can be defined - words are considered to be built of phones. Different factors like phone context, style of speech, speaker and many other factors affect the acoustic property of the waveform corresponding to the phones. Moreover, co articulation makes phones sound very different from their canonical representation. Furthermore, the transitions between the phones are also very important, and thus come the concept of diphones- parts of speech between two

consecutive phones.

Often, three states in phone are selected for HMM recognition, with the first part being dependent on the previous phones, the middle part being stable and the last part being dependent on the subsequent phone.

The non-linguistic sounds are usually referred to as fillers (breath, um, uh, cough, etc.). The words and these other non-linguistic sounds form utterances- which are separate chunks of audio between pauses.

2 Sound Recording and Word Detection

This component as described in [5] is responsible for capturing the input from a microphone and then forwarding it to the feature extraction component of the system. Before forwarding to the feature extraction module, it also identifies the segments of the sound containing the words, that is, word detection. It is the responsibility of this module to save the audio files in the .WAV format which is required for further training the model.

2.1 Sound Recorder

Voice recording is the first step in speech recognition. The sound recorder takes the input from the microphone, saves these audio files in the .WAV format and finally forwards them to the next module. It also supports change of different factors like the sampling rate, the number of channels and the size of the sample. In this project, the sampling rate of the sound recorder has been kept at 16000 samples per second.

The sound recorder actually has two classes or functions implemented in it - sound reader class and sound recorder class.

1. It is the responsibility of the **Sound Reader** class to receive the input audio from the microphone, with its parameters being sampling rate, number of channels and the sample size. It has three basic functions- Open, Read and Close. The Open function opens the device in the read mode, the Read function checks if there is some valid sound present in

the audio signal and then returns the content, while the Close function releases the device.

2. It is the responsibility of the **Sound Recorder** class to convert the input audio signal from its raw format to .WAV format.

2.2 Word Detector

In speech recognition system, detection of words in sound signals is very important. It is able to detect the silence region, and thus, anything other than the region of silence is considered to be a word. Usually, the zero crossing rate is used to detect the region of silence.

For word detection, the audio signal is sampled over a particular time interval, for which the energy and the zero crossing rates are calculated. Zero crossing rate is defined as the number of times the wave goes from the positive value to the negative value and vice-versa. The first 100 milli-seconds are considered to be the region of silence during which the average zero crossing rate is calculated to determine the background noise. Upper threshold value for zero crossing rate is fixed as 2 times this value while the lower threshold value is set to 0.75 times the upper threshold.

During word detection, if the zero crossing rate goes above the upper threshold value and stays there for consecutive three sample periods, then it is assumed that word is present. Thus, recording is started and continues till the zero crossing rate falls below the lower threshold value and stays there for a minimum period of 30 milli-seconds continuously.

3 Feature Extraction

The feature extraction module [5] is one of the most important modules in the speech recognition system. To identify a spoken word, the speech recognition system has to identify and differentiate between different sounds in the same way as human beings do. However, the most important point is that although the same word, if spoken by different people, produces different types of sound waves, human beings still have the ability to understand that the spoken word is the same. This is primarily because of the fact that these words, although

spoken by different people, tend to have some common features, which makes recognition possible. Thus, in the same way, these features have to be extracted by the feature extractor module for efficient speech recognition.

The features actually get extracted over definite intervals of time, this time interval is called

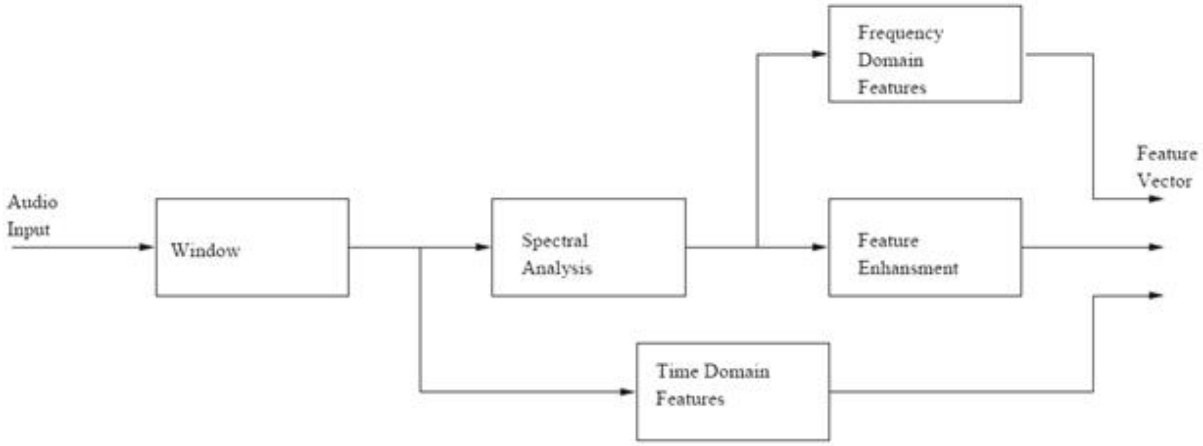

Figure 2: Block diagram of Feature Extraction module

a window and the data extracted during this interval is called the frame. Usually the features get extracted every 10 milli-seconds, this is known as the frame rate, while the window duration is usually 25 milli-seconds. Different types of windows used for feature extraction are:-

1. **Rectangular Window** : w(n) = 1
2. **Bartlett window** : w(n) = 0.5 (1 - cos (2*pi*n / (N-1)))
3. **Hamming Window** : w(n) = 0.54 - 0.46 cos (2*pi*n / (N-1)))
4. **Cosine Window** : w(n) = cos ((pi*n/n-1) - (pi/2))

Out of these windows, the Hamming window is the most commonly used window for

speech recognition system as it involves the least possible amount of disturbance or distortion.

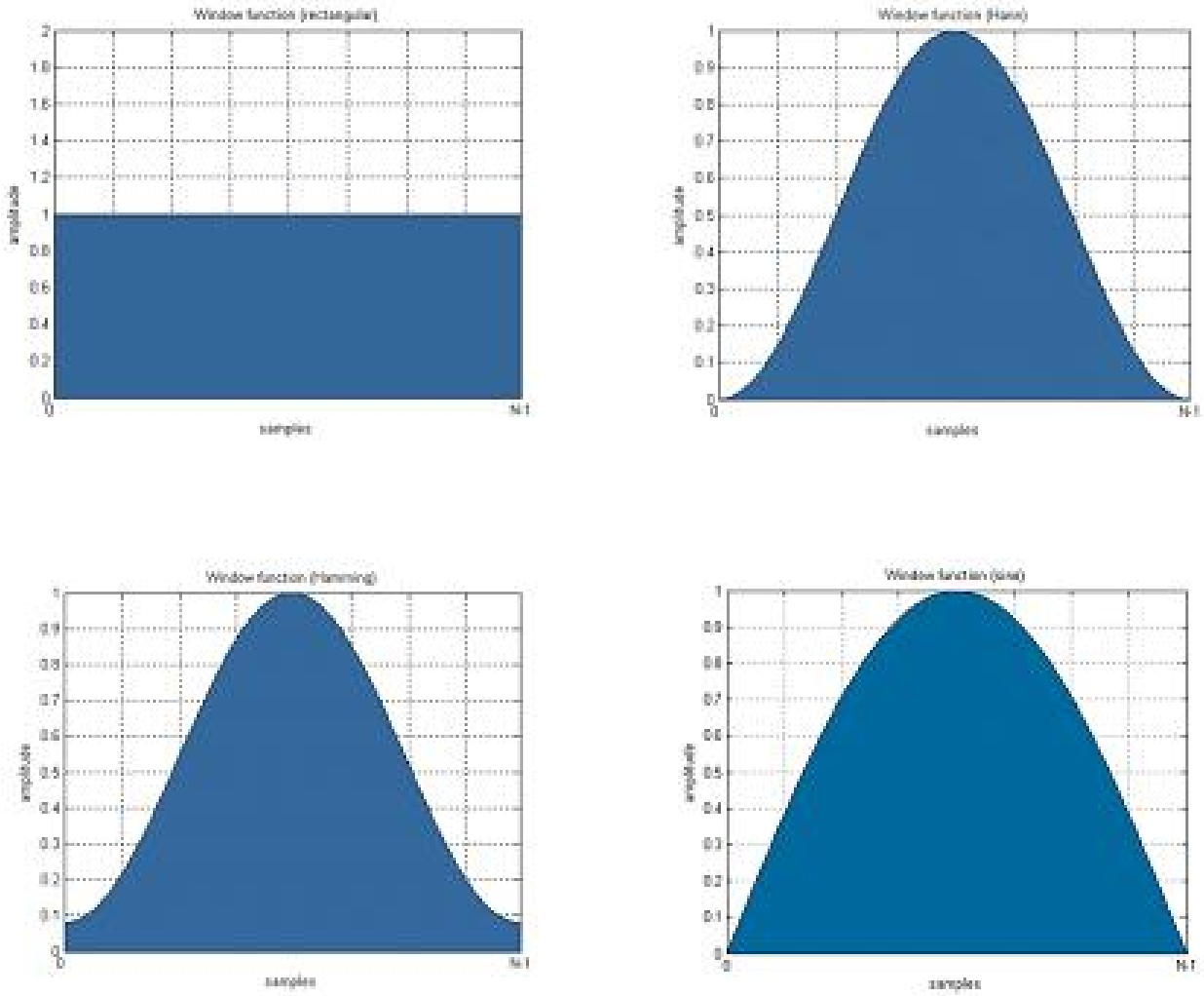

Figure 3: Different window functions (a)Rectangle (b)Bartlett (c)Hamming (d)Cosine

3.1 Mel Frequency Cepstrum Coefficients computation

Different temporal and spectral analysis is done on the sound signals to extract the useful features, the most important of them being the Mel Frequency Cepstrum Coefficients

(MFCC). MFCC is considered to be the closest possible approximation to the human ear. MFCC is generated from the sound signal by passing through high band pass filters which results in higher frequencies becoming more distinct than the lower frequencies. MFCCs are the coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip.

4 Knowledge Models

Before the speech recognition system is able to recognise the input, it is necessary to train the system. During training, the system generates the acoustic and language models from the data given by the user. These models have been described in [5].

4.1 Acoustic Model

Features that were extracted from the input sound by the extraction module have to be compared with some predefined model to identify the spoken word. This model is known as the Acoustic Model.

4.1.1 Word Model

In word model, the words are modelled as a whole. During recognition, the input sound is matched against each word present in the model and the best possible match is then considered to be the spoken word.

4.1.2 Phone Model

In phone model, only parts of words called phones are modelled instead of modelling the word as a whole. Instead of matching the sound with each word, we match the sound with

the words and recognise the parts. These parts are then put together to form a word.
In both these models, the silence and the filler words have to be modelled. Filler words are actually the ones that are produced by the human beings between two words.

4.2 Language Model

Providing a fair idea about the context and the words that can occur in the context, to the speech recognition system, is the main idea of the language model. It provides an idea about the different words that are possible in the language and the sequence in which these words may occur.
The language model developed and used in this project has been described in detail in the next chapter.

5 Implementation

5.1 Requirements for System preparation

5.1.1 Phone Set

Phoneme is the basic or the smallest unit of sound in any language. In the phone set that we have used to develop the speech recognition system for Hindi language, the phone set consists of 59 phonemes. The different phonemes that have been used are:-
A AA R I K K~N EI S T OO L P II N: Y M V U D G G B H J T: CH SH D: UU DH SHH ND~BH AI AU TH PH PH~KH KH~D~RX CHH T:H GH D:H JH DH~O O- NJ~NG~L: H: E- E J~SIL

5.1.2 Dictionary

A dictionary (also known as the pronunciation lexicon) specifies the pronunciations of the words as linear sequence of phonemes in such a way that each line in the dictionary contains

exactly one pronunciation specification. An example dictionary has been shown below:-

ADHYAYANA A DH Y A Y A N A

TARAHA T A R A H A

SAMAADHI S A M AA DH I

YUDHISHHT:HIRA Y U DH I SHH T:H I R A

KHAADA KH AA D A

The pronunciation is completely case-insensitive, that is, it is not possible to have two different pronunciations **KHAADA** and **khaada** in the dictionary.

5.1.3 Multiple Pronunciations

A word may have more than one pronunciation with each one on a separate line. They are distinguished by a unique parenthesised suffix for the word string, with the first appearance in unparenthesised form. For example:-

BAHUTA B A H U T A

BAHUTA(2) B A H OO T A

5.1.4 Filler Dictionary

In addition to the regular lexicon that contains the words in the language for which we are developing the recognition system, the Sphinx3 decoders also need a filler lexicon to define the words that are not present in the language. To be more specific, the filler dictionary specifies those words which are not present in the language model, but are still encountered during normal speech. This filler dictionary includes the special beginning-of-sentence and the end-of-sentence tokens <s> and </s> respectively as well as the silence word <sil>. However, all of them have the same SIL (silence-phone) as their pronunciation.

Example:-

<s> SIL

</s> SIL

<sil> SIL

It is important to make sure that there are no blank spaces after any line in the dictionary.

5.1.5 Transcript File

This file included the sentences (or utterances) ,consisting of the spoken text to the corresponding audio files in the following format.
<s> KRXPAYAA KARAKEI YAHA LIKHA DIIJIEI </s> (test39)
<s> MUJHEI ISA CHIIJA KAA KUCHHA PATAA NAHIIN: HAI </s> (test40)
Each word in the transcript file has to be present in the language dictionary.
Example:
PATAA P A T AA should be present in dictionary.

5.1.6 Train fileids and Test fileids

The train fileids file contained each name of the .WAV file that was spoken by the speaker for training present in the .WAV directory.
On the other hand, the test fileids contained the name of the .WAV file which was to be tested by recording.
It is importnant that each word of the corpus is present in dictionary and also in the transcript file.
For each phoneme , there must be at least one word which is present in the dictionary as well as in the transcript file.

5.2 Language Model

5.2.1 Components provided for Training

The SPHINX trainer trains the recognition system using a set of sample acoustic signals. The trainer also needs to know which sound units whose parameters it has to learn and in what sequence they appear, this information is usually provided in the transcript file using

a tag. The trainer then looks at the trainer dictionary to map the words to a sequence of sound units, besides it also uses the filler dictionary to map the non-speech sounds to the corresponding non-speech or speech-like sound units. Thus the components provided for training as given in [7] are:-

1. Trainer source code
2. Acoustic signals
3. Transcript file
4. Language dictionary
5. Filler dictionary

5.2.2 Components provided for Decoding

The data to be recognised are usually referred to as the test data. Altogether, the different tools provided for decoding are:-

1. Decoder source code
2. Language dictionary
3. Filler dictionary
4. Language model
5. Test data
6. Acoustic models (provided during training the system)

5.2.3 Setting up the system

First of all, a directory tutorial had to be created and then move to that directory. The detailed procedure for setting up the data and the trainer and then performing a preliminary training and decode run are explained in detail in [7]. The Sphinxbase usually provides two databases for testing the data initially from which either one can be chosen. In this project, AN4 database was chosen, which is a quite small database. AN4 was downloaded to the tutorial directory and the contents were then extracted using the commands as given below:-

- **mkdir tutorial**

- **cd tutorial**
- **gunzip -c an4_sphere.tar.gz | tar xf -**

5.2.4 Setting up the trainer

The SphinxTrain, Sphinxbase and Sphinx3 were also downloaded to the tutorial directory and installed from the respective tar files and their contents were extracted by using the commands:-

- **gunzip -c SphinxTrain.nightly.tar.gz | tar xf -**
- **gunzip -c sphinxbase.nightly.tar.gz | tar xf -**
- **gunzip -c sphinx3.nightly.tar.gz | tar xf -**

For compilation, the following commands were used:-

- **cd SphinxTrain**
- **./configure**
- **make**
- **cd ../sphinxbase**
- **./configure**
- **make**
- **cd ../sphinx3**
- **make**
- **make install**

After the code compilation, the tutorial was set up by copying all the executables and scripts to the same folder where the data was present. For this, the following command is required:-

- **cd sphinx3**
- **perl scripts/setup_tutorial.pl an4**

5.2.5 Performing a preliminary training run

After setting up the trainer we carry out a preliminary training run. The speech recognition component cannot work directly with the input test acoustic signals. These signals have to be first converted to a sequence of feature vectors, in this case the MFCCs which are compared with the features of the stored signals and the test data is identified. To perform this parameterization, the following command is used:-

- **perl scripts_pl/make_feats.pl -ctl etc/an4_train.fileids**

For each word, a sequence of 13-dimensional vectors called feature vectors consisting of the MFCCs will be computed by the above command. The MFCCs generated will be stored in the /feats directory.

Now within the /scripts_pl directory there were several directories numbered from 00 to 99. We enter each directory and run the slave*.pl file or in case there is a single .pl file in the directory, we run it.

- perl scripts_pl/00.verify/verify_all.pl
- perl scripts_pl/10.vector_quantize/slave.VQ.pl
- perl scripts_pl/20.ci_hmm/slave_convg.pl
- perl scripts_pl/30.cd_hmm_untied/slave_convg.pl
- perl scripts_pl/40.buildtrees/slave.treebuilder.pl
- perl scripts_pl/45.prunetree/slave.state-tying.pl
- perl scripts_pl/50.cd_hmm_tied/slave_convg.pl
- perl scripts_pl/90.deleted_interpolation/deleted_interpolation.pl
- perl scripts_pl/99.make_s2_models/make_s2_models.pl

We could have otherwise run the RunAll.pl script as following which also does the same:-

- perl scripts_pl/RunAll.pl

After we ran these commands, several new directories were created within the current direc-

tory which contained files which were generated and were extremely essential in the course of training. A .html file appeared in the directory which contained the status report of the job being launched and its progress. It was checked that the current slave*.pl script got completed successfully before launching the next slave*.pl. These steps have been explained in [7].

It has to be noted that during this process some of the steps are not required for the creation of semi-continuous models and these scripts when invoked do nothing and are skipped.

perl scripts_pl/00.verify/verify_all.pl:

It is mainly responsible for checking whether the dictionary and the filler dictionary agree with the phone list. It also checks whether all the words in the transcript file are also there in the dictionary and also ensures that there are no duplicate entries present in the dictionary. Besides it also ensures that all the phones present in the phone list appear at least once.

perl scripts_pl/10.vector_ vector_quantize/slave.VQ.pl:

This step was skipped because of continuous model.

perl scripts_pl/20.ci_hmm/slave_convg.pl:

This step trains the context independent (CI) models for the sub-words present in our dictionary.

perl scripts_pl/30.cd_hmm_untied/slave_convg.pl:

This step trains the Context-Dependent models with untied states. They are required to build the decision trees so that the states can be tied.

perl scripts_pl/40.buildtrees/slave.treebuilder.pl:

This script generates for each subword unit their corresponding decision trees.

perl scripts_pl/45.prunetree/slave.state-tying.pl:

This step prunes the decision trees and ties the states.

perl scripts_pl/50.cd_hmm_tied/slave_convg.pl:
This script trains the final models for the triphones in the corpus.txt file which are called the CD-tied models.

During training of the model, a few noncritical errors appeared like:
This step had 6 ERROR messages and 2 WARNING messages. Please check the log file for details.
The main causes of such errors were small amount of data in an4 and the bad quality of recordings.

6 Pruning and Performance Parameters

To restrict the active search space to manageable limits, pruning is perormed, so that the less promising state likelihoods are discarded during the recognition process.

6.1 Beam Pruning

At each instant, the decoder has a number of active HMMs to match with the next frame of input speech. But it has to first eliminate those whose state likelihoods fall below certain threshold value, with respect to the best HMM state likelihood at that instant.

6.2 Absolute Pruning

Even with beam pruning, sometimes the number of active entities may become very large to compute. If a large number of HMMs come within the pruning threshold, they are kept active. However, when their number grows beyond certain limits, the chances of finding out the correct word are considerably reduced. In such cases, the active search space is not allowed to grow beyond limits. It can be restricted using pruning parameters that restrict

the number of active entities at any point of time.

6.3 Pruning Parameters

The pruning parameters are the following:

- **-beam:** Determines which HMMs remain active at any given instant during recognition.
- **-pbeam:** Determines which active HMM can undergo transition to its successor in the tree at any instant.
- **-wbeam:** Determines which words are recognized by the decoder.
- **-maxhmmpf:** Determines the number of HMMs that can remain active at any instant.
- **-maxwpf:** Determines the number of distinct words recognized at any given frame.
- **-subvqbeam:** Determines its active mixture components at any frame.

7 Creating Language Model for Hindi

A new experiment folder was created and named as /final directory under the /tutorial directory.. The setup is copied from the directory /tutorial/an4 to the directory /tutorial/final, using the simple command:-

- **cd an4**
- **perl scripts_pl/copy_setup.pl -task final**

For language modelling, we used the Version 2 of the CMU Cambridge Statistical Language Modelling Toolkit. This toolkit has been explained in detail in [8].

The generation of the language model consists of the following steps as explained in [8]:
1. Initially the corpus.txt which contains the sentences and utterances in the spoken text corresponding to the acoustic signals in the ./wav directory, is used to compute the word frequencies

- **cat corpus.txt | ./text2wfreq > a.wfreq**

2. Then the vocabulary (a.vocab file) is generated from the a.wfreq file using the wfreq2vocab

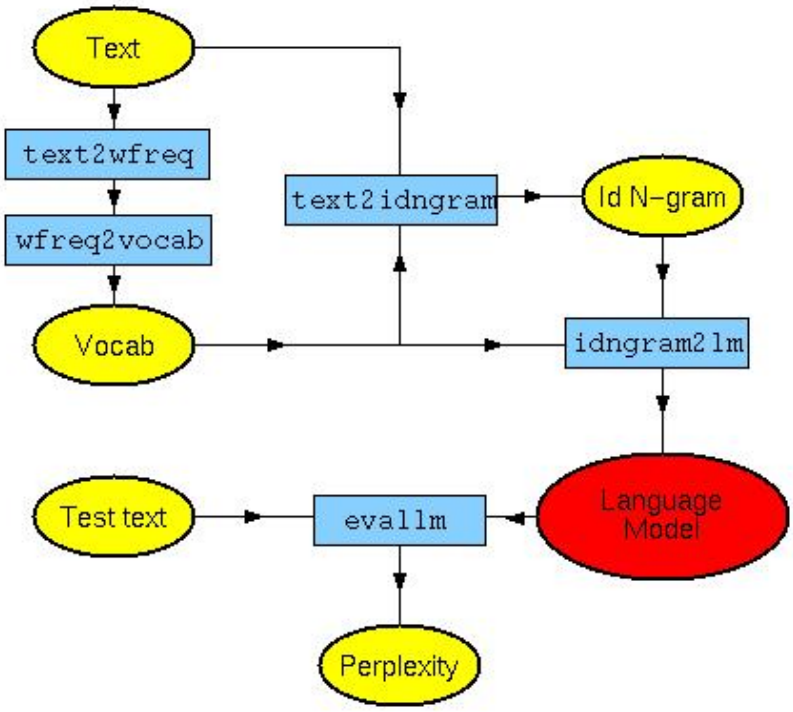

Figure 4: Flow diagram of Language Model

script.

- **cat a.wfreq | ./wfreq2vocab -top 20000 > a.vocab**

3. Based on this vocabulary, a binary id 3-gram of the training text is created using the text2idngram script.

- **cat a.text | ./text2idngram -vocab a.vocab > .idngram**

4. The idngram is then converted into a binay format language model using the idngram2lm script.

- **./idngram2lm -idngram a.idngram -vocab a.vocab -binary a.binlm**

5. Then the language model is created which is in the .arpa format which is recognised by the sphinx speech recogniser using the binlm2arpa script.

- **./binlm2arpa -binary a.binlm -arpa final.arpa**

6. The Binary Dump file necessary for recognition is created by installing the lm3g2dmp utility from the link mentioned below:

https://cmusphinx.svn.sourceforge.net/svnroot/cmusphinx/trunk/share/lm3g2dmp

Sphinx recognises the .DMP format which is generated by the following command

- **./lm3g2dmp input directory/corpus.arpa output directory**

Then the different sound units that we used to train the system are kept in the language dictionary /final/etc/final.dic and the filler dictionary /final/etc/final.filler, and the sound units in these files are noted. The list of all the standard phonemes is also mentioned in the file /final/etc/final.phone. After re-designing these units, the file /final/etc/sphinx_train.cfg was altered accordingly:-

CFG_DICTIONARY = our training dictionary with full path.

CFG_FILLERDICT = our filler dictionary with full path.

CFG_RAWPHONEFILE = our phone list with full path.

CFG_HMM_TYPE = this variable could have the values .semi. or .cont.. The most common choice for SPHINX-3 is .cont..

CFG_STATESPERHMM = it could be any integer, but the values recommended are 3 or 5. Here we have used the value 5.

CFG_SKIPSTATE = this can be set to either no or yes. Here we have set this value to yes.

CFG_N_TIED_STATES = this number can be set to any value between 500 and 2500. In this case, we have set the value to 1000.

CFG_CONVERGENCE_RATIO = this value is set to a value lying between 0.1 and 0.001. This value is defined as the ratio of the difference in likelihood values between the current iteration and the previous iteration of Baum-Welch and the total likelihood value in the previous iteration of Baum-Welch. In this case, we have set this value to 0.04

Once all the desired changes were made, we trained a new set of models, by re-running all the slave*.pl scripts from the directories /final/scripts_pl/00* through /final/scripts_pl/99* , or simply by running perl scripts_pl/RunAll.pl.

Then we changed the decoder parameters, affecting the recognition results, by editing the file /final/etc/sphinx_decode.cfg . Changing the decoding parameters present in the configuration file, we decoded several times without retraining the acoustic models to find out as to which values of the different parameters give the best results.

The different configuration parameters of sphinx_decode.cfg file are:-

DEC_ CFG_ GAUSSIANS = it counts the number of densities used by the decoder in the recognition model. In this case, since we trained continuous models,it created intermediate models where the number of Gaussians was 1, 2, 4, 8, etc.

DEC_ CFG_ MODEL_ NAME = it is the model name. By default, it uses the context dependent (CD) tied state models but can also use the CD untied and CI models to analyse how accuracy changes.

DEC_ CFG_ LANGUAGEWEIGHT = it defines the language weight and usually the value ranges between 6 and 13.

7.1 lt-sphinx3_decode parameters

7.1.1 Primary Flags

The minimum parameters that we needed to provide were the input and output databases or files:

Model definition input file: **-mdef**

Acoustic model files: **-mean, -var, -mixw, -tmat, -subvq**

Main and filler lexicons: **-dict, -fdict**

Language model binary dump file: **-lm**

Filler word probabilities: **-fillpen, -fillprob, -silprob**

7.1.2 Additional Configuration Flags

These are the additional parameters needed to obtain the right decoder configuration.
Feature type configuration: **-cmn, -agc, -varnorm, -lowerf, -upperf, -nfilt, -sampre**
control file: **-ctloffset, -ctlcount**

8 Results

To test data we used the following shell script:
##For recording data
rec -r 16000 /home/bablu/tutorial/final/wav/test.wav

For playing the test data.
play /home/bablu/tutorial/final/wav/test.wav

##For converting wav file to MFCC
./wave2feat -verbose yes -ei wav -di /home/bablu/tutorial/final/wav -mswav
yes -eo mfc -do /home/bablu/tutorial/final/feat/ -alpha 0.97 -dither yes - doublebw
no -n_lt 40 -ncep 13 -lowerf 133.33334 -upperf 6855.4976 -n_t 512 -wlen 0.0256 -c
/home/bablu/tutorial/final/etc/project_test.fileids

For decoding test data
lt-sphinx3_decode
-mdef /home/bablu/tutorial/final/model_architecture/final.ci.mdef -senmgau .cont.
-mean /home/bablu/tutorial/final/model_parameters /final.ci_cont/means
-var /home/bablu/tutorial/final/model_parameters/final.ci_cont/variances
-mixw /home/bablu/tutorial/final/model_parameters/final.ci_cont/ mixture_weights
-tmat /home/bablu/tutorial/final/model_parameters/final.ci_cont/ transition_matrices
-lw 23 -feat 1s_c_d_dd -beam 1e-120 -wbeam 1e-80 -/home/bablu/tutorial/final/etc/final.dic
-fdict /home/bablu/tutorial/final/etc/final.filler

-lm /home/bablu/tutorial/final/etc/final.ug.lm.DMP

-wip 0.2 -ctl /home/bablu/tutorial/final/etc/final_test.fileids

-ctlo_set 0 -ctlcount 1 -cepdir /home/bablu/tutorial/final/feat

-cepext .mfc -hyp /home/bablu/tutorial/final/result/final-1-1.match

-agc none -varnorm no -cmn current

Input data :

ISAKEI ATIRIKTA KACHCHII VA ADHAPAKII MACHHALIYOON: KAA SEIVANA KADAAPI NA KIYAA JAAYEI

Output:

INFO: utt.c(195): Processing: testINFO: feat.c(1134): At directory /home/bablu/tutorial/final /featINFO: feat.c(377): Reading mfc _le: /home/bablu/tutorial/final/feat/test.mfc[0..-1]INFO: cmn.c(175): CMN: 4.56 0.28 0.08 0.22 -0.17 -0.12 -0.01 -0.15 -0.01 -0.13 0.02 -0.11 -0.05

.....

INFO: fast_algo_struct.c(397): HMMHist[0..0](test): 767(100)INFO: lm.c(950): 2269307 tg(), 2133753 tgcache, 135545 bo, 2816 _lls, 63 in mem (15.3Backtrace(test) FV:test> WORD SFrm EFrm AScr(UnNorm) LMScore AScr+LScr AScale fv:test> <sil> 0 114 20062080 -59089 20002991 20575472 fv:test> ISA 115 136 478572 -95685 382887 678950 fv:test> <sil> 137 143 810830 -59089 751741 913218 fv:test> KARA 144 187 553258 -163914 389344 1502484 fv:test> KII 188 212 -304787 -123174 -427961 168131 fv:test> <sil> 213 223 1365539 -59089 1306450 1601337 fv:test> AN:KA 224 246 19773 -169283 -149510 525230

fv:test> THII 247 276 1109971 -151293 958678 1597874 fv:test> VA 277 306 782948 -160827 622121 1140490 fv:test> PAN:PA 307 344 745489 -169773 575716 1413909

fv:test> KI 345 365 -184050 -169794 -353844 87331

fv:test> MACHHALIYOON: 366 422 291205 -169773 121432 1665013

fv:test> KAA 423 442 -155944 -21527 -177471 -27273

fv:test> DEIRA 443 487 1962580 -183409 1779171 2911046

fv:test> PAN:PA 488 525 142783 -169794 -27011 738801

fv:test> AADHII 526 566 359401 -169794 189607 1115775

fv:test> NA 567 583 -109915 -161863 -271778 187951

fv:test> KIYAA 584 612 -190718 -37711 -228429 251749

fv:test> JAAYEI 613 652 1084740 -21527 1063213 1740168

fv:test> <sil> 653 766 19854433 -59089 19795344 20388884

FV:test> TOTAL 48678188 -2375497

FWDVIT: ISA KARA ATIRIKTA AN:KA THII VA ADHAPAKII MACHHALIYOON: KAA DEIRA PAN:PA AADHII NA KIYAA JAAYEI

(test)FWDXCT: test S 60257593 T 48354157 A 48678188 L -324031 0 20062080 -7675 <sil> 115 478572 -12903 ISA 137 810830 -7675 <sil> 144 553258 -22650 KARA 188 -304787 -16830 KII 213 1365539 -7675 <sil> 224 19773 -23417 AN:KA 247 1109971 -20847 THII 277 782948 -22209 VA 307 745489 -23487 PAN:PA 345 -184050 -23490 KI 366 291205 -23487 MACHHALIYOON: 423 -155944 -2309 KAA 443 1962580 -25435 DEIRA 488 142783 -23490 PAN:PA 526 359401 -23490 AADHII 567 -109915 -22357 NA 584 -190718 -4621 KIYAA 613 1084740 -2309 JAAYEI 653 19854433 -7675 <sil> 767 INFO: stat.c(156): 767 frm; 0 cdsen/fr, 177 cisen/fr, 0 cdgau/fr, 158 cigau/fr, Sen 0.01, CPU 0.01 Clk [Ovrhd 0.01 CPU 0.01 Clk]; 526 hmm/fr, 67 wd/fr, Search: 0.07 CPU 0.07 Clk (test) INFO: corpus.c(661): test: 0.6 sec CPU, 0.6 sec Clk; TOT: 0.6 sec CPU, 0.6 sec Clk INFO: stat.c(206): SUMMARY: 767 fr; 0 cdsen/fr, 177 cisen/fr, 0 cdgau/fr, 158 cigau/fr, 0.01 xCPU 0.01 xClk [Ovhrd 0.01 xCPU 0 xClk]; 526 hmm/fr, 66 wd/fr, 0.07 xCPU 0.07 xClk; tot: 0.08 xCPU, 0.08 xClkroot 23425 64.0 0.2 4932 2636 pts/0 S+ 03:24 0:00 lt-sphinx3_decode -mdef

/home/bablu/tutorial/final/model_architecture/final.ci.mdef -senmgau .cont. -mean /home/bablu/tutorial/final/model_parameters/final.ci_cont/means -var /home/bablu/tutorial/final/model_parameters/final.ci_cont/variances -mixw /home/bablu/tutorial/final/model_parameters/final.ci_cont/mixture_weights -tmat /home/bablu/tutorial/final/model_parameters/final.ci_cont/transition_matrices -lw 7 -feat 1s_c_d_dd -beam 1e-120 -wbeam 1e-80 -dict

/home/bablu/tutorial/final/etc/final.dic -fdict /home/bablu/tutorial/final/etc/final.filler -lm /home/bablu/tutorial/final
/etc/final.ug.lm.DMP -wip 0.2 -ctl /home/bablu/tutorial/final/etc/final_test.fileids -ctlo_set 0 -ctlcount 1 -cepdir
/home/bablu/tutorial/final/feat -cepext .mfc -hyp /home/bablu/tutorial/final/result/final-1-1.match -agc none -varnorm no -cmn currentro ot 23426 0.0 0.1 4652 1088 pts/0 S+ 03:24 0:00 sh -c ps aguxwww | grep sphinx3_decoderoot 23428 0.0 0.0 4044 708 pts/0 R+ 03:24 0:00 grep sphinx3_decode

9 Conclusion

In this project, sound in the form of Hindi speech was recorded with the help of microphone. A Hindi language dictionary and a filler dictionary was created along with the transcript file, which was used for training a word-based acoustic model.

This model was then used to recognise words spoken by other speakers. This speech recognition system gave quite satisfactory results when it was tested for words whose corresponding sound signals had been used to train the model.

10 Future Work

In this project, the speech recognition system in Hindi along with the Hindi language model has been developed and tested. The trainer and the decoder configuration files have several parameters, these parameters can be tested to improve the efficieny of the speech recognition system.

Besides, other than Hindi, this speech recognition system can be further extended to include other regional languages as well, by developing their corresponding language dictio-

naries. This will add to the Development of Multimodal User Interface (DMU).

References

[1] Samudravijaya K and Maria Barot.*A comparison of public domain software tools for speech recognition.* Workshop on Spoken Language Processing, 2003, pages 125-131.

[2] N. Rajput M. Kumar and A. Verma.*A large-vocabulary continuous speech recognition system for hindi.*IBM Journal for Research and Development

[3] L.R. Bahl, P.F. Brown, P.V. deSouza, R.L. Mercert. *A tree-based statistical language model for natural language speech recognition* IEEE Transaction on Acoustic, Speech, Signal Processing, Vol.37, July 1989, pages 1001-1008.

[4] L. R. Bahl, S.V. De Gennaro, P.S.Gopalakrishnan, R.L. Mercer.*A fast approximate acoustic match for large vocabulary speech recognition* IEEE Transactions on Speech and Audio Processing, Jan. 1993, pages 59-67.

[5] Ripul Gupta.*Speech Recognition for Hindi* M.Tech Thesis, IIT Bombay

[6] L. R. Rabiner.*A tutorial on hidden markov models and selected applications in speech recognition* Proceedings of the IEEE, 1989, pages 256-286.

[7] http://www.speech.cs.cmu.edu/sphinx/tutorial.html *Robust group's Open Source Tutorial.* Carnegie Mellon University.

[8] http://www.speech.cs.cmu.edu/SLM/toolkit_documentation.html *The CMU-Cambridge Statistical Language Modeling Toolkit v2*