

Project Report On

Hindi Speech Recognition

Submitted By:

Niket Khandelwal	111070008
Tanmay Inamdar	111070014
Gaurav Deshmukh	111070016
Jignesh Jain	111070050

Bachelor of Technology, Computer Engineering
2014-2015

Under the guidance of
Prof. Manasi Kulkarni



Department Of Computer Engineering and Information Technology
Veermata Jijabai Technological Institute
(Autonomous Institute Affiliated to Mumbai University)
Matunga, Mumbai - 400019

Statement by the Candidates

We wish to state the work embodied in this report titled “**Hindi Speech Recognition**” forms our group contribution to the work carried out under the Guidance of **Prof. Manasi Kulkarni** at **Veermata Jijabai Technological Institute**. This work has not been submitted for any other Degree or Diploma of any University/Institute. Wherever references have been made to previous works of others, it has been clearly indicated.

Mr. Niket Khandelwal

Mr. Tanmay Inamdar

Mr. Gaurav Deshmukh

Mr. Jignesh Jain



VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE
(Autonomous Institute Affiliated to University of Mumbai)
Matunga, Mumbai - 400019

CERTIFICATE

This is to certify that the following students have satisfactorily carried out work for the Project
entitled

Hindi Speech Recognition

in partial fulfilment of B.Tech in Computer Engineering of the University of Mumbai during the
academic year 2014-2015

Group Members:

Niket Khandelwal	111070008
Tanmay Inamdar	111070014
Gaurav Deshmukh	111070016
Jignesh Jain	111070050

Internal Examiner

External Examiner

CERTIFICATE

This to certify that this team of B. Tech students has completed the project
“Hindi Speech Recognition”

Prof. Mrs. Manasi Kulkarni

Project Guide

Dr. G. P. Bhole

Head of Department

ACKNOWLEDGEMENT

We would like to thank our project guide and mentor, Prof. Manasi Kulkarni for contributing her time and effort to help us with our project. Her constant support and encouragement, along with the numerous inputs and suggestions she gave, have been invaluable and have been a driving force which has constantly motivated us to explore different aspects of our project at each phase to make sure that the project is designed and implemented in an appropriate manner and that our conclusions are appropriate.

We also want to thank our Head of Department, Dr. G. P. Bhole, for encouraging us and providing us guidance on documenting our thesis.

ABSTRACT

Hindi is a widely used language in India that is written in Devanagari script. It is spoken by about 260 million speakers, and yet the input methods for Hindi are obscure and are difficult to learn.

This project aims to solve this problem by providing a finite vocabulary dictionary of Hindi speech-to-text system. We use MFCC (Mel Frequency Cepstral Coefficients) as features which are given input to DTW (Dynamic Time Warping) algorithm and HMM (Hidden Markov Model) for single speaker and multiple speaker voice recognition respectively. It is a proof-of-concept project that can be extended to develop a full-fledged speech-to-text system. Our system can also be used as a voice command system.

ORGANISATION OF REPORT

This report is organized in the form of following five chapters:

Chapter 1: Introduction

This chapter gives a brief overview about the project in terms of its motivation, importance, goal and the approach that is used to achieve the goal. It also provides definitions and terms that are widely used throughout the book.

Chapter 2: Literature Review

This chapter provides summary of the most important research done in the field of Speech Recognition. It also states some limitations in the research performed and list of features adopted in to the project from already existing research. A list of Algorithms used in the project have been provided and explained.

Chapter 3: Analysis and Design

This chapter gives details about system design by presenting the architecture diagram, the use-case diagram as well as activity diagram for proposed system. Brief explanation of each module within the architecture diagram has also been provided.

Chapter 4: Implementation and Testing

This chapter presents the implementation of the approach stated in earlier chapters. It gives details about how the training and testing dataset was created. Source code and screenshots of the working of individual modules are also provided along with the observations from testing.

Chapter 5: Conclusion and Future Work

This chapter provides the conclusions drawn from testing. It also states the work that can be done in future in order to further enhance the proposed system.

TABLE OF CONTENTS

1. Introduction	9
1.1. Introduction	10
1.2. Motivation	10
1.3. Problem Statement	11
1.4. Definitions	12
2. Literature Review	13
2.1. Existing solutions and their limitations	14
2.2. Existing work in Speech Recognition	14
2.3. Features to Adopt	15
2.4. Initial approach	16
2.5. Algorithms Used	16
2.6. Python Libraries	20
3. Analysis and Design	22
3.1. Architecture Diagram	23
3.2. Use case Diagram	26
3.3. Activity Diagram	29
4. Implementation	33
4.1. Source Code	34
4.2. Screenshots	60
4.3. Training Dataset	64
4.4. Testing	64
5. Conclusion and Future Work	69
5.1. Conclusion	70
5.2. Limitations and future work	70
5.3. References	71

Chapter 1

Introduction

1.1 Introduction:

Hindi (हिन्दी), or more precisely Modern Standard Hindi, is a standardised and Sanskritized register of the Hindustani language. Hindustani is the native language of most people living in Delhi, Uttar Pradesh, Uttarakhand, Chhattisgarh, Himachal Pradesh, Chandigarh, Bihar, Jharkhand, Madhya Pradesh, Haryana, and Rajasthan. Hindi is one of the two official languages of India (the other being English).

Hindi is written in Devanagari script, which is also used to write Marathi, Konkani, Bodo, Maithili and Nepali, possibly with little modifications. Devanagari consists of 11 vowels and 33 consonants and is written from left to right. The order of letters in Devanagari is based on phonetic principles, that consider both the manner and place of articulation of the consonants and vowels they represent. In Devanagari, there is one-to-one correspondence between the sound and the character that represents that sound.

1.2 Motivation:

In spite of such a large number of users of Hindi, as well as structure of Devanagari script, the input methods for Devanagari remain obscure, both on computers as well as mobile devices.

The traditional method to write Hindi in Devanagari is to use keyboard layouts which map the keys on QWERTY keyboard to Devanagari characters - InScript, Typewriter, Phonetic layouts being the notable ones. These keyboard layouts require a lot of training to achieve acceptable speeds, and are incompatible with each other. The same problem exists with the specialized Devanagari keyboards.

The other alternative, which is commonly used to write Hindi language is what is termed as transliteration systems. In this method, Latin (Roman) characters are used to represent Devanagari characters, which are then converted to Devanagari. However, since the standard latin alphabet contain only 26 characters, which are used to map to 44 Devanagari characters, the mapping is not one-to-one. Several transliteration conventions are proposed to resolve this

problem, for example Hunterian system, ISO 15919, and WX. However, there is no general agreement to resolve this, and different methods are incompatible to each other.

1.3 Problem Statement:

The primary aim of the project is to develop an easier alternative for existing input systems for Hindi language. We propose a speech-to-text solution to solve this problem.

The major reason is the obscurity of the existing solutions. As explained in the previous section, the keyboard layouts, or dedicated keyboards for Devanagari are often not intuitive, require a lot of training and different layouts are incompatible with each other. The other class of input methods, i.e. transliteration systems, also disagree in the conventions and thus are incompatible with each other.

The second reason is that none of the existing systems seem to focus on the key fact of Hindi -- there is one-to-one correspondence between the character and sound associated with it. This is the key feature of Indic languages (amongst others), which is not true for many Latin derived languages such as English. This fact can be exploited by a speech-to-text system.

Speech-to-text system addresses both of the issues -- 1) there is no convention involved in this approach, and 2) it can exploit the phonetic nature of the Devanagari script. We aim to develop an isolated word detection system for finite vocabulary, which is an elementary solution. However, with using sophisticated approaches like nested HMMs, and language model to develop a full-fledged speech-to-text system for Hindi language.

We also note that this system has many advantages. It can be used by an illiterate person to convert his spoken words to textual format, thus replacing a human interpreter. It can also be used by handicapped people who cannot use conventional keyboards to input in Hindi.

1.4 Definitions:

Following is the list of definitions commonly used terms throughout this document. Other terms will be defined before their usage.

1. **Isolated word detection:** the process of mapping an audio input to its corresponding word.
2. **MFC (Mel-Frequency Cepstrum):** a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.
3. **MFCC (Mel-Frequency Cepstral Coefficients):** coefficients that make up an MFC.
4. **DTW (Dynamic Time Warping):** an algorithm for measuring similarity between two temporal sequences which may vary in time or speed.
5. **Markov Model:** a statistical model used to model randomly changing systems where it is assumed that future states depend only on the present state.
6. **HMM (Hidden Markov Model):** a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states.

Chapter 2

Literature Review

2.1 Existing solutions and their limitations:

There are several attempts of speech to text for Indic languages. However, almost all of them are closed source as well as paid, therefore analysis and study of these systems is infeasible. We describe the most notable two:

The work by IBM and C-DAC has resulted in a sophisticated speech to text system [1] for Hindi called 'Shrutlekhan-Rajbhasha'. The techniques used in this implementation are unknown, but the main feature of this software is that it is claimed to work over various dialects and variants of Hindi language.

Recently, Google keyboard [2] has included voice typing support for Hindi. This is a free (but closed source) application for Android platform. Basic testing has demonstrated low degree of accuracy.

2.2 Existing Work in Speech Recognition:

MFCC:

The first step in any automatic speech recognition system is to extract features from the input audio data i.e. identify the components of the audio signal that are good for identifying the linguistic content and discarding all the other stuff which carries information like background noise, emotion etc.

Mel Frequency Cepstral Coefficients (MFCCs) are a feature widely used in automatic speech and speaker recognition. They were introduced by Davis and Mermelstein in the 1980's, and have been state-of-the-art ever since.

One of the approaches to speech recognition is to study the human speech. Sounds generated by a human are filtered by the shape of vocal tract i.e. tongue, teeth etc. The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.

HMM for Speech Recognition:

‘A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition’ by Rabiner [3] is a recognized and very useful paper. It explains the mathematical foundations of HMMs and the three problems and their solutions using HMM in the field of Speech Recognition:

1. Compute probability of observation sequence given the HMM
2. Choose state sequence given the observation sequence and the HMM
3. Adjust the parameters of HMM to maximize the probability of observation sequence

In our application, relevant problems are 1 and 3.

2.3 Features to adopt:

The audio is recorded at the sample rate of 44100 Hz, i.e. there are 44100 audio samples recorded per second. The number of channels used for recording is 1.

We use MFCC, delta and delta delta coefficients as the features in this implementation. The information about MFCC is given in an earlier section. Here we explain how to extract features from the recorded audio. Following is the basic overview of MFCC feature extraction. The detailed instructions can be found in [4]

1. Frame the signal into short frames.
2. For each frame calculate the periodogram estimate of the power spectrum.
3. Apply the mel filterbank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filterbank energies.
5. Take the DCT of the log filterbank energies.
6. Keep DCT coefficients 2-13, discard the rest.
7. Calculate delta and delta-delta coefficients (first and second order differences)

These features are used to train the HMM using Baum Welch algorithm [5]. There is one HMM per word, in finite vocabulary implementation. In more complex implementations, there are successive sophisticated layers of HMMs, from phones to language models. This is related to the problem number 3 described in [4]. Then, these trained HMMs are used to find the best matching word. This is the problem number 1 in [4].

2.4 Initial approach:

Phone is the basic unit of sound that possesses distinct physical or perceptual properties, and it serves as the basic unit of phonetic speech analysis. Our initial approach was to divide the recording into individual word, and divide each word into individual phones. Then the system would identify the phone and match it to corresponding Devanagari character. Thus, it would reconstruct the Devanagari representation of word.

The identification was to be done by DTW algorithm[6], which compares two temporal sequences for similarity. However, the duration of each phone is different, and the pronunciation of phones is context sensitive. Therefore, segmentation of words into phones turned out to be a very difficult task. This approach is called beads-on-a-string model, and is known to be a difficult and inefficient approach.

2.5 Algorithms used:

MFCC:

The calculation of MFCC coefficients has been explained in an earlier section.

DCT:

Discrete Cosine Transform is one of the most important steps of MFCC calculation. A **discrete cosine transform (DCT)** expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering. Formally, the discrete cosine transform is a linear, invertible function $f : \mathbb{R}^N \rightarrow \mathbb{R}^N$ (where \mathbb{R} denotes the set of real numbers), or equivalently an invertible $N \times N$ square matrix.

The N real numbers x_0, \dots, x_{N-1} are transformed into the N real numbers X_0, \dots, X_{N-1} :

$$X_k = \frac{1}{2}(x_0 + (-1)^k x_{N-1}) + \sum_{n=1}^{N-2} x_n \cos \left[\frac{\pi}{N-1} nk \right] \quad k = 0, \dots, N-1.$$

The DCT step in **MFCC** decorrelates the energies which means diagonal covariance matrices can be used to model the features in e.g. a HMM classifier. But notice that only 12 of the 26 DCT coefficients are kept. This is because the higher DCT coefficients represent fast changes in

the filterbank energies and it turns out that these fast changes actually degrade ASR performance, so we get a small improvement by dropping them.

Dynamic Time Warping (DTW):

Dynamic time warping is an algorithm for measuring similarity between two sequences that may vary in time or speed. For instance, similarities in walking patterns would be detected, even if in one video the person was walking slowly and if in another he or she were walking more quickly, or even if there were accelerations and deceleration during the course of one observation. DTW has been applied to video, audio, and graphics – indeed, any data that can be turned into a linear representation can be analyzed with DTW.

A well-known application has been automatic speech recognition, to cope with different speaking speeds. In general, it is a method that allows a computer to find an optimal match between two given sequences (e.g., time series) with certain restrictions. That is, the sequences are "warped" non-linearly to match each other. This sequence alignment method is often used in the context of hidden Markov models.

This example illustrates the pseudocode of the dynamic time warping algorithm when the two sequences s and t are strings of discrete symbols.

For two symbols x and y , $d(x, y)$ is a distance between the symbols, e.g. $d(x, y) = |x - y|$

```
int DTWDistance(s: array [1..n], t: array [1..m]) {
    DTW := array [0..n, 0..m]

    for i := 1 to n
        DTW[i, 0] := infinity
    for i := 1 to m
        DTW[0, i] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := 1 to m
            cost:= d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j],
                                         DTW[i, j-1],
                                         DTW[i-1, j-1])

    return DTW[n, m]
}
```

Hidden Markov Model (HMM):

A **hidden Markov model (HMM)** is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (*hidden*) states.

In simpler Markov models (like a Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a *hidden* Markov model, the state is not directly visible, but output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. Note that the adjective 'hidden' refers to the state sequence through which the model passes, not to the parameters of the model; the model is still referred to as a 'hidden' Markov model even if these parameters are known exactly.

1) **Forward-backward Algorithm:**

The forward-backward algorithm is an inference algorithm for hidden Markov models which computes the posterior marginals of all hidden state variables given a sequence of observations/emissions $o_{1:t} := o_1, \dots, o_t$, i.e. it computes, for all hidden state variables $X_k \in \{X_1, \dots, X_t\}$, the distribution $P(X_k \mid o_{1:t})$. This inference task is usually called

smoothing. The algorithm makes use of the principle of dynamic programming to compute efficiently the values that are required to obtain the posterior marginal distributions in two passes. The first pass goes forward in time while the second goes backward in time; hence the name *forward-backward algorithm*.

In the first pass, the forward-backward algorithm computes a set of forward probabilities which provide, for all $k \in \{1, \dots, t\}$, the probability of ending up in any particular state given the first k observations in the sequence, i.e. $P(X_k | o_{1:k})$. In the second pass, the algorithm computes a set of backward probabilities which provide the probability of observing the remaining observations given any starting point k , i.e. $P(o_{k+1:t} | X_k)$. These two sets of probability distributions can then be combined to obtain the distribution over states at any specific point in time given the entire observation sequence:

$$P(X_k | o_{1:t}) = P(X_k | o_{1:k}, o_{k+1:t}) \propto P(o_{k+1:t} | X_k)P(X_k | o_{1:k})$$

The last step follows from an application of the Bayes' rule and the conditional independence of $o_{k+1:t}$ and $o_{1:k}$ given X_k .

As outlined above, the algorithm involves three steps:

1. computing forward probabilities
2. computing backward probabilities
3. computing smoothed values.

2) **Baum Welch Algorithm:**

The Baum-Welch algorithm is used to find the unknown parameters of a hidden Markov model (HMM). The Baum-Welch algorithm uses the well-known EM algorithm to find the maximum likelihood estimate of the parameters of a hidden Markov model given a set of observed feature vectors.

Let X_t be a discrete hidden random variable with N possible values. We assume the $P(X_t | X_{t-1})$ is independent of time t , which leads to the definition of the time independent stochastic transition matrix

$$A = \{a_{ij}\} = P(X_t = j | X_{t-1} = i).$$

The initial state distribution (i.e. when $t = 1$) is given by

$$\pi_i = P(X_1 = i).$$

The observation variables Y_t can take one of K possible values. The probability of a certain observation at time t for state j is given by

$$b_j(y_t) = P(Y_t = y_t | X_t = j).$$

Taking into account all the possible values of Y_t and X_t we obtain the K by N matrix $B = \{b_j(y_i)\}$.

An observation sequence is given by $Y = (Y_1 = y_1, Y_2 = y_2, \dots, Y_t = y_t)$.

Thus we can describe a hidden Markov chain by $\theta = (A, B, \pi)$. The Baum–Welch algorithm finds a local maximum for $\theta^* = \max_{\theta} P(Y|\theta)$. (i.e. the HMM parameters θ that maximise the probability of the observation.)

2.6 Python libraries:

1. NumPy:

NumPy is the fundamental package for scientific computing with Python. This is required by some other libraries used, and it is also used for implementation of algorithms like DCT (Discrete Cosine Transform). [14]

2. SciPy:

SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering. We use it for calculation of commonly used mathematical operations like calculating mean and standard deviation of arrays. [15]

3. Matplotlib:

Matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. We use this library for plotting graphs. [16]

4. PyAudio:

PyAudio provides Python bindings for PortAudio v19, the cross-platform audio I/O library. We used PyAudio for recording the training data as well as for recording the input audio.

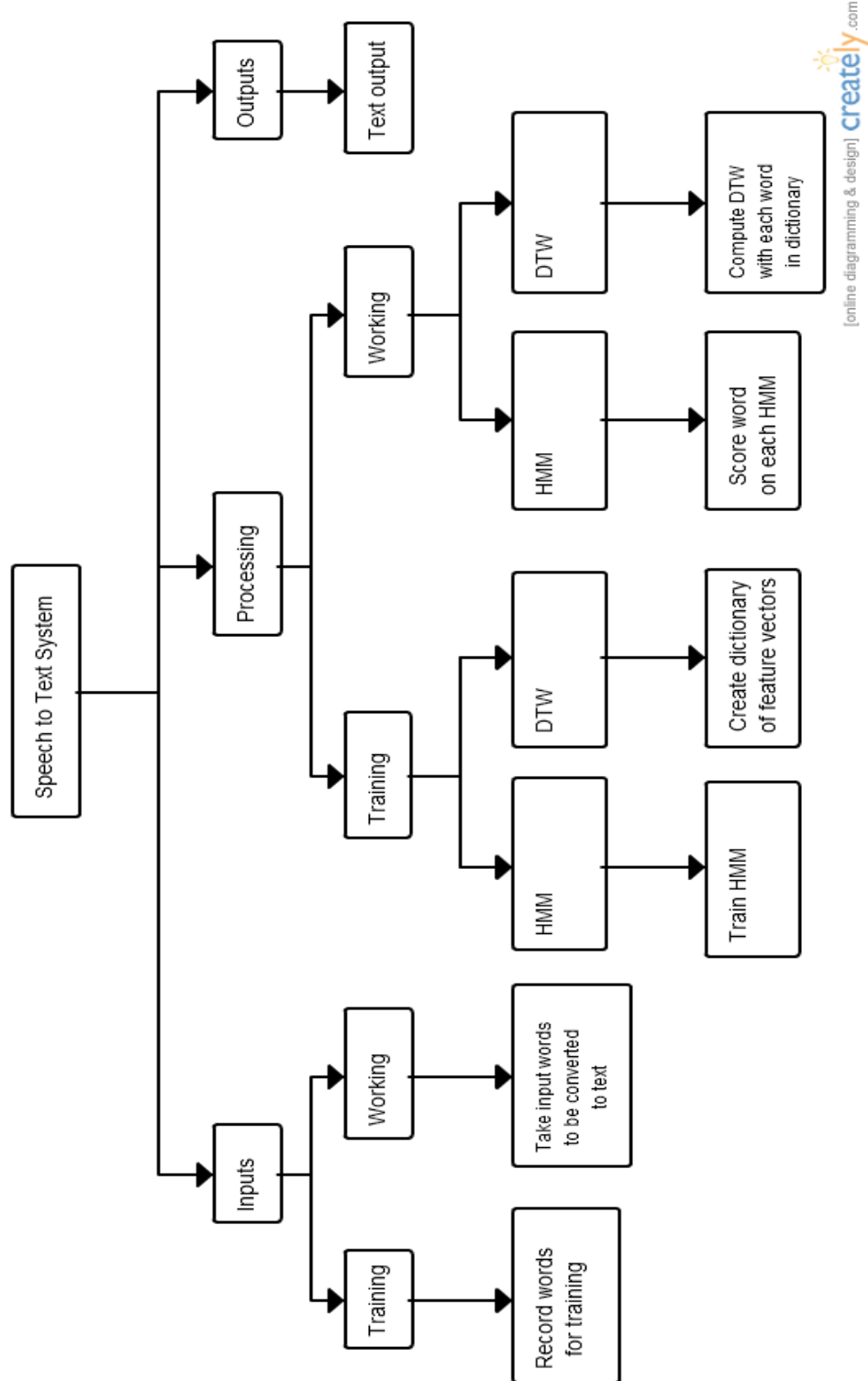
5. Hmmlearn

hmmlearn is a set of algorithm for learning and inference of Hidden Markov Models. This library provides the implementations of algorithms like Forward-Backward Algorithm and Baum Welch Algorithm.

Chapter 3

Analysis and Design

3.1 Architecture Diagram



1. **Inputs:**

a. **Training:**

For training, developers of the system collect the samples from different users with different qualities of voices and dialects.

b. **Working:**

When the system is in the working stage, the user gives his/her input in the form of audio that he/she intends to convert into textual format.

2. **Processing:**

a. **Training:**

Training of the system happens for two algorithms, HMM and DTW. Initially, the input audio is split into words by identifying silence, and non-silence regions. Then, the MFCC features are extracted from each word.

i. **HMM:**

One HMM is created for every word in the dictionary, that is initialized to a random state. Then it is trained using the sample data using Baum Welch algorithm that is described earlier. This algorithm sets the parameters of the HMM for the particular word.

ii. **DTW:**

For DTW, the extracted features of words are stored in the form of dictionary to process later.

b. **Working:**

In working, the system runs one of the two algorithms: HMM and DTW. HMM gives good results for multiple speakers, whereas DTW works better for single speaker.

i. **HMM:**

Each word is scored against every HMM, using Forward-Backward algorithm. It outputs the log-likelihood that the word matches the word corresponding to the word of HMM. Then the word corresponding to the maximum log-likelihood is chosen.

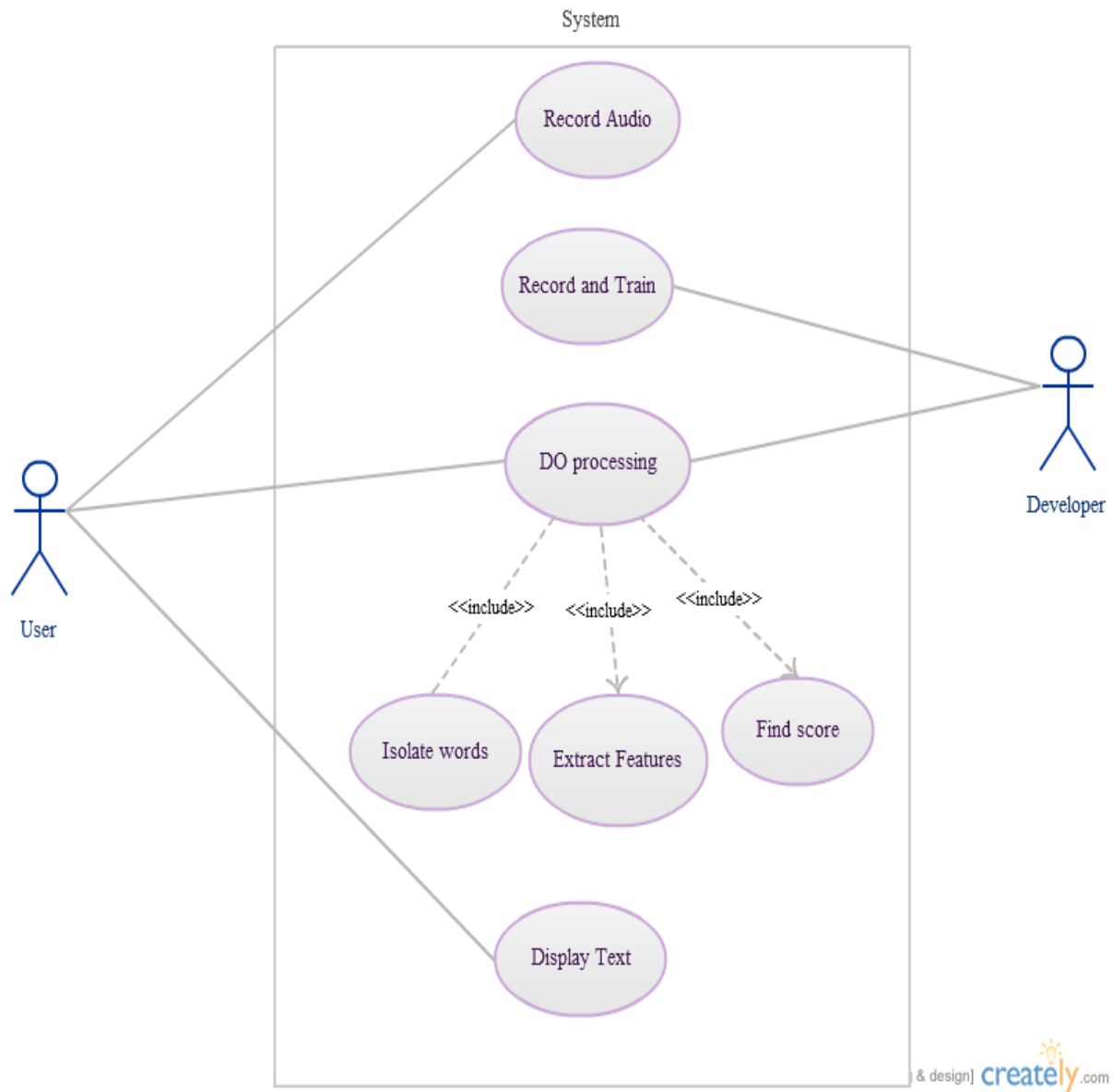
ii. DTW:

The DTW algorithm is run on the input word and each word in the dictionary. It outputs the “distance” between the two samples by dynamic programming approach. The word with the least value of “distance” is chosen.

3. Output:

The word that is output by the HMM and DTW algorithms is output to the user in textual format in Devanagari script.

3.2 Use Case Diagram



1. **Use Case: Record Audio**

Preconditions: System requires hardware support for audio input

Description: User records audio using microphone

Postconditions: Recorded audio saved in permanent storage

2. **Use Case: Do Processing**

Preconditions: System takes recorded audio and do further processing

Description: System does all processing required for speech recognition. This includes

- a. isolating words from audio, extract features(MFCC) for each word, and find score
- b. corresponding to each word (based on approach selected)

Included use cases: Isolate Words, Extract Features, Find Score

Postconditions: System returns separated words along with their feature vectors

3. **Use Case: Isolate Words**

Preconditions: System takes saved audio recording file

Description: System takes recorded audio and isolate words in it

Postconditions: System returns separate audio files for each word in audio recording

4. **Use Case: Extract Features**

Preconditions: System takes audio file as input

Description: System takes each isolated word and extract features for each word

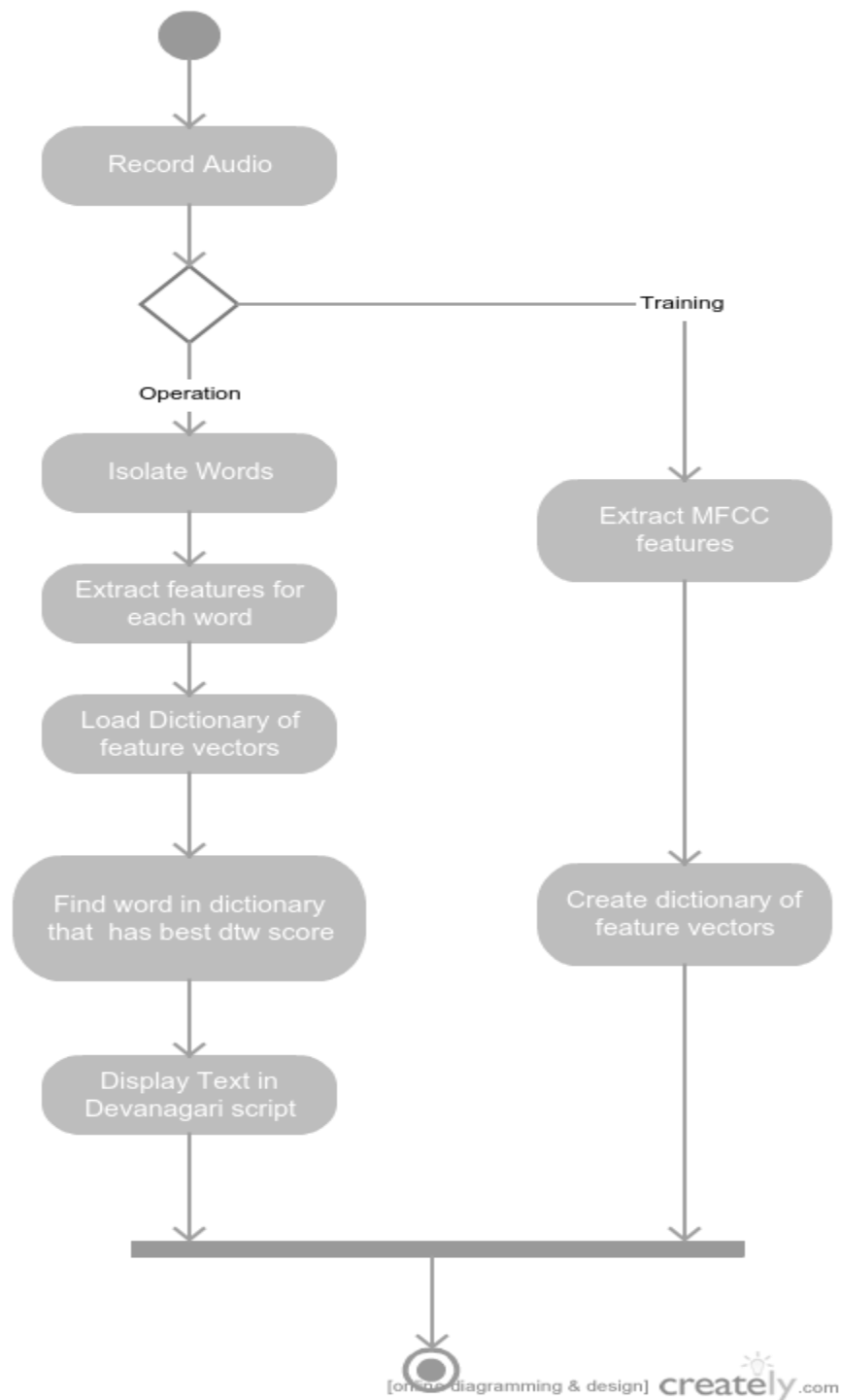
Postconditions: System saves feature vectors for each audio file

5. **Use Case:** Find score (based on approach selected)
 Preconditions: System takes feature vectors of audio files as input
 Description: Based on selected approach (DTW or HMM), system finds word with best score
 Postconditions: System returns index of word in dictionary having best score

6. **Use Case:** Record and Train
 Preconditions: System takes recorded audio files for each word as input
 Description: Developer records templates for training the system. Then features of each audio file are extracted. These features are used to training HMM model of words
 Postconditions: System saves trained HMM model for given training data set

7. **Use Case:** Display Text (In Devanagari Script)
 Preconditions: Input of list of words in dictionary saved in Devanagari script
 Description: System finds word written in Devanagari corresponding to index returned by 'Find Score'
 Postconditions: Word corresponding to returned index is printed on output screen in Devanagari script

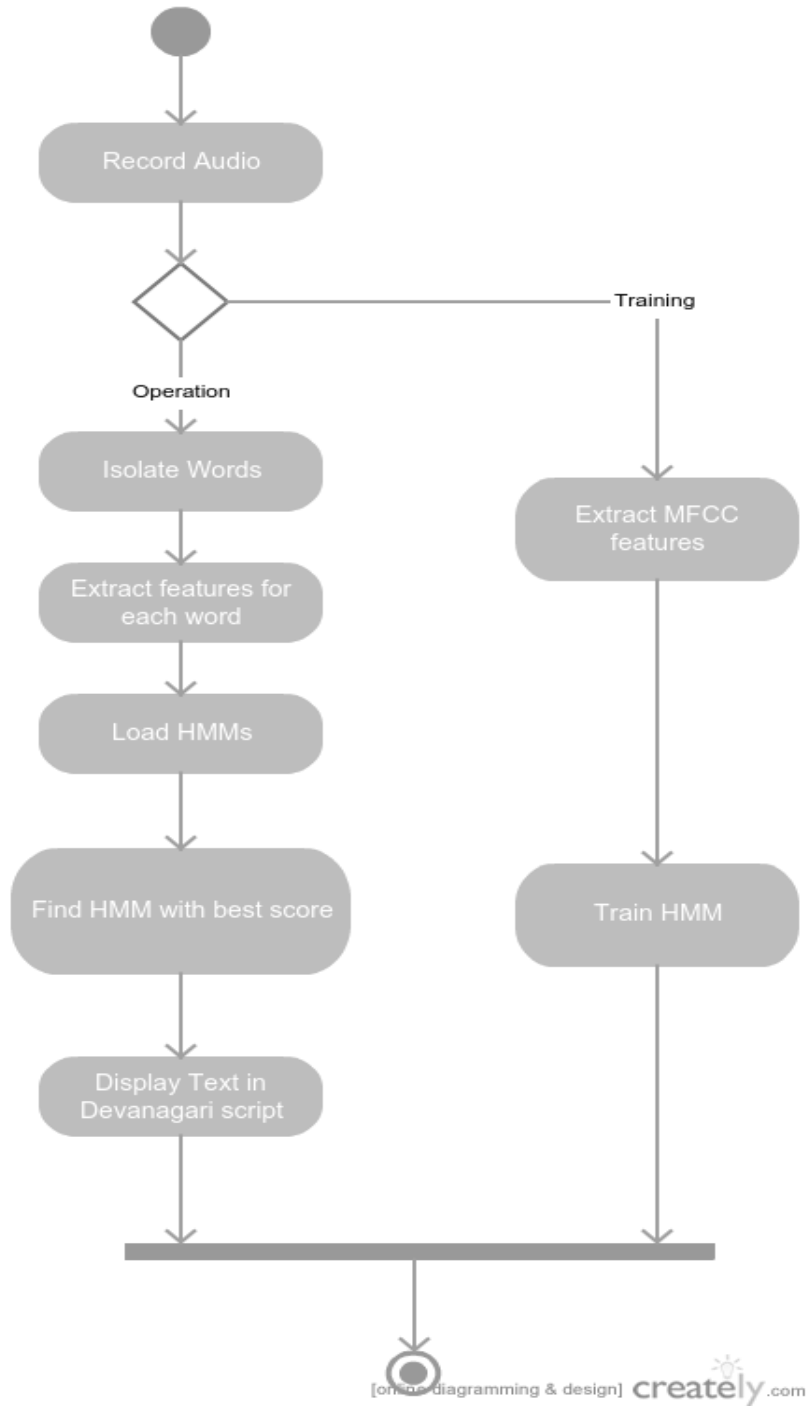
3.3.1 Activity Diagram (DTW)



Steps in Activity Diagram (DTW):

1. Record audio using microphone
2. Training
 - a. Extract features(MFCC) from recorded samples
 - b. Create Dictionary of feature vectors
3. Operation
 - a. Isolate words from recorded audio
 - b. Extract features for each word
 - c. Load Dictionary of feature vectors
 - d. Find word in dictionary that has best dtw score
 - e. Display word with best score in Devanagari Script

3.3.2 Activity Diagram (HMM)



Steps in Activity Diagram (HMM):

1. Record audio using microphone
2. Training
 - a. Extract features(MFCC) from recorded samples
 - b. Train HMM
3. Operation
 - a. Isolate words from recorded audio
 - b. Extract features for each word
 - c. Load HMMs
 - d. Find HMM with best score
 - e. Display word with best score in Devanagari Script

Chapter 4

Implementation

4.1 Source Code

Isolate Words(isolate_words.py) :

```
import numpy
import scipy.io.wavfile as wav
from features.sigproc import framesig
import math, sys

winlen = 0.01
winstep = 0.01
sample_rate = 44100

minSignal = 0.0
threshold = 13.0
average_number=1.0
level = 0.0
adjustment = 0.003
background = 300.0

# in terms of number of 10 ms frames
start_speech = 10
end_silence = 10
speech_leader = 5
speech_trailer = 5

voiced = []
sig = []

def get_signal(start, end):
    res = numpy.array([], dtype=numpy.int16)
    for i in xrange(start, end+1):
        if voiced[i]:
            res = numpy.append(res, sig[441*i : 441*(i+1)])
    return res

def log_root_mean_square(samples):
    sum_of_squares = sum(1.0*x*x for x in samples)
    root_mean_square = math.sqrt(sum_of_squares*1.0 /
len(samples))
    root_mean_square = max(root_mean_square,1)
    return math.log10(root_mean_square) * 20

def classify(audio):
    global level, background, average_number, threshold
    current = log_root_mean_square(audio)
    is_speech = False
    if current >= minSignal:
```

Hindi Speech Recognition

```
level = ((level * average_number) + current) / (average_number + 1)
if current < background:
    background = current
else:
    background += (current - background) * adjustment
if level < background:
    level = background
    is_speech = (level - background > threshold)
return is_speech

def chop(output_folder = 'chopped-words', audio_file =
'recording.wav'):
    global voiced
    global sig

    (rate,sig) = wav.read(audio_file)

    frames = framesig(sig, winlen*sample_rate,
winstep*sample_rate,lambda x:numpy.ones((1,x)))

    for i in range(0,len(frames)):
        if classify(frames[i]):
            voiced.append(True)
        else:
            voiced.append(False)

    started = False
        start_index = 0
        start_cnt = 0
        stop_index = 0
        stop_cnt = 0
    words = 0

    print ">> START_FRAME END_FRAME LENGTH_IN_SECONDS"

    for i in range(0,len(frames)):
        if not started:
            if voiced[i]:
                start_cnt += 1
            else:
                start_cnt = 0
        if start_cnt == start_speech:
            started = True
            start_index = i - start_speech + 1 -
speech_leader
            start_index = max(0, start_index)
        else:
```

Hindi Speech Recognition

```
if voiced[i]:
    stop_cnt = 0
else:
    stop_cnt += 1
if stop_cnt == end_silence:
    stop_index = i - end_silence + 1 +
speech_trailer
    stop_index = min(len(frames)-1, stop_index)

print ">>", start_index, stop_index, (stop_index - start_index +
1 ) * 10
wav.write(output_folder + "/word" + str(words) + ".wav",rate,
sig[441*start_index : 441*(stop_index+1)])

words += 1
started = False
    start_index = start_cnt = 0
    stop_index = stop_cnt = 0

if started:
    stop_index = len(frames)-1
print ">>", start_index, stop_index, (stop_index - start_index
+ 1 ) * 10
wav.write(output_folder + "/word" + str(words) + ".wav",rate,
sig[441*start_index : 441*(stop_index+1)])

words += 1

return words
```

Feature Extraction(sigproc.py) :

```
import numpy
import math

def framesig(sig, frame_len, frame_step, winfunc=lambda
x: numpy.hamming(x)):
    """Frame a signal into overlapping frames.

    :param sig: the audio signal to frame.
    :param frame_len: length of each frame measured in samples.
    :param frame_step: number of samples after the start of the
    previous frame that the next frame should begin.
    :param winfunc: the analysis window to apply to each frame. By
    default no window is applied.
    :returns: an array of frames. Size is NUMFRAMES by frame_len.
    """
    slen = len(sig)
    frame_len = int(round(frame_len))
    frame_step = int(round(frame_step))
    if slen <= frame_len:
        numframes = 1
    else:
        numframes = 1 + int(math.ceil((1.0*slen -
        frame_len)/frame_step))

    padlen = int((numframes-1)*frame_step + frame_len)

    zeros = numpy.zeros((padlen - slen,))
    padsignal = numpy.concatenate((sig, zeros))

    indices =
    numpy.tile(numpy.arange(0, frame_len), (numframes, 1)) +
    numpy.tile(numpy.arange(0, numframes*frame_step, frame_step), (fram
    e_len, 1)).T
    indices = numpy.array(indices, dtype=numpy.int32)
    frames = padsignal[indices]
    win = numpy.tile(winfunc(frame_len), (numframes, 1))
    return frames*win

def deframesig(frames, siglen, frame_len, frame_step, winfunc=lambda
x: numpy.hamming(x)):
    """Does overlap-add procedure to undo the action of
    framesig.

    :param frames: the array of frames.
```

Hindi Speech Recognition

```
:param siglen: the length of the desired signal, use 0 if
unknown. Output will be truncated to siglen samples.
:param frame_len: length of each frame measured in samples.
:param frame_step: number of samples after the start of the
previous frame that the next frame should begin.
:param winfunc: the analysis window to apply to each frame. By
default no window is applied.
:returns: a 1-D signal.
    """
    frame_len = round(frame_len)
    frame_step = round(frame_step)
    numframes = numpy.shape(frames)[0]
    assert numpy.shape(frames)[1] == frame_len, '"frames" matrix is
wrong size, 2nd dim is not equal to frame_len'

    indices =
numpy.tile(numpy.arange(0,frame_len), (numframes,1)) +
numpy.tile(numpy.arange(0,numframes*frame_step,frame_step), (fram
e_len,1)).T
    indices = numpy.array(indices, dtype=numpy.int32)
    padlen = (numframes-1)*frame_step + frame_len

    if siglen <= 0: siglen = padlen

    rec_signal = numpy.zeros((1,padlen))
    window_correction = numpy.zeros((1,padlen))
    win = winfunc(frame_len)

    for i in range(0,numframes):
        window_correction[indices[i,:]] =
window_correction[indices[i,:]] + win + 1e-15 #add a little bit
so it is never zero
        rec_signal[indices[i,:]] = rec_signal[indices[i,:]] +
frames[i,:]

    rec_signal = rec_signal/window_correction
    return rec_signal[0:siglen]

def magspec(frames,NFFT):
    """Compute the magnitude spectrum of each frame in frames.
    If frames is an NxD matrix, output will be NxNFFT.

    :param frames: the array of frames. Each row is a frame.
    :param NFFT: the FFT length to use. If NFFT > frame_len, the
frames are zero-padded.
    :returns: If frames is an NxD matrix, output will be NxNFFT.
    Each row will be the magnitude spectrum of the corresponding
frame.
```

Hindi Speech Recognition

```
    """
    complex_spec = numpy.fft.rfft(frames,NFFT)
    return numpy.absolute(complex_spec)

def powspec(frames,NFFT):
    """Compute the power spectrum of each frame in frames. If
    frames is an Nx D matrix, output will be NxNFFT.

    :param frames: the array of frames. Each row is a frame.
    :param NFFT: the FFT length to use. If NFFT > frame_len, the
    frames are zero-padded.
    :returns: If frames is an Nx D matrix, output will be NxNFFT.
    Each row will be the power spectrum of the corresponding frame.
    """
    return 1.0/NFFT * numpy.square(magspec(frames,NFFT))

def logpowspec(frames,NFFT,norm=1):
    """Compute the log power spectrum of each frame in frames.
    If frames is an Nx D matrix, output will be NxNFFT.

    :param frames: the array of frames. Each row is a frame.
    :param NFFT: the FFT length to use. If NFFT > frame_len, the
    frames are zero-padded.
    :param norm: If norm=1, the log power spectrum is normalised so
    that the max value (across all frames) is 1.
    :returns: If frames is an Nx D matrix, output will be NxNFFT.
    Each row will be the log power spectrum of the corresponding
    frame.
    """
    ps = powspec(frames,NFFT);
    ps[ps<=1e-30] = 1e-30
    lps = 10*numpy.log10(ps)
    if norm:
        return lps - numpy.max(lps)
    else:
        return lps

def preemphasis(signal,coeff=0.95):
    """perform preemphasis on the input signal.

    :param signal: The signal to filter.
    :param coeff: The preemphasis coefficient. 0 is no filter,
    default is 0.95.
    :returns: the filtered signal.
    """
    return numpy.append(signal[0],signal[1:]-coeff*signal[:-1])
```

Feature Extraction(base.py) :

```
import numpy
import scipy.io.wavfile as wav
from features import sigproc
from scipy.fftpack import dct

def get_features(audio_file):
    """ Computes feature vectors from a wav singal
        : (1 energy + 12 mfcc + 13 delta + 13 double delta)
    :returns: A numpy array of size NUMFRAMES * 39
    """
    (rate,sig) = wav.read(audio_file)
    mfcc_feat = mfcc(sig)
    delta_feat = delta(mfcc_feat)
    delta_delta_feat = delta(delta_feat)

    F = numpy.append(mfcc_feat,delta_feat,1)
    F = numpy.append(F,delta_delta_feat,1)
    return F

def
mfcc(signal,samplerate=44100,winlen=0.025,winstep=0.01,numcep=13
,
nfilt=26,nfft=512,lowfreq=0,highfreq=None,preemph=0.97,ceplifter
=22,appendEnergy=True):
    """Compute MFCC features from an audio signal.

: param signal: the audio signal from which to compute features.
Should be an N*1 array
: param samplerate: the samplerate of the signal we are working
with.
: param winlen: the length of the analysis window in seconds.
Default is 0.025s (25 milliseconds)
: param winstep: the step between successive windows in seconds.
Default is 0.01s (10 milliseconds)
: param numcep: the number of cepstrum to return, default 13
: param nfilt: the number of filters in the filterbank, default
26.
: param nfft: the FFT size. Default is 512.
: param lowfreq: lowest band edge of mel filters. In Hz, default
is 0.
: param highfreq: highest band edge of mel filters. In Hz,
default is samplerate/2
: param preemph: apply preemphasis filter with preemph as
coefficient. 0 is no filter. Default is 0.97.
```


Hindi Speech Recognition

```
:param ceplifter: apply a lifter to final cepstral coefficients.
0 is no lifter. Default is 22.
:param appendEnergy: if this is true, the zeroth cepstral
coefficient is replaced with the log of the total frame energy.
:returns: A numpy array of size (NUMFRAMES by numcep) containing
features. Each row holds 1 feature vector.
"""

    feat,energy =
fbank(signal,samplerate,winlen,winstep,nfilt,nfft,lowfreq,highfr
eq,preemph)
feat = numpy.log(feat)
feat = dct(feat, type=2, axis=1, norm='ortho')[:, :numcep]
feat = lifter(feat,ceplifter)
if appendEnergy: feat[:,0] = numpy.log(energy) # replace first
cepstral coefficient with log of frame energy
return feat

def fbank(signal,samplerate=44100,winlen=0.025,winstep=0.01,
nfilt=26,nfft=512,lowfreq=0,highfreq=None,preemph=0.97):
    """Compute Mel-filterbank energy features from an audio
signal.

:param signal: the audio signal from which to compute features.
Should be an N*1 array
:param samplerate: the samplerate of the signal we are working
with.
:param winlen: the length of the analysis window in seconds.
Default is 0.025s (25 milliseconds)
:param winstep: the step between successive windows in seconds.
Default is 0.01s (10 milliseconds)
:param nfilt: the number of filters in the filterbank, default
26.
:param nfft: the FFT size. Default is 512.
:param lowfreq: lowest band edge of mel filters. In Hz, default
is 0.
:param highfreq: highest band edge of mel filters. In Hz,
default is samplerate/2
:param preemph: apply preemphasis filter with preemph as
coefficient. 0 is no filter. Default is 0.97.
:returns: 2 values. The first is a numpy array of size
(NUMFRAMES by nfilt) containing features. Each row holds 1
feature vector. The
second return value is the energy in each frame (total energy,
unwindowed). A numpy array of zero crossing rate for each frame.
"""
    highfreq= highfreq or samplerate/2
    signal = sigproc.preemphasis(signal,preemph)
```

Hindi Speech Recognition

```
frames = sigproc.framesig(signal, winlen*samplerate,
winstep*samplerate)
pspec = sigproc.powspec(frames,nfft)
energy = numpy.sum(pspec,1) # this stores the total energy in
each frame
energy = numpy.where(energy == 0,numpy.finfo(float).eps,energy)
# if energy is zero, we get problems with log

fb = get_filterbanks(nfilt,nfft,samplerate)
feat = numpy.dot(pspec,fb.T) # compute the filterbank energies
feat = numpy.where(feat == 0,numpy.finfo(float).eps,feat) # if
feat is zero, we get problems with log
return feat,energy

def logfbank(signal,samplerate=44100,winlen=0.025,winstep=0.01,
nfilt=26,nfft=512,lowfreq=0,highfreq=None,preemph=0.97):
    """Compute log Mel-filterbank energy features from an audio
    signal.

    :param signal: the audio signal from which to compute features.
    Should be an N*1 array
    :param samplerate: the samplerate of the signal we are working
    with.
    :param winlen: the length of the analysis window in seconds.
    Default is 0.025s (25 milliseconds)
    :param winstep: the step between successive windows in seconds.
    Default is 0.01s (10 milliseconds)
    :param nfilt: the number of filters in the filterbank, default
    26.
    :param nfft: the FFT size. Default is 512.
    :param lowfreq: lowest band edge of mel filters. In Hz, default
    is 0.
    :param highfreq: highest band edge of mel filters. In Hz,
    default is samplerate/2
    :param preemph: apply preemphasis filter with preemph as
    coefficient. 0 is no filter. Default is 0.97.
    :returns: A numpy array of size (NUMFRAMES by nfilt) containing
    features. Each row holds 1 feature vector.
    """
    feat,energy =
fbank(signal,samplerate,winlen,winstep,nfilt,nfft,lowfreq,highfr
eq,preemph)
    return numpy.log(feat)

def ssc(signal,samplerate=44100,winlen=0.025,winstep=0.01,
nfilt=26,nfft=512,lowfreq=0,highfreq=None,preemph=0.97):
```

Hindi Speech Recognition

```
"""Compute Spectral Subband Centroid features from an audio
signal.

:param signal: the audio signal from which to compute features.
Should be an N*1 array
:param samplerate: the samplerate of the signal we are working
with.
:param winlen: the length of the analysis window in seconds.
Default is 0.025s (25 milliseconds)
:param winstep: the step between successive windows in seconds.
Default is 0.01s (10 milliseconds)
:param nfilt: the number of filters in the filterbank, default
26.
:param nfft: the FFT size. Default is 512.
:param lowfreq: lowest band edge of mel filters. In Hz, default
is 0.
:param highfreq: highest band edge of mel filters. In Hz,
default is samplerate/2
:param preemph: apply preemphasis filter with preemph as
coefficient. 0 is no filter. Default is 0.97.
:returns: A numpy array of size (NUMFRAMES by nfilt) containing
features. Each row holds 1 feature vector.
"""

highfreq= highfreq or samplerate/2
signal = sigproc.preemphasis(signal,preemph)
frames = sigproc.framesig(signal, winlen*samplerate,
winstep*samplerate)
pspec = sigproc.powspec(frames,nfft)
pspec = numpy.where(pspec == 0,numpy.finfo(float).eps,pspec) #
if things are all zeros we get problems

fb = get_filterbanks(nfilt,nfft,samplerate)
feat = numpy.dot(pspec,fb.T) # compute the filterbank energies
R =
numpy.tile(numpy.linspace(1,samplerate/2,numpy.size(pspec,1)),(n
umpy.size(pspec,0),1))

return numpy.dot(pspec*R,fb.T) / feat

def hz2mel(hz):
    """Convert a value in Hertz to Mels

    :param hz: a value in Hz. This can also be a numpy array,
    conversion proceeds element-wise.
    :returns: a value in Mels. If an array was passed in, an
    identical sized array is returned.
    """
    return 2595 * numpy.log10(1+hz/700.0)
```

Hindi Speech Recognition

```
def mel2hz(mel):
    """Convert a value in Mels to Hertz

    :param mel: a value in Mels. This can also be a numpy array,
    conversion proceeds element-wise.
    :returns: a value in Hertz. If an array was passed in, an
    identical sized array is returned.
    """
    return 700*(10**(mel/2595.0)-1)

def
get_filterbanks(nfilt=20,nfft=512,samplerate=44100,lowfreq=0,hig
hfreq=None):
    """Compute a Mel-filterbank. The filters are stored in the
    rows, the columns correspond
    to fft bins. The filters are returned as an array of size nfilt
    * (nfft/2 + 1)

    :param nfilt: the number of filters in the filterbank, default
    20.
    :param nfft: the FFT size. Default is 512.
    :param samplerate: the samplerate of the signal we are working
    with. Affects mel spacing.
    :param lowfreq: lowest band edge of mel filters, default 0 Hz
    :param highfreq: highest band edge of mel filters, default
    samplerate/2
    :returns: A numpy array of size nfilt * (nfft/2 + 1) containing
    filterbank. Each row holds 1 filter.
    """
    highfreq= highfreq or samplerate/2

    # compute points evenly spaced in mels
    lowmel = hz2mel(lowfreq)
    highmel = hz2mel(highfreq)
    melpoints = numpy.linspace(lowmel,highmel,nfilt+2)
    # our points are in Hz, but we use fft bins, so we have to
    convert
    # from Hz to fft bin number
    bin = numpy.floor((nfft+1)*mel2hz(melpoints)/samplerate)

    fbank = numpy.zeros([nfilt,nfft/2+1])
    for j in xrange(0,nfilt):
        for i in xrange(int(bin[j]),int(bin[j+1])):
            fbank[j,i] = (i - bin[j])/(bin[j+1]-bin[j])
            for i in xrange(int(bin[j+1]),int(bin[j+2])):
                fbank[j,i] = (bin[j+2]-i)/(bin[j+2]-bin[j+1])
    return fbank
```

```
def lifter(cepstra,L=22):
    """Apply a cepstral lifter the the matrix of cepstra. This
    has the effect of increasing the
    magnitude of the high frequency DCT coeffs.

    :param cepstra: the matrix of mel-cepstra, will be numframes *
    numcep in size.
    :param L: the liftering coefficient to use. Default is 22. L <=
    0 disables lifter.
    """
    if L > 0:
        nframes,ncoeff = numpy.shape(cepstra)
        n = numpy.arange(ncoeff)
        lift = 1+ (L/2)*numpy.sin(numpy.pi*n/L)
        return lift*cepstra
    else:
        # values of L <= 0, do nothing
        return cepstra

def delta(m):
    """ Computes delta coefficients with N=1
    :returns: A numpy array of size same as that of m
    """
    x,y = m.shape[0], m.shape[1]
    delta_feat = numpy.empty([x,y])
    for i in range(0,x):
        for j in range(0,y):
            delta_feat[i,j] = ( m[i,min(j+1,y-1)] - m[i,max(0,j-
1)] ) + 2*(m[i,min(j+2,y-1)] - m[i,max(0,j-2)]) ) / 10.0
    return delta_feat

'''
def zero_crossing_rate(frames):
    """calculates zero crossing rate in signal
    :param frames: frames of signal for which
    zero_crossing_rate is calculated.
    :return: zero crossing rate for signal
    """
    Nframes = len(frames)
    zcrs = numpy.zeros(Nframes)
    for i in range(0,Nframes):
        zcr = 0.0
        N = len(frames[i])
        for j in range(1,N):
```

Hindi Speech Recognition

```
        zcr = zcr + abs(sgn(frames[i][j]) -
sgn(frames[i][j-1]))
        zcr = zcr/(2.0*N)
        zcrs[i] = zcr
    return zcrs

def calc_energy(frames):
    """calculates absolute anergy in each frame
    :param frames: frames of signal for which
zero_crossing_rate is calculated.
    :return: zero crossing rate for signal
    """
    Nframes = len(frames)
    abs_energy = numpy.zeros(Nframes)
    for i in range(0,Nframes):
        N = len(frames[i])
        energy = sum(1.0*x*x for x in frames[i])
        energy = energy*1.0/N
        abs_energy[i] = energy
    return abs_energy

def sgn(value):
    """checks sign of value

    :param value: any real number.
    :returns: -1 if value is negaitive, 1 otherwise
    """
    if value>=0:
        return 1
    else:
        return -1
'''
```

Dynamic Time Warping (DTW)

Creating Dictionary of Feature Vectors(dictionary.py):

```
import numpy as np
import pickle
from dtw import dtw
from features import get_features

WORD_COMPARE_TR = 2.5

class Dword:

    def __init__(self, path):

        with open(path + '/meta.txt') as meta:
            meta_data = [x for x in meta]

            self.name = meta_data[0].strip()
            self.repeat = int(meta_data[1])
            self.path = path
        self.load()

    def load(self):

        self.features = []
        for i in xrange(self.repeat):
            self.features.append(get_features(self.path + "/" + str(i+1) +
            ".wav"))

            self.mean = np.mean([len(x) for x in self.features])
            self.std = np.std([len(x) for x in self.features])

    def compare(self, w):
        cost = np.inf
        if abs(w.frame_cnt - self.mean) <= WORD_COMPARE_TR * self.std:
            for f in self.features:
                cost = min(cost, dtw(f, w.features))
            return cost

class Dictionary:

    def __init__(self, path):

        with open(path + '/meta.txt') as meta:
            meta_data = [x for x in meta]

            self.path = path
```

Hindi Speech Recognition

```
        self.dword_cnt = int(meta_data[0])
self.load()

def load(self):
    self.dwords = []
    for i in xrange(self.dword_cnt):

        dword_path = self.path + '/' + str(i+1)
        serialized = open(dword_path + '/serialized.txt', 'r')
        is_serial = (serialized.read() == '1')
        serialized.close()

    if is_serial:
        with open(dword_path + '/dword.pkl', 'rb') as input:
            dword = pickle.load(input)
    else:
        dword = Dword(dword_path)
        with open(dword_path + '/dword.pkl', 'wb') as output:
            pickle.dump(dword, output, 0)
        serialized = open(dword_path + '/serialized.txt', 'w')
        serialized.write("1")
        serialized.close()

    self.dwords.append(dword)

def find_match(self, w):
    best = np.inf
    index = 0
    for i in xrange(self.dword_cnt):
        cost = self.dwords[i].compare(w)
        if cost < best:
            best = cost
            index = i
    return self.dwords[index].name

class Word:

    def __init__(self, audio_path):
        self.audio_path = audio_path
        self.load()

    def load(self):
        self.features = get_features(self.audio_path)
        self.frame_cnt = len(self.features)
```


DTW Algorithm Implementation(dtw.py) :

```
from numpy import array, zeros, argmin, inf
from numpy.linalg import norm

def dtw(x, y, dist=lambda x, y: norm(x-y, ord=1)):

    """ here dist is same as sum((x-y)**2)/len(x)
    Computes the DTW of two sequences.

:param array x: N1*M array
:param array y: N2*M array
:param func dist: distance used as cost measure (default L1
norm)

    Returns the minimum distance, the accumulated cost matrix
and the wrap path.

    """
    x = array(x)
    if len(x.shape) == 1:
        x = x.reshape(-1, 1)
    y = array(y)
    if len(y.shape) == 1:
        y = y.reshape(-1, 1)

    r, c = len(x), len(y)

    D = zeros((r + 1, c + 1))
    D[0, 1:] = inf
    D[1:, 0] = inf

    for i in range(r):
        for j in range(c):
            D[i+1, j+1] = dist(x[i], y[j])

    for i in range(r):
        for j in range(c):
            D[i+1, j+1] += min(D[i, j], D[i, j+1], D[i+1, j])

    D = D[1:, 1:]

    dist = D[-1, -1] / sum(D.shape)

    #return dist, D, _trackback(D)
    return dist
```

```
def _trackeback(D):
    i, j = array(D.shape) - 1
    p, q = [i], [j]
    while (i > 0 and j > 0):
        tb = argmin((D[i-1, j-1], D[i-1, j], D[i, j-1]))

        if (tb == 0):
            i = i - 1
            j = j - 1
        elif (tb == 1):
            i = i - 1
        elif (tb == 2):
            j = j - 1

    p.insert(0, i)
    q.insert(0, j)

    p.insert(0, 0)
    q.insert(0, 0)
    return (array(p), array(q))
```

Speech to Text conversion(main.py):

```
import pickle, sys
from chop_words import chop
from record import rec
from dictionary import *

def load_dictionary(path):
    serialized = open(path + '/serialized.txt', 'r')
    is_serial = (serialized.read() == '1')
    serialized.close()

    if is_serial:
        with open(path + '/dictionary.pkl', 'rb') as input:
            dictionary = pickle.load(input)
        else:
            dictionary = Dictionary(path)
        with open(path + '/dictionary.pkl', 'wb') as output:
            pickle.dump(dictionary, output, 0)
        serialized = open(path + '/serialized.txt', 'w')
        serialized.write("1")
        serialized.close()

    return dictionary

if __name__ == '__main__':

    if len(sys.argv) < 3:
        print "Usage: python main.py [DICTIONARY] [AUDIO-FILE]"
    else:
        dictionary = load_dictionary(sys.argv[1])
        audio_file = sys.argv[2]

        word_cnt = chop('chopped-words', audio_file)
        print "querying", word_cnt, "words"

        for i in xrange(word_cnt):
            w = Word('chopped-words/word' + str(i) + '.wav')
        print dictionary.find_match(w),
```

Hidden Markov Model (HMM)

Training HMM models(training.py):

```
import numpy as np
import scipy.io.wavfile as wav
from features import get_features
import pylab as pl
from hmmlearn.hmm import GaussianHMM
import os
import sys
import math
import pickle

test_folder = sys.argv[1]
dict_folder = sys.argv[2]
model_folder = sys.argv[3]

fpaths = []
labels = []
spoken = []
for f in os.listdir(dict_folder):
    temp = []
    for w in os.listdir(dict_folder+'/'+ f):
        temp.append(dict_folder+'/'+ f + '/' + w)
    labels.append(f)
    if f not in spoken:
        spoken.append(f)
    fpaths.append(temp)
print 'Words spoken:',spoken

n_samples = 20
#####
# Run Gaussian HMM
models = []
means = []
std_devs = []

print "Generating models..."
for i in range(len(spoken)):

    n_components = 3
    arr = []

    # make an HMM instance and execute fit
    model = GaussianHMM(n_components, covariance_type="diag",
n_iter=1000)
```

```
for j in range(n_samples):
    (rate,sig) = wav.read(fpaths[i][j])
    features = get_features(sig)
    arr.append(len(features))
    model.fit([features])

models.append(model)
means.append(np.mean(arr))
std_devs.append(np.std(arr))

with open('Models/models.pkl', 'wb') as output:
    pickle.dump(models, output, 0)

with open('Models/means.pkl', 'wb') as output:
    pickle.dump(means, output, 0)

with open('Models/std_devs.pkl', 'wb') as output:
    pickle.dump(std_devs, output, 0)

serialized = open('Models/serialized.txt', 'w')
serialized.write("1")
```

Speech to Text conversion using HMM(main.py):

```
import numpy as np
import scipy.io.wavfile as wav
from features import get_features
import pylab as pl
from hmmlearn.hmm import GaussianHMM
from isolate_words import get_words
from chop_words import chop
import os
import sys
import math
import pickle
import subprocess

test_file = sys.argv[1]
dict_folder = sys.argv[2]
model_folder = sys.argv[3]
meta_file = sys.argv[4]

fpaths = []
labels = []
spoken = []
for f in os.listdir(dict_folder):
    temp = []
    for w in os.listdir(dict_folder+'/'+ f):
        temp.append(dict_folder+'/'+ f + '/' + w)
    labels.append(f)
    if f not in spoken:
        spoken.append(f)
    fpaths.append(temp)

words = []
with open(meta_file+'.txt') as answers:
    for entry in answers:
        words.append(entry)

#print 'Words spoken:',words

serialized = open(model_folder+'/serialized.txt', 'r')
isSerial = False
if (serialized.read() == '1'):
    isSerial = True

if isSerial == False:
    print "Generating models..."
    n_samples = 20
```

Hindi Speech Recognition

```
#####  
#####  
# Run Gaussian HMM  
models = []  
means = []  
std_devs = []  
  
for i in range(len(spoken)):  
  
    n_components = 3  
    arr = []  
  
    # make an HMM instance and execute fit  
    model = GaussianHMM(n_components,  
covariance_type="diag", n_iter=1000)  
  
    for j in range(n_samples):  
        (rate,sig) = wav.read(fpaths[i][j])  
        features = get_features(sig)  
        arr.append(len(features))  
        model.fit([features])  
  
    models.append(model)  
    means.append(np.mean(arr))  
    std_devs.append(np.std(arr))  
  
with open(model_folder+'/models.pkl', 'wb') as output:  
    pickle.dump(models, output, 0)  
  
with open(model_folder+'/means.pkl', 'wb') as output:  
    pickle.dump(means, output, 0)  
  
with open(model_folder+'/std_devs.pkl', 'wb') as output:  
    pickle.dump(std_devs, output, 0)  
  
serialized = open(model_folder+'/serialized.txt', 'w')  
serialized.write("1")  
print "Done\n"  
else:  
    print "Loading models..."  
    with open(model_folder+'/models.pkl', 'rb') as input:  
        models = pickle.load(input)  
  
    with open(model_folder+'/means.pkl', 'rb') as input:  
        means = pickle.load(input)  
  
    with open(model_folder+'/std_devs.pkl', 'rb') as input:  
        std_devs = pickle.load(input)
```

Hindi Speech Recognition

```
    print "Done\n"

tot_words = chop(test_file)
right = 0.0
threshold = 5

print "Detected Words:"

for i in xrange(tot_words):
    try:
        (rate,sig) = wav.read("word" + str(i) + ".wav")
        features = get_features(sig)
        word_len = len(features)
        ans = -1
        j = -1
        max_ans = -1e9
        for model in models:
            j = j+1
            if math.fabs(word_len - means[j]) <= threshold *
std_devs[j]:
                temp = model.score(features)
                if temp>max_ans:
                    max_ans = temp
                    ans = j

        print words[ans],
    except:
        break

os.system("rm word*.wav")
```


Calculating accuracy for sample tests (detect_accuracy.py):

```
import numpy as np
import scipy.io.wavfile as wav
from features import get_features
from hmmlearn.hmm import GaussianHMM
import os
import sys
import math
import pickle

test_folder = sys.argv[1]
model_folder = sys.argv[2]

fpaths = []
labels = []
spoken = []
for f in os.listdir('dictionary'):
    temp = []
    for w in os.listdir('dictionary/' + f):
        temp.append('dictionary/' + f + '/' + w)
    labels.append(f)
    if f not in spoken:
        spoken.append(f)
    fpaths.append(temp)
print 'Words spoken:', spoken

serialized = open('Models/serialized.txt', 'r')
isSerial = False
if (serialized.read() == '1'):
    isSerial = True

if isSerial == False:
    print "Generating models..."
    n_samples = 20
    #####
    #####
    # Run Gaussian HMM
    models = []
    means = []
    std_devs = []

    for i in range(len(spoken)):

        n_components = 3
        arr = []
```

Hindi Speech Recognition

```
# make an HMM instance and execute fit
model = GaussianHMM(n_components,
covariance_type="diag", n_iter=1000)

for j in range(n_samples):
    (rate,sig) = wav.read(fpaths[i][j])
    features = get_features(sig)
    arr.append(len(features))
    model.fit([features])

models.append(model)
means.append(np.mean(arr))
std_devs.append(np.std(arr))

with open('Models/models.pkl', 'wb') as output:
    pickle.dump(models, output, 0)

with open('Models/means.pkl', 'wb') as output:
    pickle.dump(means, output, 0)

with open('Models/std_devs.pkl', 'wb') as output:
    pickle.dump(std_devs, output, 0)

serialized = open('Models/serialized.txt', 'w')
serialized.write("1")
print "Done\n"
else:
    print "Loading models..."
    with open(model_folder+'/models.pkl', 'rb') as input:
        models = pickle.load(input)

    with open(model_folder+'/means.pkl', 'rb') as input:
        means = pickle.load(input)

    with open(model_folder+'/std_devs.pkl', 'rb') as input:
        std_devs = pickle.load(input)
    print "Done\n"

correct_answers = []
with open('Test/'+test_folder+'/answer.txt') as answers:
    for entry in answers:
        correct_answers.append(entry.split())

tot_words = len(correct_answers)
right = 0.0
threshold = 1.5

for i in xrange(tot_words):
```

Hindi Speech Recognition

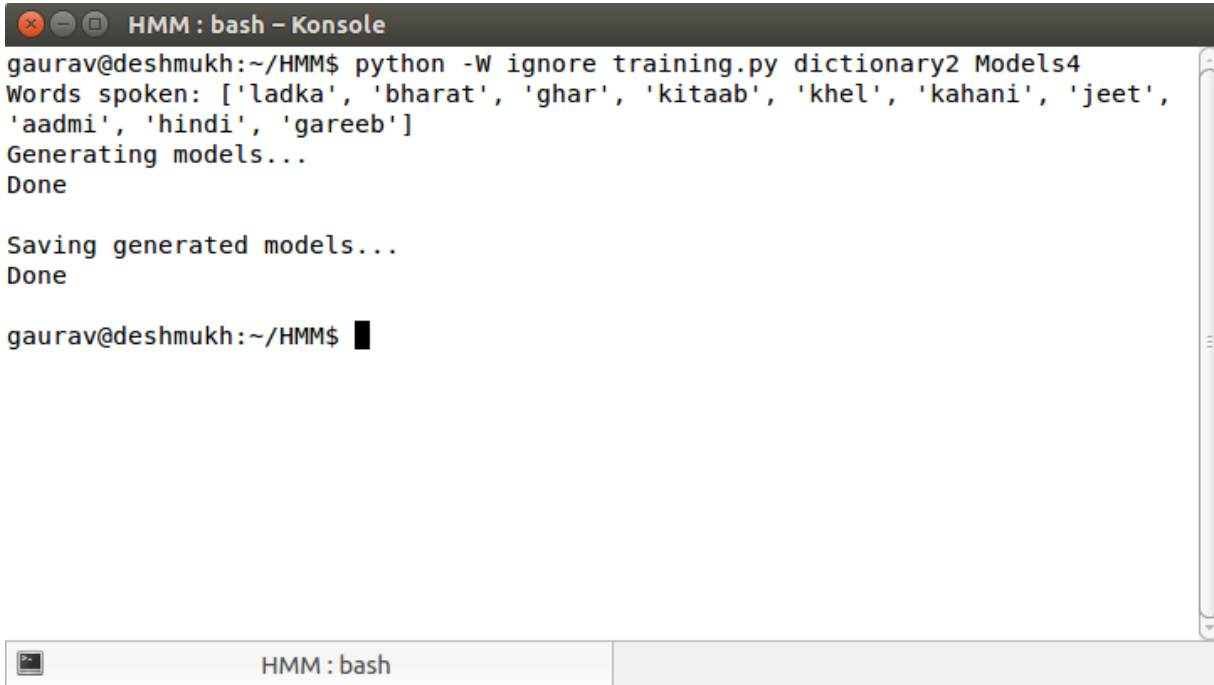
```
try:
    (rate,sig) = wav.read('Test/'+test_folder+"/word" +
str(i) + ".wav")
    features = get_features(sig)
    word_len = len(features)
    ans = -1
    j = -1
    max_ans = -1e9
    for model in models:
        j = j+1
        if math.fabs(word_len - means[j]) <= threshold *
std_devs[j]:
            temp = model.score(features)
            if temp>max_ans:
                max_ans = temp
                ans = j

    print str(i+1)+". Detected word: "+spoken[ans]
    if spoken[ans] == correct_answers[i][0]:
        right = right + 1
except:
    break

print "Accuracy = "+str((right/tot_words)*100)+"%"
```

4.2 Screenshots:

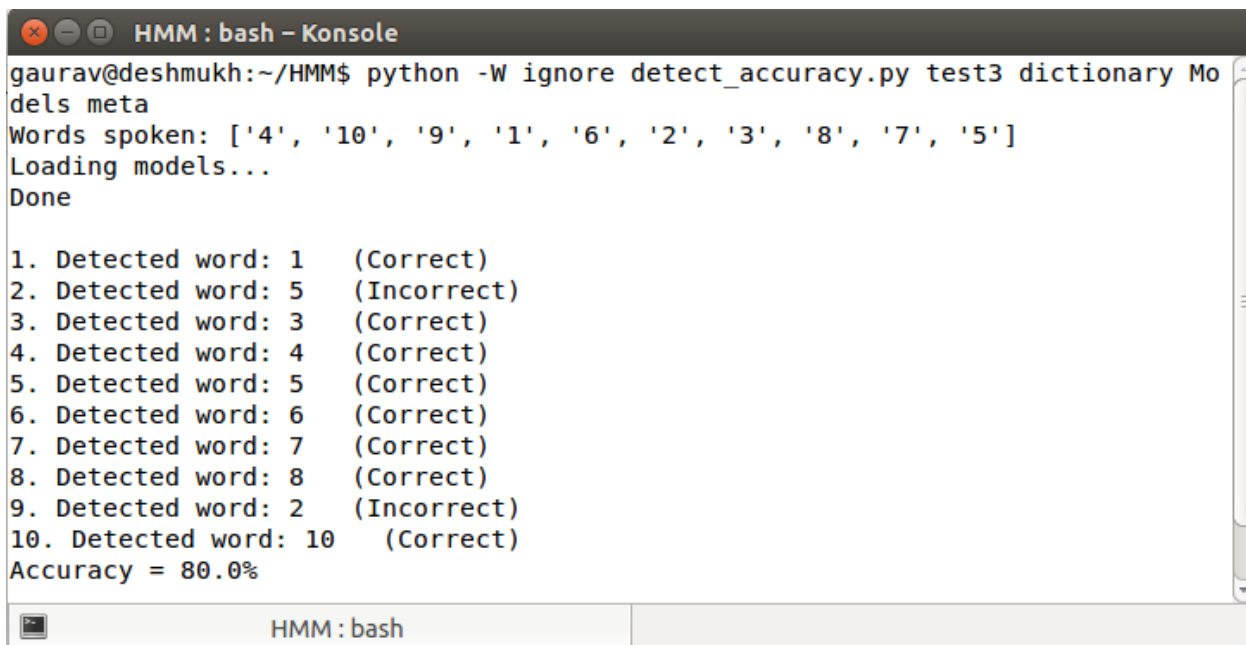
1. HMM



```
HMM : bash - Konsole
gaurav@deshmukh:~/HMM$ python -W ignore training.py dictionary2 Models4
Words spoken: ['ladka', 'bharat', 'ghar', 'kitaab', 'khel', 'kahani', 'jeet',
'aadmi', 'hindi', 'gareeb']
Generating models...
Done

Saving generated models...
Done

gaurav@deshmukh:~/HMM$
```

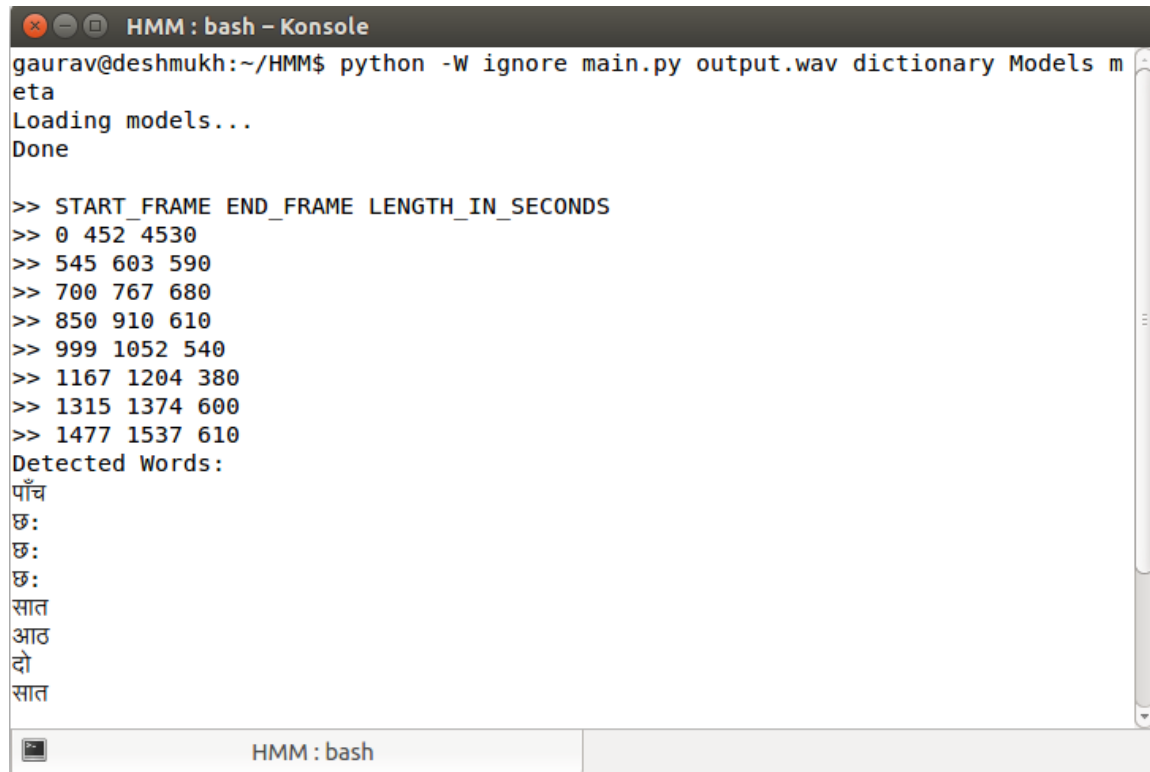


```
HMM : bash - Konsole
gaurav@deshmukh:~/HMM$ python -W ignore detect_accuracy.py test3 dictionary Mo
dels meta
Words spoken: ['4', '10', '9', '1', '6', '2', '3', '8', '7', '5']
Loading models...
Done

1. Detected word: 1    (Correct)
2. Detected word: 5    (Incorrect)
3. Detected word: 3    (Correct)
4. Detected word: 4    (Correct)
5. Detected word: 5    (Correct)
6. Detected word: 6    (Correct)
7. Detected word: 7    (Correct)
8. Detected word: 8    (Correct)
9. Detected word: 2    (Incorrect)
10. Detected word: 10   (Correct)
Accuracy = 80.0%
```

Training HMM using dictionary

Calculating accuracy of sample test

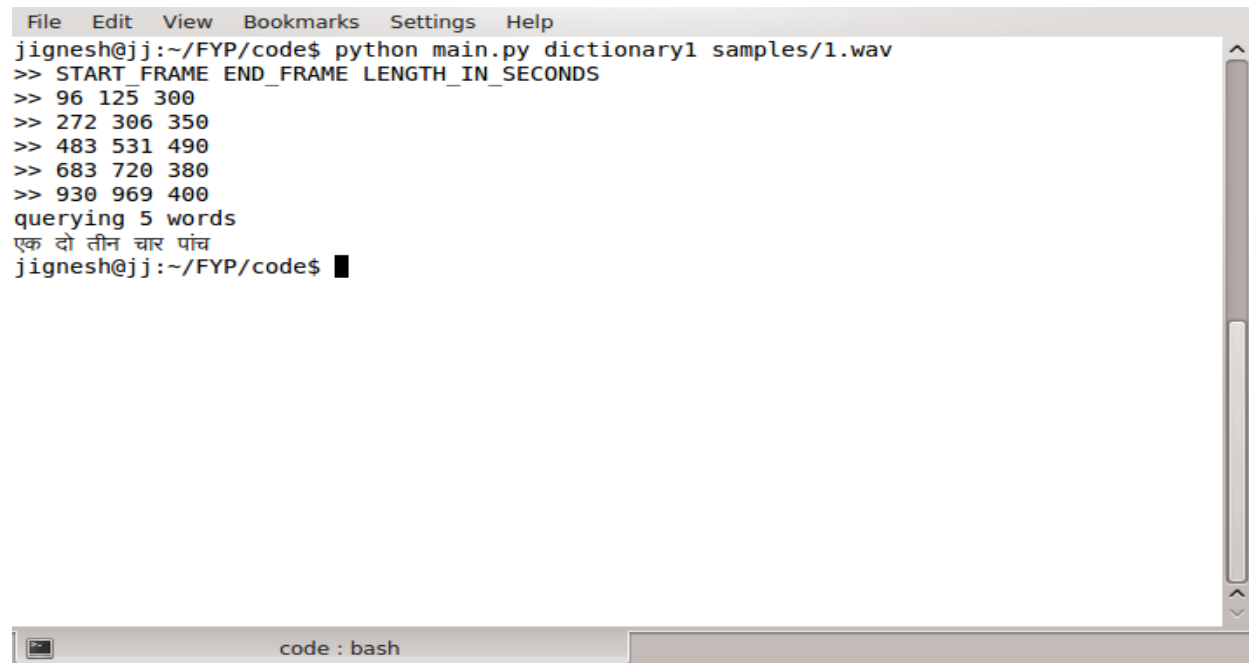


```
HMM : bash - Konsole
gaurav@deshmukh:~/HMM$ python -W ignore main.py output.wav dictionary Models m
eta
Loading models...
Done

>> START_FRAME END_FRAME LENGTH_IN_SECONDS
>> 0 452 4530
>> 545 603 590
>> 700 767 680
>> 850 910 610
>> 999 1052 540
>> 1167 1204 380
>> 1315 1374 600
>> 1477 1537 610
Detected Words:
पाँच
छः
छः
सात
आठ
दो
सात
```

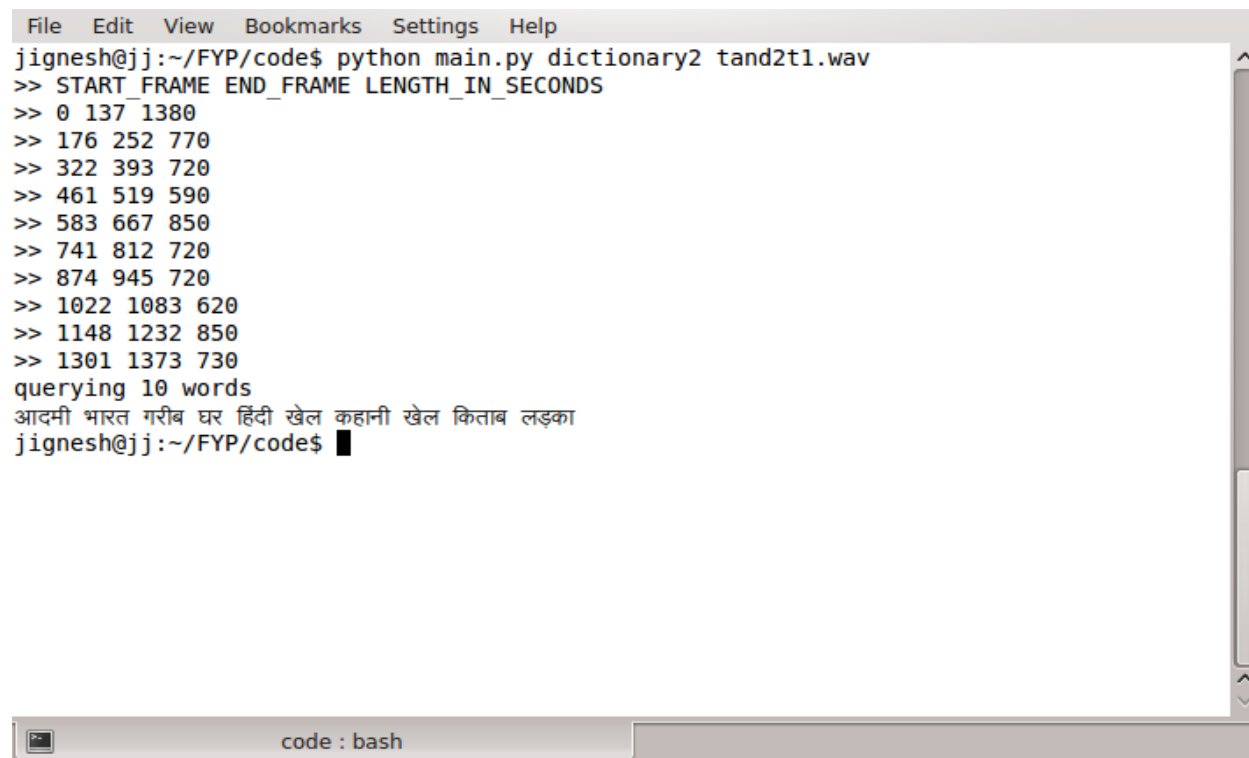
Detecting words from recorded audio file 'output.wav'

2. DTW



```
File Edit View Bookmarks Settings Help
jignesh@jj:~/FYP/code$ python main.py dictionary1 samples/1.wav
>> START_FRAME END_FRAME LENGTH_IN_SECONDS
>> 96 125 300
>> 272 306 350
>> 483 531 490
>> 683 720 380
>> 930 969 400
querying 5 words
एक दो तीन चार पांच
jignesh@jj:~/FYP/code$
```

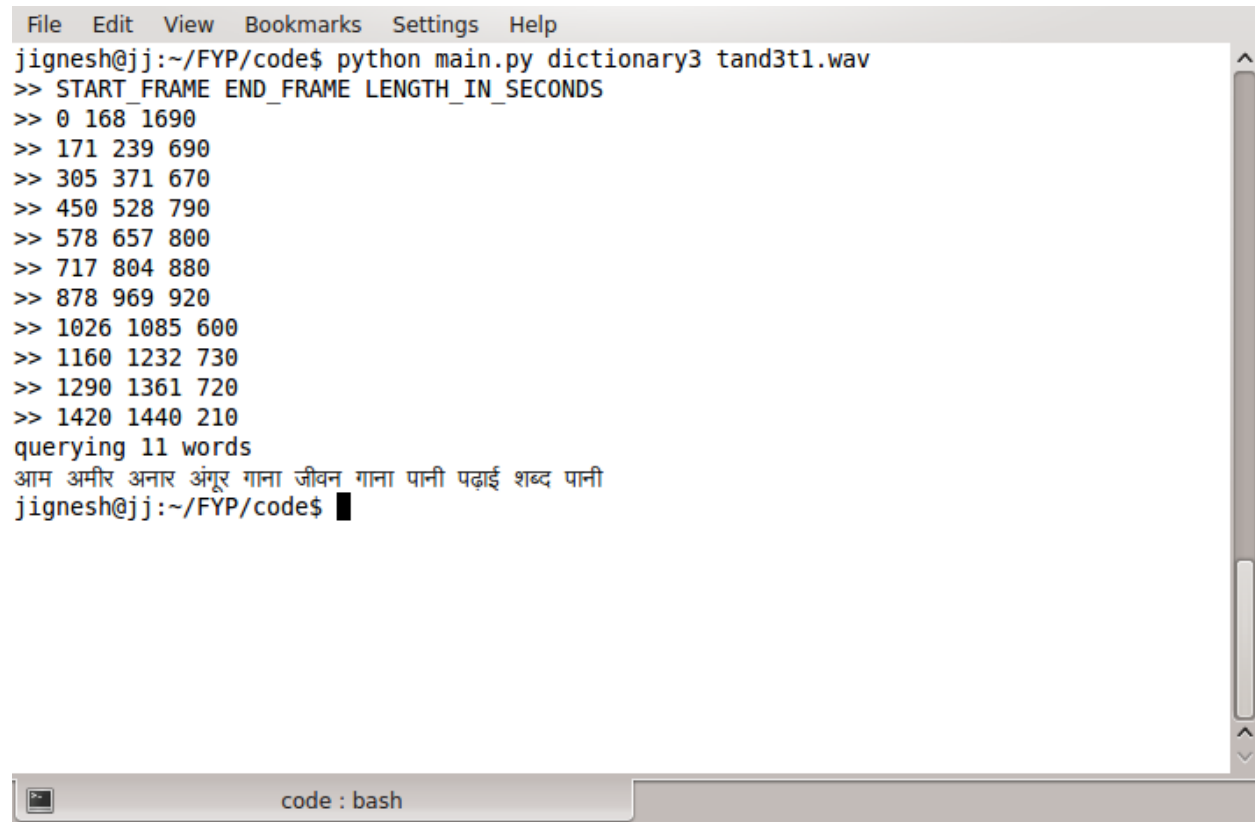
Querying Set 1 words



```
File Edit View Bookmarks Settings Help
jignesh@jj:~/FYP/code$ python main.py dictionary2 tand2t1.wav
>> START_FRAME END_FRAME LENGTH_IN_SECONDS
>> 0 137 1380
>> 176 252 770
>> 322 393 720
>> 461 519 590
>> 583 667 850
>> 741 812 720
>> 874 945 720
>> 1022 1083 620
>> 1148 1232 850
>> 1301 1373 730
querying 10 words
आदमी भारत गरीब घर हिंदी खेल कहानी खेल किताब लड़का
jignesh@jj:~/FYP/code$
```

Querying Set 2 words

Hindi Speech Recognition



```
File Edit View Bookmarks Settings Help
jignesh@jj:~/FYP/code$ python main.py dictionary3 tand3t1.wav
>> START_FRAME END_FRAME LENGTH_IN_SECONDS
>> 0 168 1690
>> 171 239 690
>> 305 371 670
>> 450 528 790
>> 578 657 800
>> 717 804 880
>> 878 969 920
>> 1026 1085 600
>> 1160 1232 730
>> 1290 1361 720
>> 1420 1440 210
querying 11 words
आम अमीर अनार अंगूर गाना जीवन गाना पानी पढ़ाई शब्द पानी
jignesh@jj:~/FYP/code$
```

Querying Set 3 words

4.3 Training Dataset:

Our training dataset contains of 3 sets of words containing 10 words each. Each word is spoken 5 times by 4 different people, thus there are 20 different samples for each word. The 3 sets are as follows:

1. Set 1 (Hindi Numerals):

एक, दो, तीन, चार, पांच, छः, सात, आठ, नौ, दस

2. Set 2 (Commonly used Hindi words):

लड़का, भारत, घर, किताब, खेल, कहानी, जीत, आदमी, हिंदी, गरीब

3. Set 3: (Commonly used Hindi words):

अमीर, खाना, अनार, शब्द, पानी, आम, गाना, अंगूर, पढ़ाई, जीवन

4.4 Testing:

Recognition was tried on three kinds of sounds

1. **Seen sound:** The sound files used to train the models
2. **Unseen sound seen user:** Unused sound file of the user whose other sound files were used for training.
3. **Unseen user:** The user whose voice we not used for training.

The result of the experiment were as shown:

1. Set 1 (Hindi Numerals):

Using HMM

Type of Sound	Number of Sounds	Correct Recognition
Seen sound	30	24
Unseen sound seen user	30	18
Unseen user	20	8

Using DTW

Type of Sound	Number of Sounds	Correct Recognition
Seen sound	30	30
Unseen sound seen user	30	21
Unseen user	20	5

2. Set 2 (Commonly used Hindi Words)

Using HMM

Type of Sound	Number of Sounds	Correct Recognition
Seen sound	30	20
Unseen sound seen user	30	15
Unseen user	20	7

Using DTW

Type of Sound	Number of Sounds	Correct Recognition
Seen sound	30	28
Unseen sound seen user	30	22
Unseen user	20	6

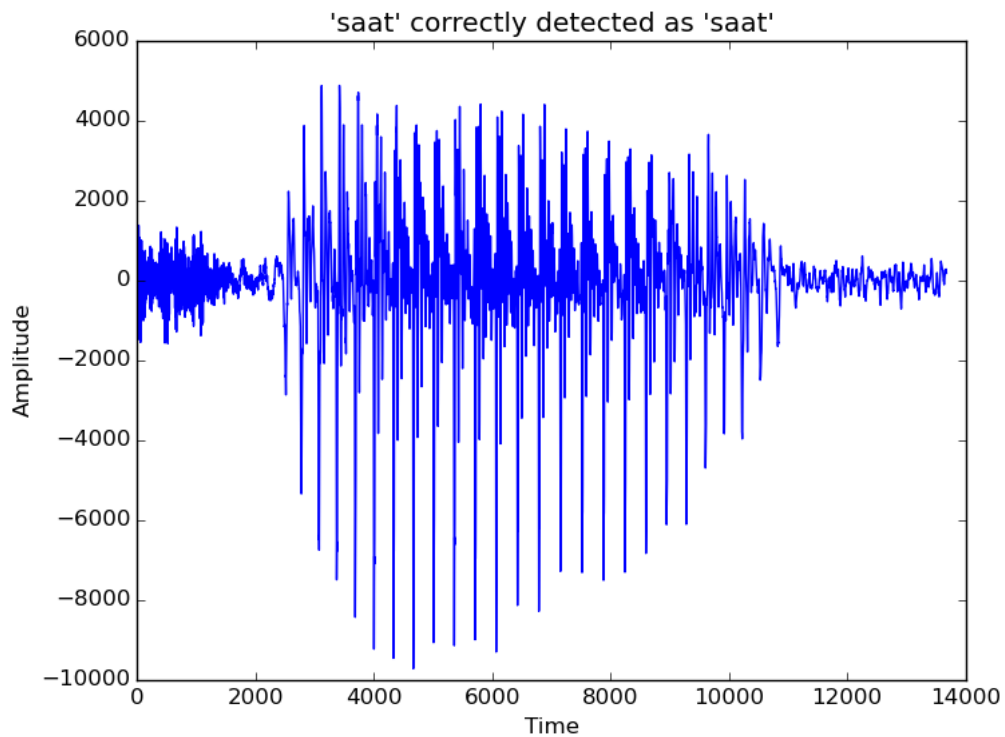
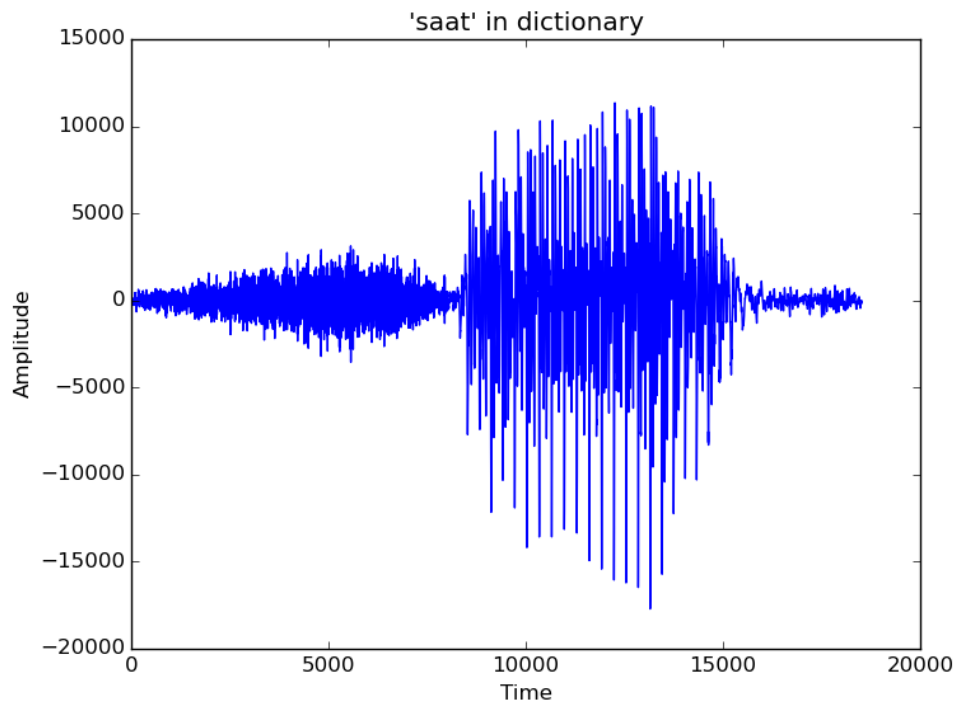
3. Set 3 (Commonly used Hindi Words)

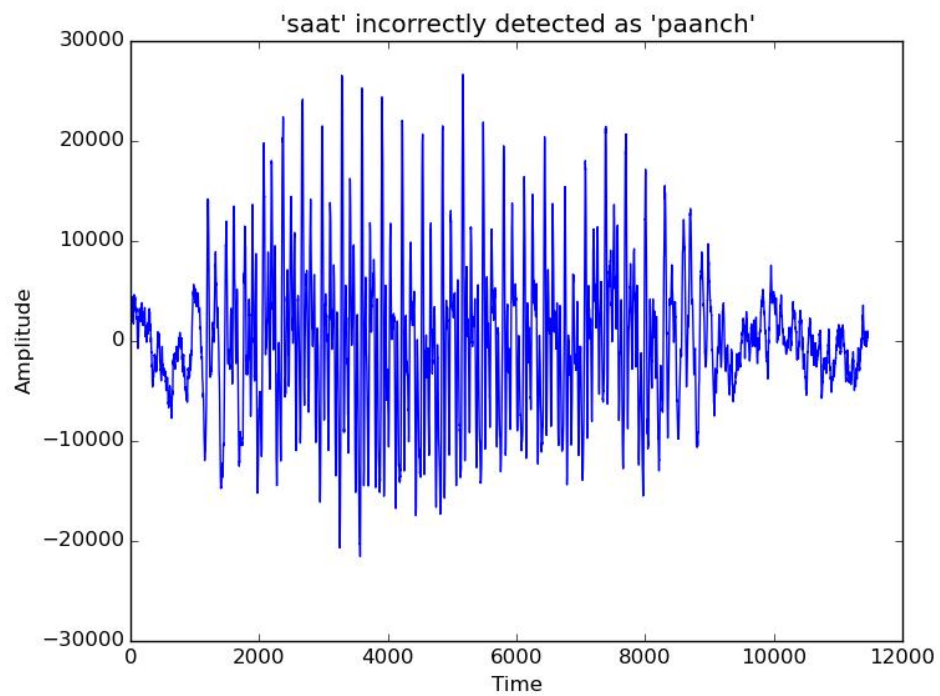
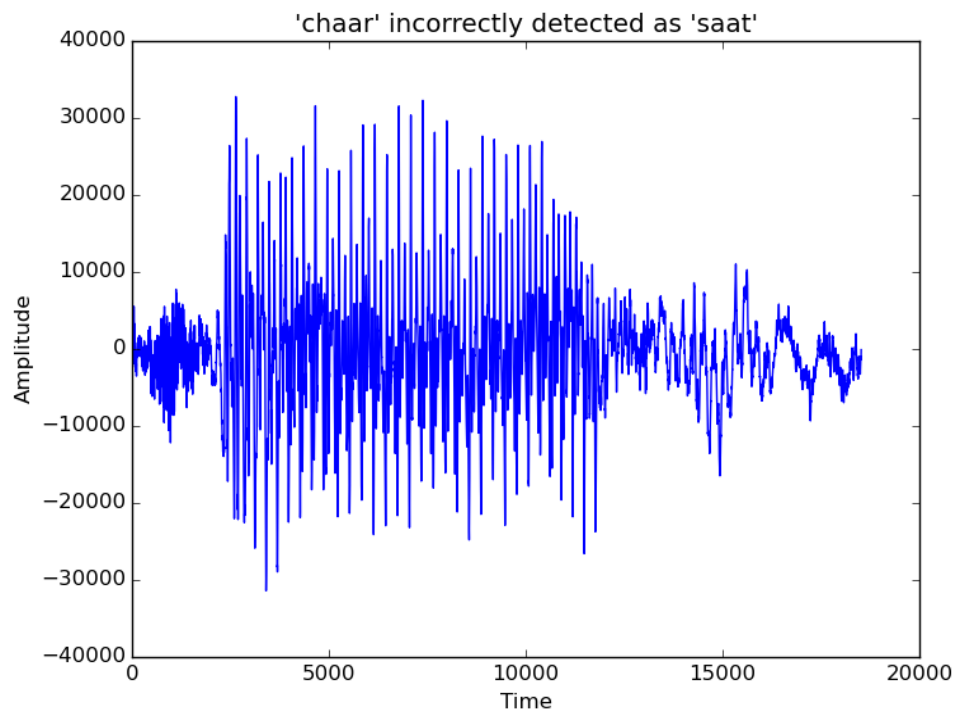
Using HMM

Type of Sound	Number of Sounds	Correct Recognition
Seen sound	30	21
Unseen sound seen user	30	17
Unseen user	20	5

Using DTW

Type of Sound	Number of Sounds	Correct Recognition
Seen sound	30	29
Unseen sound seen user	30	19
Unseen user	20	4





Chapter 5

Conclusion and Future Work

5.1 Conclusion:

Our finite vocabulary system uses 30 words as training data. It achieves 72% accuracy for seen sounds, 56% accuracy for seen users' unseen sounds, and 50% accuracy for unseen users' sounds using HMM algorithm. Whereas the DTW algorithm achieves 100% accuracy for seen sounds, 67% accuracy for seen users' unseen sounds, and 25% accuracy for unseen users' sounds. Therefore, the system works reasonably well for finite vocabulary using the two algorithms.

The system can further be improved by addressing the issues described in the next section. The said improvements can improve the accuracy and efficiency of the system.

5.2 Limitations and future work:

Our speech-to-text system contains only one layer that learns finite vocabulary from the given training data, and predicts isolated words. However, an ideal speech-to-text system should contain multiple layers stacked on each other for maximum efficiency.

The lowest layer is predicting phones from a given word, considering that pronunciation of a particular phone depends (at least) on the preceding and succeeding phone. This consideration can remove the finite vocabulary constraint, and predict arbitrary sequence of words, with the help of other layers. Another layer is HMMs of HMMs created in the previous layer, thereby making it a continuous speech recognition system. This requires a language model for Hindi which takes grammatical structure of sentences of Hindi into consideration, and predicts next word given the current sequence of words. However, there has been practically no research performed for language model of Hindi, and since it is a combination of linguistics and formal language theory, it is out of scope of this project.

The number of speakers involved in the training data is 4. This is the reason of size of training dataset as well as amount of words in the vocabulary being limited. Given a large dataset, the HMMs can be trained for variations in speech, dialects etc.

Another slight improvement can be achieved by taking feedback from the user. The system can be modified to take an optional feedback about the words that the system failed to identify correctly. If it is observed that a specific word is particularly low accuracy, then the recording from the user can be used to improve the training of the system.

5.3 References:

1. IBM - CDAC Shrutlekhan - Rajyabhasha:
<https://www.research.ibm.com/irl/cdachindi.html>
2. Google Keyboard on Google Play:
<https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=en>
3. Rabiner L. *'A Tutorial On Hidden Markov Models and Selected Applications in Speech Recognition'*
4. Mel Frequency Cepstrum Coefficients (MFCC) Tutorial:
<http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>
5. Baum-Welch Algorithm - Wikipedia:
https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm
6. Dynamic Time Warping Algorithm - Wikipedia:
http://en.wikipedia.org/wiki/Dynamic_time_warping
7. R. Makhijani, R. Gupta, *'Isolated Word Speech Recognition System Using Dynamic Time Warping'*
8. J. Cernoky, *'Speech Recognition -- Intro and DTW'*
9. B. Gawali, et al, *'Marathi Isolated Word Recognition System Using MFCC and DTW Features'*
10. M. Gales, S. Young, *'The Application of Hidden Markov Models in Speech Recognition'*
11. R. Gupta, *'Speech Recognition For Hindi'*
12. R. Kumar et al., *'Development of Indian Language Speech Databases for Large Vocabulary Speech Recognition Systems'*
13. NPTEL lectures on Hidden Markov Models:
<https://www.youtube.com/watch?v=1evnHDZfgs4>
<https://www.youtube.com/watch?v=dc8a-uOKe3I>

14. Single Speaker Speech Recognition in Hidden Markov Models:

<http://kastnerkyle.github.io/blog/2014/05/22/single-speaker-speech-recognition/>

15. CMU Sphinx, Open Source Speech Recognition Toolkit:

<http://cmusphinx.sourceforge.net/>

16. Python DTW Module: *<http://cmusphinx.sourceforge.net/>*