Indian Institute of Technology, Guwahati

Department of Computer Science and Engineering

Project Report
On

"Speech-based Medical Prescription Generator"

Based on

Speech Recognition System

Course: CS566 Speech Processing

Submitted to:
Prof. P. K. Das

Submitted by:
Darshika Verma (214101014)
Sandeep Agri (214101047)

TABLE OF CONTENT

ABSTRACT

This document defines a set of evaluation criteria and test methods for speech recognition systems used to generate the medical prescription for some common symptoms. This report is on the project which detects the symptom and displays the appropriate medication for the particular symptom.

INTRODUCTION

In this report, we concentrate on the speech recognition programs that are human-computer interactive. When software evaluators observe humans testing such software programs, they gain valuable insights into technical problems and barriers that they may never witness otherwise. . Testing speech recognition products for universal usability is an important step before considering the product to be a viable solution for its customers later.

This document concerns Speech Recognition accuracy in detecting the spoken symptom and generating the appropriate medical prescription, which is a critical factor in the development of hands-free human-machine interactive devices.

There are two separate issues that we want to test through this project:

- Accuracy of word recognition
- Software friendliness.

Major factors that impede the accuracy of word recognition in the environment are noise sources and system noise.

But, what is speech recognition?

Speech recognition is the ability of a machine or program to identify words spoken aloud and convert them into readable text. The working of the Speech Recognition system is: Speech recognition software works by breaking down the audio of a speech recording into individual sounds, analyzing each sound, using algorithms to find the most probable word fit in that language, and transcribing those sounds into text. You speak into a microphone and the computer transforms the

sound of your words into the text used by your word processor or other applications available on your computer. The computer may repeat what you just said or it may give you a prompt for what you are expected to say next. This is the central promise of interactive speech recognition. You also have to correct errors virtually as soon as they happen.

The new voice recognition systems are certainly much easier to use. You can speak at a normal pace without leaving distinct pauses between words. However, you cannot really use “natural speech” as claimed by the manufacturers. You must speak clearly, as you do when you speak to a Dictaphone or when you leave someone a telephone message. Remember, the computer is relying solely on your spoken words. It cannot interpret your tone or inflection, and it cannot interpret your gestures and facial expressions, which are part of everyday human communication. Some of the systems also look at whole phrases, not just the individual words you speak. They try to get information from the context of your speech, to help work out the correct interpretation.

The goal of this project is to define a set of evaluation criteria and test methods for the interactive voice recognition systems for detecting the correct symptom and generating the corresponding medical prescription for the symptom.

PROPOSED METHODOLOGY

Basic requirements to develop this project are as follows:

- Windows OS
- Microsoft Visual Studio 2010
- C++ 11 integrated with VS2010
- Recording Module

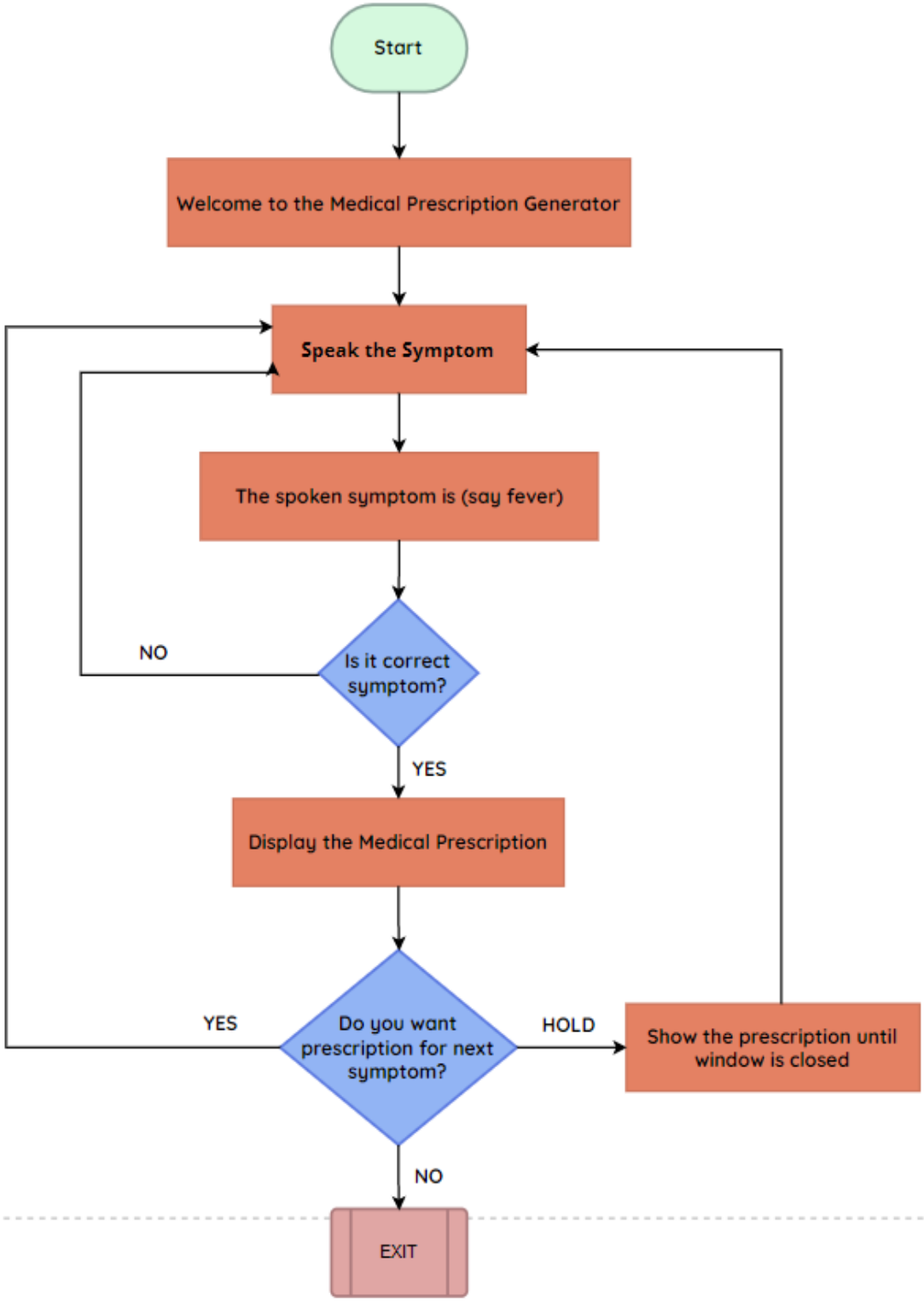
With the availability of the above software, we further proceed in modeling the logic.

The prerequisites of this project are:

- ➢ Basic i/o operations on file
- ➢ Pre-processing of speech data
- ➢ Feature extraction
- ➢ Modeling of extracted feature

Enhancing model above discussed topics are broadly elaborated in the experimental setup section.

With the availability of the above tools, we further proceeded. Below is the flow chart for our project.



EXPERIMENTAL SETUP

This project is divided into the following modules:

- ➔ **Training Module**
- ➔ **Testing Module**
- ➔ **Live Training Module**

1. Training Module

The flow for training over data is as follows:

i. Record the 20 utterances of each word as data for the system.

ii. Extract stable frames for every utterance of each word.

iii. Using local distance analysis (in vector quantization) calculate the observation sequence for each utterance.

iv. Pass the obtained observation sequence to HMM for model designing.

v. Now enhance the model using HMM re-estimation algorithm.

Now, a reference model is ready for our project. The training of data is not integrated with the GUI applications. This is a different module that will just evaluate the reference model.

2. Testing Module

The system will give instructions to the user and the user is required to follow them.

The flow of testing is as follows:

i. Live recording of data is done when the system instructs using the live recording module provided to the user.

ii. Testing the data with pre-trained models.

iii. Verifying the symptom is detected with the user.

iv. If verification is successful display a medical prescription for the symptom.

v. If verification fails, record the input again.

3. Live Training Module

The system will give instructions to the user and the user is required to follow them.

The flow of live training the model with the new speaker is as follows:

i. The speaker has to speak 20 utterances of each word in the recording module provided.

ii. Train the model again with the newly recorded data.

iii. New lambda models are stored in the folder.

RESULT

For offline testing, we took 20 recordings of each word, with deterministic difference of maximum and second maximum P(O/lambda) we got 70% accuracy and without considering deterministic difference we got 95% accuracy.

The prescription list of spoken symptom is successfully generated.

SOURCE CODE

```
#ifndef hmm_h
#define hmm_h

#include "stdafx.h"
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "fever.h"
#include <Windows.h>
#include <MMSystem.h>


#define N 5;
#define M 32;



// variables for hmm
static int T = 0, end = 0;
static int i = 0, j = 0, k = 0, l = 0, y = 0, z = 0, obs = 0, itr = 0;
static char c[200];
static int si = N;
static int O[121];
static long double silence[13];
static long double codebook[33][13];
static double store[10][5][12], inp[48000], dc_shift, max = 0.0, min = 0.0, temp[320],
check = 0.0, R[13], C[13], A_f[12];
static double tok[12] = {1.0, 3.0, 7.0, 13.0, 19.0, 22.0, 25.0, 33.0, 42.0, 50.0, 56,
61.0}; // tokhura's weight
static double tok_d = 0.0, min_d = 999999.0;
static int ind = -1;
static double ref[5][12];
static int count = 0;
static char word[5] = "1";
static char iteration[5] = "1";

//////////////////////////////////////////////////////
// lambda's declared for each digit

// for initial lambda
static long double A_l_i[6][6];
static long double B_l_i[6][33];
static long double Pi_l_i[6];

// for digit 0
static long double A_l_0[6][6];
static long double B_l_0[6][33];
static long double Pi_l_0[6];

// for word 1
static long double A_l_1[6][6];
static long double B_l_1[6][33];
static long double Pi_l_1[6];
```

```
// for word 2
static long double A_l_2[6][6];
static long double B_l_2[6][33];
static long double Pi_l_2[6];

// for word 3
static long double A_l_3[6][6];
static long double B_l_3[6][33];
static long double Pi_l_3[6];

// for word 4
static long double A_l_4[6][6];
static long double B_l_4[6][33];
static long double Pi_l_4[6];

// for digit 5
static long double A_l_5[6][6];
static long double B_l_5[6][33];
static long double Pi_l_5[6];

////////////////////////////////////////////////////////////////////////////

static long double A_1[6][6];
static long double B_1[6][33];
static long double Pi_1[6];
static long double A_bar[6][6];
static long double B_bar[6][33];
static long double Pi_bar[6];
static long double delta_1[121][6];
static int si_1[121][6];
static long double max_delta = 0.0;
static int max_si = 0;
static long double temp_delta = 0.0;
static long double p_star = 0.0;
static long double p_star_new = 0.0;
static int q_star = 0;
static int Q_star[121];
static long double alpha[121][6];
static long double beta[121][6];
static long double P_O_L = 0.0;
static long double gama[121][6];
static long double Zi[121][6][6];

////////////////////////////////////////////////////////////////////////////

static FILE *text;


static void initialise()
{
    if (itr == 1)
    {
        Pi_1[1] = 1;
        for (i = 2; i <= 5; i++)
        {
            Pi_1[i] = 0;
        }
```

```
        for (i = 1; i <= 5; i++)
        {
            for (j = 1; j <= 5; j++)
            {
                A_1[i][j] = 0;
            }
        }
        A_1[1][1] = A_1[2][2] = A_1[3][3] = A_1[4][4] = 0.8;
        A_1[1][2] = A_1[2][3] = A_1[3][4] = A_1[4][5] = 0.2;
        A_1[5][5] = 1;

        for (i = 1; i <= 5; i++)
        {
            for (j = 1; j <= 32; j++)
            {
                B_1[i][j] = 1.0 / 32.0;
            }
        }
    }
    else
    {
        for (i = 1; i <= 5; i++)
        {
            Pi_1[i] = Pi_l_i[i];
        }

        for (i = 1; i <= 5; i++)
        {
            for (j = 1; j <= 5; j++)
            {
                A_1[i][j] = A_l_i[i][j];
            }
        }

        for (i = 1; i <= 5; i++)
        {
            for (j = 1; j <= 32; j++)
            {
                B_1[i][j] = B_l_i[i][j];
            }
        }
    }
}

static void state_seq() // problem 2 (viterbi)
{
    // Viterbi's algo

    // initialisation

    for (j = 1; j <= 5; j++)
    {
        delta_1[1][j] = Pi_1[j] * B_1[j][O[1]];
        si_1[1][j] = 0;
    }

    // recurssion
```

```
    for (i = 2; i <= T; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            for (k = 1; k <= 5; k++)
            {
                temp_delta = (delta_1[i - 1][k] * A_1[k][j]);

                if (temp_delta > max_delta)
                {
                    max_delta = temp_delta;
                    max_si = k;
                }
            }
            delta_1[i][j] = max_delta * B_1[j][O[i]];
            si_1[i][j] = max_si;
            max_delta = 0.0;
            max_si = 0;
        }
    }

    // termination
    for (i = 1; i <= 5; i++)
    {
        if (delta_1[T][i] > p_star)
        {
            p_star = delta_1[T][i];
            q_star = i;
        }
    }

    // backtrekking
    Q_star[T] = q_star;
    for (i = T - 1; i >= 1; i--)
    {
        Q_star[i] = si_1[i + 1][Q_star[i + 1]];
    }
    printf("State sequence is\n\n");
    for (i = 1; i <= T; i++)
    {
        printf("%d ", Q_star[i]);
    }
    printf("\n");
    q_star = 0;
}

// forward pass
static void forward_pass() // problem 1
{

    long double temp = 0.0;

    // initialisation
    for (i = 1; i <= 5; i++)
    {
        alpha[1][i] = Pi_1[i] * B_1[i][O[1]];
    }
```

```
    // induction
    for (i = 2; i <= T; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            for (k = 1; k <= 5; k++)
            {
                temp += (alpha[i - 1][k] * A_1[k][j]);
            }
            alpha[i][j] = temp * B_1[j][O[i]];
            temp = 0.0;
        }
    }

    P_O_L = 0.0;
    // termination
    for (i = 1; i <= 5; i++)
    {
        P_O_L += (alpha[T][i]);
    }
    printf("P(O/lambda)------->%e \n", P_O_L);
}

// backward pass
static void backward_pass()
{

    long double temp = 0.0;
    // initialisation
    for (i = 1; i <= 5; i++)
    {
        beta[T][i] = 1;
    }

    // induction
    for (i = T - 1; i >= 1; i--)
    {
        for (j = 1; j <= 5; j++)
        {
            for (k = 1; k <= 5; k++)
            {
                temp += A_1[j][k] * B_1[k][O[i + 1]] * beta[i + 1][k];
            }
            beta[i][j] = temp;
            temp = 0.0;
        }
    }
}

// create gama
static void create_gama()
{
    long double temp = 0.0;
    for (k = 1; k <= T; k++)
    {
        for (i = 1; i <= 5; i++)
        {
            temp += alpha[k][i] * beta[k][i];
```

```
        }
        for (i = 1; i <= 5; i++)
        {
            gama[k][i] = (alpha[k][i] * beta[k][i]) / temp;
        }
        temp = 0.0;
    }
}

// re-estimation
static void reestimation()
{
    // creating Zi
    long double temp = 0.0;
    long double temp2 = 0.0;
    for (k = 1; k <= T - 1; k++)
    {
        for (i = 1; i <= 5; i++)
        {
            for (j = 1; j <= 5; j++)
            {
                temp += alpha[k][i] * A_1[i][j] * B_1[j][O[k + 1]] * beta[k + 1][j];
            }
        }
        for (i = 1; i <= 5; i++)
        {
            for (j = 1; j <= 5; j++)
            {
                Zi[k][i][j] = (alpha[k][i] * A_1[i][j] * B_1[j][O[k + 1]] * beta[k +
1][j]) / temp;
            }
        }
        temp = 0.0;
    }

    // calculating Pi_bar
    for (i = 1; i <= 5; i++)
    {
        Pi_bar[i] = gama[1][i];
    }

    // calculating A_bar
    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            for (k = 1; k <= T - 1; k++)
            {
                temp += Zi[k][i][j];
                temp2 += gama[k][i];
            }
            A_bar[i][j] = temp / temp2;
            temp = 0.0;
            temp2 = 0.0;
        }
    }

    int max_ind = -1;
```

```
long double sum = 0;
long double max = -1;
long double diff = 0;

// making A_bar stichiometric
for (i = 1; i <= 5; i++)
{
    for (j = 1; j <= 5; j++)
    {
        sum += A_bar[i][j];
        if (A_bar[i][j] > max)
        {
            max = A_bar[i][j];
            max_ind = j;
        }
    }
    diff = 1.0 - sum;
    A_bar[i][max_ind] += diff;
    sum = 0;
    max = -1;
    max_ind = -1;
}

// calculating B_bar
for (i = 1; i <= 5; i++)
{
    for (j = 1; j <= 32; j++)
    {
        for (k = 1; k <= T - 1; k++)
        {
            if (O[k] == j)
            {
                temp += gama[k][i];
            }
            temp2 += gama[k][i];
        }

        B_bar[i][j] = temp / temp2;
        if (B_bar[i][j] == 0)
        {
            B_bar[i][j] = 1e-030;
        }
        temp = 0.0;
        temp2 = 0.0;
    }
}

// making B_bar stichiometric
for (i = 1; i <= 5; i++)
{
    for (j = 1; j <= 32; j++)
    {
        sum += B_bar[i][j];
        if (B_bar[i][j] > max)
        {
            max = B_bar[i][j];
            max_ind = j;
        }
```

```
        }
        diff = 1.0 - sum;
        B_bar[i][max_ind] += diff;
        sum = 0;
        max = -1;
        max_ind = -1;
    }
}

// for calculating CC
//  normalisation
static void normalise()
{
    int i = 0;
    for (i = 0; i < 320; i++)
    {
        if (temp[i] > 0)
        {
            temp[i] = (5000.0 / max);
        }
        else
        {
            temp[i] = (-5000.0 / min);
        }
    }
}

// For calculating Ri's
static void C_R()
{
    double sum = 0;
    int i = 0, j = 0;
    for (i = 0; i < 13; i++)
    {
        sum = 0;
        for (j = 0; j < 320 - i; j++)
        {
            sum += temp[j] * temp[i + j];
        }
        R[i] = sum;
    }
}

// For calculating Ai's
static void C_A()
{
    double A[13][13]; // store values for alpha's, the last row wil have all A values
from index 1 to 12
    int i = 0, j = 0;
    double E[13], K[13], temp = 0.0; // E is to store error matrix
    for (i = 0; i < 13; i++)
    {
        E[i] = 0.0;
        K[i] = 0.0;
        for (j = 0; j < 13; j++)
        {
            A[i][j] = 0.0;
        }
```

```
    }
    E[0] = R[0];
    for (i = 1; i < 13; i++)
    {
        temp = 0.0;
        for (j = 1; j <= i - 1; j++)
        {
            temp += A[i - 1][j] * R[i - j];
        }
        K[i] = R[i] - temp;
        K[i] /= E[i - 1];
        A[i][i] = K[i];
        for (j = 1; j <= i - 1; j++)
        {
            A[i][j] = A[i - 1][j] - (K[i] * A[i - 1][i - j]);
        }
        E[i] = (1 - (K[i] * K[i])) * E[i - 1];
    }
    for (i = 1; i <= 12; i++)
    {
        A_f[i - 1] = A[12][i];
    }
       //
}

// for calculating Ci's
static void C_C()
{
    int i = 0, j = 0;
    double temp = 0.0;
    for (i = 0; i < 13; i++)
        C[i] = 0.0;
    C[0] = log10(R[0] * R[0]);
    for (i = 1; i < 13; i++)
    {
        temp = 0.0;
        for (j = 1; j <= i - 1; j++)
        {
            temp += (double(j) / double(i)) * A_f[i - 1 - j] * C[j];
        }
        C[i] += temp + A_f[i - 1];
    }
}

static void get_obs_new()
{
       FILE *text;
    count = 0;
    dc_shift = 0.0;
    j = 0;

    j = 0;
    i = 0;
    char file[50] = "new-lambda/words/word_";
    strcat(file, word);
    strcat(file, "/word_");
    strcat(file, word);
    strcat(file, "_");
```

```
strcat(file, iteration);
strcat(file, ".txt");
printf("%s\n", file);

text = fopen(file, "r");
while (!feof(text))
{
    fscanf(text, "%s", c);
    if (i > 100)
    {
        for (j = 0; j < 500; j++)
        {
            fscanf(text, "%s", c);
            count += 1;
            dc_shift += atof(c);
        }
        break;
    }
    i++;
}
dc_shift /= count;
fclose(text);

i = 0;
j = 0;

text = fopen(file, "r");
while (!feof(text))
{
    fscanf(text, "%s", c);
    int val = atof(c);
    if (i > 1000 && val >= 500)
    {

        for (j = 0; j < 10000; j++)
        {
            fscanf(text, "%s", c);
            inp[j] = atof(c) - dc_shift;
        }
        break;
    }
    i++;
}
fclose(text);

for (j = 9999; j > 0; j--)
{
    if (inp[j] > 500)
    {
        end = j;
        break;
    }
}

T = (((end + 1) - 320) / 80) + 1;
if (T > 120)
{
    T = 120;
```

```
    }
    // calculating Ci's for each frame and storing it in 3-D array store
    for (j = 0; j < T; j++) // iterating for each frame
    {
        min = 0.0;
        max = 0.0;
        l = 0;
        for (k = 80 * j; k < (80 * j) + 320; k++) // seperating values frame wise,
sliding window with shift of 80
        {
            temp[l] = inp[k];
            if (temp[l] < min)
                min = temp[l];
            if (temp[l] > max)
                max = temp[l];
            l++;
        }

        // normalising samples
        normalise();

        // calculating Ri's
        C_R();

        // calculating Ai's
        C_A();

        // calculating Ci's
        C_C();

        // Applying Raised SIN window to Ci's
        for (l = 1; l <= 12; l++)
        {
            C[l] *= (1 + 6 * (sin((3.14 * l) / 12)));
        }

        // calculating tohura's distance from codebook to get observation number
        for (l = 1; l <= 32; l++)
        {
            for (k = 1; k <= 12; k++)
            {
                tok_d += ((C[k] - codebook[l][k]) * (C[k] - codebook[l][k]) * tok[k]);
            }
            if (tok_d < min_d)
            {
                min_d = tok_d;
                ind = l;
            }
            tok_d = 0.0;
        }
        O[j + 1] = ind;
        min_d = 99999;
        ind = -1;
    }
    // loc[24]='\0'; //reseting the file name
}

static void create_lambda_new()
```

```
{
       FILE *text;
    for (itr = 1; itr <= 3; itr++)
    {
        printf("itr %d \n\n", itr);
        for (obs = 1; obs <= 20; obs++)
        {
            // getting observation sequence
            itoa(obs, iteration, 10);

            get_obs_new();

            // initialising with starting lambda
            initialise();

printf("================================================================================
==================================================\n");
            printf("FOR OBSERVATON SEQUENCE\n\n");
            for (i = 1; i <= T; i++)
            {
                printf("%d  ", O[i]);
            }
            printf("\n\n");
            int l = 0;
            long double p_star_old = 0.0;
            while (l++ < 200) // iterating for max 200 times
            {

                printf("Iteration is %d \n", l);


                // calculating the all matrices and variable again to get new p_star
                forward_pass();
                backward_pass();

                create_gama();
                state_seq();

                reestimation();

                // putting again lambda_bar to lambda
                for (i = 1; i <= 5; i++)
                {
                    Pi_1[i] = Pi_bar[i];
                }

                for (i = 1; i <= 5; i++)
                {
                    for (j = 1; j <= 5; j++)
                    {
                        A_1[i][j] = A_bar[i][j];
                    }
                }

                for (i = 1; i <= 5; i++)
                {
                    for (j = 1; j <= 32; j++)
                    {
```

```
            B_1[i][j] = B_bar[i][j];
        }
    }
    printf("P_star is ----> %g \n\n", p_star);
    if (p_star < p_star_old) // condition for saturation
    {
        break;
    }
    p_star_old = p_star;
    p_star_new = 0.0;
    p_star = 0.0;
}

// saving in optimal lambda

if (atoi(word) == 1)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_1[i] += Pi_1[i];
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            A_l_1[i][j] += A_1[i][j];
        }
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 32; j++)
        {
            B_l_1[i][j] += B_1[i][j];
        }
    }
}
if (atoi(word) == 2)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_2[i] += Pi_1[i];
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            A_l_2[i][j] += A_1[i][j];
        }
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 32; j++)
        {
            B_l_2[i][j] += B_1[i][j];
```

```
        }
    }
}
if (atoi(word) == 3)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_3[i] += Pi_1[i];
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            A_l_3[i][j] += A_1[i][j];
        }
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 32; j++)
        {
            B_l_3[i][j] += B_1[i][j];
        }
    }
}
if (atoi(word) == 4)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_4[i] += Pi_1[i];
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            A_l_4[i][j] += A_1[i][j];
        }
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 32; j++)
        {
            B_l_4[i][j] += B_1[i][j];
        }
    }
}
if (atoi(word) == 5)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_5[i] += Pi_1[i];
    }

    for (i = 1; i <= 5; i++)
    {
```

```
            for (j = 1; j <= 5; j++)
            {
                A_l_5[i][j] += A_1[i][j];
            }
        }

        for (i = 1; i <= 5; i++)
        {
            for (j = 1; j <= 32; j++)
            {
                B_l_5[i][j] += B_1[i][j];
            }
        }
    }


}

// averaging final lambda

if (atoi(word) == 1)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_1[i] /= 20;
        Pi_l_i[i] = Pi_l_1[i];
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            A_l_1[i][j] /= 20;
            A_l_i[i][j] = A_l_1[i][j];
        }
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 32; j++)
        {
            B_l_1[i][j] /= 20;
            B_l_i[i][j] = B_l_1[i][j];
        }
    }
}
if (atoi(word) == 2)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_2[i] /= 20;
        Pi_l_i[i] = Pi_l_2[i];
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
```

```
            A_l_2[i][j] /= 20;
            A_l_i[i][j] = A_l_2[i][j];
        }
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 32; j++)
        {
            B_l_2[i][j] /= 20;
            B_l_i[i][j] = B_l_2[i][j];
        }
    }
}
if (atoi(word) == 3)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_3[i] /= 20;
        Pi_l_i[i] = Pi_l_3[i];
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            A_l_3[i][j] /= 20;
            A_l_i[i][j] = A_l_3[i][j];
        }
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 32; j++)
        {
            B_l_3[i][j] /= 20;
            B_l_i[i][j] = B_l_3[i][j];
        }
    }
}
if (atoi(word) == 4)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_4[i] /= 20;
        Pi_l_i[i] = Pi_l_4[i];
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            A_l_4[i][j] /= 20;
            A_l_i[i][j] = A_l_4[i][j];
        }
    }

    for (i = 1; i <= 5; i++)
```

```
    {
        for (j = 1; j <= 32; j++)
        {
            B_l_4[i][j] /= 20;
            B_l_i[i][j] = B_l_4[i][j];
        }
    }
}
if (atoi(word) == 5)
{
    for (i = 1; i <= 5; i++)
    {
        Pi_l_5[i] /= 20;
        Pi_l_i[i] = Pi_l_5[i];
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            A_l_5[i][j] /= 20;
            A_l_i[i][j] = A_l_5[i][j];
        }
    }

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 32; j++)
        {
            B_l_5[i][j] /= 20;
            B_l_i[i][j] = B_l_5[i][j];
        }
    }
}


int max_ind = -1;
long double sum = 0;
long double max = -1;
long double diff = 0;

// making A__l_i stichiometric
for (i = 1; i <= 5; i++)
{
    for (j = 1; j <= 5; j++)
    {
        sum += A_l_i[i][j];
        if (A_l_i[i][j] > max)
        {
            max = A_l_i[i][j];
            max_ind = j;
        }
    }
    diff = 1.0 - sum;
    A_l_i[i][max_ind] += diff;
    sum = 0;
    max = -1;
    max_ind = -1;
```

```
        }

        // making B_bar stichiometric
        for (i = 1; i <= 5; i++)
        {
            for (j = 1; j <= 32; j++)
            {
                sum += B_l_i[i][j];
                if (B_l_i[i][j] > max)
                {
                    max = B_l_i[i][j];
                    max_ind = j;
                }
            }
            diff = 1.0 - sum;
            B_l_i[i][max_ind] += diff;
            sum = 0;
            max = -1;
            max_ind = -1;
        }
    }

    // creating lambda file
    char file[50] = "new-lambda/lambda_";
    strcat(file, word);
    strcat(file, ".txt");
    text = fopen(file, "w");
    for (i = 1; i <= 5; i++)
    {
        fprintf(text, "%g  ", Pi_l_i[i]);
    }
    fprintf(text, "\n\n");

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 5; j++)
        {
            fprintf(text, "%g  ", A_l_i[i][j]);
        }
        fprintf(text, "\n");
    }
    fprintf(text, "\n\n");

    for (i = 1; i <= 5; i++)
    {
        for (j = 1; j <= 32; j++)
        {
            fprintf(text, "%g  ", B_l_i[i][j]);
        }
        fprintf(text, "\n\n");
    }

    fprintf(text, "\n\np_star  %g ", p_star);
    fclose(text);
}

static void record_new()
{
```

```
    printf("You have to utter 20 recordings of each of the following word");
    PlaySound(TEXT("Test fever.wav"), NULL, SND_SYNC);
    // Recording_Module.exe 3 test.wav new-lambda/words/word_1/word_1_1.txt;
    for (int i = 1; i <= 5; i++)
    {
        //printf("\n for word %s\n", digit_to_word(i));
        char file[100] = "Recording_Module.exe 3 test.wav new-lambda/words/word_";
        itoa(i, word, 10);
        strcat(file, word);
        strcat(file, "/word_");
        strcat(file, word);
        strcat(file, "_");
        char itr[5];
        for (int j = 1; j <= 20; j++)
        {
            printf("\nFor iteration %d \n\n", j);
            itoa(j, itr, 10);
            strcat(file, itr);
            strcat(file, ".txt");
            system(file);
            file[63] = '\0';
        }
    }

    word[0] = '1';

    create_lambda_new();

    //-----------------------------------------------------------------------------------
---------------

printf("\n\n\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n\n");
    printf("\n\nFor digit 2\n\n");

    word[0] = '2';

    create_lambda_new();

    //-----------------------------------------------------------------------------------
---------------

printf("\n\n\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n\n");
    printf("\n\nFor digit 3\n\n");

    word[0] = '3';

    create_lambda_new();

    //-----------------------------------------------------------------------------------
---------------

printf("\n\n\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n\n");
    printf("\n\nFor digit 4\n\n");
```

```
    word[0] = '4';

    create_lambda_new();

    //------------------------------------------------------------------------------------
---------------

printf("\n\n\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n\n");
    printf("\n\nFor digit 5\n\n");

    word[0] = '5';

    create_lambda_new();

       printf("\n\n New Lambda are generated in folder \"new lambda\" , you can paste it
in main folder for testing \n\n");
}


static void testing()
{
       FILE *text;
    count = 0;
        i = 0;
        long double s_prob = 0.0;
        system("Recording_Module.exe 3 test.wav test.txt");
        printf("\n Press ENTER\n");
        // taking silent part from recording
        text = fopen("test.txt", "r");
        while (!feof(text))
        {
            fscanf(text, "%s", c);
            if (i > 100)
            {
                for (j = 0; j < 500; j++)
                {
                    fscanf(text, "%s", c);
                    count += 1;
                    dc_shift += atof(c);
                }
                break;
            }
            i++;
        }
        dc_shift /= count;
        fclose(text);

        i = 0;
        j = 0;
        // taking stable frame from recording
        text = fopen("test.txt", "r");
        while (!feof(text))
        {
            fscanf(text, "%s", c);
            int val = atof(c);
            if (i > 1000 && val >= 500)
            {
```

```
                for (j = 0; j < 10000; j++)
                {
                    fscanf(text, "%s", c);
                    inp[j] = atof(c) - dc_shift;
                }
                break;
            }
            i++;
        }
        fclose(text);

        for (j = 9999; j > 0; j--)
        {
            if (inp[j] > 500)
            {
                end = j;
                break;
            }
        }
        T = (((end + 1) - 320) / 80) + 1;
        if (T > 120)
        {
            T = 120;
        }

        // calculating Ci's for each frame and storing it in 3-D array store
        for (j = 0; j < T; j++) // iterating for each frame
        {
            min = 0.0;
            max = 0.0;
            l = 0;
            for (k = 80 * j; k < (80 * j) + 320; k++) // seperating values frame wise,
sliding window with shift of 80
            {
                temp[l] = inp[k];
                if (temp[l] < min)
                    min = temp[l];
                if (temp[l] > max)
                    max = temp[l];
                l++;
            }

            // normalising samples
            normalise();

            // calculating Ri's
            C_R();

            // calculating Ai's
            C_A();

            // calculating Ci's
            C_C();

            // Applying Raised SIN window to Ci's
            for (l = 1; l <= 12; l++)
            {
                C[l] *= (1 + 6 * (sin((3.14 * l) / 12)));
```

```
            }

            // calculating tohura's distance from codebook to get observation number
            for (l = 1; l <= 32; l++)
            {
                for (k = 1; k <= 12; k++)
                {
                    tok_d += ((C[k] - codebook[l][k]) * (C[k] - codebook[l][k]) *
tok[k]);
                }
                if (tok_d < min_d)
                {
                    min_d = tok_d;
                    ind = l;
                }
                tok_d = 0.0;
            }
            O[j + 1] = ind;
            min_d = 99999;
            ind = -1;
        }
        printf("FOR OBSERVATON SEQUENCE\n\n");
        for (i = 1; i <= T; i++)
        {
            printf("%d  ", O[i]);
        }
        int max_ind = -1;
        long double max_prob = -1;
        for (itr = 1; itr <= 5; itr++)
        {
            char file[50] = "lambda_";
            itoa(itr, iteration, 10);
            strcat(file, iteration);
            strcat(file, ".txt");
            text = fopen(file, "r");
            for (i = 1; i <= 5; i++)
            {
                fscanf(text, "%s", c);
                Pi_1[i] = atof(c);
            }

            for (i = 1; i <= 5; i++)
            {
                for (j = 1; j <= 5; j++)
                {
                    fscanf(text, "%s", c);
                    A_1[i][j] = atof(c);
                }
            }

            for (i = 1; i <= 5; i++)
            {
                for (j = 1; j <= 32; j++)
                {
                    fscanf(text, "%s", c);
                    B_1[i][j] = atof(c);
                }
            }
```

```
            fclose(text);

            long double temp = 0.0;

            // initialisation
            for (i = 1; i <= 5; i++)
            {
                alpha[1][i] = Pi_1[i] * B_1[i][O[1]];
            }

            // induction
            for (i = 2; i <= T; i++)
            {
                for (j = 1; j <= 5; j++)
                {
                    for (k = 1; k <= 5; k++)
                    {
                        temp += (alpha[i - 1][k] * A_1[k][j]);
                    }
                    alpha[i][j] = temp * B_1[j][O[i]];
                    temp = 0.0;
                }
            }

            P_O_L = 0.0;
            // termination
            for (i = 1; i <= 5; i++)
            {
                P_O_L += (alpha[T][i]);
            }
            printf("\nlambda %d is giving P_O_L------->%e \n", itr, P_O_L);

            if (max_prob < P_O_L)
            {
                s_prob = max_prob;
                max_prob = P_O_L;
                max_ind = itr;
            }
            if (max_prob > P_O_L && P_O_L > s_prob)
            {
                s_prob = P_O_L;
            }
        }
        if (max_prob / s_prob < 1e5)
        {
            //printf("\n\n Word wasn't identified, try again!! \n and difference is %g
\n", max_prob / s_prob);
                    //textBox1->Text  =  "\n Word wasn't identified, try again!! ";
                    //cout<<"\n Word wasn't identified, try again!! ";
                    system("start default_page.html");
                    Sleep(5000);
                    system("TASKKILL /IM chrome.exe /F");

        }
        //printf("\n\n The said word is %s \n with difference of %g \n",
digit_to_word(max_ind), max_prob / s_prob);
             else
             {
```

```
		if(max_ind==1)
		{
			system("start fever.html");
			Sleep(3000);
			system("TASKKILL /IM chrome.exe /F");
		}
		else if(max_ind==2)
		{
			system("start flu.html");
			Sleep(3000);
			system("TASKKILL /IM chrome.exe /F");
		}
		else if(max_ind==3)
		{
			system("start headache.html");
			Sleep(3000);
			system("TASKKILL /IM chrome.exe /F");
		}
		else if(max_ind==4)
		{
			system("start allergy.html");
			Sleep(3000);
			system("TASKKILL /IM chrome.exe /F");
		}
		else if(max_ind==5)
		{
			system("start vomitting.html");
			Sleep(3000);
			system("TASKKILL /IM chrome.exe /F");
		}
		}
   //     printf("\n\n The said word is %s \n with difference of %g \n",
digit_to_word(max_ind), max_prob / s_prob);

printf("\n\n================================================================================
=========\n\n");

}

static void read_codebook()
{
	//long double codebook[33][13];
	for (int i=1;i<=32;i++)
	{
		for(int j=1;j<=12;j++)
		{
			codebook[i][j]=0;
		}
	}
	FILE *text;
	//  reading codebook
    text = fopen("codebook_pro.txt", "r");
    i = 1, j = 1;
	long double in=0.0;
    while (!feof(text))
    {
        fscanf(text, "%s", c);
		in=atof(c);
```

```
        codebook[i][j] = in;
        j++;
        if (j == 13)
        {
            j = 1;
            i++;
        }
    }
    fclose(text);
}


#endif
```