PROJECT REPORT
ON
SPEECH BASED CAPTCHA SOLVER

Name: Hriman Krishna Mahanta	**Roll no.:** 224101024
Name: Kushal Chakraborty	**Roll no.:** 224101032
Name: Vishwas Paikra	**Roll no.:** 224101058

TABLE OF CONTENTS

1. INTRODUCTION

CAPTCHA is an acronym which stands for “Completely Automated Public Turing Test to tell Computers and Humans Apart”. CAPTCHA has become an increasingly popular tool which is used to avoid malicious activities on the Internet. Their main application is to determine if an online user is a bot or a human. We often encounter CAPTCHA while browsing through the Internet. Some of the most popular CAPTCHAs are text based captchas in which the user has to type the text which is given in an image. The text might be distorted to make it difficult for a bot to detect it. In the last few years, there has also been a lot of development in other types of captcha like image recognition in which the user has to select from among multiple boxes which contains certain objects in it.

In our project, we have developed a speech based captcha solver. In this captcha, a mathematical expression is displayed to the user. The user then has to correctly pronounce the result of that expression. The captcha is used in a login form in which the user is only allowed to login after he/she passes the captcha test. For sound recognition, we have used the Hidden Markov Model. We have trained the model on the digits from 0 to 9. The aim of this project is to develop an alternate type of captcha which might solve some of the drawbacks of the aforementioned captchas.

2. OVERVIEW

2.1 Front End

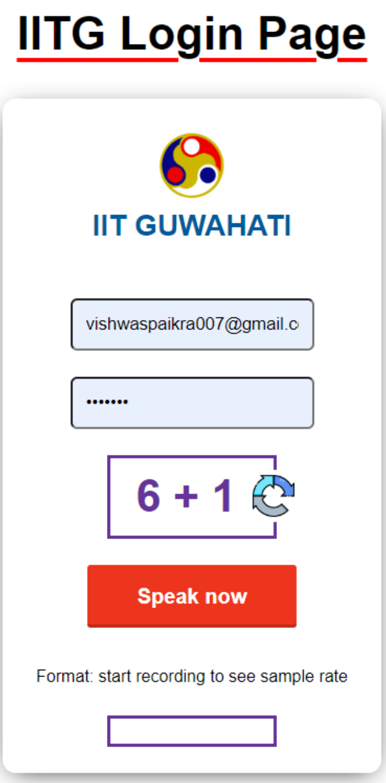
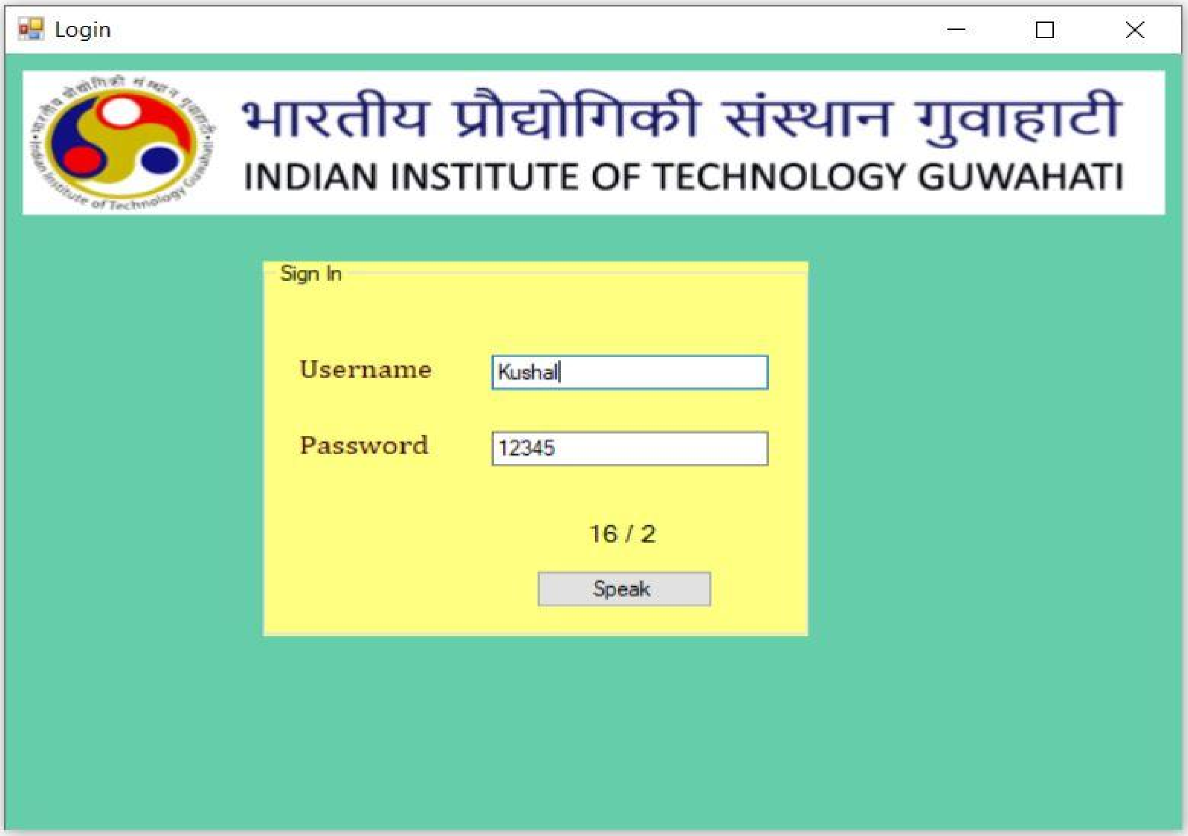


Fig 2.1: Screenshot of the web based captcha solver

The front end of this project is a login form in which the user has to give a username, password and solve a mathematical expression and pronounce the corresponding result. If the pronounced result(which will be a number between 0 and 9) is correct, then the user is logged in. We have used HTML, CSS and Javascript to design the web page which contains the login form. To integrate the Hidden Markov Model into the webpage, we have implemented it in Javascript.


Login
भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
Sign In
Username
Password
Speak

Kushal

12345

16 / 2

Fig 2.2: Screenshot of the windows form based captcha solver

In addition to the web based framework, we have also created a windows form based framework using Microsoft Visual Studio 2010. For this, we have used the Windows Form Application project in visual studio to create the front end of the application.

2.2 Back End

The back end of this project contains the implementation of the Hidden Markov Model for speech recognition. To train the model, we have used 90 utterances of each digit from 0 to 9. We have created a codebook containing the Ci values for each of the frames from utterances of the digits. Then, we have executed the LBG algorithm to create a codebook which contains 32 rows. Using the codebook, we generate the observation sequences of the utterances. This observation sequence will be used as an input to the Hidden Markov Model for training the data. In the training process, we have executed the reestimation function for a total of 100 iterations. After the training phase, we have created a live testing module which predicts the pronounced digit. The output of the predicted digit is used to solve the captcha.

3. IMPLEMENTATION

3.1 Recording the utterance of the digit

We have recorded 90 utterances of each digit from 0 to 9. Thus, there are a total of 900 utterances of the digits. We have used the Cool Edit 2000 software to record the digits. All the utterances are saved as a .txt file. This file contains the amplitude of each sample. We have normalized the amplitudes to within the range -5000 to +5000 and then subtracted the DC shift.

3.2 Creating the codebook vector using LBG algorithm

We have extracted frames from the utterances by taking 320 samples for each frame and then sliding the samples by 80 for the next frame. For each frame, we have calculated the cepstral coefficients (C1 to C12). Then, we have calculated the cepstral coefficients of each of the frames from all the utterances and store them in the universe. Now, we have initialized a codebook with one entry which is the average of all the cepstral coefficients of the universe. Then, we executed the LBG algorithm to iteratively double the codebook entries until the codebook size is 32.

3.3 Finding the observation sequence of each utterance

For each utterance, we find the corresponding observation sequence of each frame using the codebook vector by finding the minimum Tokhura distance from each row in the codebook vector. This observation sequence denotes the cluster in the codebook vector which is closest to the corresponding frame.

3.4 Creating the Hidden Markov Model

We have used the inertia model to initialize the A, B and PI matrix of the hidden markov model. For the matrix A, initially the probability of remaining in the same state is taken as 0.8 while the probability of going to the succeeding state is 0.2 for each state. For the matrix B, the value of each element is 1/32. And for PI, the initial values are 1 for the first state and 0 for all the other states. For each of the utterances, we have taken the observation sequences as input to the model for training the model.

3.5 Training the data

Using the observation sequences and feeding them into the model, we have trained the Hidden Markov Model for each of the digits. We have executed the re estimation procedure for 100 iterations to train the model. At the end of the training procedure we have the models (i.e the final values of A, B and PI matrix) for each of the utterances. We average these models for each digit to create a total of 10 models each representing a digit from 0 to 9. We will use these models to predict the digit in our live recognition module by finding the probability of an observation sequence for each model.

3.6 Creating the live recognition module

For recognising the digits on the webpage, we have used the live recognition module which records a sound for 2 seconds and then stores it in .txt format. This file can then be processed using the models we have created to predict the digit. In the training phase, we have created 10 models each representing digits from 0 to 9. We find the probability of the observation sequences of the file given each of the models. The model which will give the maximum probability will be taken as the predicted digit for that file.

3.7 Creating the web framework for captcha

- Web App is being used for the testing purpose developed using HTML, CSS and Javascript.
- Firstly we generated codebook and lambda a and b models using Windows App and the generated codebook and lambda a and b model is saved inside the web app as static variables.
- Web App contains 3 main modules
    - Recorder.js and App.js contains logic to record and create sample data

from the voice sample of the user

- ○ Recognize.js contains functions to generate observation sequence using cepstral coefficients and codebook and to predict the said captcha's answer digit using lambda a and b and forward procedure.
- ○ User Interface contains mainly HTML and CSS. It shows username and password as inputs to identify the user and a random mathematical captcha (addition, subtraction, multiplication and division) and a button to start recorder for 3 seconds. This button once clicked records audio from the user and calls recognise(sample, sample_size) function after generating sample text data from audio.

3.8 Creating the Windows Forms framework for captcha

Windows Forms is a Graphical User Interface(GUI) class library which is bundled in *.Net Framework*. Its main purpose is to provide an easier interface to develop the applications for desktop, tablet, PCs. It is also termed as the WinForms. The applications which are developed by using Windows Forms or WinForms are known as the Windows Forms Applications that run on the desktop computer. WinForms can be used only to develop the Windows Forms Applications not web applications. WinForms applications can contain different types of controls like labels, list boxes, tooltips etc.

It has three pages:

- Login page where the captcha is also shown and the user speaks the result by pressing the speak button
- New User voice Training Page where user can record all the 10 digits 0-9 in continuous to train model with this new data set
- And finally the user's personal page where the user can read and write text like a notebook.

4. SOFTWARE TOOLS USED

4.1 Cool Edit 2000

Cool Edit is a software program which is used to record sound in different formats such as .wav and .txt. It is a very useful software which can be used in a lot of applications related to speech processing. We can set different sample rate (16kHz) , channels (mono) and resolution (16 bit). It was used to get sample text of spoken digits from 0 to 9 to train our model with those files.

4.2 Microsoft Visual Studio 2010

Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs including websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code. We used it as our primary code editor to develop our windows app and also to write the implementation of the Hidden Markov Model.

5. CONCLUSION AND FUTURE SCOPE

In this project, we have used speech processing to build a captcha solver. We have developed two different types of front end models for our project. One model is web based and the other is a windows application. For the speech recognition module, we have used the principle of Hidden Markov Model to train and recognise the data. The Hidden Markov Model gives relatively good results in the prediction of digits for our project. We have restricted the numbers from 0 to 9 in this project, although in future we can expand it to more numbers. Also, there are still a few shortcomings in the live recognition of digits in the phone version of the web app as sometimes it is not able to correctly predict the spoken digit. However it gives above 90% correct prediction in windows app and in web app (Laptop). We can further improve the prediction accuracy by adding even more data to the model.

6. CODE USED IN THE PROJECT

We have used the following header files in the implementation of Hidden Markov Model. This header files contain the definition of all the variables and functions used in the project.

(i) config.h:

```
#define mx_amp 5000
#define frame_size 320
#define max_no_frames 85
#define PI 3.142857142857
#define p 12
//#define no_of_training_samples 25
//#define total_samples 30
#define max_samples 100
#define no_of_digits 10
#define window_shift 80


#define codebook_size 32
#define kmeans_thresh 0.0001
#define epsilon 0.03
#define N 5
#define M 32
#define MAX_T 160
#define RESET_VAL 1e-30
#define thresh 3
#define zcr_thresh 0.8
#define f_size 320
#define ignore_initial_frames 100


extern int no_of_training_samples;
extern int no_of_testing_samples;
```

(ii) LBG.h:

```
#include "stdafx.h"
#include "config.h"

/*
long double universe_data[100000][p];
long double codebook[codebook_size][p],
newcodebook[codebook_size][p];
int cluster_size[codebook_size];
long double Weights[]={1.0, 3.0, 7.0, 13.0, 19.0, 22.0,
25.0, 33.0, 42.0, 50.0, 56.0, 61.0};
long int assigned_cluster[100000]={0};
long int no_of_rows=0;
int cbook_size=1;*/


extern long double universe_data[500000][p];
extern long double codebook[codebook_size][p],
newcodebook[codebook_size][p];
extern int cluster_size[codebook_size];
extern long double Weights[];
extern long int assigned_cluster[500000];
extern long int no_of_rows;
extern int cbook_size;

void initialise_codebook();
long double calculate_tokhura(long double A[p], long
double B[p]);
void assign_cluster_to_data();
long double calculate_distortion();
void update_codebook();
void kmeans();
void display_codebook();
void LBG_initialization();
void LBG();
void display_universe_data();
void store_final_codebook(char * output_file);
void read_universe_data(char *input_file);
void generate_codebook();
```

(iii) HMM_functions.h

```
#include "stdafx.h"
#include "config.h"

/*
 long double aij[N][N], alpha[MAX_T][N],
beta[MAX_T][N], gamma[MAX_T][N], delta[MAX_T][N],
psi[MAX_T][N];
 long double trained_aij[N][N];
 long double trained_bjk[N][M];
 int Qstar[MAX_T];
long double bjk[N][M], Pstar, xi[MAX_T][N][N],
prob_of_obs_given_lambda;
long double aij_bar[N][N];
long double bjk_bar[N][M], Pi_bar[N];
long double Pi[N]={1.00, 0.00, 0.00, 0.00, 0.00};
long double prob_arr[max_samples];
int obs[MAX_T];
int T;*/


extern long double aij[N][N], alpha[MAX_T][N],
beta[MAX_T][N], gamma[MAX_T][N], delta[MAX_T][N],
psi[MAX_T][N];
extern long double trained_aij[N][N];
extern long double trained_bjk[N][M];
extern int Qstar[MAX_T];
extern long double bjk[N][M], Pstar, xi[MAX_T][N][N],
prob_of_obs_given_lambda;
extern long double aij_bar[N][N];
extern long double bjk_bar[N][M], Pi_bar[N];
extern long double Pi[N];
extern long double prob_arr[max_samples];
extern int obs[MAX_T];
extern int T;

void display_aij();
void read_aij(char * filename);
void display_bjk();
void read_bjk(char *filename);
void read_obs_seq(char *filename);
void set_aij_bjk();
void set_allmatrix();
long double forward_prc();
void backward_prc();
void calculate_gamma();
void viterbi();
void baum_welch_calculate_xi();
void reestimate_parameters();
void update_aij();
void update_bjk();
void HMM_model(char *filename_aij, char *filename_bjk,
char *filename_obs_seq );
void generate_HMM_model();
void store_aij_to_file(char *filename);
void store_bjk_to_file(char *filename);
void read_trained_aij(char *filename);
void read_trained_bjk(char *filename);
void set_matrix_trained_aij();
void set_matrix_trained_bjk();
void avg_out_aij_bjk(int digit);
void write_to_aij_file(char *filename);
void write_to_bjk_file(char *filename);
void copy_file_aij(char * input_file, char
*output_file);
void copy_file_bjk(char * input_file, char
*output_file);
void store_probabilities(char *filename);
void HMM_training();
```

(iv) durbin_functions.h

```
#include "stdafx.h"
#include "config.h"

/*
//long double tokhuraWeights[]={1.0, 3.0, 7.0, 13.0,
19.0, 22.0, 25.0, 33.0, 42.0, 50.0, 56.0, 61.0};
long double my_weights[]={0.3, 0.6, 1.0, 0.6, 0.3};
long int file_no, acc=0, no_of_frames;
long double nfactor, dcshift, sum=0.0, d, dist,
final_dist, mindst;
long int dsize, start, end, i, j, k, m, f, index, v,
fno;
long double data[100000],
frames_data[max_no_frames+10][frame_size],
energy_data[100000];
long double R[max_no_frames+10][p+1], E[p+1],
a[p+1][p+1], K[p+1], C[max_no_frames+10][p+1],
Alpha[max_no_frames+10][p+1],
read_Ci[max_no_frames+10][p+1];
*/

extern long double my_weights[];
extern long int file_no, acc, no_of_frames;
extern long double nfactor, dcshift, sum, d, dist,
final_dist, mindst;
extern long int dsize, start, end, i, j, k, m, f,
index, v, fno;
extern long double data[500000],
frames_data[max_no_frames+10][frame_size],
energy_data[500000];
extern long double R[max_no_frames+10][p+1], E[p+1],
a[p+1][p+1], K[p+1], C[max_no_frames+10][p+1],
Alpha[max_no_frames+10][p+1],
read_Ci[max_no_frames+10][p+1];


void calc_dcshift(char *filename);
void calc_nfactor(char *filename);
void normalize_data(char * inpfile);
void apply_hamming_window();
void apply_raisedsine_window();
void add_Cis_to_universe(char *filename);
void store_Cis_to_file(char *filename);
void calculate_cis();
void apply_durbins_algo();
void calculate_Ris();
void extract_frames(char * filename);
void compute_cepstral_testing();
void build_universe();
int max(int a, int b);
int min(int a, int b);
```

(v) hmm_backend.h

```
#include "stdafx.h"
#pragma once
#include "config.h"
#include "durbin_functions.h"
#include "HMM_functions.h"
#include "LBG.h"

/*
long double cepstral_values[MAX_T][p];
long double raw_data[100000];
long double *avg;
int result;
string captcha_str;*/

extern long double cepstral_values[MAX_T][p];
extern long double raw_data[500000];
extern long double *avg;
extern int result;
extern string captcha_str;
extern int arg1, arg2;


void read_cep_values(char *filename);
void generate_sequence(char *filename);
void read_codebook(char *filename);
void generate_observation_sequence_training();
void generate_observation_sequence_testing();
void testing_with_stored_data();
void helper2(char *filename);
void seggregation_and_detection(long double *avg, char
* input_file, char *output_file);
int test_with_live_data();
void live_training();
int generate_captcha();
void delay(int time);
```