भारतीय प्रौद्योगिकी संस्थान हैदराबाद

Indian Institute of Technology Hyderabad

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR SPEECH PROCESSING

INTERNSHIP PROJECT REPORT

BY

Y V SAI DINESH

2017IEN48

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CENTRAL UNIVERSITY OF KARNATAKA

KADAGANCHI

JULY 2019

ACKNOWLEDGMENT

I respect and thank Dr. GVV SHARMA, for providing me an opportunity to do the project work in IIT Hyderabad and giving me all support and guidance which made me complete the project duly. I am extremely thankful to him for providing such a nice support and guidance, although he had busy schedule managing the corporate affairs.

I owe my deep gratitude to Mr. K PRASANNA KUMAR, who took keen interest on my project work and guided me all along, till the completion of my project work by providing all the necessary information for developing a good system.

I am thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs of Electrical Department, IITH which helped me in successfully completing my project work. Also, I would like to extend my sincere esteems to all staff in laboratory for their timely support.

Y V SAI DINESH

CONTENTS

INTRODUCTION

Speech recognition has its roots in research done at Bell Labs in the early 1950s. Early systems were limited to a single speaker and had limited vocabularies of about a dozen words. Modern speech recognition systems have come a long way since their ancient counterparts. They can recognize speech from multiple speakers and have enormous vocabularies in numerous languages.

The first component of speech recognition is, of course, speech. Speech must be converted from physical sound to an electrical signal with a microphone, and then to digital data with an analog-to-digital converter. Once digitized, several models can be used to transcribe the audio to text.

Most modern speech recognition systems rely on what is known as a Hidden Markov Model (HMM). This approach works on the assumption that a speech signal, when viewed on a short enough timescale (say, ten milliseconds), can be reasonably approximated as a stationary process—that is, a process in which statistical properties do not change over time.

In a typical HMM, the speech signal is divided into 10-millisecond fragments. The power spectrum of each fragment, which is essentially a plot of the signal's power as a function of frequency, is mapped to a vector of real numbers known as cepstral coefficients. The dimension of this vector is usually small—sometimes as low as 10, although more accurate systems may have dimension 32 or more. The final output of the HMM is a sequence of these vectors.

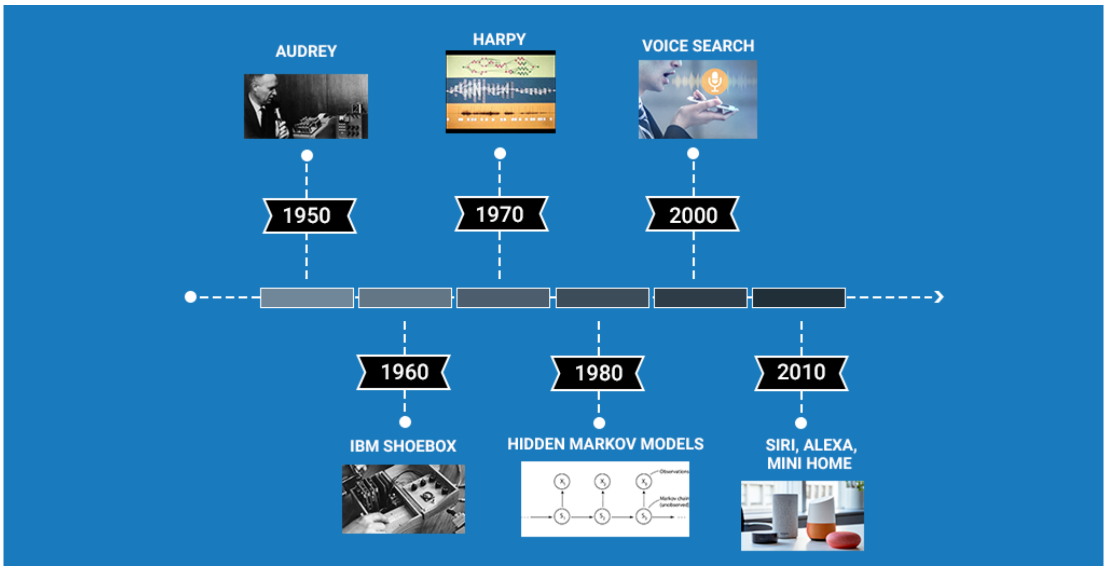
To decode the speech into text, groups of vectors are matched to one or more phonemes- a fundamental unit of speech. This calculation requires training, since the sound of a phoneme varies from speaker to speaker, and even varies from one utterance to another by the same speaker. A special algorithm is then applied to determine the most likely word (or words) that produce the given sequence of phonemes.

One can imagine that this whole process may be computationally expensive. In many modern speech recognition systems, neural networks are used to simplify the speech signal using techniques for feature transformation and dimensionality reduction *before* HMM recognition. Voice activity detectors (VADs) are also used to reduce an audio signal to only the portions that are likely to contain speech. This prevents the recognizer from wasting time analysing unnecessary parts of the signal.

This will sound familiar to anyone who has owned a smartphone in the last decade. I can't remember the last time I took the time to type out the entire query on Google Search. I simply ask the question – and Google lays out the entire weather pattern for me.

It saves me a ton of time and I can quickly glance at my screen and get back to work. A win-win for everyone! But how does Google understand what I'm saying? And how does Google's system convert my query into text on my phone's screen?

This is where the beauty of speech-to-text models comes in. Google uses a mix of deep learning and Natural Language Processing (NLP) techniques to parse through our query, retrieve the answer and present it in the form of both audio and text.

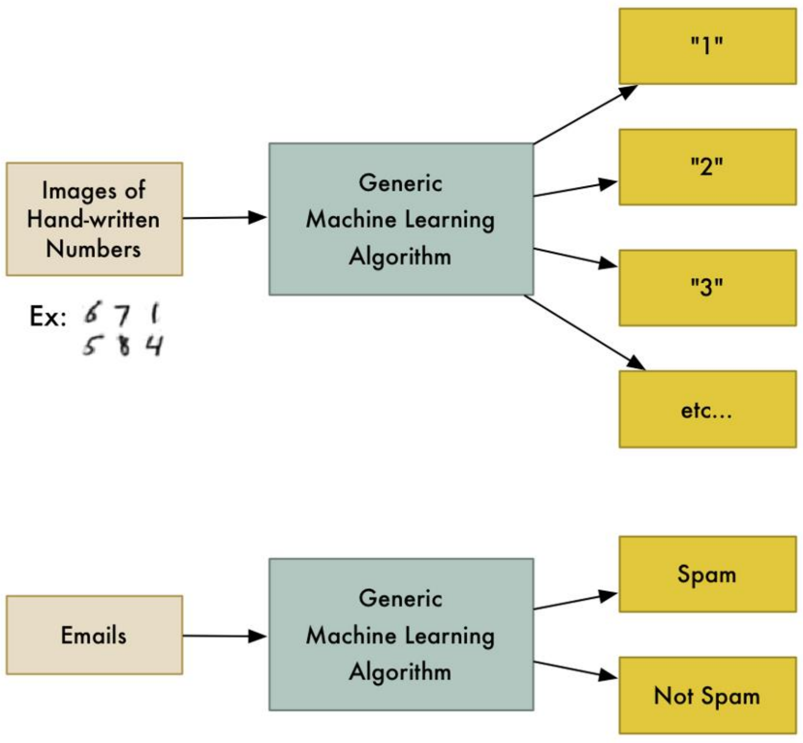


- The first speech recognition system, **Audrey**, was developed back in 1952 by three Bell Labs researchers. Audrey was designed to recognize only digits
- Just after 10 years, IBM introduced its first speech recognition system **IBM Shoebox**, which was capable of recognizing 16 words including digits. It could identify commands like "Five plus three plus eight plus six plus four minus nine, total," and would print out the correct answer, i.e., 17
- The Defence Advanced Research Projects Agency (DARPA) contributed a lot to speech recognition technology during the 1970s. DARPA funded for around 5 years from 1971-76 to a program called **Speech Understanding Research** and finally, **Harpy** was developed which was able to recognize 1011 words. It was quite a big achievement at that time.
- In the 1980s, the **Hidden Markov Model** (HMM) was applied to the speech recognition system. HMM is a statistical model which is used to model the problems that involve sequential information. It has a pretty good track record in many real-world applications including speech recognition.
- In 2001, Google introduced the **Voice Search** application that allowed users to search for queries by speaking to the machine. This was the first voice-enabled application which was very popular among the people. It made the conversation between the people and machines a lot easier.
- By 2011, Apple launched **Siri** that offered a real-time, faster, and easier way to interact with the Apple devices by just using your voice. As of now, **Amazon's Alexa** and **Google's Home** are the most popular voice command based virtual assistants that are being widely used by consumers across the globe.

To be more generalized, these devices use a special set of codes written and are trained by user to machine, which makes the machine more efficient and reduces the user's effort.

Machine learning is the idea that there are generic algorithms that can tell you something interesting about a set of data without you having to write any custom code specific to the problem. Instead of writing code, you feed data to the generic algorithm and it builds its own logic based on the data.

For example, one kind of algorithm is a classification algorithm. It can put data into different groups. The same classification algorithm used to recognize handwritten numbers could also be used to classify emails into spam and not-spam without changing a line of code. It's the same algorithm but it's fed different training data so it comes up with different classification logic.


Images of Hand-written Numbers
Ex: 6 7 1 5 8 4
Generic Machine Learning Algorithm
"1"
"2"
"3"
etc...
Emails
Generic Machine Learning Algorithm
Spam
Not Spam

MATH FOR MACHINE LEARNING

LINEAR ALGEBRA (USING MATRICES AND VECTORS)

1. ### GEOMETRIC CONSTRUCTIONS THROUGH PYTHON LIBRARIES
   This module includes: the construction of right angled triangles with varying triangle congruence properties (SAS,ASA,SSS,AAS..etc). Construction of Circumcircle using different triangles. Drawing Tangents to the circumcircles. Drawing incircles from three given points and line equation.

2. ### PYTHON WITH LINEAR ALGEBRA: 2D
   Drawing line, altitudes of triangle using vectors, medians of triangle using vectors, circumcircle using vectors, incircle, and excircle.

3. ### PYTHON WITH LINEAR ALGEBRA THROUGH COORDINATE GEOMENTRY
   Solving vector equations and obtaining a straight line, verifying orthogonality between the curves, finding locus using vectors, finding normal and tangent vector for any given conic equation, finding center and radius from circle equation (from tangents and normals), finding eccentricity from parabola equation and finding tangents and normal from different points, finding eccentricity from ellipse and hyperbola equation and finding tangents and normal from different points. To represent same coordinate system like other sets, it needs to transform one or more set of coordinates. Conflation is often used for the process of making two geographic data sets fit, or combining the contents of both. For this we used Affine Transformation.

4. ### 3D GEOMENTRY THROUGH LINEAR ALGEBRA
   Solving equations of lines and planes, finding orthogonality of planes, projections on a plane, finding coplanar planes, using least squares in 3D geometry.

MATRIX ANALYSIS

MATRIX ANALYSIS THROUGH PYTHON

Finding exact error and random error using Least squares method, system of equations with no solution, which is solved using Moore-Penrose pseudo inverse in Python. An alternative method for obtaining the pseudo inverse using SVD is also employed. In the process, all basic concepts in matrix analysis like eigenvalues, eigenvectors, orthogonality, Gram-Schmidt orthogonalization, symmetric matrices and SVD are covered, checking orthogonality of matrices and finding the quadratic forms.

DATA ANALYTICS

RANDOM VARIABLES THROUGH PYTHON

This manual provides probability and random variables. This is done by generating random variables in Python and computing metrics like the CDF and PDF for some random variables. In the process, basic concepts like hypothesis testing, transformation of random variables, central limit theorem, etc.. are introduced. Transforming the Domain of Random variables.

LINEAR CLASSIFICATION

This manual covers the guassian distribution for random variables using python. Finding CDF and PDF of guassian random variables. Estimation and detection of random variables using scatterplot. Using Bayes classifier for classification. Finding probability of error. Using linear discriminant analysis for multivariate guassian. Optimum classifier using python.

OPTIMIZATION

CONVEX OPTIMIZATION

This manual provides a simple introduction to convex optimization through graphical and numerical computation using python libraries. The Karuch-Kuhn-Tucker (KKT) conditions are explained through examples. Linear Programming (LP) and Semi Defininte Programming (SDP) problems are also introduced and shown to be convex optimization problems through examples. Further, the freely available Covex optimization python library is used for solving LP and SDP problems. It also explains the Simplex Method for solving Linear Programming problems through examples and also the Northwest corner cell method, Modi Method, and using convex optimization for solving Transportation problems through examples

DIGITAL SIGNAL PROCESSING

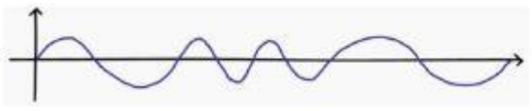
This manual covers the filter design in DSP using python libraries. Using butterworth filter for noise cancellation.

```
import soundfile as sf
from scipy import signal
#read .wav file input signal,
fs = sf.read('Sound Noise.wav' )
#sampling frequency of Input signal
sampl_freq=fs
#order of the filter
order=4
#cutoff frquency 4kHz
cutoff freq=4000.0
#digital frequency
Wn=2*cutoff freq/sampl_freq
# b and a are numerator and denominator polynomials respectively
b, a = signal.butter(order,Wn, 'low')
#filter the input signal with butterworth filter
output signal = signal.filtfilt(b, a, input signal)
#output signal = signal.lfilter(b, a, input_signal)
#write the output signal into .wav file
sf.write('Sound With ReducedNoise.wav', output signal, fs)
```

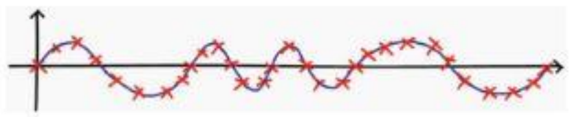
An audio signal is a continuous representation of amplitude as it varies with time. Here, time can even be in picoseconds. That is why an audio signal is an analog signal.

Analog signals are memory hogging since they have an infinite number of samples and processing them is highly computationally demanding. Therefore, **we need a technique to convert analog signals to digital signals so that we can work with them easily.**

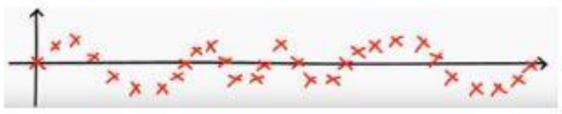
Step 1: Analog audio signal - Continuous representation of signal



Step 2: Sampling - Samples are selected at regular time intervals



Step 3: Digital audio signal - The way it is stored in memory



Sampling the signal is a process of converting an analog signal to a digital signal by selecting a certain number of samples per second from the analog signal. We are converting an audio signal to a discrete signal through sampling so that it can be stored and processed efficiently in memory.

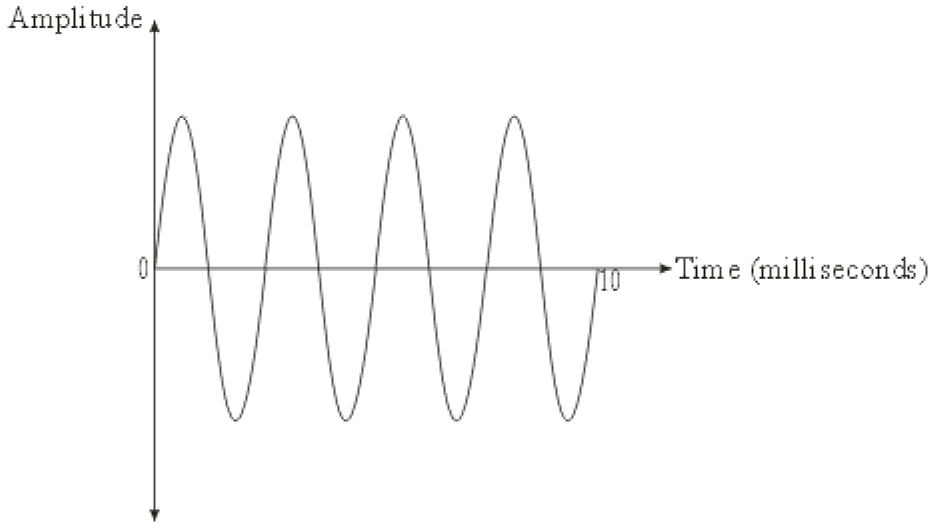
The key thing to take away from the above figure is that we are able to reconstruct an almost similar audio wave even after sampling the analog signal since I have chosen a high sampling rate. The **sampling rate or sampling frequency** is defined as the number of samples selected per second.

Different Feature Extraction Techniques for an Audio Signal

The first step in speech recognition is to extract the features from an audio signal which we will input to our model later. So now, I will walk you through the different ways of extracting features from the audio signal.

Time-domain

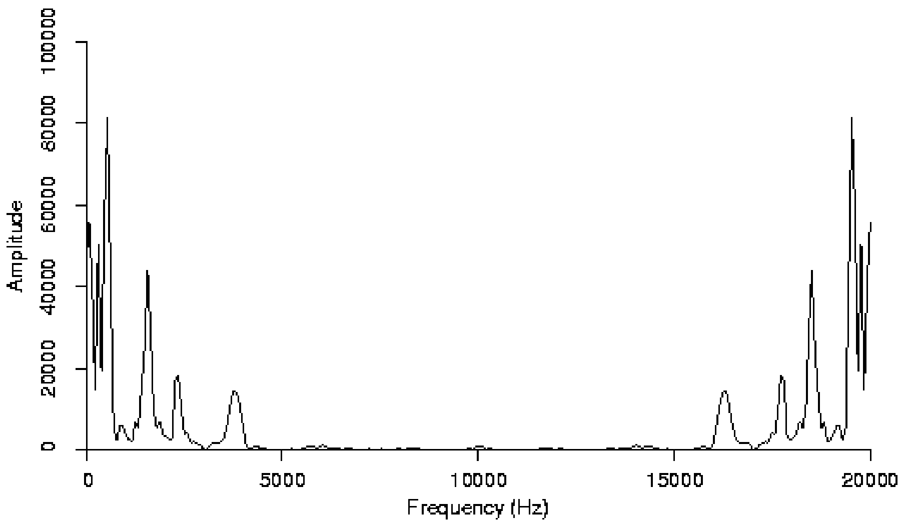
Here, the audio signal is represented by the amplitude as a function of time. In simple words, it is **a plot between amplitude and time**. The features are the amplitudes which are recorded at different time intervals.



The limitation of the time-domain analysis is that it completely ignores the information about the rate of the signal which is addressed by the frequency domain analysis. So let's discuss that in the next section.

Frequency domain

In the frequency domain, the audio signal is represented by amplitude as a function of frequency. Simply put – it is **a plot between frequency and amplitude**. The features are the amplitudes recorded at different frequencies.



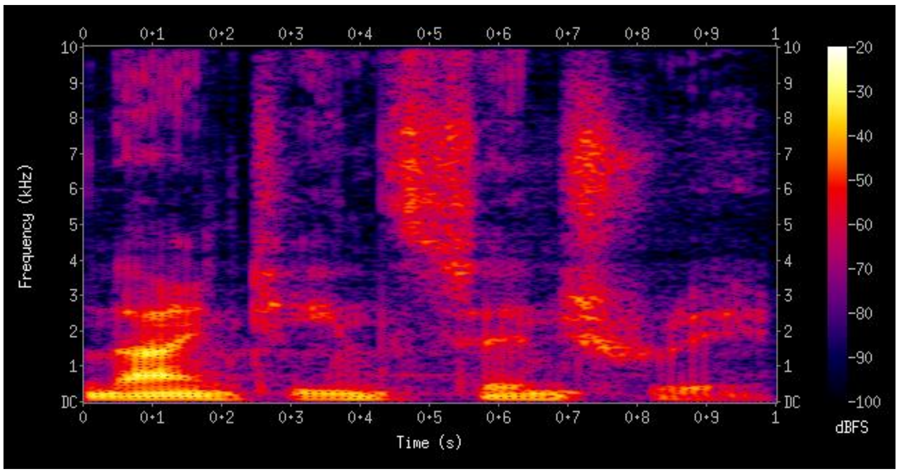
The limitation of this frequency domain analysis is that it completely ignores the order or sequence of the signal which is addressed by time-domain analysis.

Time-domain analysis completely ignores the frequency component whereas frequency domain analysis pays no attention to the time component.

We can get the time-dependent frequencies with the help of a spectrogram.

Spectrogram

It's a 2D plot between time and frequency where each point in the plot represents the amplitude of a particular frequency at a particular time in terms of intensity of color. In simple terms, the spectrogram is a spectrum (broad range of colors) of frequencies as it varies with time.



HARDWARE SKILLS

EMBEDDED C USING ARDUINO

Programming in Arduino AVR-GCC. Interfacing seven segment display using embedded C. Using AVRA, AVR DUDE and AVR GCC. Arduino Pin Control (ATMega368), mapping between Arduino pins and microcontroller ports.

ASSEMBLY PROGRAMMING USING ARDUINO

Writing assembly program in Arduino and interfacing that with sensors and actuators. Controlling the seven segment display using assembly code and designing the decade counter. Boolean operations using IC 7447. Combinational logic using assembly codes.

DIGITAL DESIGN THROUGH ARDUINO

This manual covers the entire breadth of digital design by building a decade counter using an arduino. In the process, boolean logic, combinational logic and sequential logic are covered.

FPGA PROGRAMMING

This manual provides an introduction to Verilog programming using the Icoboard-Lattice FPGA. This is done by implementing a decade counter using verilog. The process is likely to be similar for other FPGA boards as well. Installing yosys and other dependencies for icestorm.

```
cd $HOME git clone git ://git.drogon.net/wiringPi
cd wiringPi && ./build
cd $HOME sudo apt -get install subversion svn co
http://svn.clifford.at/handicraft/2015/icoprog
cd icoprog && make install
sudo apt -get install build - essential clang bison flex libreadline -dev
sudo apt -get install gawk tcl-dev libffi-dev
git mercurial graphviz
sudo apt -get install xdot pkg- config python python3 libftdi-dev
cd $HOME git clone https://github.com/cliffordwolf/icestorm.git
icestorm cd icestorm && make && sudo make install
cd $HOME git clone https://github.com/cseed/arachne -pnr.git arachne-pnr
cd arachne -pnr && make && sudo make install
cd $HOME git clone https://github.com/cliffordwolf/yosys.git
cd yosys && make && sudo make install
```

PROJECTS

VOICE CONTROLLED TOY CAR USING GOOGLE API

HARDWARE REQUIREMENTS

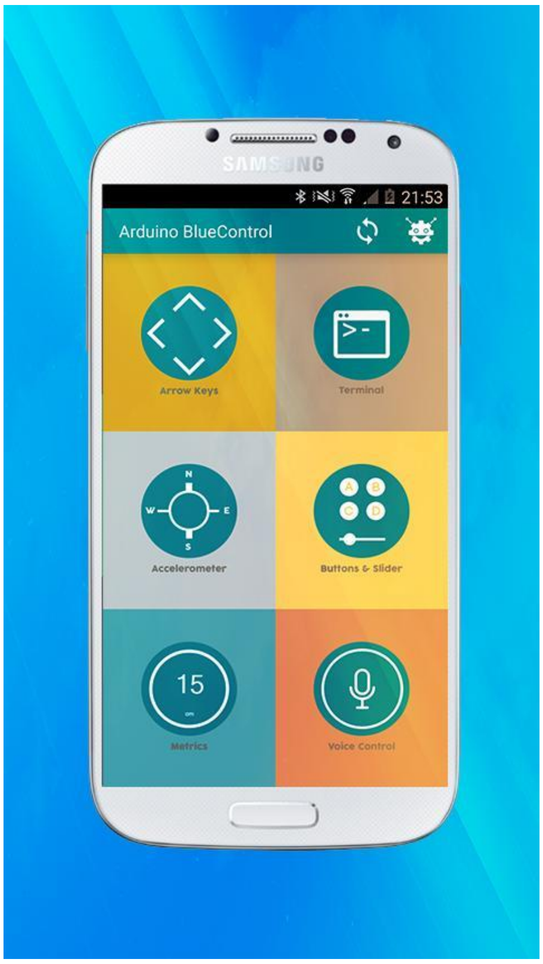
1. L293D (MOTOR DRIVER)
2. POWER SOURCE (BATTERY)
3. BLUETOOTH MODULE (HC-05 AND ABOVE)
4. ARDUINO UNO/NANO
5. CHASIS KIT WITH DC GEARED MOTORS

SOFTWARE REQUIREMENT

1. ARDUINO IDE
2. ARDUINO BLUETOOTH CONTROLLER APP USING GOOGLE API

CODE IN ARDUINO IDE

```
int motor_input1=2;
int motor_input2=3;
int motor_input3=4;
int motor_input4=5;
String voice="";

void setup() {
Serial.begin(9600);
pinMode(motor_input1, OUTPUT); //RIGHT MOTOR
pinMode(motor_input2, OUTPUT); //RIGHT MOTOR
pinMode(motor_input3, OUTPUT); //LEFT MOTOR
pinMode(motor_input4, OUTPUT); //LEFT MOTOR }

void loop() {
while(Serial.available()>0) {
delay(10);
char c=Serial.read();
voice+=c;
}
if (voice.length() > 0) {
//Serial.println(voice);
if(voice=="forward") {
digitalWrite(motor_input1, HIGH);
digitalWrite(motor_input2, LOW);
digitalWrite(motor_input3, HIGH);
digitalWrite(motor_input4, LOW);
Serial.println("forward");
delay(800);
voice = ""; }

else if(voice=="back") {
digitalWrite(motor_input1, LOW);
digitalWrite(motor_input2, HIGH);
digitalWrite(motor_input3, LOW);
digitalWrite(motor_input4, HIGH);
Serial.println("back"); delay(800);
voice = ""; }
else if(voice=="left") {
digitalWrite(motor_input1, HIGH);
digitalWrite(motor_input2, LOW);
digitalWrite(motor_input3, LOW);
digitalWrite(motor_input4, HIGH);
Serial.println("left");
delay(800);
 voice = ""; }

else if(voice=="right") {
digitalWrite(motor_input1, LOW);
digitalWrite(motor_input2, HIGH);
digitalWrite(motor_input3, HIGH);
digitalWrite(motor_input4, LOW);
Serial.println("right");
delay(800);
voice = ""; }

else if(voice=="stop") {
digitalWrite(motor_input1, LOW);
digitalWrite(motor_input2, LOW);
digitalWrite(motor_input3, LOW);
digitalWrite(motor_input4, LOW);
Serial.println("stop");
delay(800);
voice = "";
}
else {
voice = "";
}
}
}
```



TOY CAR USING VOICE RECOGNITION MACHINE LEARNING ALGORITHM

HARDWARE REQUIRED

1. L293D (MOTOR DRIVER)
2. POWER SOURCE (BATTERY)
3. RASPBERRY PI 3B+
4. BLUETOOTH MODULE (HC-05 AND ABOVE)
5. ARDUINO UNO/NANO
6. CHASIS KIT WITH DC GEARED MOTORS

Implementing the Speech-to-Text Model in Python

Import the Libraries

```
import os import librosa #for audio processing
import IPython.display as ipd
import matplotlib.pyplot as plt
import numpy as np from scipy.io
import wavfile #for audio processing
import warnings warnings.filterwarnings("ignore")
```

DATASET

Record ‘forward’ 80 times using you phone and save as ‘forwardi.wav’ for i = 1, . . .,80. The recording duration should be between 1-3 seconds. 1.3 Repeat by recording ‘left’, ‘right’, ‘back’ and ‘stop’. Make sure that the audio files for each command are in separate directories. The datasets are generated through zero padding.

```
import numpy as np import
soundfile as sf
new_data = np.empty([25000,]) #creating an empty array for new file to be
generated from original file
y1 = np.empty([25000,])
for j in range(0,80):
        b= "back"+str(j)+".wav" data, samplerate = sf.read(b) #reading
                                      audio file using soundfile library
        print len(data), samplerate
        x= len(data)
        p = 25000-x
        for y in range(1 ,p):
                for i in range(0,y-1): #adding empty elements in the array
                                      in the start
                        new_data[i] =y1[i]
                for i in range(y,25000-x+y-1):
                        new_data[i] =data[i-y]
                for i in range(25000-y , 24999): #adding empty elements in
                                                  the array in the end
                        new_data[i] = y1[i]
                a = "back"+str(j)+"_"+str(y)+".wav" #total length becomes
                                                  25000
                sf.write(a, new_data, samplerate) #audio files are written
                                                  back to harddisk
                print len(new_data)
```

CODE (USING LINEAR REGRESSION AND GRADIENT DESCENT OPTIMIZATION)

```
import numpy as np
from matplotlib import pyplot as plt
import random
import soundfile as sf
from python_speech_features import mfcc

def sigmoid(x):                                   #defining sigmoid function
    x = np.array(x,dtype=np.float128)
    x = x.reshape(nOut,1)
    x = x
    for  i in range (0,5):
        if x[i] < -700:                               # to prevent overflow
error, we have manually defined it to be 0, when input is very low
            x[i]=0
        else:
            x[i] = 1/(1+np.exp(-x[i]))
    x=x.reshape(-1,nOut)
    return x
nOut = 5


def sigmoidprime(x):                             # derivative of sigmoid
function
    return sigmoid(x)*(1-sigmoid(x))
#    return 1.*(x>0)




y0 = np.empty([6250,4043])                       # reading all the back.wav
files, coverting to mfcc format, adding labels and storing in an array
for j in range(0,6250):
    b = "back"+str(j)+".wav"
    #print b
    data, samplerate = sf.read(b)
    data1 = mfcc(data,samplerate)
    data = data1.reshape(4043,)
    y0[j]=data
y = np.empty([6250,5])
for i in range (0,6250):                         # manually assigning labels
    y[i][0]=1.0
    y[i][1]=0.0
    y[i][2]=0.0
    y[i][3]=0.0
    y[i][4]=0.0
y01 = np.append(y0,y,axis=1)
print("y01 shape {}".format(y01.shape))

y1 = np.empty([6250,4043])

for j in range(0,6250):                          # reading all the forward.wav
files, coverting to mfcc format, adding labels and storing in an array
    b = "forward"+str(j)+".wav"
    #print b
    data, samplerate = sf.read(b)
    data1 = mfcc(data,samplerate)
    data = data1.reshape(4043,)
    y1[j]=data
```

```
y = np.empty([6250,5])
for i in range (0,6250):                      # manually assigning labels
    y[i][0]=0.0
    y[i][1]=1.0
    y[i][2]=0.0
    y[i][3]=0.0
    y[i][4]=0.0
y1l = np.append(y1,y,axis=1)
print("y1l shape {}".format(y1l.shape))

y2 = np.empty([6250,4043])

for j in range(0,6250):                        # reading all the left.wav
files, coverting to mfcc format, adding labels and storing in an array
    b = "left"+str(j)+".wav"
    #print b
    data, samplerate = sf.read(b)
    data1 = mfcc(data,samplerate)
    data = data1.reshape(4043,)
    y2[j]=data
y = np.empty([6250,5])
for i in range (0,6250):                        # manually assigning labels
    y[i][0]=0.0
    y[i][1]=0.0
    y[i][2]=1.0
    y[i][3]=0.0
    y[i][4]=0.0
y2l = np.append(y2,y,axis=1)
print("y2l shape {}".format(y2l.shape))

y3 = np.empty([6250,4043])                      # reading all the right.wav
files, coverting to mfcc format, adding labels and storing in an array
for j in range(0,6250):
    b = "right"+str(j)+".wav"
    #print b
    data, samplerate = sf.read(b)
    data1 = mfcc(data,samplerate)
    data = data1.reshape(4043,)
    y3[j]=data
y = np.empty([6250,5])
for i in range (0,6250):                        # manually assigning labels
    y[i][0]=0.0
    y[i][1]=0.0
    y[i][2]=0.0
    y[i][3]=1.0
    y[i][4]=0.0
y3l = np.append(y3,y,axis=1)
print("y3l shape {}".format(y3l.shape))

y4 = np.empty([6250,4043])                      # reading all the stop.wav
files, coverting to mfcc format, adding labels and storing in an array
for j in range(0,6250):
    b = "stop"+str(j)+".wav"
    #print b
    data, samplerate = sf.read(b)
    data1 = mfcc(data,samplerate)
    data = data1.reshape(4043,)
    y4[j]=data
y = np.empty([6250,5])
for i in range (0,6250):                         # manually assigning
labels
```

```
    y[i][0]=0.0
    y[i][1]=0.0
    y[i][2]=0.0
    y[i][3]=0.0
    y[i][4]=1.0
y4l = np.append(y4,y,axis=1)
print("y4l shape {}".format(y4l.shape))


trains = np.empty([27500,4048])                    # using the first 5500
elements of each word in the train set
k=0
for j in range(0,5500):
    trains[j]=y0l[k]
    k=k+1
k=0
for j in range(5500,11000):
    trains[j]=y1l[k]
    k=k+1
k=0
for j in range(11000,16500):
    trains[j]=y2l[k]
    k=k+1
k=0
for j in range(16500,22000):
    trains[j]=y3l[k]
    k=k+1
k=0
for j in range(22000,27500):
    trains[j]=y4l[k]
    k=k+1
print("trains shape {}".format(trains.shape))
np.random.shuffle(trains)


tests = np.empty([3750,4048])                       # using the last 750
elements of each array in the test set
k = 5500
for j in range(0,750):
    tests[j]=y0l[k]
    k=k+1
k = 5500
for j in range(750,1500):
    tests[j]=y1l[k]
    k=k+1
k = 5500
for j in range(1500,2250):
    tests[j]=y2l[k]
    k=k+1
k = 5500
for j in range(2250,3000):
    tests[j]=y3l[k]
    k=k+1
k = 5500
for j in range(3000,3750):
    tests[j]=y4l[k]
    k=k+1
print("tests shape {}".format(tests.shape))
np.random.shuffle(tests)
```

```
nIn = 4043                                              # nIn = no. of
inputs, nOut = no. of outputs, lr = learning rate, nEpochs = no. of epochs,
losses = a list to store losses in each epoch
nOut = 5
lr = 0.01
nEpochs = 10
losses = []


def nn_forward(X, Y, W1, b):                            # function which is
called to predict output
    x = X.reshape(-1, nIn)
    Y = Y.reshape(-1, nOut)
    #print x.shape
    layer2 = np.dot(x,W1) + b
    out= sigmoid(layer2)
    loss = np.sum(0.5 * (Y - out)**2)
    #losses1.append(loss)
    return out

def train(X, Y, W1, b):                                 # function used to
train the dataset
    x = X.reshape(-1, nIn)
    Y = Y.reshape(-1, nOut)

    layer2 = np.dot(x,W1) + b
    a_out = sigmoid(layer2)


    #out = np.dot(a_layer2, W2)
    #a_out = sigmoid(out)

    loss = np.sum(0.5 * (Y - a_out)**2)                  # cost function
    #losses.append(loss)
    delta_loss = (Y - a_out) * -sigmoidprime(a_out)      # finding delta
loss
    delta_W1 = np.dot(x.T, delta_loss)                   # delta W1 =
X(transpose) dot deltaloss
    delta_b  = delta_loss

    #delta_loss = np.dot(delta_loss, W2.T) * sigmoidprime(layer2)
    #delta_W1 = np.dot(X.T.reshape(nIn, 1), delta_loss)

    #print ("a_out",a_out)
    #print Y

    W1 = W1 - lr * delta_W1                              # updating value of
W1 using gradient descent
    b = b - lr*delta_b                                   # updating value of
b using gradient descent

    #aw1.append(W1)
    #aw2.append(W2)

    return W1, loss, a_out, b


W1 = np.random.rand(nIn, nOut) * 0.5                     # W1 and b randomly
initialised
b  = np.random.rand(1 , nOut)
```

```
trainX = np.empty([27500,4043])                         # spliting of train
set into features and labels
trainY = np.empty([27500,5])
for i in range(0,27500):
    trainX[i]=trains[i][:4043]
for i in range(0,27500):
    trainY[i]=trains[i][4043:]
print("trainX shape {}".format(trainX.shape))
print("trainY shape {}".format(trainY.shape))

testX = np.empty([3750,4043])                           # spliting of test
set into features and labels
testY = np.empty([3750,5])
for i in range(0,3750):
    testX[i]=tests[i][:4043]
for i in range(0,3750):
    testY[i]=tests[i][4043:]
print("testX shape {}".format(testX.shape))
print("testY shape {}".format(testY.shape))




for j in range(nEpochs):                                # traing the
dataset
    for i in range(trainX.shape[0]):
        W1, loss, a_out, b = train(trainX[i], trainY[i], W1, b)
    print("Epoch {} Loss: {}".format(j, loss))
    #print a_out
    losses.append(loss)



correct = 0
total = len(testX)


#print pred
for i in range(testX.shape[0]):                        # making predictions
and calculating accuracy
    pred = np.argmax(nn_forward(testX[i],testY[i], W1, b))
    actual = np.argmax(testY[i])

    print("Prediction: Type {}".format(pred))
    print("Actual: Type {}\n".format(actual))

    if pred == actual:
        correct +=1

print("Accuracy: {}%".format((correct*1.0)/total * 100))

np.savetxt('W1.out',W1,delimiter = ',')                # values of W1 and b
stored in different files to be used in the raspberry pi
np.savetxt('b.out',b,delimiter = ',')
#nE = np.linspace(1,500,500)
nE1 = np.linspace(1,nEpochs,nEpochs)                   # plotting loss with
respect to no. of epochs
plt.plot(nE1,losses)
#plt.plot(nE,losses1)
```

```
#print W1
#print W1.shape
#plt.grid()
#plt.show()

#print aw2
```