Classification of Signals Using Machine Learning Techniques

Pradyumna Mukunda (1PI14EC045)

PES Institute of Technology, Bangalore

Guide: Dr. V.K. Agrawal,

Department of Information Science and Engineering,

Crucible of Research and Innovation (CORI),

PES University, Bangalore-560085.

PES University,

Bangalore-560085

Acknowledgements

On the submission of my project entitled **Classification of Signals Using Machine Learning Techniques**, I would like to express my gratitude to the project supervisor **Dr. V.K. Agrawal**, Director of CORI, PES University for his guidance and inspiration during the course of the project work.

Also, I would like to thank my family and friends for their support and encouragement.

Abstract

The main focus in this project is to demonstrate the use of machine learning techniques in solving a classification problem.

This project aims to show how an artificial learning machine can classify a bunch of time domain signals into two or more groups based on their frequency spectra. The machine learning techniques used are k-means clustering, backpropagation algorithm and extreme learning machine algorithm. The latter two fall under the category of Artificial Neural Networks or ANNs.

Table of Contents

1 Introduction

Machine learning is an emerging technology that can aid in the discovery of rules and patterns in sets of data. It has frequently been observed that the volume of recorded data is growing at an astonishing rate that far outstrips our ability to make sense of it. Machine learning comes as the solution by proposing clever alternatives to analyzing huge volumes of data. It is a step forward from all of statistics, computer science and all other emerging applications in the industry. By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning is able to produce accurate results and analysis.

As of 2017, Machine learning is already being applied in instances like the self-driving Google car, cyber fraud detection, online recommendation engines - like friend recommendations on Facebook, movie recommendations on Netflix and offers recommendations from Amazon.

A recent news item went as follows: 'Apple buys machine learning firm Perceptio Inc., a startup, in an attempt to bring advanced image-classifying artificial intelligence to smartphones by reducing data overhead which is typically required of conventional methods'. Another recent development was that MIT researchers were working on object recognition through flexible machine learning. Yet another tech enthusiast, David Auerbach, claims that, 'Machine learning is starting to reshape how we live and it's time we understood what it was and why it matters.

All of this echoes the vitality of the role machine learning can play in today's data-rich world.

2 Concepts

2.1 Machine Learning

Machine Learning has been defined in many ways:

> Field of study that gives computers the ability to learn without being explicitly programmed **- Arthur Samuels**
>
> A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance of T, as measured by P, improves with experience E. **- Tom Mitchell**

Machine learning algorithms are typically classified into three broad categories, depending on the nature of the learning data available to a learning system. These are:

> **Supervised Learning:** The computer is presented with example inputs and their desired outputs and the goal is to learn a general rule that maps inputs to outputs.
>
> **Unsupervised Learning:** No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (like discovering hidden patterns in data) or a means towards an end (like feature learning).
>
> **Reinforcement learning:** A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The program is provided feedback in terms of rewards and punishments as it navigates its problem space.

Another categorization of machine learning tasks arises when one considers the desired output of a machine-learned system.

> In **classification**, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".

In **regression**, also a supervised problem, the outputs are continuous rather than discrete.

In **clustering**, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.

Density estimation finds the distribution of inputs in some space.

Dimensionality reduction simplifies inputs by mapping them into a lower-dimensional space. **Topic modeling** is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.

There are many different approaches to machine learning, namely:

1. Decision tree learning
2. Association rule learning
3. Artificial neural networks
4. Deep learning
5. Inductive logic programming
6. Support vector machines
7. Clustering
8. Bayesian networks
9. Reinforcement learning
10. Representation learning
11. Similarity and metric learning
12. Sparse dictionary learning
13. Genetic algorithms
14. Rule-based machine learning
15. Learning classifier systems

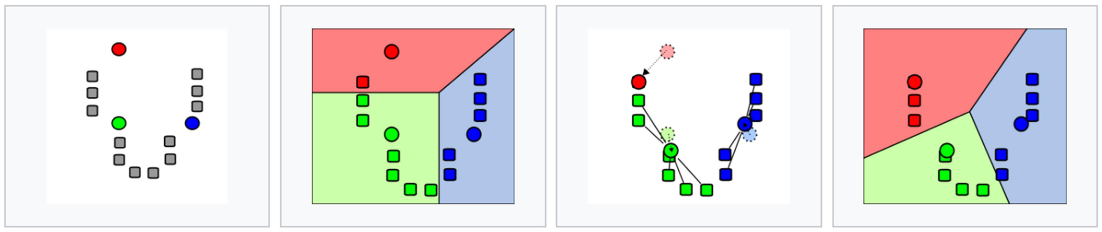
For this project we will discuss two of these, k-means clustering and artificial neural networks.

2.2 k-means clustering:

k-means clustering is an algorithm to classify n d-dimensional vectors into k clusters(groups), where each of the n observation vectors is classified into the cluster with the nearest mean. The algorithm proceeds as follows

1. Initially choose k d-dimensional mean points at random, which represent k clusters. These are the prototypes of the final k clusters.
2. For each of the n observation vectors, find the mean point which is nearest to it (i.e. the mean point which is at least Euclidean distance from the given observation vector).
3. Classify all the observation vectors with the same nearest mean point into a single cluster. In this way we obtain k clusters.
4. Calculate the centroid (i.e mean) of each of the k clusters. The k points so obtained are the new mean points.
5. Repeat steps 2 to 4 until convergence is reached (i.e. there is no more change in mean points). These are the final k clusters.

The k-means clustering algorithm is an example of unsupervised machine learning.



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Fig. 1: Demonstration of k-means clustering algorithm (Source: Wikipedia)

2.3 Artificial Neural Networks (ANNs)

Artificial neural networks (ANNs) or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming.

An ANN is based on a collection of connected units called artificial neurons, (analogous to axons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. Further, they may have a threshold such that only if the aggregate signal is below (or above) that level is the downstream signal sent.

Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

The original goal of the neural network approach was to solve problems in the same way that a human brain would. Over time, attention focused on matching specific mental abilities, leading to deviations from biology such as backpropagation, or passing information in the reverse direction and adjusting the network to reflect that information.

Neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games, medical diagnosis and in many other domains.

2.3.1 Feedforward neural network

A feedforward neural network is an artificial neural network wherein connections between the units do not form a cycle. As such, it is different from recurrent neural networks.

In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

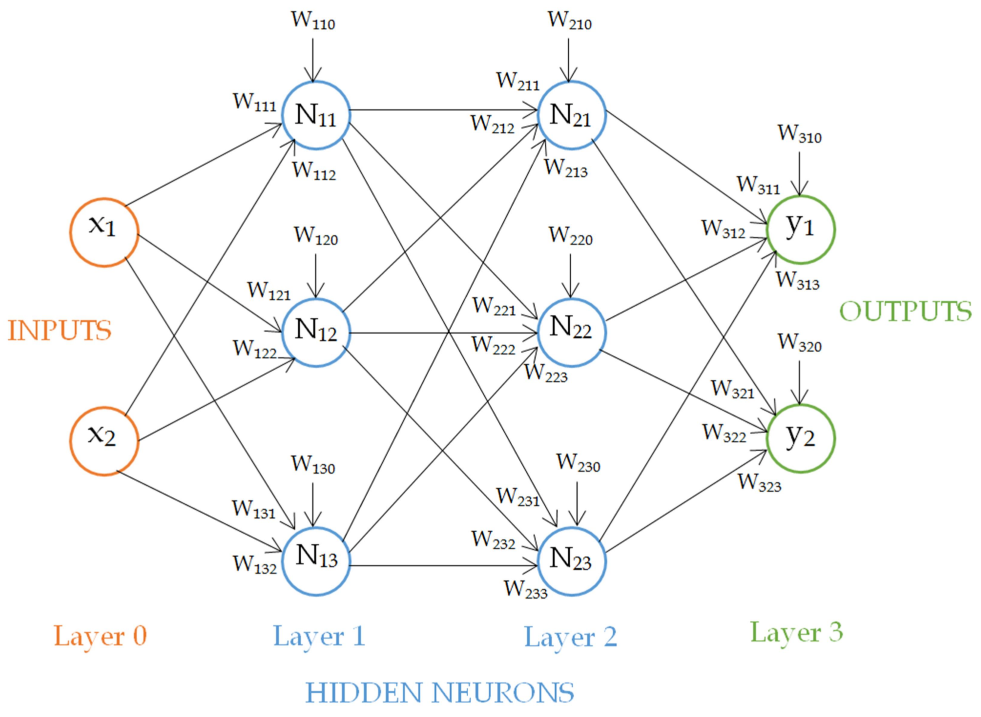

INPUTS
OUTPUTS
HIDDEN NEURONS
Layer 0
Layer 1
Layer 2
Layer 3
x_1
x_2
N_{11}
N_{12}
N_{13}
N_{21}
N_{22}
N_{23}
y_1
y_2
W_{110} W_{111} W_{112} W_{120} W_{121} W_{122} W_{130} W_{131} W_{132}
W_{210} W_{211} W_{212} W_{213} W_{220} W_{221} W_{222} W_{223} W_{230} W_{231} W_{232} W_{233}
W_{310} W_{311} W_{312} W_{313} W_{320} W_{321} W_{322} W_{323}

Fig. 2.1: A 2:3:3:2 feedforward neural network

2.3.2 Structure of a feedforward neural network

The basic unit of a neural network is a neuron, which is a processing unit that takes multiple inputs and returns a single output. One or more neurons which are fed the same set of inputs form a layer. A neural network is formed when the outputs of one layer of neurons are fed as inputs to another layer of neurons.

The internal structure of a general neuron is given below:

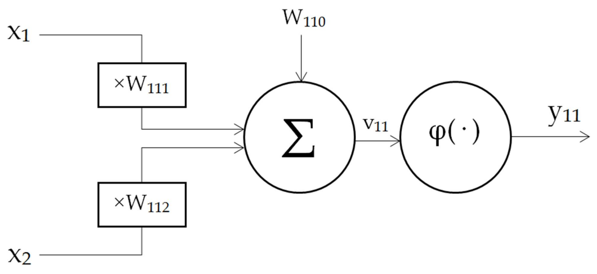


Fig. 2.2: Structure of neuron N_{11}

In the Fig 2.2, the signals x_1 and x_2 serve as the inputs to the neuron. A neuron may have any number of inputs. The inputs x_1 and x_2 are multiplied by the synaptic weights W_{111} and W_{112} respectively and fed to a summation block. A constant term W_{110} is also fed as an input to this block, which is known as a bias. The output v_{11} of the summation block is known as the induced field. This output signal is fed to a function block $\varphi_1(\cdot)$, where the function φ_1 is known as the activation function. The output y_{11} of the function block becomes the output of the neuron. Therefore:

$$v_{11} = W_{110} + W_{111}x_1 + W_{112}x_2$$
$$y_{11} = \varphi_1(v_{11})$$

The neuron demonstrated in Fig. 2.2 is actually the neuron N_{11} in the Fig. 2.1. Therefore it is only part of a larger neural network.

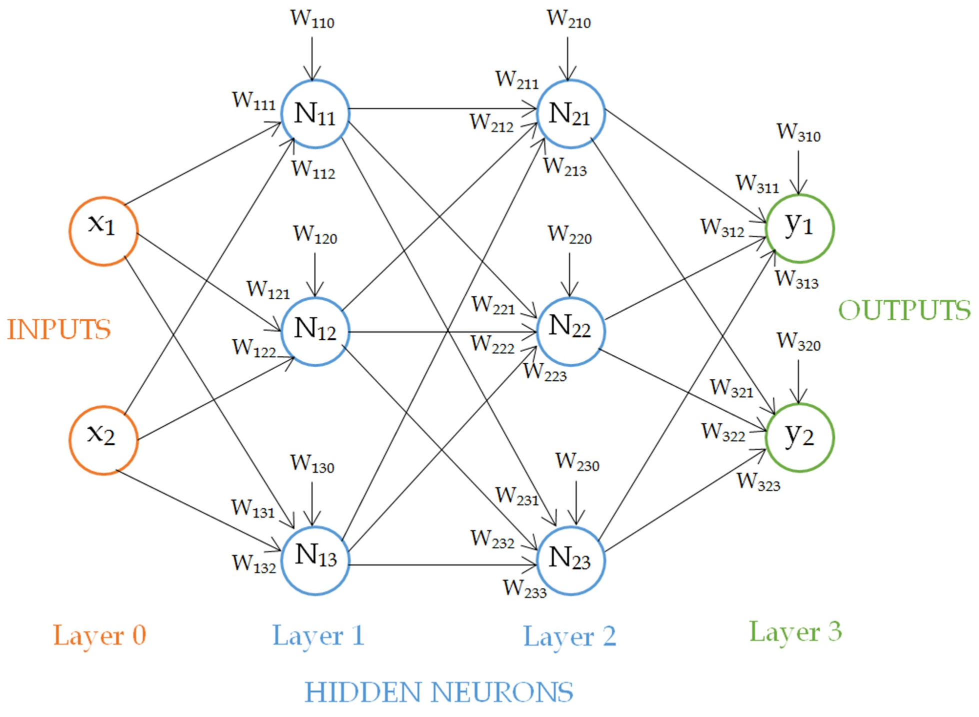


Fig. 2.1: A 2:3:3:2 feedforward neural network

In the Fig. 2.1, the blue circles represent what are known as hidden neurons and the green circles represent output neurons, which are called so because the outputs y_1 and y_2 of these neurons form the outputs of the neural network as a whole. The red circles just represent the input signals x_1 and x_2 to the neural network (they are not neurons). The synaptic weights of each neuron are also explicitly shown.

In the figure, the inputs x_1 and x_2 fed to neuron N_{11} are also fed to the neurons N_{12} and N_{13}, therefore forming a layer of neurons. The neurons N_{12} and N_{13} have the same structure as that of N_{11}, which implies that we can extend the two equations for the neuron N_{11} into a set of six equations.

$$
\begin{aligned}
v_{11} &= W_{110} + W_{111}x_1 + W_{112}x_2 \\
y_{11} &= \varphi_1(v_{11})
\end{aligned}
$$

$$
\begin{aligned}
v_{12} &= W_{120} + W_{121}x_1 + W_{122}x_2 \\
y_{12} &= \varphi_1(v_{12})
\end{aligned}
$$

$$
\begin{aligned}
v_{13} &= W_{130} + W_{131}x_1 + W_{132}x_2 \\
y_{13} &= \varphi_1(v_{13})
\end{aligned}
$$

This can be simplified into matrix equations:

$$
\begin{aligned}
\boldsymbol{v_1} &= \boldsymbol{W_1}\boldsymbol{\bar{y}_0} \\
\boldsymbol{y_1} &= \varphi_1(\boldsymbol{v_1})
\end{aligned}
$$

Where,

$$
\boldsymbol{W_1} = \begin{pmatrix} W_{110} & W_{111} & W_{112} \\ W_{110} & W_{121} & W_{122} \\ W_{110} & W_{131} & W_{132} \end{pmatrix}
$$

$$
\boldsymbol{\bar{y}_0} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \qquad \boldsymbol{v_1} = \begin{pmatrix} v_{11} \\ v_{12} \\ v_{13} \end{pmatrix} \quad \boldsymbol{y_1} = \begin{pmatrix} y_{11} \\ y_{12} \\ y_{13} \end{pmatrix}
$$

The outputs of the 1st layer of neurons are fed as inputs to the next layer. The matrix equations of the 2nd layer are:

$$
\begin{aligned}
\boldsymbol{v_2} &= \boldsymbol{W_2}\boldsymbol{\bar{y}_1} \\
\boldsymbol{y_2} &= \varphi_2(\boldsymbol{v_2})
\end{aligned}
$$

Where,

$$
\boldsymbol{W_1} = \begin{pmatrix} W_{110} & W_{111} & W_{112} & W_{113} \\ W_{110} & W_{121} & W_{122} & W_{123} \\ W_{110} & W_{131} & W_{132} & W_{133} \end{pmatrix}
$$

$$\bar{\boldsymbol{y}}_{\mathbf{1}} = \begin{pmatrix} 1 \\ y_{11} \\ y_{12} \\ y_{13} \end{pmatrix} \qquad \boldsymbol{v}_{\mathbf{2}} = \begin{pmatrix} v_{21} \\ v_{22} \\ v_{23} \end{pmatrix} \quad \boldsymbol{y}_{\mathbf{2}} = \begin{pmatrix} y_{21} \\ y_{22} \\ y_{23} \end{pmatrix}$$

For a general layer l, the equations become:

$$\boldsymbol{v}_{\boldsymbol{l}} = \boldsymbol{W}_{\boldsymbol{l}} \bar{\boldsymbol{y}}_{\boldsymbol{l-1}}$$
$$\boldsymbol{y}_{\boldsymbol{l}} = \varphi_l(\boldsymbol{v}_{\boldsymbol{l}})$$

where $\bar{\boldsymbol{y}}_{\boldsymbol{l}} = \begin{pmatrix} 1 \\ \boldsymbol{y}_{\boldsymbol{l}} \end{pmatrix}$ and $\bar{\boldsymbol{y}}_{\mathbf{0}} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \end{pmatrix}$

The $\bar{\boldsymbol{y}}$ matrix of the final layer, represented as $\bar{\boldsymbol{y}}_{\boldsymbol{L}}$, becomes the output matrix of the network.

Now we have formed a neural network, the next step is the learning.

2.4 Backpropagation algorithm (BPA)

Backpropagation is a method to calculate the gradient of the loss function with respect to the weights in an artificial neural network. It is commonly used as a part of algorithms that optimize the performance of the network by adjusting the weights, for example in the gradient descent algorithm.

The optimization algorithm repeats a two phase cycle, propagation and weight update. When an input vector is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired output, using a loss function, and an error value is calculated for each of the neurons in the output layer. The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output.
The importance of this process is that, as the network is trained, the neurons in the intermediate layers organize themselves in such a way that the different neurons learn to recognize different characteristics of the total input space. After training, when an arbitrary input pattern is presented, neurons in the hidden layer of the network will respond with an active output if the new input contains a pattern that resembles a feature that the individual neurons have learned to recognize during their training.

Backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient – it is therefore usually considered to be a supervised learning method.

In Fig. 2.1, the network returns the output value $\overline{\boldsymbol{y}}_{\boldsymbol{L}} = \binom{y_1}{y_2}$ for the input value $\boldsymbol{x} = \binom{x_1}{x_2}$.

Suppose this is not our desired output and instead we want the network to return the value $\boldsymbol{y_d} = \binom{y_{d1}}{y_{d2}}$ for the same input value.

According to BPA, to achieve this, we define an error function, and then we calculate the gradient of the error function with respect to each of the weights in the network and then we minimize the error function by adjusting the weights according to gradient descent algorithm.

First we define the error function, which in this case is just the difference between the obtained output and desired output.

$$\boldsymbol{e} = \boldsymbol{y_d} - \overline{\boldsymbol{y}}_{\boldsymbol{L}}$$

Then we minimize the error by adjusting the weight matrices of every layer using gradient descent algorithm. The derivation of this is lengthy, so we will just give the formulas. Note all quantities in bold are matrices, L represents final layer and l represents general layer:

$$\boldsymbol{\delta_L} = \boldsymbol{e} \cdot \left(\frac{\partial\varphi(\boldsymbol{x})}{\partial\boldsymbol{x}}\right)_{\boldsymbol{x}=\boldsymbol{v_L}}$$

$$where\ \cdot represents\ Hadamard\ product$$

$$\Delta\boldsymbol{W_L} = \eta_L \boldsymbol{\delta_L} \boldsymbol{y_{L-1}}^T$$

$$where\ T\ represents\ matrix\ transpose\ \&\ \eta\ is\ a\ parameter\ called\ step\ size$$

$$\boldsymbol{\delta_l} = \overline{\boldsymbol{W}}_{\boldsymbol{l+1}}{}^T \boldsymbol{\delta_{l+1}} \cdot \left(\frac{\partial\varphi(\boldsymbol{x})}{\partial\boldsymbol{x}}\right)_{\boldsymbol{x}=\boldsymbol{v_l}}$$

$$where\ \overline{\boldsymbol{W}} is\ the\ matrix\ \boldsymbol{W}\ without\ the\ first\ column$$

$$\Delta\boldsymbol{W_l} = \eta_l \boldsymbol{\delta_l} \boldsymbol{y_{l-1}}^T$$

$$Use\ first\ two\ eqns.for\ final\ layer\ and\ repeat\ last\ 2\ eqns.for\ all\ other\ layers\ till\ l = 1$$

And then finally once we have calculated the $\Delta\boldsymbol{W}$ matrix for all the layers we use the update equation:

$$\boldsymbol{W_l} = \boldsymbol{W_l} + \Delta\boldsymbol{W_l}$$

For l = 1 to L

So to implement one cycle of training, the equations are:

For l = 1 to L

$$\boldsymbol{v_l} = \boldsymbol{W_l}\boldsymbol{\bar{y}_{l-1}}$$
$$\boldsymbol{y_l} = \varphi_l(\boldsymbol{v_l})$$

where $\boldsymbol{\bar{y}_l} = \begin{pmatrix} 1 \\ \boldsymbol{y_l} \end{pmatrix}$ and $\boldsymbol{\bar{y}_0} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \end{pmatrix}$

then,

$$\boldsymbol{e} = \boldsymbol{y_d} - \boldsymbol{\bar{y}_L}$$
$$\boldsymbol{\delta_L} = \boldsymbol{e} \cdot \left(\frac{\partial \varphi_L(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)_{\boldsymbol{x}=\boldsymbol{v_L}}$$
$$\Delta \boldsymbol{W_L} = \eta_L \boldsymbol{\delta_L} \boldsymbol{y_{L-1}}^T$$
$$\boldsymbol{W_L} = \boldsymbol{W_L} + \Delta \boldsymbol{W_L}$$

then for l = L-1 to 1

$$\boldsymbol{\delta_l} = \boldsymbol{\overline{W}_{l+1}}^T \boldsymbol{\delta_{l+1}} \cdot \left(\frac{\partial \varphi_l(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)_{\boldsymbol{x}=\boldsymbol{v_l}}$$
$$\Delta \boldsymbol{W_l} = \eta_l \boldsymbol{\delta_l} \boldsymbol{y_{l-1}}^T$$
$$\boldsymbol{W_l} = \boldsymbol{W_l} + \Delta \boldsymbol{W_l}$$

But all of this is only one cycle of training, also known as an epoch. We have to repeat this *several* times to get our desired result.

In most applications, we have not one but multiple input vectors $\mathbf{x_a}$, $\mathbf{x_b}$, etc. with their corresponding desired output vectors $\mathbf{y_{da}}$, $\mathbf{y_{db}}$, etc., so we just use the same algorithm while alternating between different values of $\mathbf{x}$ and $\mathbf{y_d}$ on every repetition.

2.5 Extreme Learning Machine (ELM)

Extreme learning machines are feedforward neural network for classification or regression with a single layer of hidden nodes, where the weights connecting inputs to hidden nodes are randomly assigned and never updated (i.e. they are a random projection). The weights between hidden nodes and outputs are learned in a single step, which essentially amounts to learning a linear model. The name "extreme learning machine" (ELM) was given to such models by Guang-Bin Huang.
According to their creators, these models are able to produce good generalization performance and learn thousands of times faster than networks trained using backpropagation.

ELM uses the same network structure as BPA, with the restriction that there is only one hidden layer of neurons. Only the learning technique is different.

In BPA, when we have multiple input vectors $\mathbf{x_a}$, $\mathbf{x_b}$, etc. with their corresponding desired output vectors $\mathbf{y_{da}}$, $\mathbf{y_{db}}$, etc., we are required to repeat the algorithm several times alternating between the different values of **x** and $\mathbf{y_d}$. In ELM, we get the result in a single step.

First we have to horizontally concatenate the input vectors $\mathbf{x_a}$, $\mathbf{x_b}$, etc. into a single large matrix **X**. Likewise we horizontally concatenate all the desired output vectors $\mathbf{y_{da}}$, $\mathbf{y_{db}}$, etc. into a single large matrix $\mathbf{Y_d}$. Since there is a restriction that we use only a single layer of hidden nodes, we have only two weight matrices $\mathbf{W_1}$ and $\mathbf{W_2}$. Then the algorithm goes like this:

1. Fill $\mathbf{W_1}$ with random values
2. With $\mathbf{W_1}$ initialized, calculate $\mathbf{Y_2}$ using the formula
$$\boldsymbol{Y_1} = \varphi_1(\boldsymbol{W_1}\boldsymbol{\overline{Y}_0})\ where\ \boldsymbol{\overline{Y}_0} = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \cdots \\ & \boldsymbol{X} & \end{pmatrix}$$
3. Calculate $\mathbf{W_2}$ using the formula
$$\boldsymbol{W_2} = \boldsymbol{Y_d}\boldsymbol{\overline{Y}_1}^T(\boldsymbol{\overline{Y}_1}\boldsymbol{\overline{Y}_1}^T)^{-1}\ where\ \boldsymbol{\overline{Y}_1} = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \cdots \\ & \boldsymbol{Y_1} & \end{pmatrix}$$

And that's it.

For applications where we have all of the data beforehand, ELM is far faster and also far more accurate than BPA. However, in real time applications, where the machine captures data from the environment in real time, the use of ELM is not suitable.

3 Overview of Data

Given a set of signals, the objective is to classify them based on their frequency spectrum using k-means clustering, BPA and ELM
We have a set of 6 signals:
“Eb-Marshall.wav”
“Eb-5150.wav
“Eb-Mesa.wav”
“Ab-Marshall.wav”
“Ab-5150.wav”
“Ab-Mesa.wav”
which are recordings of guitar chords played on high-distortion guitar using different cabinet impulses (Marshall, 5150 and Mesa). The first 3 are of an E-flat chord and the last 3 are of an A-flat chord.

4 Implementation & Results

The project was entirely implemented using MATLAB

4.1 Classification using K-means clustering

MATLAB Code:

Filename: “Soundkm.m”

```
clc
clear all
close all

% Read signals
X{1} = wavread('Eb-Marshall.wav');
X{2} = wavread('Ab-5150.wav');
X{3} = wavread('Ab-Mesa.wav');
X{4} = wavread('Eb-5150.wav');
X{5} = wavread('Eb-Mesa.wav');
X{6} = wavread('Ab-Marshall.wav');

% Take Fourier Transform of the signals
for i = 1:6
    temp = fft(X{i},524288);
    temp = abs(temp);
    temp = temp(1:10000);
    Y(:,i) = temp;
end

% Plot spectra of the signals
f = (1:10000)/524288*44100;
subplot(231)
plot(f,Y(:,1));
```

```
title('Eb-Marshall.wav');
xlabel('Frequency in Hz');
subplot(232)
plot(f,Y(:,4));
title('Eb-5150.wav');
xlabel('Frequency in Hz');
subplot(233)
plot(f,Y(:,5));
title('Eb-Mesa.wav');
xlabel('Frequency in Hz');
subplot(234)
plot(f,Y(:,6),'r');
title('Ab-Marshall.wav');
xlabel('Frequency in Hz');
subplot(235)
plot(f,Y(:,2),'r');
title('Ab-5150.wav');
xlabel('Frequency in Hz');
subplot(236)
plot(f,Y(:,3),'r');
title('Ab-Mesa.wav');
xlabel('Frequency in Hz');

% Classify the signals based on spectra using k-means clustering
C = kmeans(Y',2)
```

Description:

```
% Read signals
X{1} = wavread('Eb-Marshall.wav');
X{2} = wavread('Ab-5150.wav');
X{3} = wavread('Ab-Mesa.wav');
X{4} = wavread('Eb-5150.wav');
X{5} = wavread('Eb-Mesa.wav');
X{6} = wavread('Ab-Marshall.wav');
```

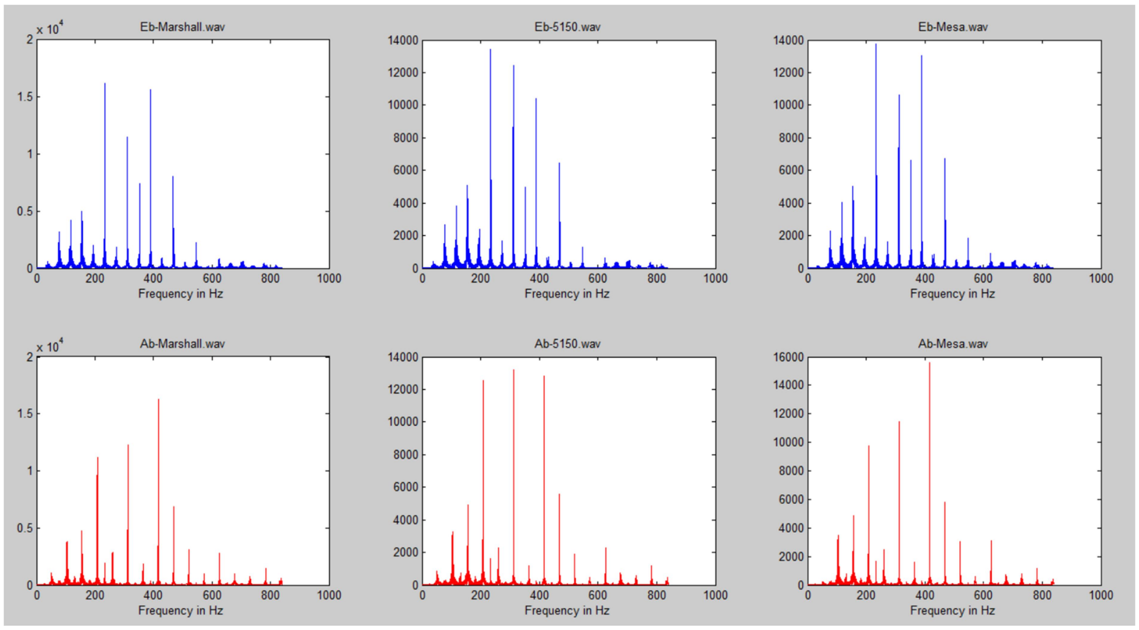
The given signals are uncompressed .wav files sampled at 44100 Hz. They are read and stored in six arrays X{1}, X{2}, ...X{6} using *wavread* function in MATLAB. The signals are stored the order:

X{1} = “Eb-Marshall.wav”
X{2} = “Ab-5150.wav”
X{3} = “Ab-Mesa.wav”
X{4} = “Eb-5150.wav”
X{5} = “Eb-Mesa.wav”
X{6} = “Ab-Marshall.wav”

```
% Plot spectra of the signals
f = (1:10000)/524288*44100;
subplot(231)
plot(f,Y(:,1));
title('Eb-Marshall.wav');
```

```
xlabel('Frequency in Hz');
subplot(232)
plot(f,Y(:,4));
title('Eb-5150.wav');
xlabel('Frequency in Hz');
subplot(233)
plot(f,Y(:,5));
title('Eb-Mesa.wav');
xlabel('Frequency in Hz');
subplot(234)
plot(f,Y(:,6),'r');
title('Ab-Marshall.wav');
xlabel('Frequency in Hz');
subplot(235)
plot(f,Y(:,2),'r');
title('Ab-5150.wav');
xlabel('Frequency in Hz');
subplot(236)
plot(f,Y(:,3),'r');
title('Ab-Mesa.wav');
xlabel('Frequency in Hz');
```

The program then takes the 2^{19}-point FFT of every signal and truncates the spectra obtained to the first 10000 points. The results are plotted below:



```
% Classify the signals based on spectra using k-means clustering
C = kmeans(Y',2)
```

These six spectra are the six 10000-dimensonal vectors applied as input to the kmeans function, along with the desired no. of clusters (i.e. 2). On execution the following output is obtained on the console

```
C =

   2
   1
   1
   2
   2
   1
```

This implies that:

"Eb-Marshall.wav" is classified in cluster 2
"Ab-5150.wav" is classified in cluster 1
"Ab-Mesa.wav" is classified in cluster 1
"Eb-5150.wav" is classified in cluster 2
"Eb-Mesa.wav" is classified in cluster 2
"Ab-Marshall.wav" is classified in cluster 1

Therefore the kmeans clustering algorithm classifies all the recordings of E-flat chord into cluster 2 and all the recordings of A-flat chord into cluster 1.

4.2 Classification using BPA trained ANN

MATLAB Code:

Main Script: "Soundbpa.m"

```
clc
clear all
close all

% Read signals
X{1} = wavread('Eb-Marshall.wav');
X{2} = wavread('Ab-5150.wav');
X{3} = wavread('Ab-Mesa.wav');
X{4} = wavread('Eb-5150.wav');
X{5} = wavread('Eb-Mesa.wav');
X{6} = wavread('Ab-Marshall.wav');

% Take Fourier Transform of the signals
for i = 1:6
    temp = fft(X{i},524288);
    temp = abs(temp);
    temp = temp(1:10000);
    temp = temp/max(temp);
    Y(:,i) = temp;
end

% Plot spectra of the signals
f = (1:10000)/524288*44100;
subplot(231)
plot(f,Y(:,1));
title('Eb-Marshall.wav');
xlabel('Frequency in Hz');
subplot(232)
plot(f,Y(:,4));
title('Eb-5150.wav');
xlabel('Frequency in Hz');
subplot(233)
plot(f,Y(:,5));
title('Eb-Mesa.wav');
xlabel('Frequency in Hz');
subplot(234)
plot(f,Y(:,6),'r');
title('Ab-Marshall.wav');
xlabel('Frequency in Hz');
subplot(235)
plot(f,Y(:,2),'r');
title('Ab-5150.wav');
xlabel('Frequency in Hz');
subplot(236)
plot(f,Y(:,3),'r');
title('Ab-Mesa.wav');
xlabel('Frequency in Hz');

% Classify the signals based on spectra using BPA neural network
W = bpacreate([10000,1,1]);
for i = 1:100
    W = bpatrain2(W,Y(:,1),1,[0.1 0.1],2);
    W = bpatrain2(W,Y(:,3),-1,[0.1 0.1],2);
end
C(1) = bpatest2(W,Y(:,1),2);
C(2) = bpatest2(W,Y(:,2),2);
C(3) = bpatest2(W,Y(:,3),2);
C(4) = bpatest2(W,Y(:,4),2);
C(5) = bpatest2(W,Y(:,5),2);
C(6) = bpatest2(W,Y(:,6),2);
C' > 0
```

Function: “bpacreate.m”

```
% Function to create m0:m1:m2:....:mn BPA network
% and initialize with random values
% Input M = [m0 m1 m2 .... mL]
% Output W = Weight matrices of network in cell array form

function W = bpacreate(M)
    for i = 1:length(M)-1
    W{i} = rand(M(i+1), M(i)+1) - 0.5;
    end
end
```

Function: “bpatest2.m”

```
% Function to find output of given BPA network with given input
% W = Weight matrices of the network
% x = Testing input
% L = No. of layers - 1
% yout = Output
% Activation function is y = tanh(v) for all layers

function yout = bpatest2(W,x,L)
a = 1;
y{1} = [1;x];
for i = 1:L
    v{i} = W{i} * y{i};
    yb{i+1} = tanh(a*v{i});
    y{i+1} = [1;yb{i+1}];
end
yout = yb{L+1};
end
```

Function: “bpatrain2.m”

```
% Function to implement one training iteration of BPA network
% W = Input weight matrices
% x = Training input
% yd = desired output
% n = [n1 n2 n3 .... nL]
% where n1,n2 etc. are the step sizes for corresponding layers
% L = (No. of layers) - 1
% Activation function is y = tanh(v) for all layers

function Wout = bpatrain2(W,x,yd,n,L)
a = 1;
Wout = W;
y{1} = [1;x];
for i = 1:L
    v{i} = W{i} * y{i};
    yb{i+1} = tanh(a*v{i});
    y{i+1} = [1;yb{i+1}];
end
e = yd - yb{L+1};
d{L} = a*e.*(sech(a*v{L}).*sech(a*v{L}));
for i = L:-1:1
    dW{i} = n(i)*d{i}*(y{i}');
    Wout{i} = Wout{i} + dW{i};
    if i > 1
        temp = size(W{i});
        d{i-1} = a*(((W{i}(:,2:temp(2)))')*d{i}).*(sech(a*v{i-
1}).*sech(a*v{i-1}));
    end
end
end
```

Description:

```
% Read signals
X{1} = wavread('Eb-Marshall.wav');
X{2} = wavread('Ab-5150.wav');
X{3} = wavread('Ab-Mesa.wav');
X{4} = wavread('Eb-5150.wav');
X{5} = wavread('Eb-Mesa.wav');
X{6} = wavread('Ab-Marshall.wav');
```

The given signals are uncompressed .wav files sampled at 44100 Hz. They are read and stored in six arrays X{1}, X{2}, ...X{6} using *wavread* function in MATLAB. The signals are stored the order:

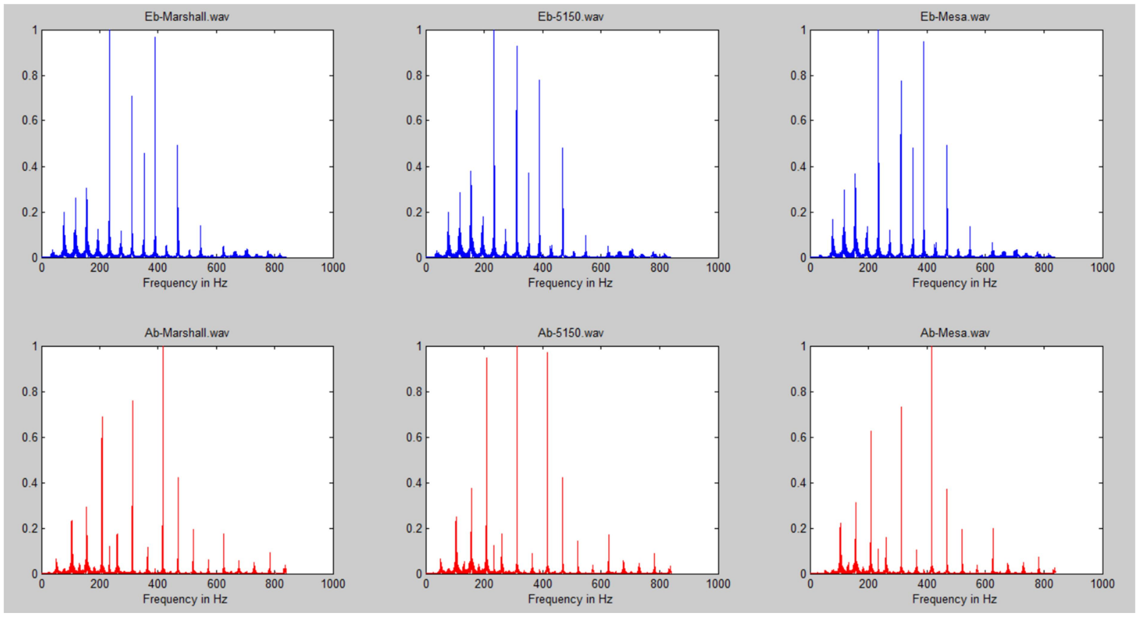
X{1} = "Eb-Marshall.wav"
X{2} = "Ab-5150.wav"
X{3} = "Ab-Mesa.wav"
X{4} = "Eb-5150.wav"
X{5} = "Eb-Mesa.wav"
X{6} = "Ab-Marshall.wav"

```
% Take Fourier Transform of the signals
for i = 1:6
    temp = fft(X{i},524288);
    temp = abs(temp);
    temp = temp(1:10000);
    temp = temp/max(temp);
    Y(:,i) = temp;
end

% Plot spectra of the signals
f = (1:10000)/524288*44100;
subplot(231)
plot(f,Y(:,1));
title('Eb-Marshall.wav');
xlabel('Frequency in Hz');
subplot(232)
plot(f,Y(:,4));
title('Eb-5150.wav');
xlabel('Frequency in Hz');
subplot(233)
plot(f,Y(:,5));
title('Eb-Mesa.wav');
xlabel('Frequency in Hz');
subplot(234)
plot(f,Y(:,6),'r');
title('Ab-Marshall.wav');
xlabel('Frequency in Hz');
subplot(235)
plot(f,Y(:,2),'r');
title('Ab-5150.wav');
xlabel('Frequency in Hz');
subplot(236)
```

```
plot(f,Y(:,3),'r');
title('Ab-Mesa.wav');
xlabel('Frequency in Hz');
```

The program then takes the 2^{19}-point FFT of every signal and truncates the spectra obtained to the first 10000 points. The signal is scaled down in amplitude to reduce the maximum amplitude to 1. The results are plotted below:



```
% Classify the signals based on spectra using BPA neural network
W = bpacreate([10000,1,1]);
```

Using the "bpacreate" function defined in the file "bpacreate.m", we create a 10000:1:1 neural network, which is basically just a single neuron that takes 10000 inputs and returns 1 output.

```
for i = 1:100
    W = bpatrain2(W,Y(:,1),1,[0.1 0.1],2);
    W = bpatrain2(W,Y(:,3),-1,[0.1 0.1],2);
end
```

We now train our neuron. The spectra Y(:,1) [of "Eb-Marshall.wav"] and Y(:,3) [of "Ab-Mesa.wav"] are used as training inputs. The neuron is trained to give an output of +1 when Y(:,1) is given as input and give an output of -1 when Y(:,3) is given as input. For this we use the "bpatrain2" function defined in the file "bpatrain2.m". The function implements only one iteration of training so we have to apply it several times repeatedly to get results. So here we have applied it 100 times using a for loop.

```
C(1) = bpatest2(W,Y(:,1),2);
C(2) = bpatest2(W,Y(:,2),2);
C(3) = bpatest2(W,Y(:,3),2);
C(4) = bpatest2(W,Y(:,4),2);
C(5) = bpatest2(W,Y(:,5),2);
C(6) = bpatest2(W,Y(:,6),2);
C = C' > 0
```

The trained neuron is now tested, with all the six spectra are applied as inputs. For testing the neuron we use the "bpatest2" function defined in "bpatest2.m". When a spectrum is applied as input if the neuron returns a positive value it is segregated into class 1, if it returns a negative value it is segregated into class 0.

On execution of the program the following output is obtained on the console.

```
C =

   1
   0
   0
   1
   1
   0
```

This implies that:

"Eb-Marshall.wav" belongs to class 1
"Ab-5150.wav" belongs to class 0
"Ab-Mesa.wav" belongs to class 0
"Eb-5150.wav" belongs to class 1
"Eb-Mesa.wav" belongs to class 1
"Ab-Marshall.wav" belongs to class 0

which is similar to the result we got for k-means clustering.

An interesting observation here is that even though only 2 out of the 6 signals were used in the training ("Eb-Marshall.wav" & "Ab-Mesa.wav"), the trained neuron is able to correctly classify the remaining 4 signals as well.

4.3 Classification using ELM

MATLAB Code:

Main Script: “Soundelm.m”

```
clc
clear all
close all

% Read signals
X{1} = wavread('Eb-Marshall.wav');
X{2} = wavread('Ab-5150.wav');
X{3} = wavread('Ab-Mesa.wav');
X{4} = wavread('Eb-5150.wav');
X{5} = wavread('Eb-Mesa.wav');
X{6} = wavread('Ab-Marshall.wav');

% Take Fourier Transform of the signals
for i = 1:6
    temp = fft(X{i},524288);
    temp = abs(temp);
    temp = temp(1:10000);
    temp = temp/max(temp);
    Y(:,i) = temp;
end

% Plot spectra of the signals
f = (1:10000)/524288*44100;
subplot(231)
plot(f,Y(:,1));
title('Eb-Marshall.wav');
xlabel('Frequency in Hz');
subplot(232)
plot(f,Y(:,4));
title('Eb-5150.wav');
xlabel('Frequency in Hz');
subplot(233)
plot(f,Y(:,5));
title('Eb-Mesa.wav');
xlabel('Frequency in Hz');
subplot(234)
plot(f,Y(:,6),'r');
title('Ab-Marshall.wav');
xlabel('Frequency in Hz');
subplot(235)
plot(f,Y(:,2),'r');
title('Ab-5150.wav');
xlabel('Frequency in Hz');
subplot(236)
plot(f,Y(:,3),'r');
title('Ab-Mesa.wav');
xlabel('Frequency in Hz');

% Classify the signals based on spectra using ELM neural network
W = elmcreate(10000,1,1);
```

```
Z = [1 -1 -1 1 1 -1];
W = elmtrain(W,Y,Z);
C(1) = elmtest(W,Y(:,1));
C(2) = elmtest(W,Y(:,2));
C(3) = elmtest(W,Y(:,2));
C(4) = elmtest(W,Y(:,4));
C(5) = elmtest(W,Y(:,5));
C(6) = elmtest(W,Y(:,6));
C' > 0
```

Function: “elmcreate.m”

```
% Function to create ELM network and initialize with random values
% m0,m1,m2 is number of neurons in Input, Hidden and Output layer
% W = Weight matrices of network in cell array form

function W = elmcreate(m0,m1,m2)
    M = [m0 m1 m2];
    for i = 1:2
        W{i} = -0.5 + rand(M(i+1), M(i)+1);
    end
end
```

Function: “elmtest.m”

```
% Function to obtain output of given ELM network
% W = Weight matrices of network
% X = Matrix of training input vectors
% Yout = Matrix of outputs
% Activation function is y = tanh(v) for hidden layers
% and y = v for output layer

function Yout = elmtest(W,X)
    [m0,N] = size(X);
    Y{1} = [ones(1,N);X];
    Y{2} = tanh(W{1}*Y{1});
    Y{2} = [ones(1,N);Y{2}];
    Y{3} = W{2}*Y{2};
    Yout = Y{3};
end
```

Function: “elmtrain.m”

```
% Function to train given ELM network
% Win = Input weight matrices
% X = Matrix of training input vectors
% Yd = Matrix of desired outputs
% W = Output weight matrices
% Activation function is y = tanh(v) for hidden layers
% and y = v for output layer

function W = elmtrain(Win,X,Yd)
    W = Win;
    W{1} = -0.5 + rand(size(W{1}));
    [m0,N] = size(X);
    Y{1} = [ones(1,N);X];
    Y{2} = tanh((W{1}*Y{1}));
    Y{2} = [ones(1,N);Y{2}];
    Y{3} = W{2}*Y{2};
    W{2} = (Yd*(Y{2}')) * inv(Y{2}*(Y{2}'));
end
```

Description:

The first 48 lines of the ELM code ["Soundelm.m"] where the program reads the signals, takes FFT and plots the spectra are the same as that of BPA ["Soundbpa.m"].

```
% Classify the signals based on spectra using ELM neural network
W = elmcreate(10000,1,1);
```

Using the "elmcreate" function defined in the file "elmcreate.m", we create a 10000:1:1 ELM, which is basically just a single neuron that takes 10000 inputs and returns 1 output.

```
Z = [1 -1 -1 1 1 -1];
```

For ELM, unlike BPA, we don't have to apply the function repeatedly. Here we get the answer in a single step. But the inputs have to be applied in the following manner:

For an ELM that takes 'm' inputs, each training pattern is an m × 1 vector. If we have 'n' training patterns, we should create an m × n matrix 'Y' by concatenating the n no. of m × 1 training patterns horizontally. Here m = 10000 and n = 6. The matrix 'Y' has already been created while we were taking FFT of the signals.

If the ELM returns 'p' outputs, the desired output corresponding to each training pattern is a p × 1 vector. Arrange the 'n' desired outputs horizontally in the same order as that of their corresponding 'n' training patterns and concatenate them horizontally into a p × n matrix 'Z'. Here p = 1.

```
W = elmtrain(W,Y,Z);
```

The training inputs matrix 'Y' and the desired outputs matrix 'Z' are applied as inputs to the "elmtrain" function defined in the file "elmtrain.m"

```
C(1) = elmtest(W,Y(:,1));
C(2) = elmtest(W,Y(:,2));
C(3) = elmtest(W,Y(:,2));
C(4) = elmtest(W,Y(:,4));
C(5) = elmtest(W,Y(:,5));
C(6) = elmtest(W,Y(:,6));
C = C' > 0
```

The trained ELM is now tested. For testing the ELM we use the “elmtest” function defined in “elmtest2.m”. If the ELM returns a positive value the input pattern is segregated into class 1, if it returns a negative value it is segregated into class 0.

On execution of the program the following output is obtained on the console.

```
C =

   1
   0
   0
   1
   1
   0
```

This implies that:

“Eb-Marshall.wav” belongs to class 1
“Ab-5150.wav” belongs to class 0
“Ab-Mesa.wav” belongs to class 0
“Eb-5150.wav” belongs to class 1
“Eb-Mesa.wav” belongs to class 1
“Ab-Marshall.wav” belongs to class 0

which is same as the BPA result.

5 Conclusion and Future Work

This project demonstrated the use of machine learning for a simple pattern recogntition and classification problem. The applications of machine learning go far beyond this:

Applications for machine learning include:

1. Adaptive websites
2. Affective computing
3. Bioinformatics
4. Brain-machine interfaces
5. Cheminformatics
6. Classifying DNA sequences
7. Computational anatomy
8. Computer vision, including object recognition
9. Detecting credit card fraud
10. Game playing
11. Information retrieval
12. Internet fraud detection
13. Linguistics
14. Marketing
15. Machine learning control
16. Machine perception
17. Medical diagnosis
18. Economics
19. Natural language processing
20. Natural language understanding
21. Optimization and metaheuristic
22. Online advertising
23. Recommender systems
24. Robot locomotion
25. Search engines
26. Sentiment analysis (or opinion mining)
27. Sequence mining
28. Software engineering
29. Speech and handwriting recognition
30. Financial market analysis
31. Structural health monitoring
32. Syntactic pattern recognition

33. Time Series Forecasting
34. User behavior analytics
35. Translation

A recent report from Mckinsey Global has claimed that machine learning will be the driving factor behind the big wave of innovation in the coming times. The technology is bound to find implementation in a wide variety of industries. Further, it could be predicted that machine learning will evolve over the years but extinction is not a thing that will be associated with it.

References

- https://en.wikipedia.org/wiki/Machine_learning
- https://www.simplilearn.com/what-is-machine-learning-and-why-it-matters-article