ABSTRACT

BRAMHASAMUDRA MALLIKARJUNA, PRATHEEK. Application of EEG in User Verification. (Under the direction of Dr. Wesley Snyder.)

Security is an important part of life. Security systems are used in many scenarios to safe-keep people, materials, information etc. Security systems like ID card, passcode are widely used in day to day life. Even though these systems are very effective, they are prone to certain risks, like loosing the ID card, or someone stealing the passcode etc. For this reason, many security systems deploy combination of these securities including bio-metric identification. This thesis investigates the feasibility of using Brain Waves (EEG signals) as an input to security system. The security system using EEG is composed of four stages, reading EEG data from the sensor, pre-processing the EEG data by filtering, extracting suitable features for classification and authenticating the users using classifiers. The performance of various classifiers for different brain tasks are studied and compared.

MindWave mobile EEG sensor is used to collect the raw EEG data from tests subjects. This requires interfacing the device with the computer through bluetooth. The raw EEG data is then pre-processed to remove DC content and other any unnecessary frequencies. Pre-processed data is then divided into subgroups of one second each and deployed to feature extraction.

EEG signals are characterized by frequencies and hence they are divided into different EEG frequency bands. Also, different brain activities give raise to different energy levels in the EEG frequency bands. For this reason, spectral energy of EEG frequency bands are used as features. This is done by computing the DFT of the pre-precessed EEG signals and calculating the energy of different EEG bands and organizing them as a feature vector. Also, the feature vectors are normalized to negate the effect of EEG sensor sensitivity to different

subjects.

The feature vectors are classified using the Mahalanobis Distance classifier, the Neural Networks classifier and the Support Vector Machines classifier. Firstly, intra-subject classification is analyzed. Here, we try to classify different tasks performed by the same subject. Then, inter-subject classification is analyzed. Here, we try to identify a subject among group of subjects performing same task. Performance of all the classifiers is evaluated for both intra-subject and inter-subject classification using classification accuracies, true positive rate (TPR) and false positive rate (FPR).

It was found that, intra-subject classification was harder compared to inter-subject classification. It was also found that the Neural Networks and Support vector machines performed superior to the Mahalanobis Distance classifier. At best, classification accuracy of 76%, TPR of 93% was achieved for inter-subject classification with four test subjects. Also, it was found that classifier performance was on average three times compared to the baseline performance. On the other hand, the performance of the system reduced with increase in number of test subject.

Application of EEG in User Verification

by

Pratheek Bramhasamudra Mallikarjuna

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Electrical Engineering

Raleigh, North Carolina

2016

APPROVED BY:

Dr. Edgar Lobaton

Dr. Xiaogang Wang

Dr. Wesley Snyder
Chair of Advisory Committee

DEDICATION

I would like to dedicate this work to my parents, Malikarjuna and Jaya; to my sister Divya; and all of my friends who have helped, encouraged and motivated me along the way.

BIOGRAPHY

The author was born in a small village, Bramhasamudra, India. He graduated from R.V. college of engineering with a Bachelor of Engineering Degree in Electronics and Computer Engineering, in June 2011. After graduating, he started working for a signal processing company, Ittiam Systems Pvt Ltd., in Bengaluru as Software Engineer in Video Communications Systems team for three years.

He continued his education at North Carolina State University, pursuing a Master of Science degree in Electrical Engineering from Fall 2014. He came in touch with Dr. Wesley Snyder when he took Computer Vision (Spring 2015) course instructed by him. He worked under Dr. Wesley Snyder as summer researcher and helped write a GUI based cross platform image processing, editing & algorithm evaluation tool. He also helped Computer Vision students of spring 2016 as the Teaching Assistant under Dr. Wesley Snyder. He currently works on EEG Based User Verification System under Dr. Wesley Snyder as part of graduate requirement for Masters with thesis. His areas of interests are Machine Learning, Computer Vision & Signal Processing and he continues to work on gaining knowledge and better understanding of techniques used in these fields.

ACKNOWLEDGEMENTS

I would like to thank my committee for all their help and guidance. First, I thank Dr. Wesley Snyder for all the time and effort he put to advise, guide and teach me. I would also like to thank him for motivating me and pushing me to excel. I would like to thanks Dr. Cliff Wang for motivating me to research on EEG based security system and all the help he provided to kick start the research. I would also like to thank Dr. Edgar Lobaton for making me a better student and researcher.

Secondly, I would like to thank all the people who were generous enough to let me note their EEG readings required for the research.

Lastly, I would like to thank all my friends who helped me in the time of need and motivated me to work hard.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

1.1 What is Security?

Security is the procedure or measure taken to ensure safety, for example, when verifying an individual who enters a secured facility or tries to log-in to a secured computer system. It is natural to consider one or all of the security types as shown in Table 1.1 for identification of an individual.

Some of the security systems might use one or more combinations of security types.

Table 1.1 Security Types

Security Type	Example
Have something	ID card
Know something	User-name/Password
Be someone	Bio-metric identification

1.1.1 ID Card Verification

An ID card or Identity Document is the document provided by the security system to identify a person. The document can be just a plain document or can be embedded with smart chip with information encoded in it. Machines can read the card and verify the user information. Even though this method is convenient, the card can be easily stolen resulting in the card being the weak link.

1.1.2 A User-name/Password Verification

An individual is provided with a User-name and a password. The user-name/password combination can be entered in the system to access approval to use the resources controlled by the system or the system itself. Even though the user doesn't have to carry any card for this method, he/she has to remember the user-name and password combination. Also, it is harder to steal the user-name/password combination.

1.1.3 Bio-metric Authentication

Bio-metric authentication involves user identification using human characteristics. Few example of such characteristics include finger print, retina, face recognition, DNA, Brain

Waves etc.

1.2 Using Brain waves for Security Systems

As we will learn in the later chapters, different thinking patterns result in different brain waves and can be distinguished using pattern recognition techniques. This can be leveraged to design a security system to identify an individual. Since same thinking patterns from different individuals result in different brain waves, cracking such security system will be hard by just knowing the thinking pattern.

1.3 Organization of Thesis

Chapter 1 provides brief introduction on Security and Security systems. It also provides information on why EEG signals will be well suited for a robust security system.

Chapter 2 provides a brief description on the human brain anatomy, Electroencephalography and pattern recognition. It discusses about EEG sensors, EEG frequency bands and MindWave mobile EEG sensor. It discusses about pre-processing the EEG signals and extracting the features. It also provides some background on Mahalanobis distance, Artificial Neural networks and Support vector machines.

Chapter 3 gives detailed description of the methodology of EEG security system. It discusses the mathematical background and implementation of pre-processing EEG signals, extracting features from the filtered signals and classifying using Mahalanobis Distance, Neural Networks and Support Vector Machines.

Chapter 4 discusses about the performance measures used to evaluate the performance of the classifiers discussed in Chapter 3. It briefly describes why classifying EEG signals is hard. It also provides the performances of all the classifiers for intra-subject and inter-subject classification.

Chapter 5 discusses few of the interesting results and the reasons behind them. It also discusses about the effect of number of classes on classifier performance.

CHAPTER

2

BACKGROUND

2.1 Human Brain

The Human Brain is an important part of the human nervous system. Along with spinal chord, the brain, as part of central nervous system, is analogous to Central processing unit (CPU) of a computer. The human brain is mostly composed of neurons which are electrically excitable cells, blood vessels and glial cells. Neurons can transmit information through electrical and chemical signals. The human brain is interconnected with following three major components,

1. Brain Stem

2. Cerebellum

3. Cerebrum

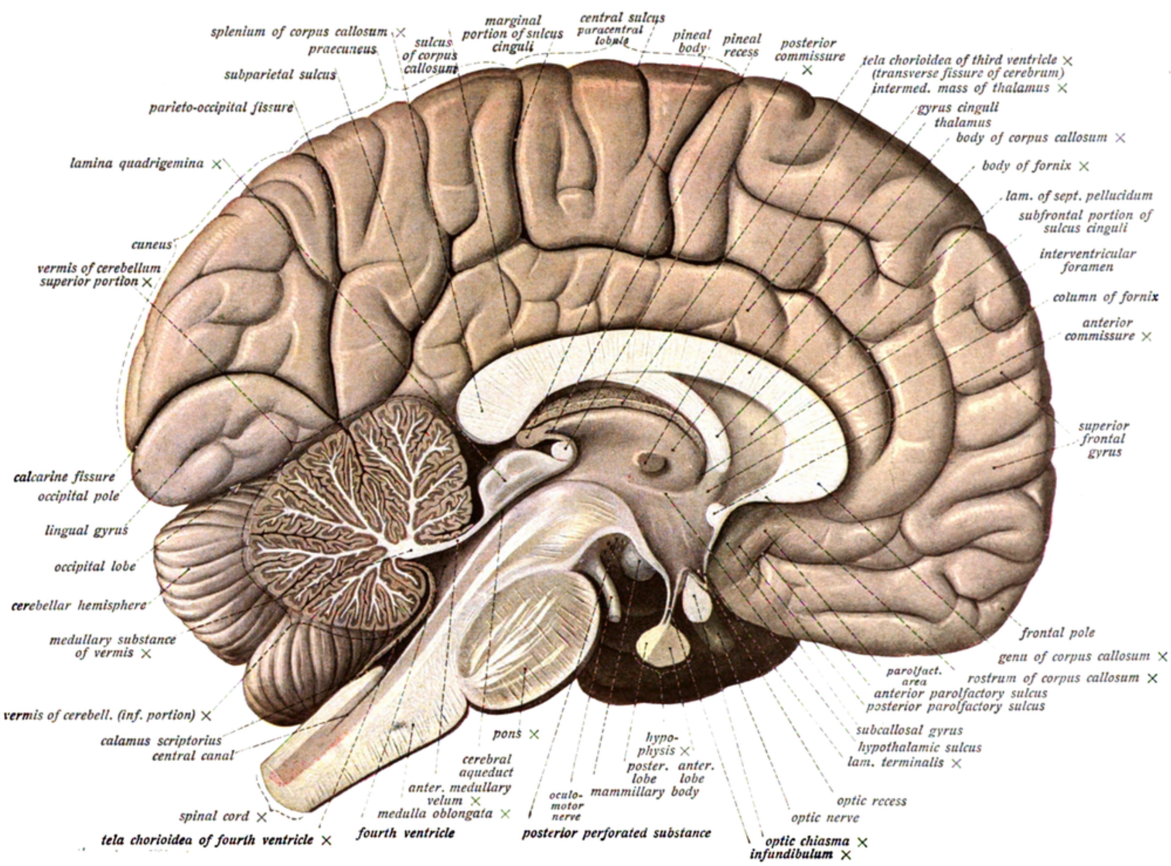

Figure 2.1 The Human Brain (Mid-line incision view) [21]

2.1.1 Brain Stem

The Brain stem connects the brain to the spinal cord and also controls autonomic processes like breathing, digestion and heart rate.

2.1.2 Cerebellum

The Cerebellum plays an important role in balance and motor control, but is also involved in some cognitive functions such as attention, language, emotional functions and in processing and storage of memories.

2.1.3 Cerebrum

The Cerebrum is divided into two hemispheres (left and right) by the longitudinal fissure. It is also covered with a layer of neural tissues known as the Cerebral Cortex which envelops organs like thalamus, hypothalamus and pituitary glands. The Thalamus helps in relying information from the brain stem and the spinal cord to the cerebral cortex. The hypothalamus and the pituitary glands control visceral functions, body temperature and behavioral responses. The Cerebral Cortex plays key role in memory, attention, thought, awareness, language and consciousness.

2.2 Electroencephalography (EEG)

Understanding how the brain works is a necessity in order to find solutions for various brain disorders like epilepsy, dementia, tumor etc. The methods to study the brain can be broadly classified into two methods,

1. An Invasive Approach - Requires physical implant of electrodes inside the brain.

2. A Non-Invasive Approach - Include methods like Magnetic Resonance Imaging(MRI) and Electroencephalography.

According to [12], both the methods give different perspectives and enable us to look inside the brain and observe what happens.

Electroencephalography (EEG) was invented by a German psychiatrist, Hans Berger, who also coined the term "Electroencephalography". An EEG is, as defined by the Mayo Clinic, "A test that detects electrical activity in your brain using small, flat metal discs (electrodes) attached to your scalp" . In a healthy human brain, the brain cells (neurons), are active all the time, even while resting. As the result of these neural activities, electrical impulses are produced. What we call "thought" is in fact ever an changing symphony of such electrical impulses. The rhythmic neural activity in the central nervous system is popularly known as *Neural Oscillation* or *Brain Waves*. For a given neuron these oscillation can occur due to rhythmic changes in the membrane potential. When these oscillations occur synchronously in a large group of neurons, macroscopic oscillations can easily be captured by EEG devices.

2.3 Pattern Recognition

According to Charles W. Therrien [22], " The goal of pattern recognition is to classify objects of interest into one of a number of categories or *classes*. The objects of interest are generally called *patterns*". The data used to discover the patterns is called the *Training set*. The data on which the predicted pattern is tested is called the *Testing set*. Pattern recognition can fall into one of the following two types,

1. **Supervised Pattern recognition:** If the classes of training set are known beforehand.

2. **Unsupervised Pattern recognition or clustering:** If the classes of the training set and maybe even number the of classes are unknown before hand.

A typical pattern recognition system is shown in Figure 2.2.

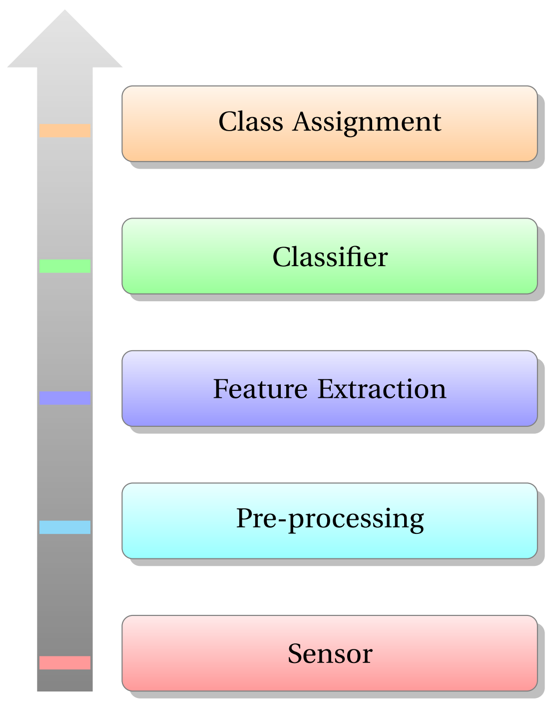

Figure 2.2 A typical pattern recognition system

2.4 Electroencephalography Sensors

In EEG sensors the voltage fluctuation due to the brain waves are read from the sensitive electrodes attached to the scalp. When neurons are electrically charged, electrons are either pushed to these electrodes or pulled from the electrodes and the voltage difference between any of two such electrodes can be measured by a voltmeter. Hence, EEG sensors will typically have a ground electrode, a system reference electrode along with one or more recording electrodes. In 1958, International Federation in Electroencephalography and Clinical Neurophysiology adopted standardization for electrode placement called 10-20 electrode placement system [11] (see Figure 2.3).

There are different types of EEG sensors available, some are sophisticated and used in labs for advance research. Some sensors are available for commercial use. Notable ones are EPOC from Emotive, MUSE and MindWave from NeuroSky. More details about the commercial EEG sensors are discussed in Section 2.7.

2.5 Raw EEG Data

When an electrode in an EEG device captures the electrical activity (which occurs due to the neural activities), it also captures the electrical activity in its proximity. The captured signal also known as "Raw EEG Data", is a result of combination of the neural activities, electrical activity of nearby muscles and ambient noise. Generally, to reduce the effect of ambient noise, the Raw EEG Data is subjected to pre-processing methods which include digital filtering (discussed in Section 2.8). Also, different frequency of the raw EEG data can

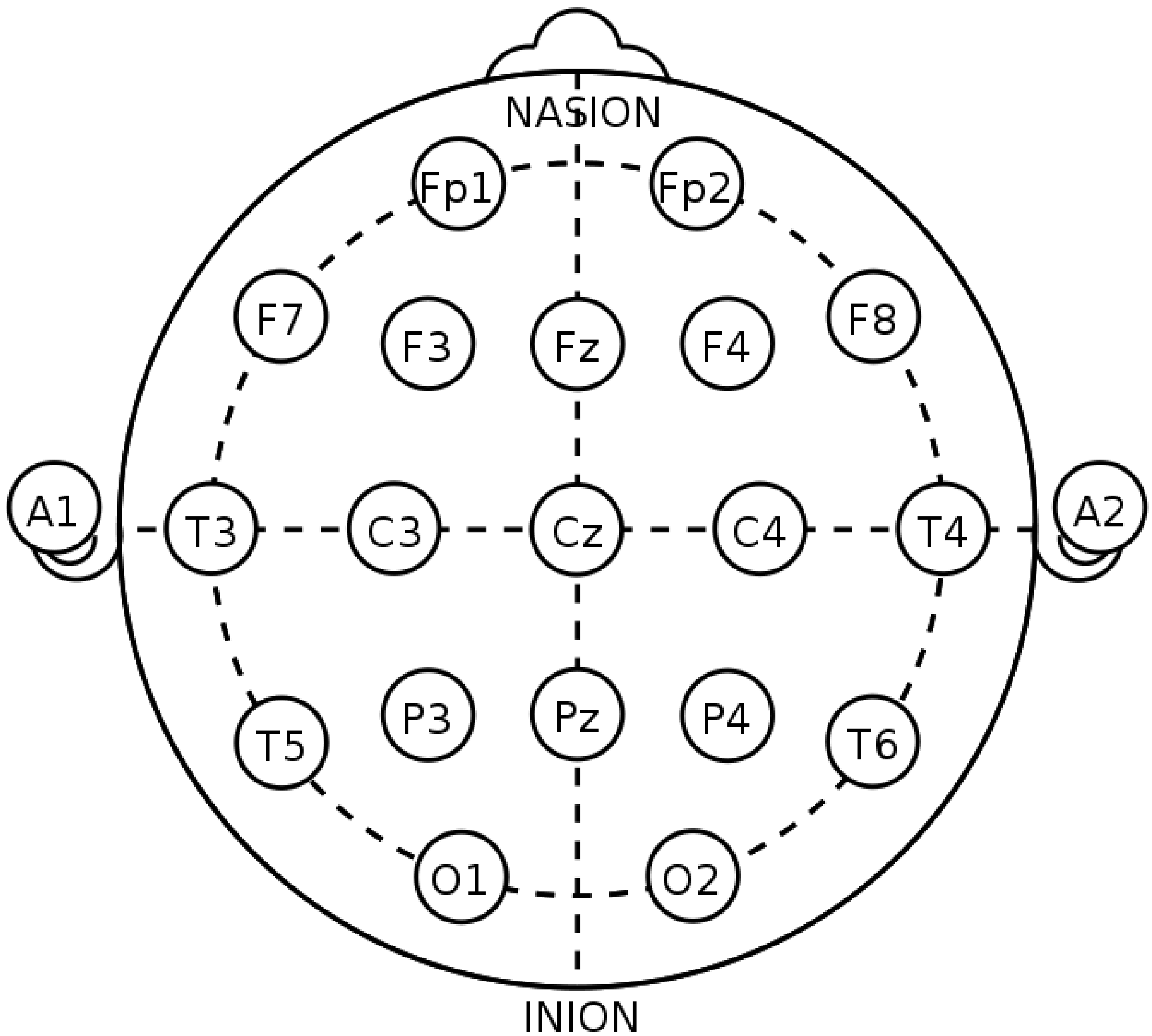

Figure 2.3 The 1020 System - Standardized placement of electrodes on scalp for EEG measurements [1]

be linked to different brain activities. More details about EEG Frequency bands is discussed in Section 2.6.

2.6 EEG Frequency Bands

The neural oscillations detected by the EEG sensors as discussed in Section 2.4 are characterized by frequency, amplitude and phase. These characteristics can be extracted by time-frequency analysis. The important frequency bands associated with the brain waves are shown in the Table 2.1.

Table 2.1 EEG Frequency Bands

Name	Frequency Band
Delta	0.1Hz - 4Hz
Theta	4Hz - 8Hz
Alpha	8Hz - 12Hz
Beta	12Hz - 30Hz
Gamma	30Hz - 48Hz

2.6.1 Delta

Delta waves are low frequency waves (0.1Hz to 4Hz) generated by the brain when the individual is in deep sleep, non-REM sleep or unconscious. The delta waves are generally not detected if the individual is awake, if detected, it is either due to artificial delta waves created due to movements or due to defects in the brain.

2.6.2 Theta

Theta waves range from 4Hz to 8Hz and are linked to Intuitive thinking, creative thinking, recall, fantasy and day dreaming. Theta waves can arise from emotional stress like frustration and disappointment [4]. According to Heinrich et al. [11] high level of Theta waves is considered abnormal among adults and possibly related to AD/HD.

2.6.3 Alpha

Theta waves range from 8Hz to 12Hz and are associated with the state of relaxation while not drowsy, being tranquil and conscious.

2.6.4 Beta

Beta waves range from 12Hz to 30Hz and are associated with performing integrative thinking, agitation, alertness, state of being relaxed yet focused and aware of self and surrounding. According to Y.Zang et al. [4], resisting or suppressing movement, or solving a math task, there is an increase of beta wave levels.

2.6.5 Gamma

Gamma waves range from 30Hz to 48Hz and are associated with state of attention, perception, and cognition.

2.7 Commercial EEG sensors

Various EEG sensors are available in the market and many of them with sophisticated design are used by a doctor to examine a patient or for medical research. Figure 2.4 shows an example EEG sensor used in research [6]. Many EEG sensors are available for commercial use as well. EPOC from Emotive [7], MUSE [18] and MindWave from NeuroSky [14] are some of the notable ones.

Figure 2.4 A Geodesic Sensor Net [6]

2.7.1 Mindwave Mobile

MindWave Mobile (shown in Figure 2.5) is an EEG headset released by NeuroSky for commercial use [14]. It has a recording sensor as part of the sensor arm which can be rested on forehead along with reference and ground sensors on the ear clip. The EEG data recorded from the sensors are transferred via Bluetooth to the Bluetooth enabled device like a Mac, a

PC, an iPhone or an Android phone.

NOTE: Along with the raw EEG data, MindWave Mobile can also transfer the brain wave frequency band readings, attention and meditation meters.

The specifications of MindWave Mobile are as given in the Table 2.2.

Table 2.2 MindWave Mobile Specifications

Parameters	Value
Raw EEG output	3 to 100Hz
Proprietary meters	Attention and Meditation
EEG Power Spectrum	Delta, Alpha, Beta, Gamma
Sampling Frequency	512Hz
Bluetooth Version	v2.1 Class 2
Bluetooth Range	10m
Bluetooth Pairing	Automatic
Headset Type	Static

2.8 Feature Extraction

Digital raw signal acquired from the EEG sensor are subjected to various pre-processing methods in order to extract features. These features are later used as the inputs to the classifiers. These features are generally the frequency spectrum energy bands shown in Table 2.1. Multi-rate Filter banks and Fast Fourier Transform (FFT) can be used to extract the average magnitude of each spectral bands.

Figure 2.5 Mindwave mobile Sensor

2.8.1 Fast Fourier Transforms(FFT)

The Fast Fourier Transforms (FFT) is an optimized and efficient algorithm to computer the Discrete Fourier Transform of a signal. Spectral energy of each EEG frequency band can then be calculated for each respective band of the FFT (additional details are presented in Section 3.2).

2.9 Classifier

The Classifier analyzes the feature vector (obtained by the passing prepossessed *input pattern* or *input vector* or *measurement vector* through feature extractor) and assigns a class to the pattern. The classifier essentially induces a partitioning of the feature vector space into a number of disjoint regions [22]. Figure 2.6 shows one such partition of the feature vector space. Here, if the feature vector falls in the region $R3$, class $c3$ is assigned to the corresponding input pattern.

2.9.1 Mahalanobis Distance

The Mahalanobis Distance is one of the measures of distance between a feature vector and a class, it is given by Eq(2.1).

$$D_x^2 = (X-\mu)^T \Sigma^{-1} (X-\mu) \tag{2.1}$$

where D_x is the Mahalanobis distance, X is the data vector, μ is estimated using $\mu = \frac{1}{n}\sum X$ over all the vectors in the class and Σ is the covariance matrix of X. As we can see,

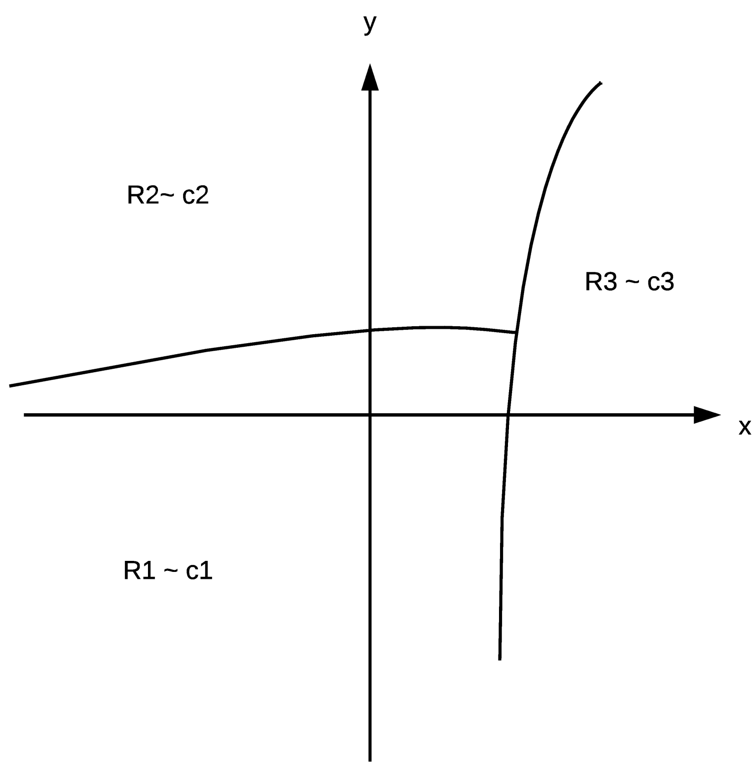

Figure 2.6 Partition of feature space [22]

the Mahalanobis distance is the argument of exponential of the multi-variate Gaussian Distribution that a given data vector (or feature vector) is a member of the set of vectors described by the Gaussian distribution with μ mean and Σ covariance.

For each class C_i of the training set, the mean vector μ_{C_i} and the covariance matrix Σ_{C_i} are calculated. Using μ_{C_i} and Σ_{C_i} of each class, the Mahalanobis Distances (D_{XC_i}) of the input vector X in the testing set is calculated. The minimum Mahalanobis distance D_{min} is then calculated using $D_{min} = min(D_{XC_i})$. The class to which the input vector X belongs to is then determined by the class C_i (with mean vector μ_{C_i} and the covariance matrix Σ_{C_i}) corresponding to the smallest Mahalanobis distance (D_{min}) for the input vector.

2.9.2 Artificial Neural Networks(ANN)

The Artificial Neural Networks are inspired by the behavior of biological neurons and are extensively used in machine learning field. A computational model for Neural Networks called *threshold logic* was created by Warren McCulloch and Walter Pitts in 1943 [13]. In 1958, Frank Rosenblatt created an algorithm called *perceptron*, which could be used for pattern recognition [19]. In 1975, Paul Werbos made one of the biggest advances in neural network research by creating the backpropagation algorithm [23], which solved the exclusive-or issue faced by the perceptron algorithm.

An *Artificial Neuron* is defined as a sum-of-products operator which produces a weighted sum of its inputs and passes it though a non-linear function such as a limiter or a sigmoid [20] as shown in the Figure 2.7. An Artificial Neural Network (ANN) consists of several of such interconnected artificial neurons. A typical artificial neural network consists of an input layer, an output layer and single or many hidden layers as shown in Figure 2.8. Following are the types of Artificial Neural Networks.

1. **Feedforward neural network -** Here the direction of data flow is from input layer to output layer and sigmoid activation is generally used.
2. **Radial basis function network -** Here the hidden layers use Radial Basis Functions (usually Gaussian).
3. **Recurrent neural network -** Here the data flow can be bi-directional.

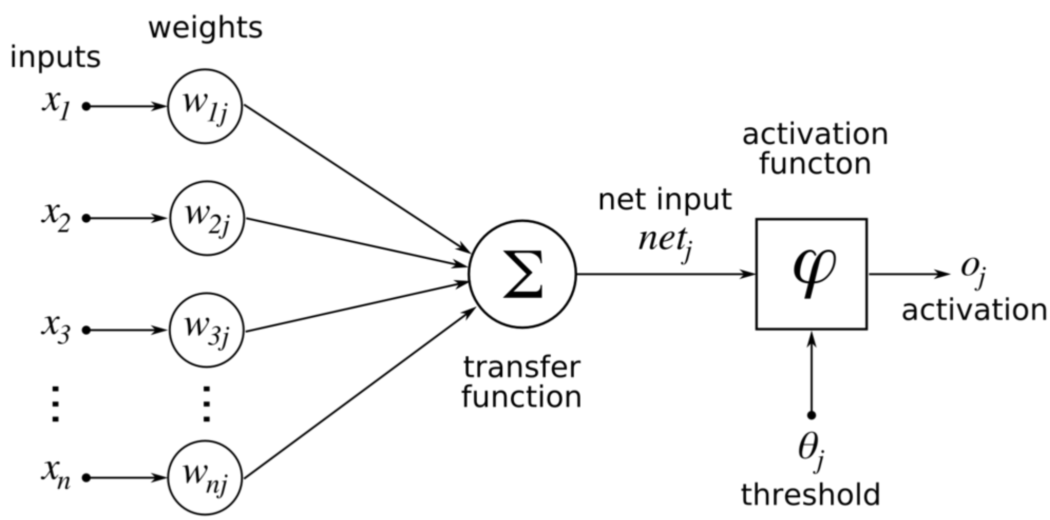

Figure 2.7 An Artificial Neuron [2]

In a typical feedforward neural network, the neurons in input layer are connected to the neurons in the first hidden layer and neurons in the first hidden layer are connected to the neurons in the second hidden layer and so on until the output layer. When the

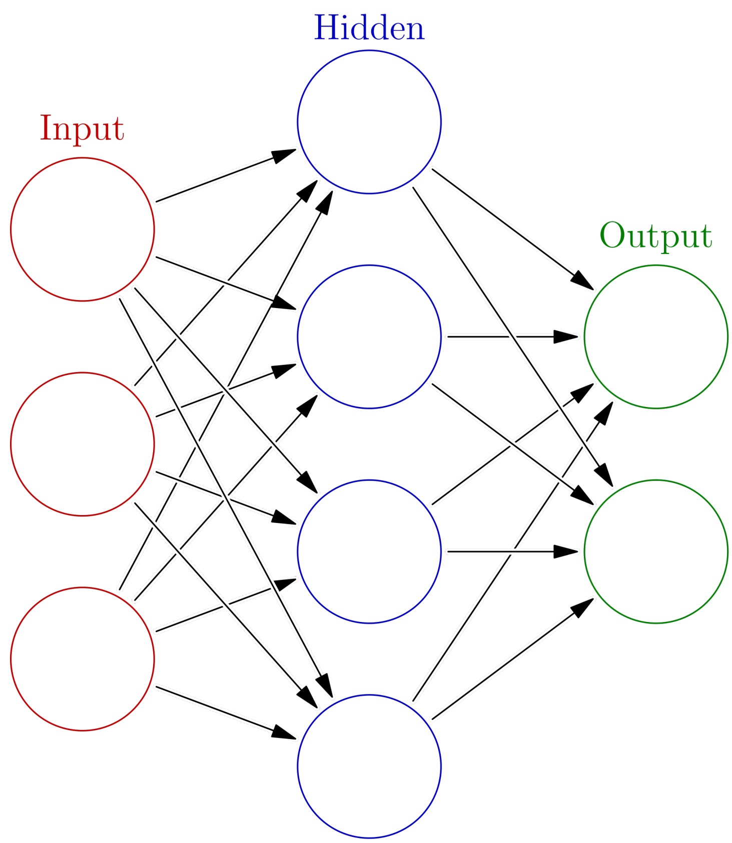

Figure 2.8 A typical Artificial Neural Network with Input, Hidden and Output Layers[8]

neural network is trained, the input activates the neurons of input layer and the activation propagates to the output layer. One of the algorithms used to train neural networks is *Backpropagation* algorithm. It is an iterative algorithm which trains the neural network and adjusts the network parameters by minimizing the output error. More details about the Neural Networks and the backpropagation algorithm is discussed in Section 3.3.2.

2.9.3 Support Vector Machines (SVM)

The Support vector machines is one of the supervised learning models used in machine learning introduced by Vapnik [3]. The SVMs preprocess the m-dimensional input vector to represent patterns in a n-dimension space - typically $n >> m$. With an appropriate non-linear mapping function to a sufficiently higher dimension, data from two categories can be separated by a hyperplane [5]. Say if we have class C_1 and C_2 which are separable by a hyperplane. Let, d_1 be the distance between closest point of C_1 and the hyperplane. Similarly, let d_2 be the distance between closest point of C_2 and the hyperplane. The *margin* is defined as $d_1 + d_2$. Support Vector Machines can be seen as an optimization problem which minimize the margin [20] (See Figure 2.9 and Figure 2.10 for a non-optimal and an optimal choice of dividing hyperplane in 2D).

Since SVMs deal with separating the input data into two by a hyperplane, it is suitable for binary classification, in fact, SVMs can be seen as a non-probabilistic binary linear classifiers. However, multi-class classification can be done by reducing the multi-class classification problem into many binary classification problems. For example, one-vs-rest or one-vs-all is one of the methods used for this. In one-vs-all classification method, a series of binary classifiers are built which distinguish between one class and the rest. The

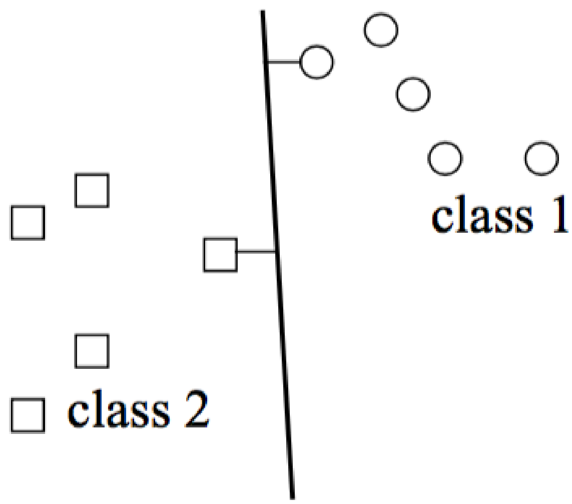

Figure 2.9 Non-optimal dividing hyperplane [20]

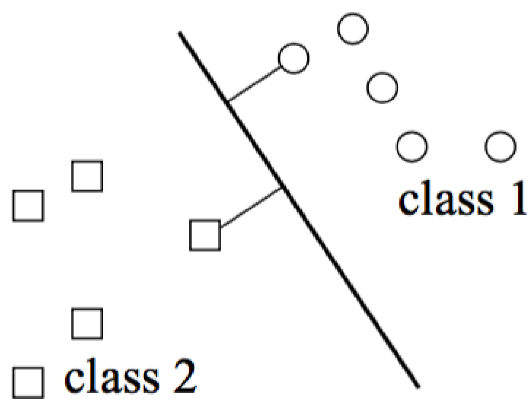

Figure 2.10 Optimal dividing hyperplane [20]

classes are then assigned using winner-take-it-all strategy. Section 3.3.3 discusses about the Support Vector Machines in greater detail.

2.10 Conclusion

In Chapter 2, we discussed about security, security typed, usage of EEG in security and methods used to achieve the same using pattern recognition methods. In Chapter 3, we will discuss in more detail about the design and methodology of pattern recognition pipeline of a security system using EEG.

CHAPTER

3

METHODOLOGY

EEG data for three different mental tasks were collected from four different test subjects. The three mental tasks are as shown in Table 3.1. Each mental task was carried out for ten seconds and repeated five times comprising fifty seconds duration of EEG signal for each task. During each task the subjects were asked to sit on a chair, close their eyes and restrict any muscle movements. The raw EEG data collected from the EEG sensor was then passed through pre-processing block, feature extraction block and classifier block consecutively. Figure 3.1 shows the overall flow of data from the EEG sensors to classifier.

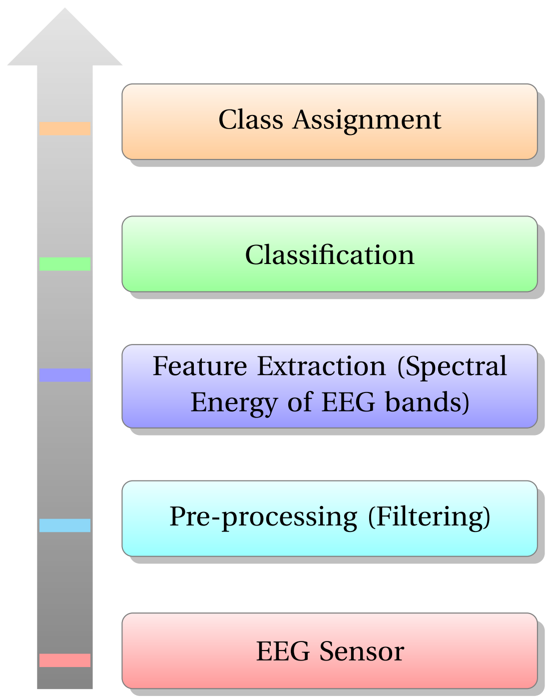

Figure 3.1 Overview of User Verification System using EEG

Table 3.1 Mental Tasks

Task	Description
Calculating	Performing a mental calculation of two digit multiplication
Breathing	Concentrating on breathing
Singing	Mentally singing a song without actually singing out loud

3.1 Pre-Processing

The data stream was read at 512 samples per second from MindWave Mobile EEG sensor and stored in a file. Different files were used for each user, each mental task and each repetition of the mental task. The stored raw EEG data was then passed through a band-pass filter to eliminate unnecessary frequency bands. If total number of samples for each repetition of a mental task for a given user was n, then the filtered data $X' = [x'_1, x'_2, \ldots, x'_n]^T$ was obtained by passing $X = [x_1, x_2, \ldots, x_n]^T$ through the band pass filter F as shown in Equation 3.1. The lower cutoff frequency and the higher cutoff frequency for the band pass filter were 0.1Hz and 48Hz respectively.

$$X' = F(X)\,. \tag{3.1}$$

The filtered data were then divided into subgroups, each subgroup with one second data. Say, if ten seconds of EEG readings were recorded, the total samples of raw EEG data stored in the file would be $512 \times 10 = 5120$. Each sub group would contain $512 \times 1 = 512$ samples. And total number of sub groups would be $5120 \div 512 = 10$. Say, if EEG readings for user i were collected for mental task j, repeated for k^{th} time, then the filtered EEG data for sub group l is given by Equation 3.2.

$$X'_{ijkl} = [x'_1, x'_2, \ldots, x'_{511}, x'_{512}]^T \; . \tag{3.2}$$

3.2 Feature Extraction

As discussed in Section 2.6 neural activities can be characterized by frequencies. Table 2.1 shows different EEG frequency bands and their frequency ranges. Since different brain activities result in different energy levels of EEG frequency bands, using spectral energy of EEG frequency bands as input feature vectors to the classifier is an excellent choice. In order to increase the dimension of input vectors, some of the EEG frequency bands were further sub-divided into low and high bands, resulting in eight EEG frequency bands as show in Table 3.2 (also see Figure 3.2).

Table 3.2 Frequency Bands used for Feature extraction

Name	Frequency Band
Delta	0.1Hz - 4Hz
Theta	4Hz - 8Hz
Low Alpha	8Hz - 10Hz
High Alpha	10Hz - 12Hz
Low Beta	12Hz - 18Hz
High Beta	18Hz - 30Hz
Low Gamma	30Hz - 40Hz
High Gamma	40Hz - 48Hz

First, we pass the each subgroup of filtered EEG data(containing 512 samples) through the 512 point Discrete Fourier Transform (DFT) block and obtain their DFT.

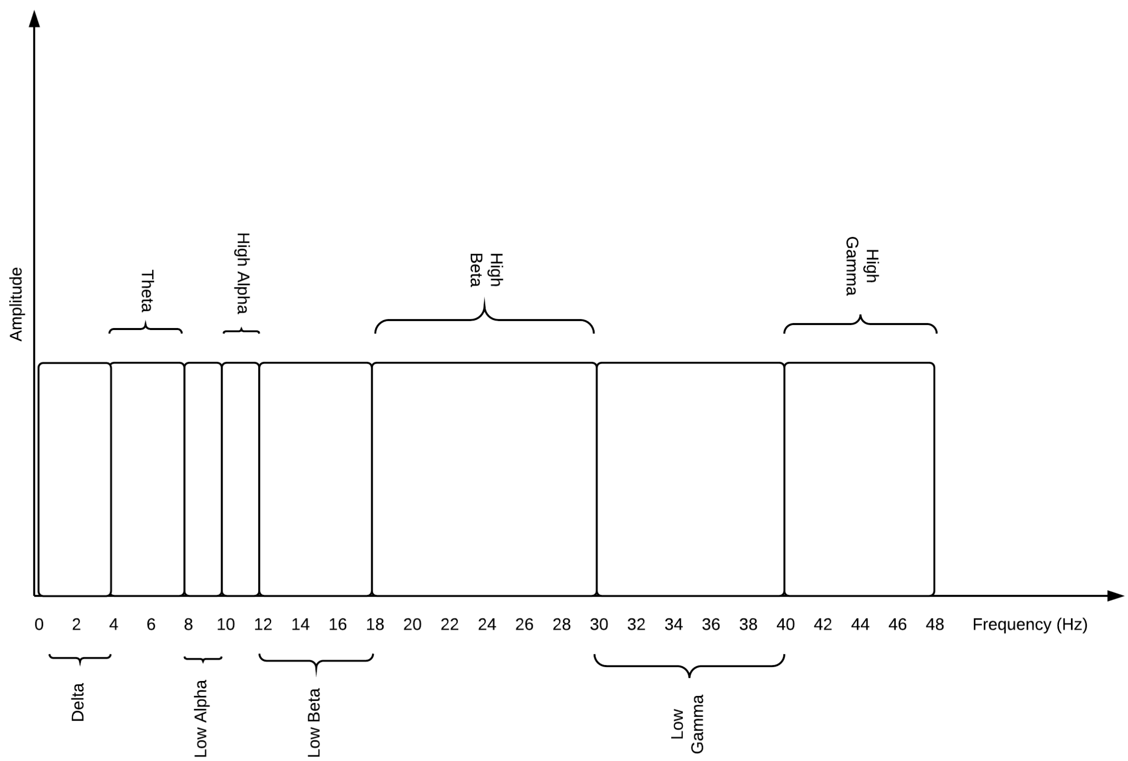

Figure 3.2 EEG Frequency Bands

If $X(n)$ is the input data sequence, $\mathscr{F}$ is the DFT operation and X_k is the DFT of $X(n)$, Equation 3.3 symbolically shows the DFT operation conducted on input sequence $X(n)$.

$$X(n) \xrightarrow{\mathscr{F}} X_k \,. \tag{3.3}$$

Say, if each subgroup of filtered EEG samples is $X(n) = [x_1, x_2, \ldots, x_{512}]^T$ and the DFT of $X(n)$ is $X(k)$, then the 512 point DFT of $X(n)$ is given by Equation 3.4.

$$X(k) = \sum_{n=0}^{n=511} X(n) \cdot \exp(-2\pi i k n/512), k \epsilon Z \,. \tag{3.4}$$

The spectral energy of each EEG frequency band i may be calculated using Equation 3.5.

$$E_i = \sqrt{\frac{1}{m_2 - m_1 + 1} \sum_{k=m_1}^{k=m_2} |X(k)|^2} \,, \tag{3.5}$$

where $m_2 > m_1$ and $k = [m_1, m_2] \in$ EEG frequency band i.

After obtaining the spectral energy of each EEG frequency band, we combine them to form a vector as shown in Equation 3.6.

$$X_{in} = [E_1, E_2, \ldots E_8] \,. \tag{3.6}$$

We then normalize the EEG spectral energy band vector X_{in} as shown in Equation 3.7 to obtain an unit vector X_n. This is next used as the input for the classifiers discussed in Section 3.3. Note that by normalizing, we were able to neutralize the effect of different sensitivity levels of EEG sensor for different users and test cases.

$$X_n = \frac{1}{|X_{in}|} X_{in} \,. \tag{3.7}$$

3.3 Classifiers

The input feature vectors obtained from the pre-processing block were randomly shuffled and split into training and testing. Following is the training and testing split percentage,

1. 70% of the feature vectors data set were used as training set.

2. 30% of the feature vectors data set were used as testing set.

For example, if each EEG mental task experiment (lasting 10 second each) is repeated 5 times, we have raw EEG data of 50 seconds. After pre-processing this data, we will have 50 input feature vectors. We then shuffle the ordering of these vectors and pick 35 (70%) as part of training set and 15 (30%) as part of testing set. The shuffling is done to randomize the training and testing split.

Since we have different users performing many mental tasks, we can try to identify the mental task given the subject or we can try to identify the subject among many subjects given the mental task. For this reason, we have conducted two different types of classification as given below,

1. **Intra - Subject Classification** : Identifying a mental task in a set of mental tasks performed by a single subject.

2. **Inter - Subject Classification** : Identifying a subject in a set of subjects performing the same mental task.

3.3.1 Mahalanobis Distance

As discussed in 2.9.1, the Mahalanobis Distance is a simple pattern recognition technique used to identify the class of the input vector. In our case, a class is either the type of task or the subject performing the mental task depending on the the classification type. The input vector $\mathbf{x}$ is the pre-processed EEG data vector given by Equation 3.8.

$$\mathbf{x} = [x_1 \quad x_2 \dots x_N], \tag{3.8}$$

where N is the number of variable in the input vector $\mathbf{x}$.

The Mahalanobis distance is computed using the Equation 3.9.

$$D_x^2 = (\mathbf{x} - E[\mathbf{x}])^T \mathbf{\Sigma}^{-1} (\mathbf{x} - E[\mathbf{x}]), \tag{3.9}$$

where $\mathbf{\Sigma}^{-1}$ is the inverse of the covariance matrix Σ and $E[\mathbf{x}]$ is the expected value of $\mathbf{x}$.

The expected value of $\mathbf{x}$ is given by Equation 3.10.

$$E[\mathbf{x}] = \boldsymbol{\mu} = [\mu_1 \mu_2 \dots \mu_N] = \sum_{i=1}^{M} \mathbf{x}_i, \tag{3.10}$$

where M is the number of input vectors.

The covariance matrix $\mathbf{\Sigma}$ is given by Equation 3.11.

$$\mathbf{\Sigma} = [(E[\mathbf{x} - E[\mathbf{x}])(E[\mathbf{x} - E[\mathbf{x}])^T] \tag{3.11}$$

As discussed in 2.9.1, we first calculate the mean vectors and sample covariance matrices for all the classes in the training set. We then calculate the Mahalanobis distance of the input testing vector with respect to each and every class using their corresponding mean vector and sample covariance matrix. We then assign the class label to the input testing vector by computing "the class which gives smallest Mahalanobis distance". Say, if we have class $C_1, C_2 \dots, C_m$ and $d_1, d_2 \dots d_m$ are the corresponding Mahalanobis distances of an input testing vector, we assign this input vector the class label C_i, if $d_i = min(d_1, d_2 \dots, d_m)$. Similarly, we then classify every single input in the testing set using Mahalanobis Distance.

3.3.2 Artificial Neural Networks

In Section 2.9.2, we briefly discussed Artificial Neural networks and how they can be used in pattern recognition. In this section, we will discuss perceptrons and multiple layer feed forward neural network.

3.3.2.1 Perceptron

Consider Figure 3.3, where input x and weights of the perceptron $\mathbf{w}$ are given by Equation 3.12 and Equation 3.13 respectively. Perceptron uses the input vector $\mathbf{x}$ and weight vector $\mathbf{w}$ such that the classification boundary, $\mathbf{w}^T\mathbf{x} = 0$ separates the classes.

$$\mathbf{x} = [1\, x_1\, x_2 x_3 \dots x_m]^T \ . \tag{3.12}$$

$$\mathbf{w} = [\, w_0\, w_1\, w_2 \dots w_m]^T \ . \tag{3.13}$$

The output of the perceptron for a given input vector is calculated using Equation 3.14.

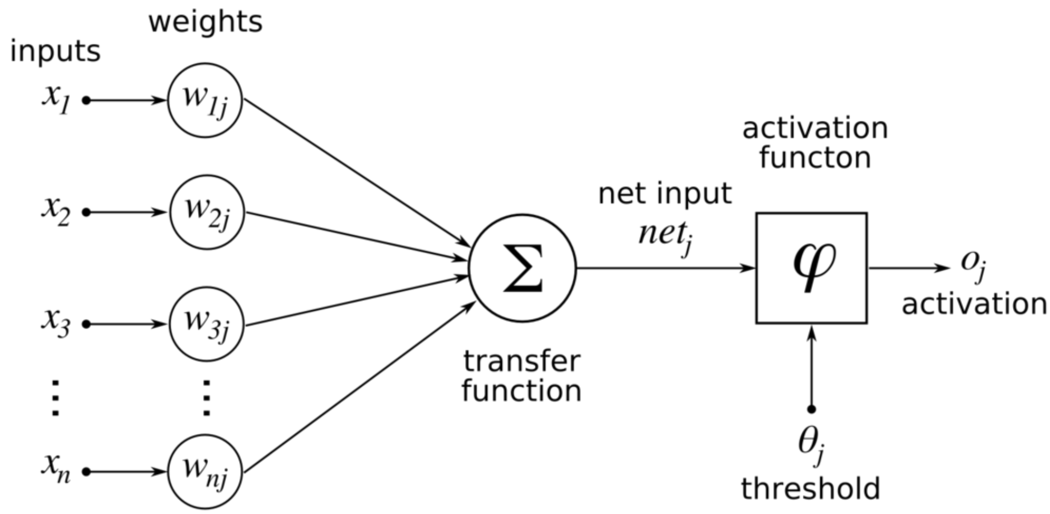

Figure 3.3 Perceptron [2]

$$F(\mathbf{x}) = \varphi(\mathbf{w}^T \mathbf{x}) \,, \tag{3.14}$$

where φ is the activation function. Different activation function like tanh (Equation 3.15), sigmoid (soft step) (Equation 3.16) etc., can be used as activation function. The choice of the activation function does not affect the training methods. For our experiments we use sigmoid activation function.

$$\varphi(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}} \,. \tag{3.15}$$

$$\varphi(v) = \frac{1}{1 + e^{-v}} \,. \tag{3.16}$$

Since the perceptron is a supervised machine learning algorithm, it requires training. Here, the training involves tuning the weight vector **w**. This can be done using the Gradient

Decent algorithm. If d_j represents the desired output and y_j is the actual output of the perceptron for j^{th} input vector x_j, we can calculate the error function for i^{th} iteration of the gradient decent algorithm using Equation 3.17.

$$E^{(i)} = \frac{1}{2m} \sum_{k=1}^{k=m} (\varphi(\mathbf{w}^T x(k)) - d(k))^2 . \tag{3.17}$$

The Gradient Decent Algorithm states that, the error function E can be minimized (given that $\varphi(\mathbf{w}^T x(k))$ is differentiable) by updating the weight vector as shown in Equation 3.18.

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \cdot \mathbf{x} \cdot (\mathbf{y}^{(i)} - \mathbf{d}) , \tag{3.18}$$

where $\mathbf{y} = [y_1 y_2 \ldots y_n]^T$ is the actual output of the perceptron in vector form, $\mathbf{d} = [d_1 d_2 \ldots d_n]^T$ is the desired output of the perceptron in vector form and η is the learning rate parameter. The learning rate parameter η determines how fast the weights converge. Keeping η too low will result in slow convergence resulting in large number of iterations to reach the optimal solution. On the other hand, large η might not guarantee the optimal solution. Hence, it is better to start η with a high value and gradually reduce it after every iteration.

3.3.2.2 Multi Layer Perceptron

The Multi layer perceptron (MLP) is an extension of the perceptron. This architecture has more neurons connected to each other and allows non-linear classification boundaries. As discussed in Section 2.9.2, the multi layer perceptron typically contains an input layer, one or more hidden layers and an output layer as shown in Figure 3.4 .

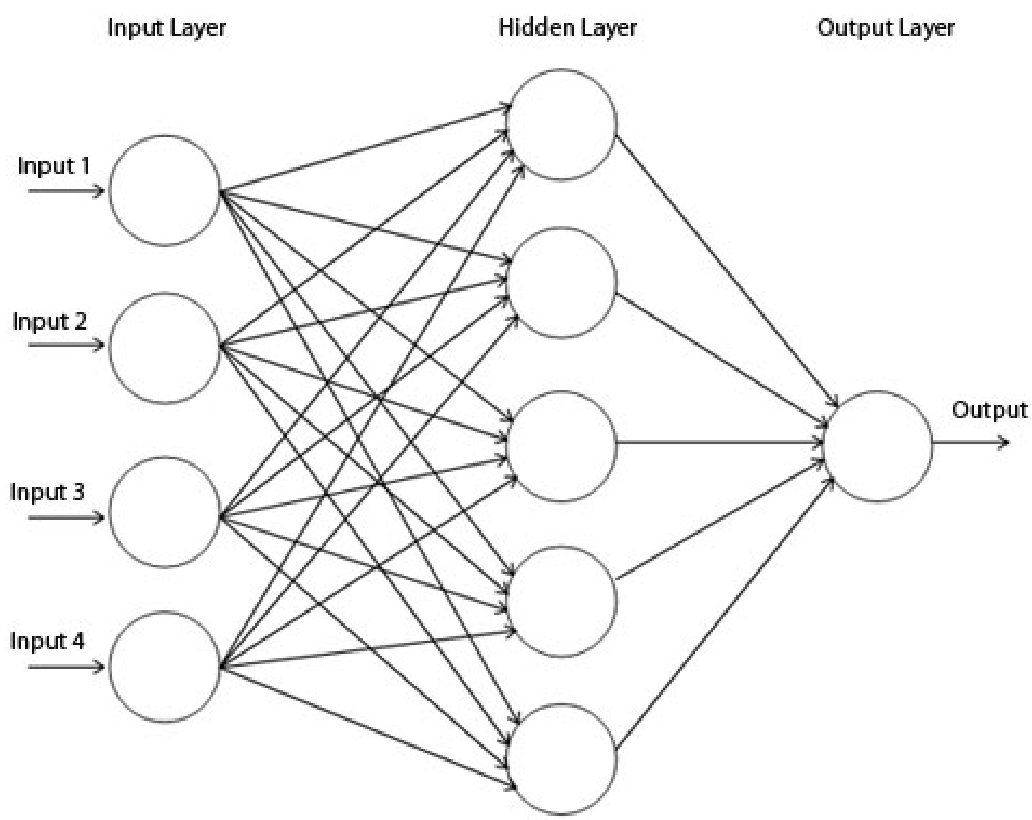

Figure 3.4 Multi Layer Perceptron [15]

According to Universal approximation theorem, a single hidden layer with finite number of neurons is sufficient to approximate a given training set [9]. Hence, a single hidden layer was used for our use case. Also, The Sigmoid (soft step) (Equation 3.16) function was used as the activation function. The features used for the Neural Network are as shown in Table 3.3.

Table 3.3 Neural Network Features

Parameters	Value
Type of network	Feed Forward network
Input layer size	8
No. of hidden layers	1
Hidden layer size	8
Output layer size	Vary based on number of classes
Activation function in hidden layer	Sigmoid
Training Algorithm used	Backpropagation Algorithm

Just like the perceptron, the MLP has a input vector $\mathbf{x} = [1\, x_1\, x_2 \dots x_{m_0}]^T$ of dimension $m_0 + 1 \times 1$. Here, $x_0 = 1$ is the bias term. Similar to the perceptron, the MLP also has weights. Since we have multiple neurons in the hidden layer connected to the input layer, we instead have a weight matrix $\mathbf{w}$ with dimension $(m_0 + 1) \times m_1$, given by Equation 3.19.

$$\mathbf{w} = \begin{bmatrix} w_{10}^{(1)} & w_{20}^{(1)} \dots & w_{m_1 0}^{(1)} \\ w_{11}^{(1)} & w_{21}^{(1)} \dots & w_{m_1 1}^{(1)} \\ \vdots & \vdots & \vdots \\ w_{1 m_0}^{(1)} & w_{2 m_0}^{(1)} \dots & w_{m_1 m_0}^{(1)} \end{bmatrix}. \tag{3.19}$$

Here the subscript in w_{ij}^k, i and j represent the connection from j^{th} input to the i^{th}

neuron in the hidden layer. The superscript k represent the k^{th} hidden layer. Since only one hidden layer was used, we have $k = 1$. Each column vector in the matrix $\mathbf{w}$ represents the weight vector for a single neuron in the hidden layer. The output of the hidden layer is calculated using Equation 3.20.

$$\mathbf{x}_{h_0} = \varphi(\mathbf{w}^T \mathbf{x}) \,, \tag{3.20}$$

where φ is the activation function.

The output of hidden layer is then used as the "input' to the output layer. After adding a bias term to the output of the hidden layer, we have $\mathbf{x}_{h1}$ given by Equation 3.21.

$$\mathbf{x}_{h_1} = [1 \quad \mathbf{x}_{h_0}]^T \,. \tag{3.21}$$

Output of the network is then calculated using Equation 3.22.

$$y(\mathbf{x}) = \varphi(\mathbf{w_o}^T \mathbf{x}_{h_1}) \,, \tag{3.22}$$

where $\mathbf{w}_o$ is is given by Equation 3.23.

$$\mathbf{w}_o = \begin{bmatrix} w^{(2)}_{10} & w^{(2)}_{20} \cdots & w^{(2)}_{m_2 0} \\ w^{(2)}_{11} & w^{(2)}_{21} \cdots & w^{(2)}_{m_2 1} \\ \vdots & \vdots & \vdots \\ w^{(2)}_{1 m_1} & w^{(2)}_{2 m_1} \cdots & w^{(2)}_{m_2 m_1} \end{bmatrix} \,, \tag{3.23}$$

where m_1 is the number of neurons in the hidden layer and m_2 is the number of neurons in the output layer.

Training the neural network was done using backpropagation algorithm. Backpropagation algorithm is an iterative algorithm, which minimizes the output error by adjusting the weight matrices of the network. The detailed description of backpropagation algorithm can be found in [5].

3.3.3 Support Vector Machines (SVMs)

As discussed in Section 2.9.3 the SVMs maximize the distance between the separating hyperplane and the nearest points of the classes to the hyperplane. The detailed derivation of how SVMs achieve this can be found in [20]. The the results in sections 3.3.3.1 and 3.3.3.2 follow the detailed derivation of SVMs in [20].

3.3.3.1 Linear SVMs

If i^{th} input vector is given by $\mathbf{x_i} = [x_1 x_2 \dots x_m]$, then the objective function $L(\lambda)$ of the SVM is given by Equation 3.24.

$$L(\boldsymbol{\lambda}) = \sum_{i=0}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j. \tag{3.24}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0. \tag{3.25}$$

$$\lambda_i \geq 0, i = 1, \dots, n. \tag{3.26}$$

Here, y_i is the desired output for the i^{th} input sample. The goal here is to find the Lagrange multipliers $\alpha_i' s$, so that the objective function $L(\boldsymbol{\lambda})$ is maximized. Also, while optimizing the objective function, the constraints shown in Equation 3.25 and Equation 3.26 are used given the training data. This is called the “dual form” of the constrained optimization problem of the support vector machines. Also, all the input vectors in the training set with $\alpha_i \neq 0$ are called the “support vectors” (hence the name).

By defining matrix A as shown in Equation 3.27 and if Λ denotes the vector of Lagrange multipliers, we can write the matrix form of $L(\lambda)$ as shown in Equation 3.28 [20].

$$A = \left[y_i y_j \mathbf{x}_i^T \mathbf{x}_j \right]. \tag{3.27}$$

$$L(\boldsymbol{\lambda}) = -\frac{1}{2}\Lambda^T A \Lambda + \mathbf{1}^T \Lambda, \tag{3.28}$$

where $\mathbf{1}$ is a vector of all ones. After finding the required Lagrange multipliers, we can compute the optimal projection vector using Equation 3.29.

$$\mathbf{w} = \sum_i \lambda_i \mathbf{x}_i y_i \,, \tag{3.29}$$

where y_i is given by Equation 3.30,

$$y_i = (\mathbf{w}^T \mathbf{x} + b) \tag{3.30}$$

and b can be solved using Equation 3.31,

$$\lambda_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) = 0 \forall i. \tag{3.31}$$

3.3.3.2 Nonlinear Support Vector Machines

Here, we apply nonlinear transformation ϑ to the input vector $\mathbf{x}_i$ to produce a vector of higher dimension $\mathbf{x}'_i$ as shown in Equation 3.32.

$$\mathbf{x}'_i = \vartheta(\mathbf{x}_i) : (\mathbb{R}^d \rightarrow \mathbb{R}^m), m > d. \tag{3.32}$$

Now, if we apply Equation 3.32 to Equation 3.27, we have,

$$A = \left[y_i y_j \vartheta(\mathbf{x}_i)^T \vartheta(\mathbf{x}_j) \right]. \tag{3.33}$$

The nonlinear operation and the inner product can be replaced by the single operation called "Kernel function" $\mathbf{K}(\mathbf{a}, \mathbf{b})$ if it satisfies the following Mercer's condition,

$$\int \mathbf{K}(\mathbf{a}, \mathbf{b}) g(\mathbf{a}) g(\mathbf{b}) \; da \; db \geq 0 \,, \tag{3.34}$$

where $g(\mathbf{x})$ has finite energy. One of the most popular kernel which satisfies the Mercers condition is the Radial Basis Function given by Equation 3.35.

$$\mathbf{K}(\mathbf{a}, \mathbf{b}) = \exp\left(-\frac{(\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b})}{2\sigma^2} \right). \tag{3.35}$$

3.4 Conclusion

In Chapter 3, we discussed the User Verification System using EEG signals pipeline in detail. We discussed about how to collect the data from Mindwave Mobile EEG sensor, how to pre-process the raw EEG signals to remove noise, how to extract feature vectors from pre-processed EEG signal, how to classify the input feature vectors using Mahalanobis distance, Neural Networks and SVMs. In next chapter, we will discuss about the results of classification.

CHAPTER

4

RESULTS

In this chapter we first discuss the performance of the Intra - Subject classification. Later we discuss the performance of Inter - subject Classification.

4.1 Measuring Performance

Apart from the classification accuracy the two measures of performances used are “True positive rate” and “False Positive rate”. Say if we must have to identify the class c_i among

Table 4.1 Confusion Matrix

		Predicted Condition	
	Total Classification	Predicted Positive	Predicted Negative
True Condition	Actually Positive	True Positive (TP)	False Negative (FN)
	Actually Negative	False Positive (FP)	True Negative (TN)

$c_1, c_2 \ldots c_n$, the true positive rate measures the percentage of patterns classified correctly as class c_1, whereas the false positive rate measures the percentage of patterns which do not belong to class c_1 but are classified as class c_1.

True Positive rate (TPR) is computed as given by Equation 4.1.

$$TPR = \frac{TP}{TP + FN}, \tag{4.1}$$

where TP is the number of true positive samples and FN is the number of false negative samples. The confusion matrix shown in Table 4.1 gives better understanding of true positives(TP), false positives(FP), true negatives(TN) and false negatives(FN).

Similar to TPR, false positive rates (FPR) can be calculated using Equation 4.2.

$$FPR = \frac{FP}{FP + TN}, \tag{4.2}$$

4.2 Similarity in EEG signals

Figure 4.1, Figure 4.2 and Figure 4.3 show the mean of amplitude of the feature vectors for calculation, breathing and singing tasks. Figure 4.4, Figure 4.5 and Figure 4.6 show the variance of amplitude of the feature vectors. The description of these tasks is given by Table 3.1. As we can see, from the “mean” graphs, EEG bands for four different test subjects follow similar patterns for the same task performed, although they vary slightly. As a result, classifying EEG signals is difficult. Also, note that the variance for Subject 1 is very low. This maybe because the brain wave signatures of Subject 1 for different tasks are closer compared to other subjects.

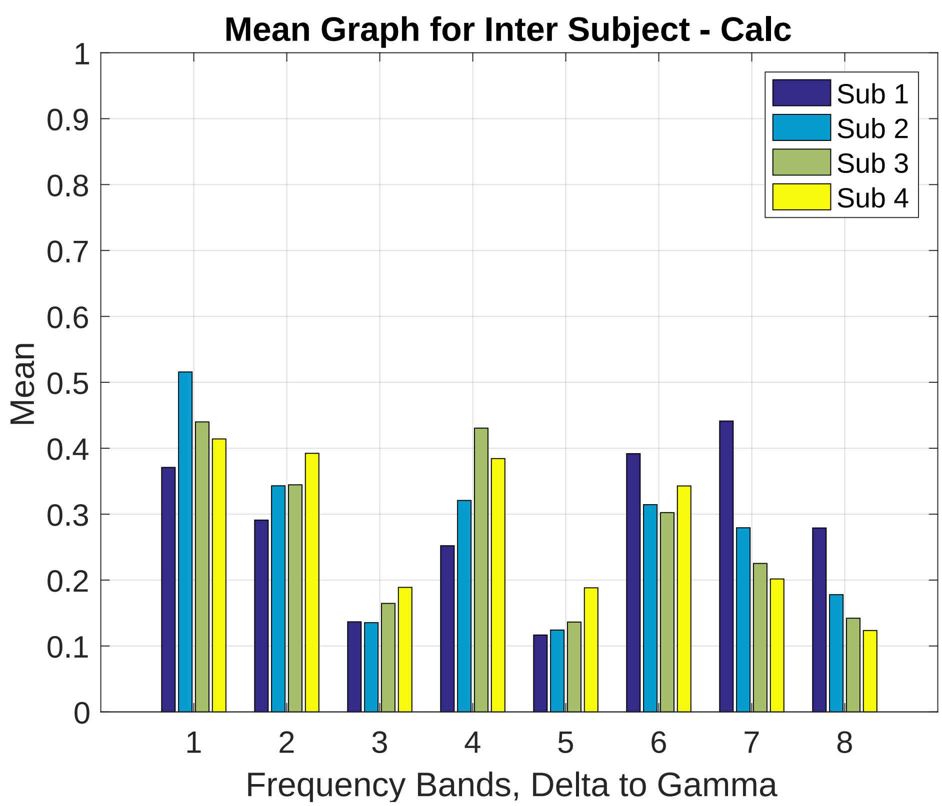

Figure 4.1 Mean of each EEG band for Calculation task

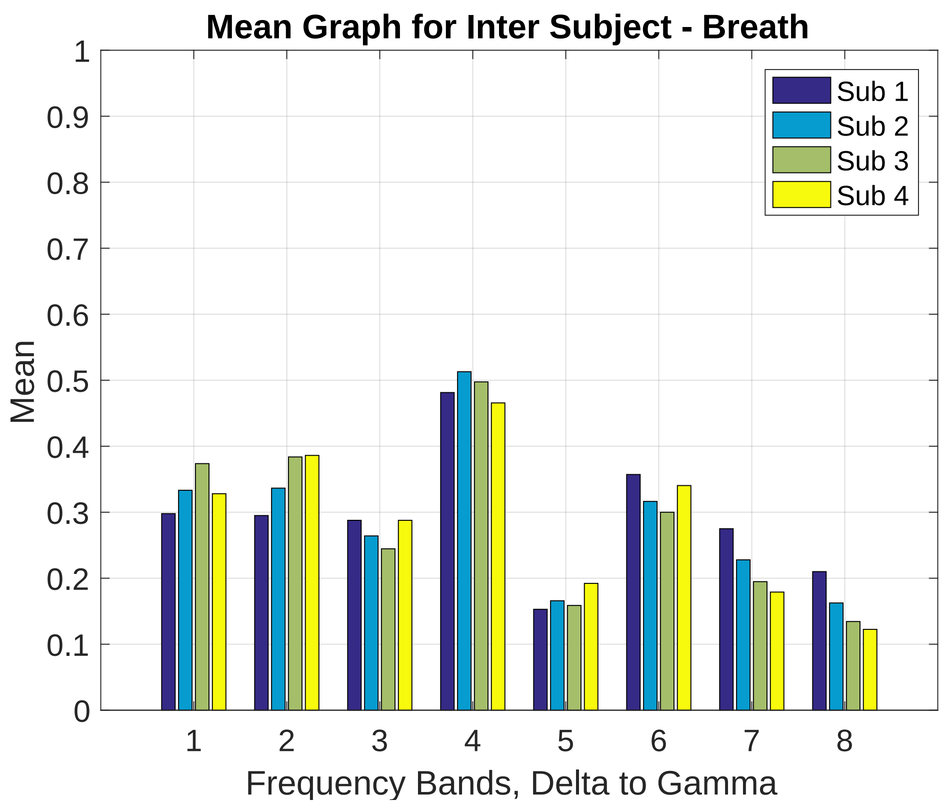

Figure 4.2 Mean of each EEG band for Breathing task

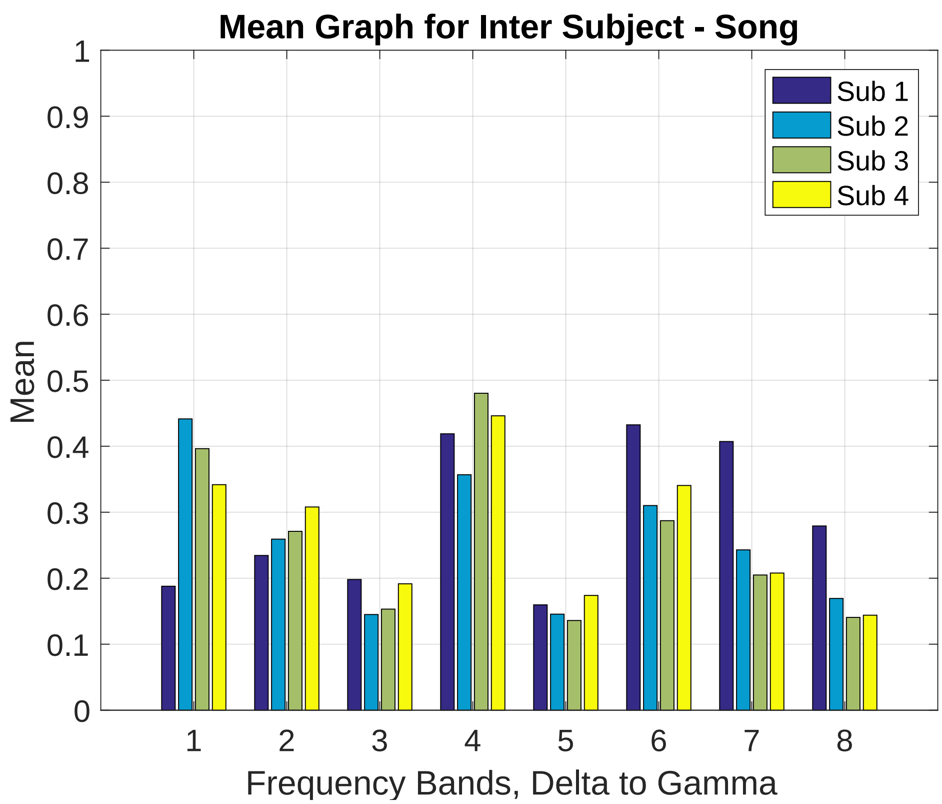

Figure 4.3 Mean of each EEG band for Singing task

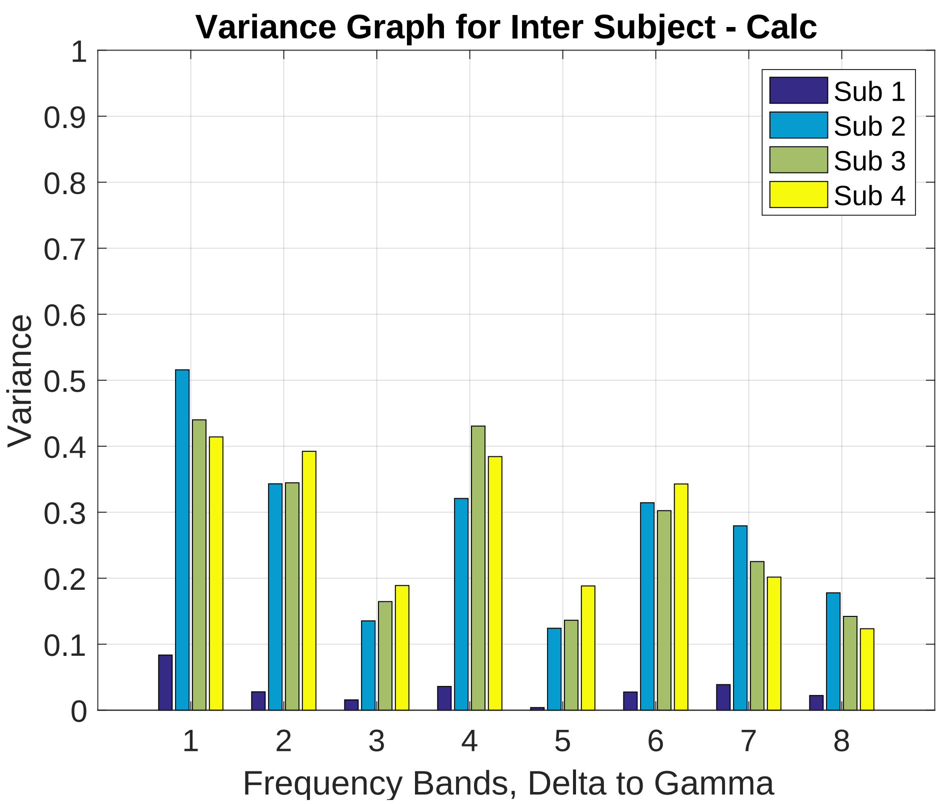

Figure 4.4 Variance of each EEG band for Calculation task

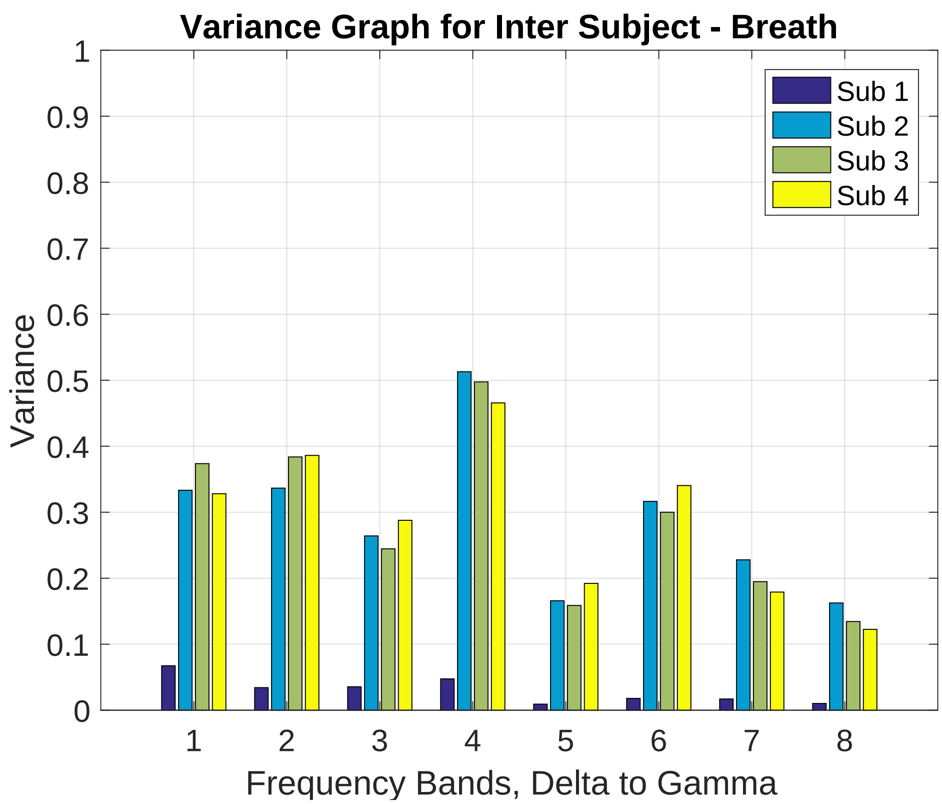

Figure 4.5 Variance of each EEG band for Breathing task

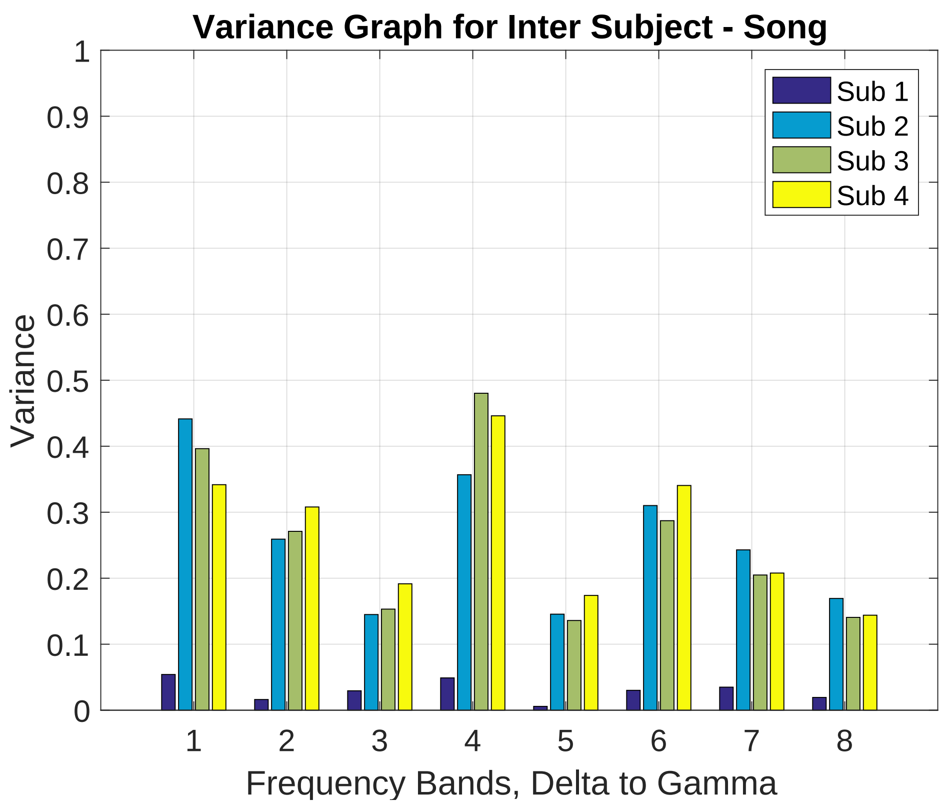

Figure 4.6 Variance of each EEG band for Singing task

4.3 Classifier Performance

Performance of the Mahalanobis Distance classifier, the Neural Network classifier and the SVM classifier are discussed in this section. As discussed in Section 3.3, we consider two types of EEG data classification. First is to identify the "task" performed by the subject. We call this "Intra-subject classification". Second type of classification is to identify the subject among group of subject performing similar tasks. We call this "Inter-subject classification". Here task refers to the brain activity like doing mathematical calculation, concentrating on breathing or mentally singing a song(refer Table 3.1 for more details).

We also need to consider comparison of baseline performance verses classifier performance. For example, if the feature vectors of all the classes are uniformly distributed, then, if we randomly choose a class among given classes, the probability of being right is given by Equation 4.3.

$$P = \frac{1}{N}\,, \tag{4.3}$$

where N is the total number of classes. We call this the "baseline performance".

4.3.1 Intra-subject Classification

For this experiment, we collected data from four different test subjects performing three different tasks repeated five times each. As discussed in Section 3.3, training and testing split was 70% and 30% respectively.

4.3.1.1 Mahalanobis Distance

Table 4.2 and Figure 4.7 show the overall accuracy of the Mahalanobis distance classifier for intra-subject classification. Table 4.3 and Figure 4.8 show the TPR for different tasks using the Mahalanobis distance classifier. Also, Table 4.4 and Figure 4.9 show the FPR for different tasks using the Mahalanobis distance classifier.

Table 4.2 Intra-subject Classification using Mahalanobis Distance, Total Accuracy

Sub	Min	Max	Average
1	33.33	55.56	44.89
2	44.44	66.67	55.78
3	35.56	46.67	42.67
4	44.44	71.11	61.56

Table 4.3 Intra-subject Classification using Mahalanobis Distance, TPR for Calculation, Breathing and Singing Task

(a) Calculation

Sub	Min	Max	Average
1	20.00	66.67	40.00
2	26.67	73.33	44.67
3	20.00	80.00	54.00
4	33.33	73.33	54.67

(b) Breathing

Sub	Min	Max	Average
1	40.00	73.33	53.33
2	60.00	86.67	72.67
3	26.67	73.33	45.33
4	20.00	73.33	46.67

(c) Singing

Sub	Min	Max	Average
1	13.33	66.67	42.00
2	26.67	73.33	42.67
3	13.33	66.67	27.33
4	60.00	93.33	77.33

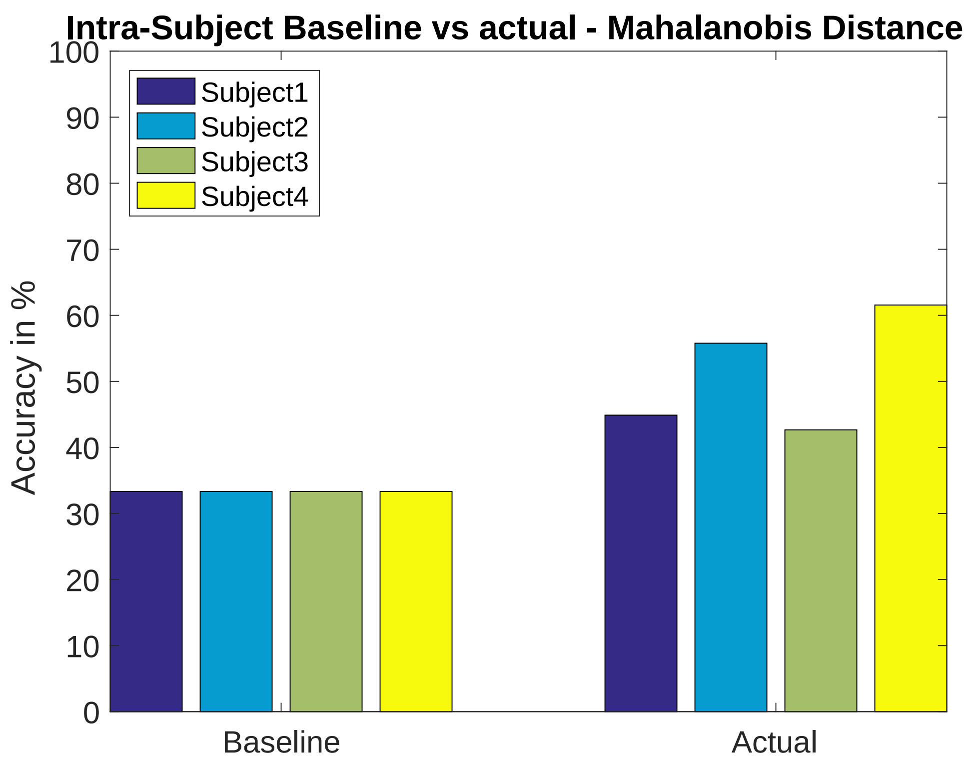

Figure 4.7 Total accuracy for Intra-subject classification using the Mahalanobis Distance Classifier

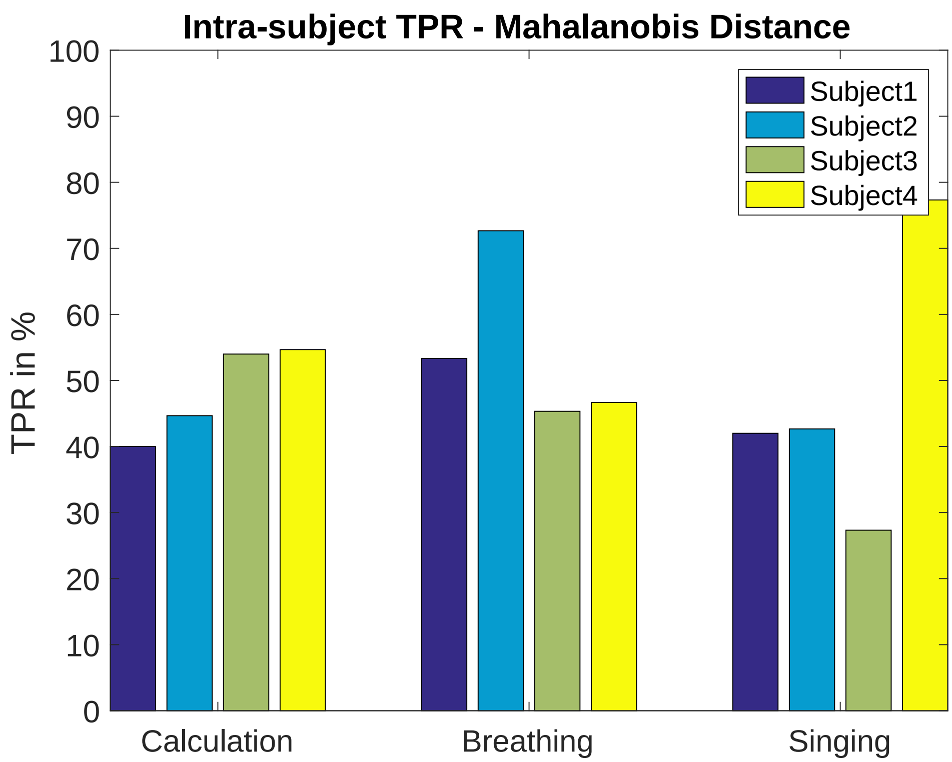

Figure 4.8 TPR for Intra-subject classification using the Mahalanobis Distance Classifier

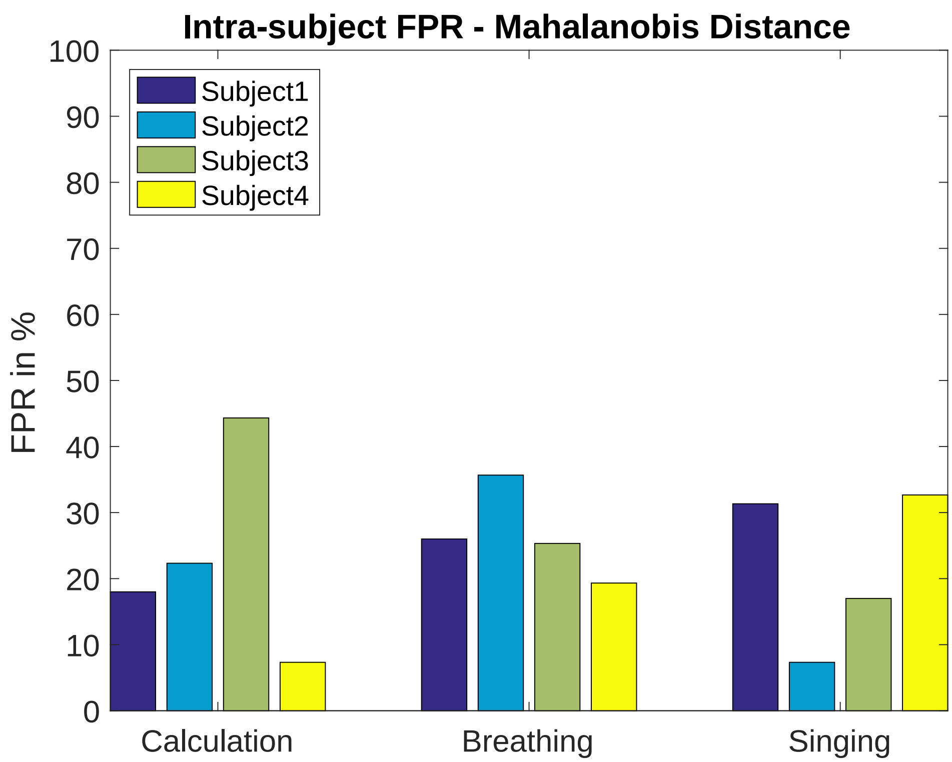

Figure 4.9 FPR for Intra-subject classification using the Mahalanobis Distance Classifier

Table 4.4 Intra-subject Classification using Mahalanobis Distance, FPR for Calculation, Breathing and Singing Task

(a) Calculation

Sub	Min	Max	Average
1	3.33	36.67	18.00
2	13.33	40.00	22.33
3	33.33	60.00	44.33
4	0.00	16.67	7.33

(b) Breathing

Sub	Min	Max	Average
1	20.00	36.67	26.00
2	23.33	53.33	35.67
3	6.67	43.33	25.33
4	3.33	36.67	19.33

(c) Singing

Sub	Min	Max	Average
1	10.00	46.67	31.33
2	0.00	13.33	7.33
3	6.67	23.33	17.00
4	20.00	53.33	32.67

4.3.1.2 Neural Networks

Table 4.5 and Figure 4.10 show the overall accuracy of the Neural Networks classifier for intra-subject classification. Table 4.6 and Figure 4.11 show the TPR for different tasks using the Neural Networks classifier. Also, Table 4.7 and Figure 4.12 show the FPR for different tasks using the Neural Networks classifier.

Table 4.5 Intra-subject Classification using Neural Networks, Total Accuracy

Sub	Min	Max	Average
1	53.33	64.44	58.22
2	57.78	73.33	65.11
3	44.44	55.56	48.44
4	62.22	73.00	66.67

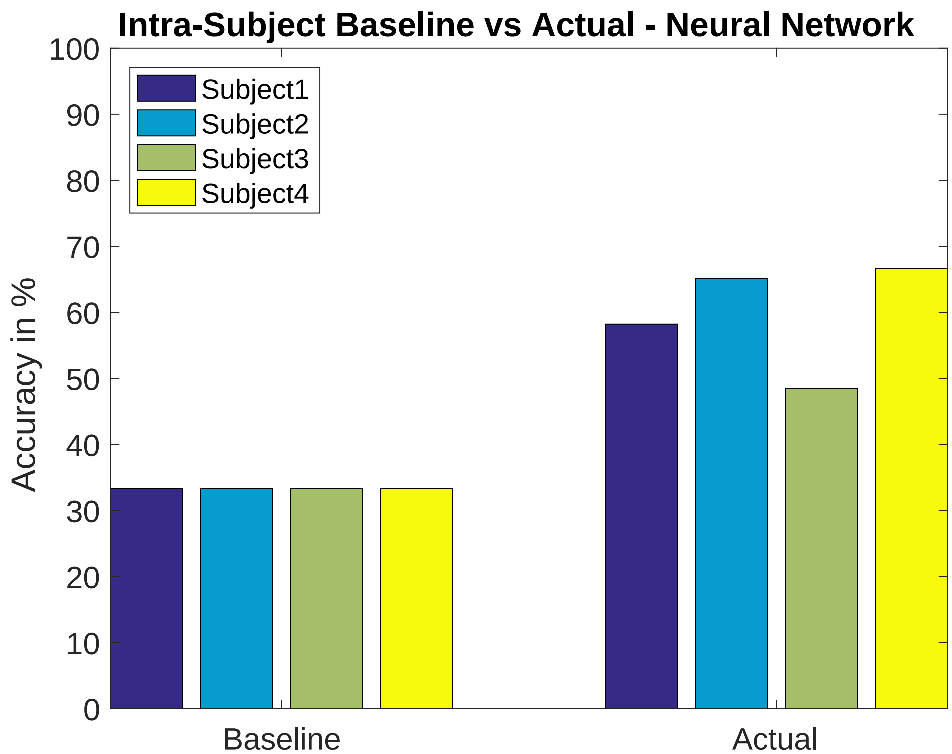

Figure 4.10 Total accuracy for Intra-subject classification using the Neural Network Classifier

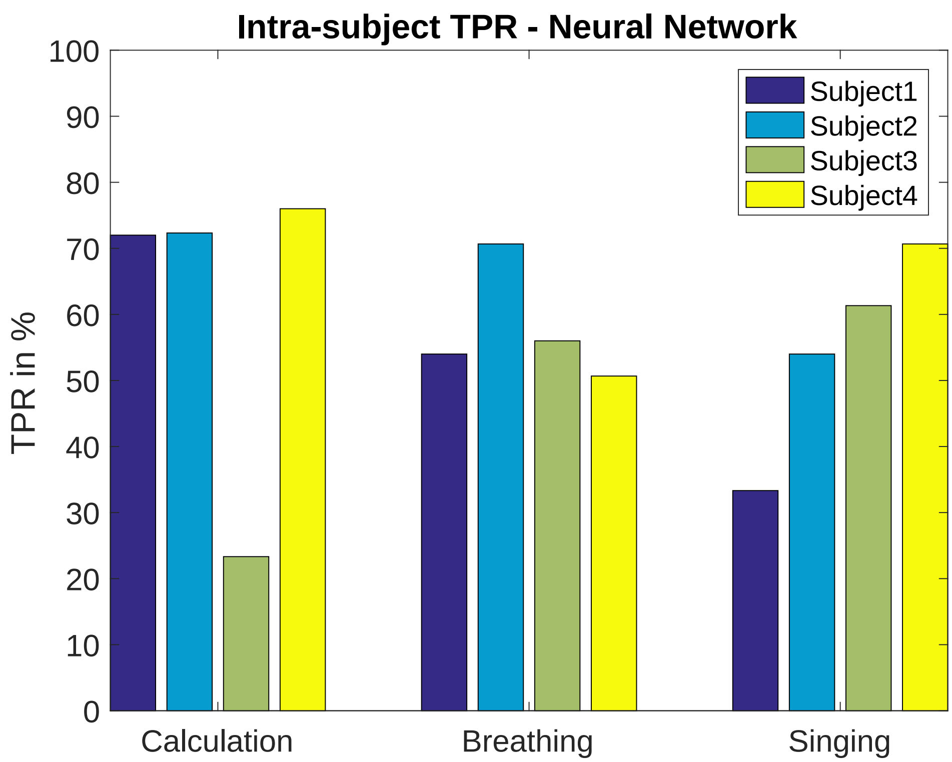

Figure 4.11 TPR for Intra-subject classification using the Neural Network Classifier

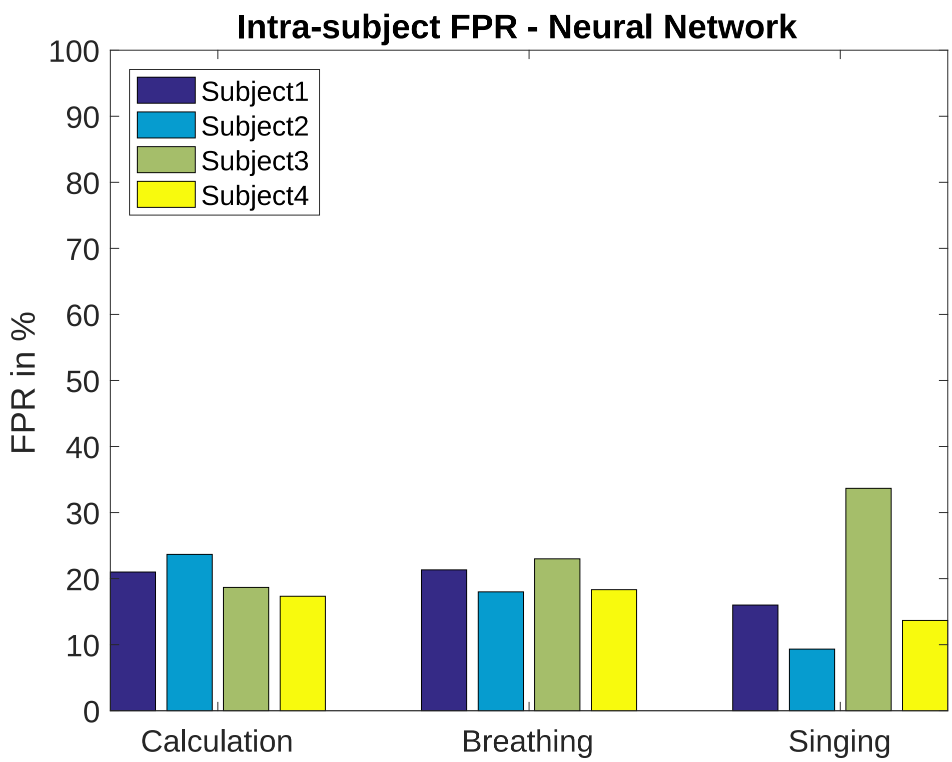

Figure 4.12 FPR for Intra-subject classification using the Neural Network Classifier

Table 4.6 Intra-subject Classification using Neural Networks, TPR for Calculation, Breathing and Singing Task

(a) Calculation

Sub	Min	Max	Average
1	60.00	86.67	72.00
2	46.67	93.33	72.33
3	6.67	33.33	23.33
4	66.67	93.33	76.00

(b) Breathing

Sub	Min	Max	Average
1	33.33	66.67	54.00
2	66.67	86.67	70.67
3	26.67	66.67	56.00
4	33.33	66.67	50.67

(c) Singing

Sub	Min	Max	Average
1	13.33	53.33	33.33
2	26.67	80.00	54.00
3	40.00	86.67	61.33
4	60.00	93.33	70.67

Table 4.7 Intra-subject Classification using Neural Networks, FPR for Calculation, Breathing and Singing Task

(a) Calculation

Sub	Min	Max	Average
1	13.33	36.67	21.00
2	16.67	30.00	23.67
3	13.33	33.33	18.67
4	10.00	33.33	17.33

(b) Breathing

Sub	Min	Max	Average
1	10.00	33.33	21.33
2	3.33	26.67	18.00
3	10.00	36.67	23.00
4	6.67	26.67	18.33

(c) Singing

Sub	Min	Max	Average
1	10.00	23.33	16.00
2	3.33	13.33	9.33
3	20.00	60.00	33.67
4	3.33	23.33	13.67

4.3.1.3 Support Vector Machines

Table 4.8 and Figure 4.13 show the overall accuracy of the SVM classifier for intra-subject classification. Table 4.9 and Figure 4.14 show the TPR for different tasks using the SVM classifier. Also, Table 4.10 and Figure 4.15 show the FPR for different tasks using the SVM classifier.

Table 4.8 Intra-subject Classification using Support Vector Machines, Total Accuracy

Sub	Min	Max	Average
1	40.00	57.78	50.67
2	55.56	71.11	66.44
3	33.33	48.89	41.11
4	57.78	73.33	65.56

Table 4.9 Intra-subject Classification using Support Vector Machines, TPR for Calculation, Breathing and Singing Task

(a) Calculation

Sub	Min	Max	Average
1	53.33	86.67	70.00
2	60.00	93.33	77.33
3	40.00	73.33	53.33
4	73.33	93.33	82.00

(b) Breathing

Sub	Min	Max	Average
1	40.00	73.33	56.00
2	33.33	73.33	61.33
3	40.00	73.33	55.33
4	40.00	66.67	45.33

(c) Singing

Sub	Min	Max	Average
1	6.67	40.00	20.67
2	33.33	66.67	48.00
4	46.67	73.33	60.00
4	68.67	77.00	77.67

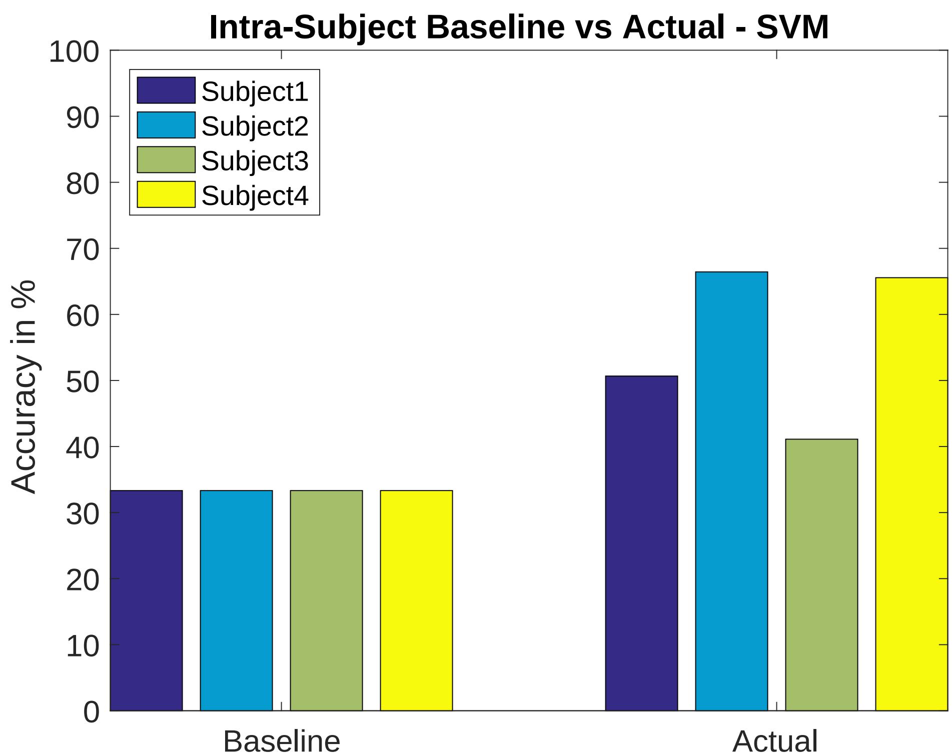

Figure 4.13 Total accuracy for Intra-subject classification using the SVM Classifier

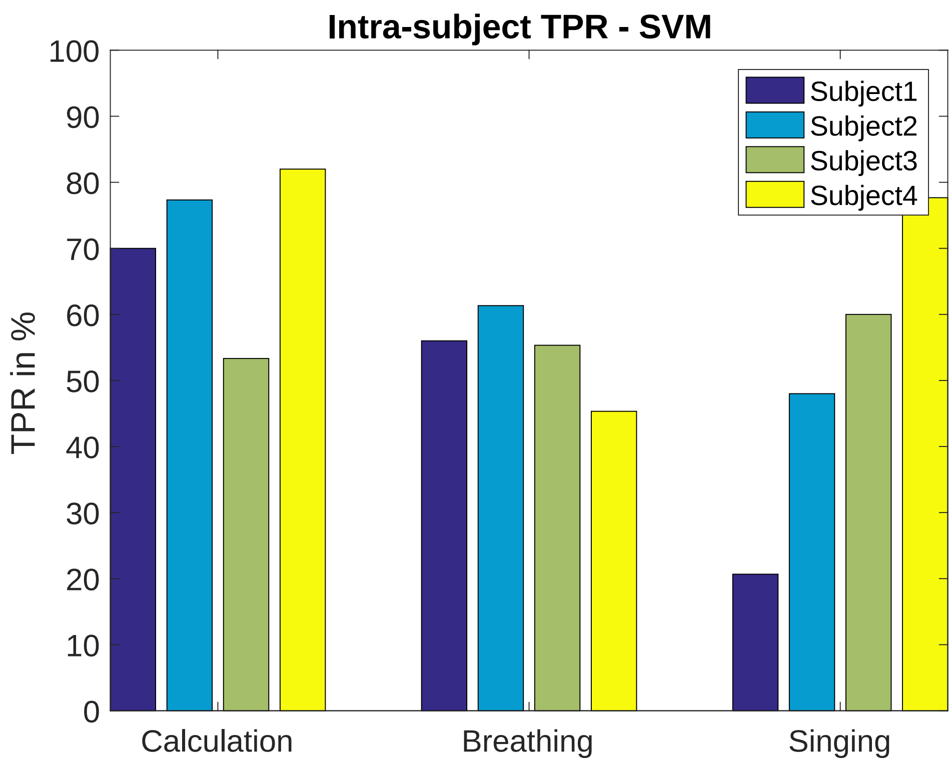

Figure 4.14 TPR for Intra-subject classification using the SVM Classifier

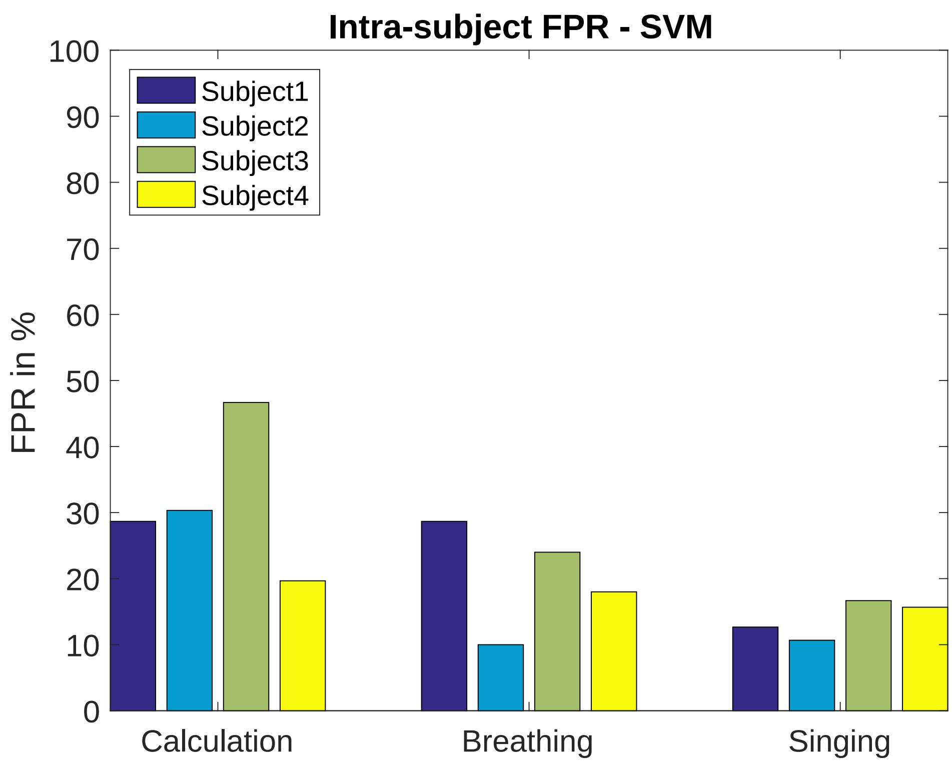

Figure 4.15 FPR for Intra-subject classification using the SVM Classifier

Table 4.10 Intra-subject Classification using Support Vector Machines, FPR for Calculation, Breathing and Singing Task

(a) Calculation

Sub	Min	Max	Average
1	20.00	36.67	28.67
2	23.33	40.00	30.33
3	36.67	60.00	46.67
4	10.00	33.33	19.67

(b) Breathing

Sub	Min	Max	Average
1	16.67	60.00	28.67
2	0.00	20.00	10.00
3	13.33	40.00	24.00
4	6.67	40.00	18.00

(c) Singing

Sub	Min	Max	Average
1	3.33	26.67	12.67
2	3.33	20.00	10.67
3	6.67	33.33	16.67
4	0.00	23.33	15.67

4.3.2 Inter-Subject Classification

For this experiment, we collected data from four different test subjects performing three different tasks repeated five times each. As discussed in Section 3.3, training and testing split was 70% and 30% respectively.

4.3.2.1 Mahalanobis Distance

Table 4.11 and Figure 4.16 show the overall accuracy of the Mahalanobis distance classifier for inter-subject classification. Table 4.12 and Figure 4.17 show the TPR for different test subjects using the Mahalanobis distance classifier. Also, Table 4.13 and Figure 4.18 show the FPR for different test subjects using the Mahalanobis distance classifier.

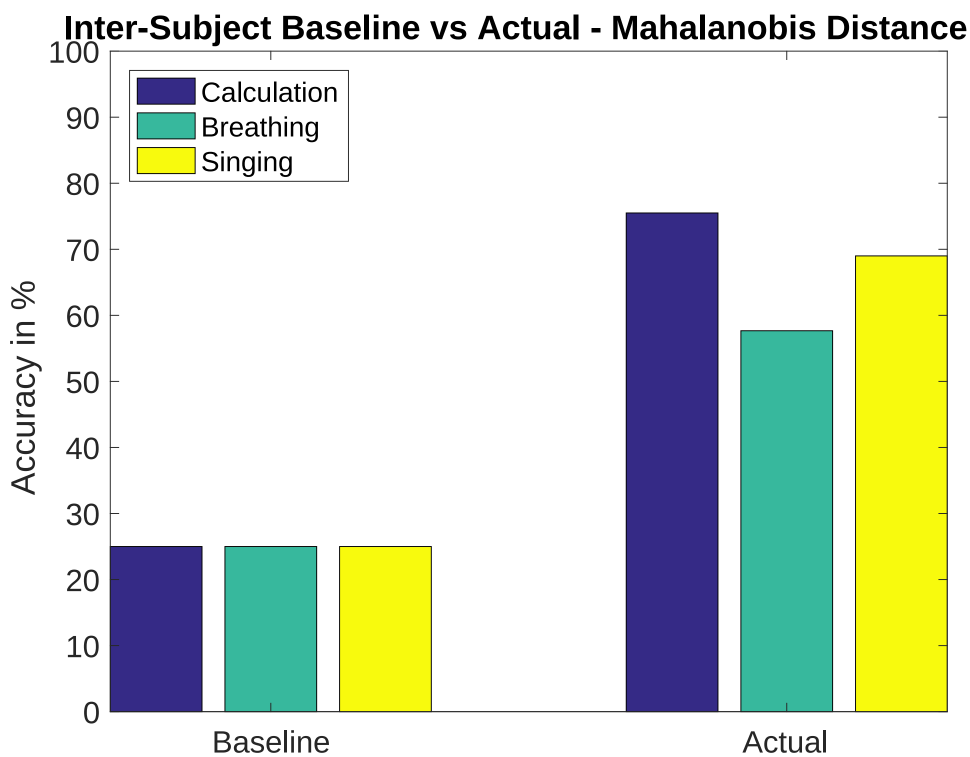

Figure 4.16 Total accuracy for Inter-subject classification using the Mahalanobis Distance Classifier

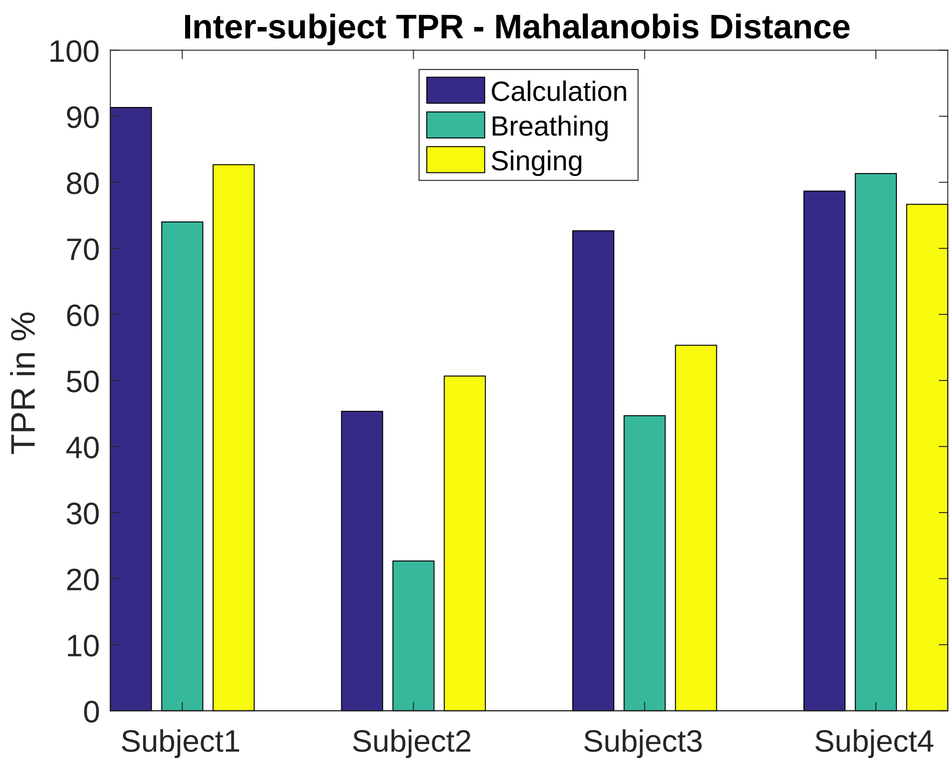

Figure 4.17 TPR for Inter-subject classification using the Mahalanobis Distance Classifier

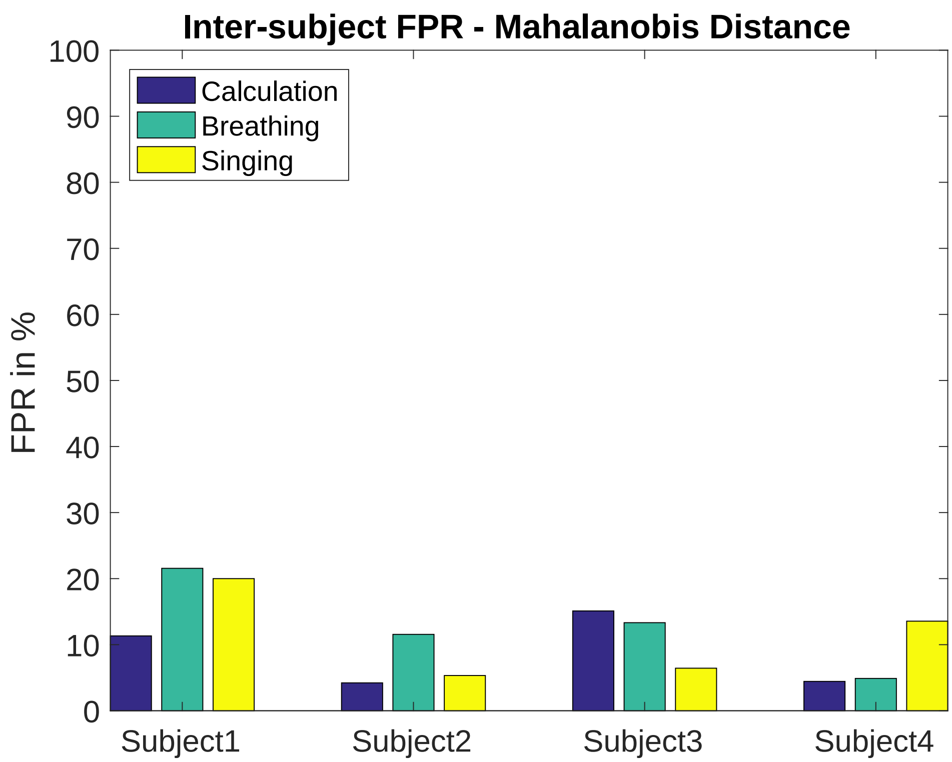

Figure 4.18 FPR for Inter-subject classification using the Mahalanobis Distance Classifier

Table 4.11 Inter-subject Classification for 4 subjects using Mahalanobis Distance - Total Accuracy

Task	Min	Max	Average
Calculation	65.00	85.00	75.50
Breathing	50.00	65.00	57.67
Singing	55.00	78.33	69.00

Table 4.12 Inter-subject Classification for 4 subjects using Mahalanobis Distance - TPR for Subject 1-4

(a) Subject 1

Task	Min	Max	Average
Calculation	80.00	100.00	91.33
Breathing	60.00	93.33	74.00
Singing	73.33	100.00	82.67

(b) Subject 2

Task	Min	Max	Average
Calculation	20.00	66.67	45.33
Breathing	6.67	40.00	22.67
Singing	33.33	73.33	50.67

(c) Subject 3

Task	Min	Max	Average
Calculation	46.67	93.33	72.67
Breathing	26.67	73.33	44.67
Singing	33.33	73.33	55.33

(d) Subject 4

Task	Min	Max	Average
Calculation	60.00	86.67	78.67
Breathing	66.67	100.00	81.33
Singing	46.67	93.33	76.67

Table 4.13 Inter-subject Classification for 4 subjects using Mahalanobis Distance - FPR for Subject 1-4

(a) Subject 1

Task	Min	Max	Average
Calculation	2.22	17.78	11.33
Breathing	13.33	31.11	21.56
Singing	6.67	35.56	20.00

(b) Subject 2

Task	Min	Max	Average
Calculation	0.00	8.89	4.22
Breathing	4.44	20.00	11.56
Singing	2.22	11.11	5.33

(c) Subject 3

Task	Min	Max	Average
Calculation	6.67	22.22	15.11
Breathing	4.44	28.89	13.33
Singing	2.22	15.56	6.44

(d) Subject 4

Task	Min	Max	Average
Calculation	2.22	6.67	4.44
Breathing	0.00	8.89	4.89
Singing	2.22	28.89	13.56

Inter subject classification with Mahalanobis distance demonstrate a wide variation of TPR for different test subjects. As you can see, Subject 2 and Subject 3 have lower TPR compared to Subject 3 and 4. Also, FPR for Subject 1 is higher compared to the rest of the subjects.

4.3.2.2 Neural Networks

Table 4.14 and Figure 4.19 show the overall accuracy of the Neural Networks classifier for inter-subject classification. Table 4.15 and Figure 4.20 show the TPR for different test subjects using the Neural Networks classifier. Also, Table 4.16 and Figure 4.21 show the FPR for different test subjects using the Neural Networks classifier.

Table 4.14 Inter-subject Classification for 4 subjects using Neural Networks - Total Accuracy

Task	Min	Max	Average
Calculation	73.33	85.00	80.17
Breathing	55.00	63.33	59.33
Singing	66.67	85.00	73.17

Inter-subject classification using Neural Networks show more consistency with TPR and FPR compared to Mahalanobis Distance. Also, the TPR and classification accuracy for calculation task is higher compared to breathing and song tasks.

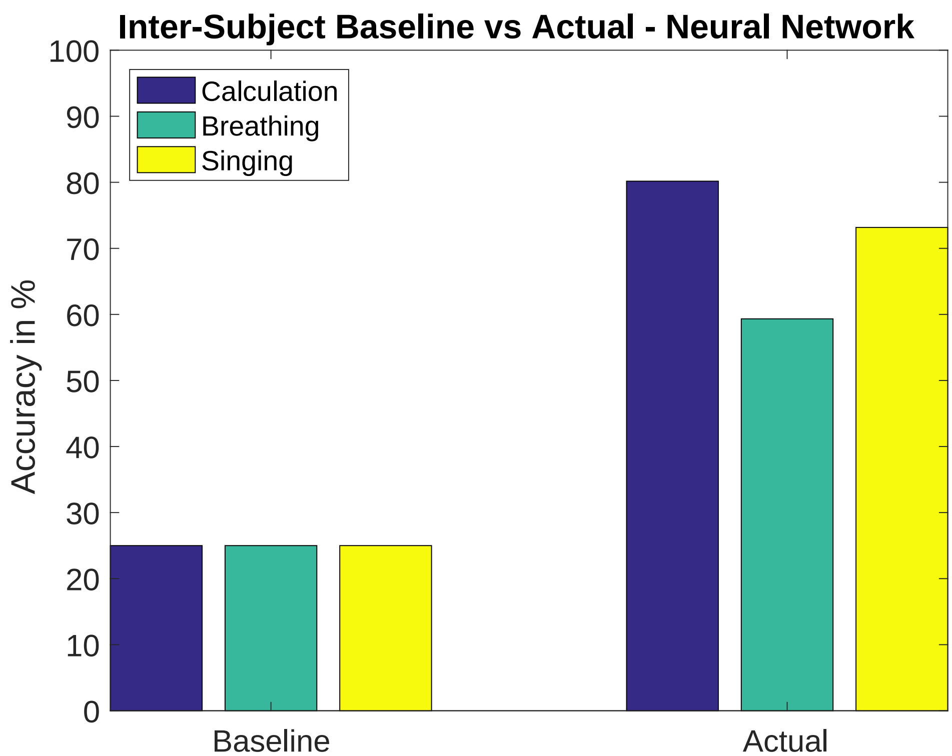

Figure 4.19 Total accuracy for Inter-subject classification using Neural Networks

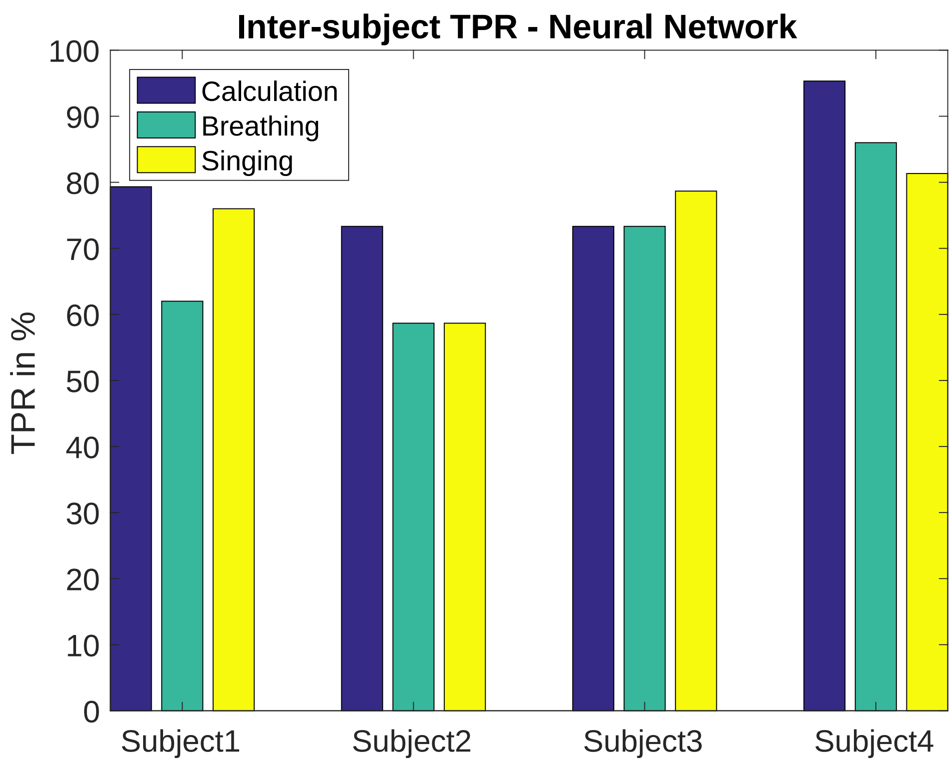

Figure 4.20 TPR for Inter-subject classification using Neural Networks

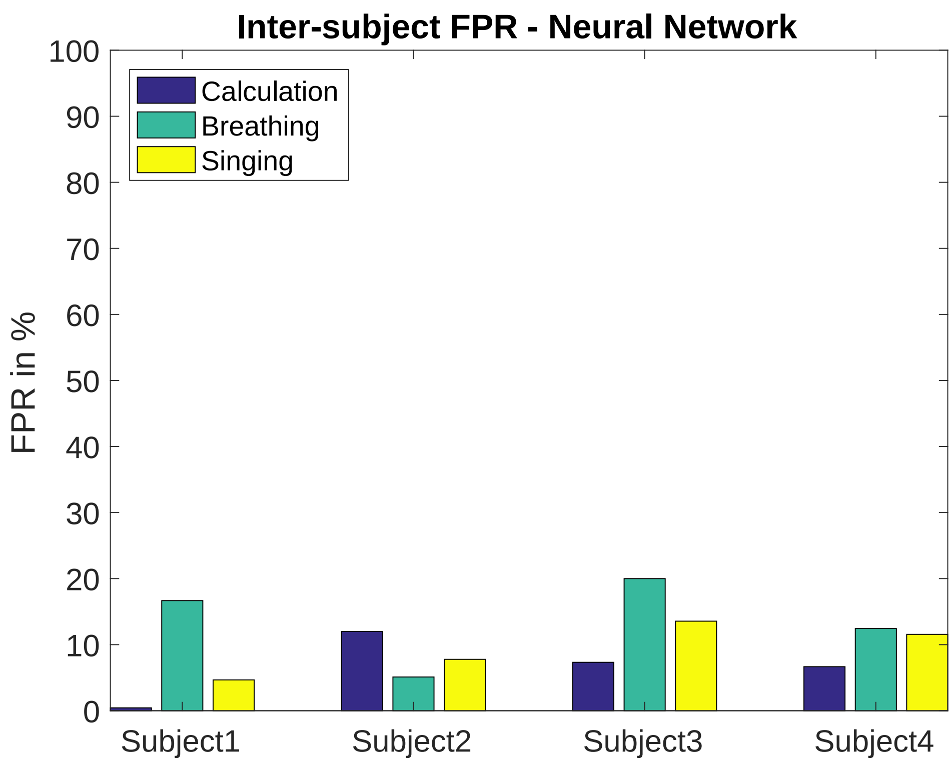

Figure 4.21 FPR for Inter-subject classification using Neural Networks

Table 4.15 Inter-subject Classification for 4 subjects using Neural Networks - TPR for Subject 1-4

(a) Subject 1

Task	Min	Max	Average
Calculation	66.67	86.67	79.33
Breathing	46.67	73.33	62.00
Singing	66.67	80.00	76.00

(b) Subject 2

Task	Min	Max	Average
Calculation	66.67	93.33	73.33
Breathing	26.67	80.00	58.67
Singing	26.67	80.00	58.67

(c) Subject 3

Task	Min	Max	Average
Calculation	60.00	80.00	73.33
Breathing	60.00	86.67	73.33
Singing	66.67	93.33	78.67

(d) Subject 4

Task	Min	Max	Average
Calculation	86.67	100.00	95.33
Breathing	73.33	100.00	86.00
Singing	66.67	100.00	81.33

Table 4.16 Inter-subject Classification for 4 subjects using Neural Networks - FPR for Subject 1-4

(a) Subject 1

Task	Min	Max	Average
Calculation	0.00	2.22	0.44
Breathing	8.89	22.22	16.67
Singing	2.22	8.89	4.67

(b) Subject 2

Task	Min	Max	Average
Calculation	4.44	15.56	12.00
Breathing	0.00	15.56	5.11
Singing	2.22	13.33	7.78

(c) Subject 3

Task	Min	Max	Average
Calculation	0.00	13.33	7.33
Breathing	13.33	24.44	20.00
Singing	8.89	17.78	13.56

(d) Subject 4

Task	Min	Max	Average
Calculation	0.00	11.11	6.67
Breathing	6.67	22.22	12.44
Singing	6.67	17.78	11.56

4.3.2.3 Support Vector Machines

Table 4.17 and Figure 4.22 show the overall accuracy of the SVM classifier for inter-subject classification. Table 4.18 and Figure 4.23 show the TPR for different test subjects using the SVM classifier. Also, Table 4.19 and Figure 4.24 show the FPR for different test subjects using the SVM classifier.

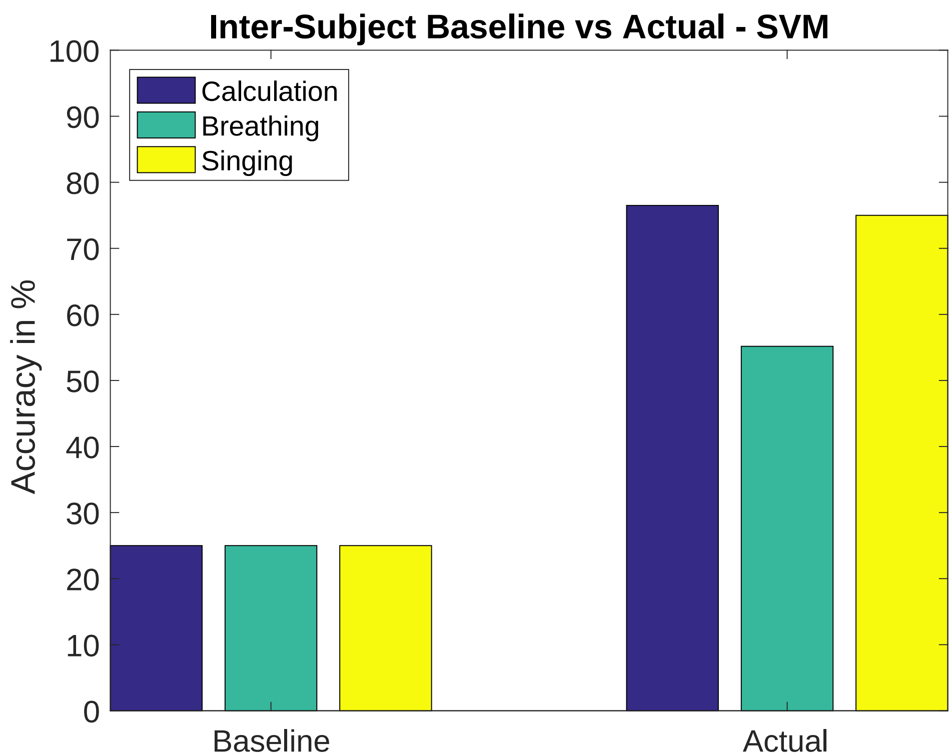

Figure 4.22 Total accuracy for Inter-subject classification using Support Vector Machines

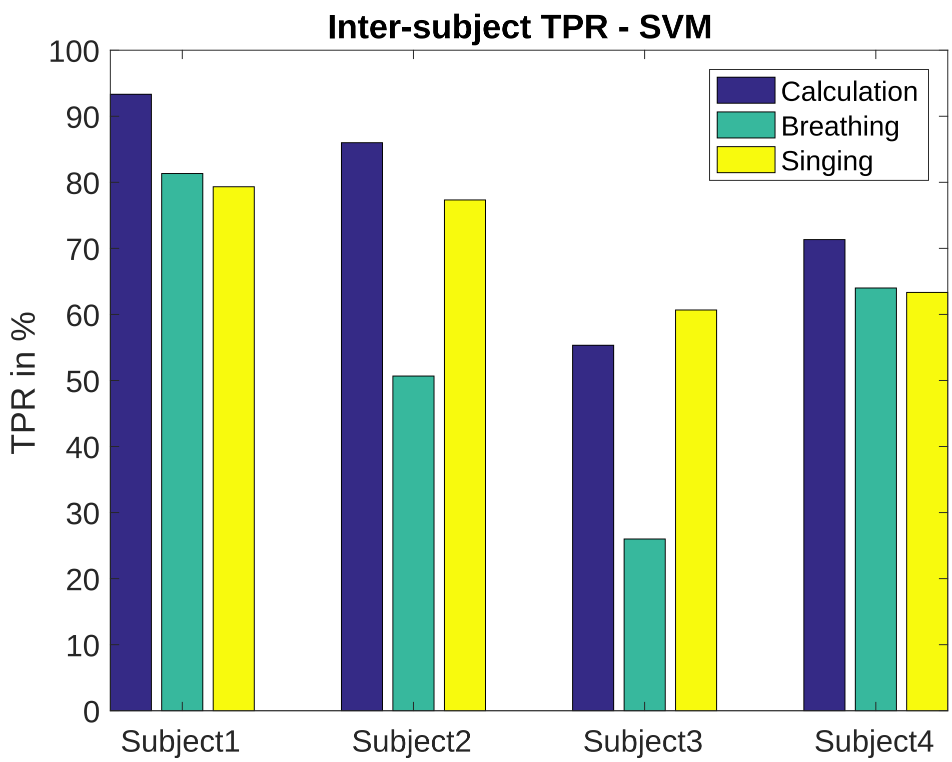

Figure 4.23 TPR for Inter-subject classification using Support Vector Machines

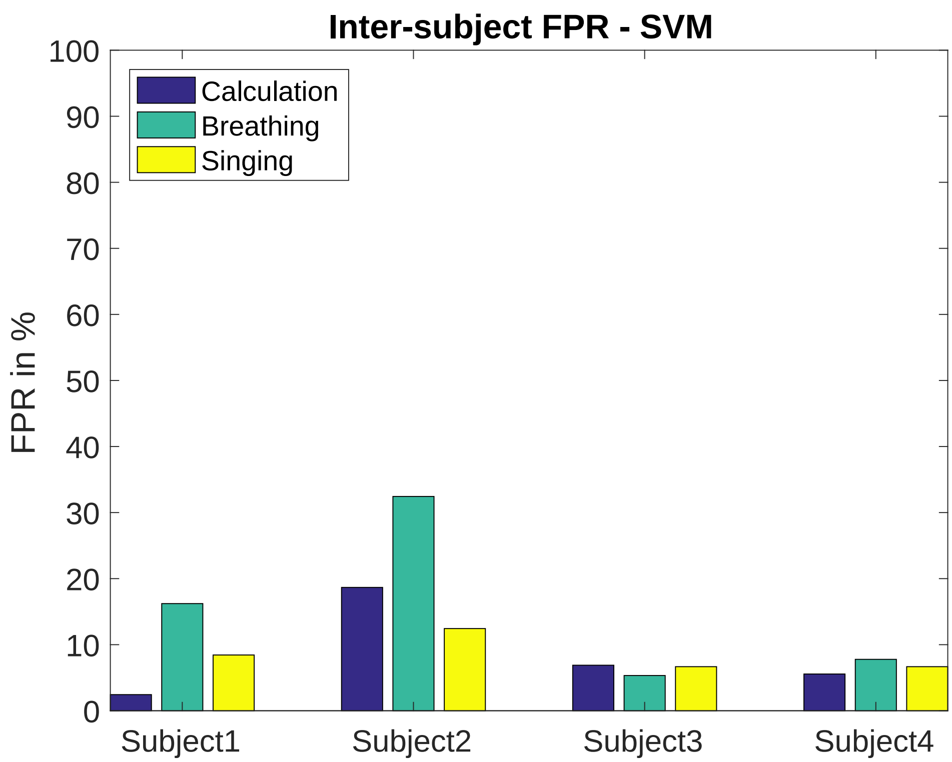

Figure 4.24 FPR for Inter-subject classification using Support Vector Machines

Table 4.17 Inter-subject Classification for 4 subjects using Support Vector Machines - Total Accuracy

Task	Min	Max	Average
Calculation	68.33	80.00	76.50
Breathing	50.00	60.00	55.17
Singing	66.67	85.00	75.00

Table 4.18 Inter-subject Classification for 4 subjects using Support Vector Machines - TPR for Subject 1

(a) Subject 1

Task	Min	Max	Average
Calculation	86.67	100.00	93.33
Breathing	66.67	93.33	81.33
Singing	60.00	100.00	79.33

(b) Subject 2

Task	Min	Max	Average
Calculation	80.00	100.00	86.00
Breathing	20.00	80.00	50.67
Singing	66.67	86.67	77.33

(c) Subject 3

Task	Min	Max	Average
Calculation	40.00	73.33	55.33
Breathing	13.33	33.33	26.00
Singing	40.00	73.33	60.67

(d) Subject 4

Task	Min	Max	Average
Calculation	46.67	93.33	71.33
Breathing	46.67	80.00	64.00
Singing	33.33	86.67	63.33

Table 4.19 Inter-subject Classification for 4 subjects using Support Vector Machines - FPR for Subject 1

(a) Subject 1

Task	Min	Max	Average
Calculation	0.00	6.67	2.44
Breathing	8.89	24.44	16.22
Singing	2.22	17.78	8.44

(b) Subject 2

Task	Min	Max	Average
Calculation	11.11	31.11	18.67
Breathing	24.44	40.00	32.44
Singing	8.89	22.22	12.44

(c) Subject 3

Task	Min	Max	Average
Calculation	2.22	8.89	6.89
Breathing	0.00	13.33	5.33
Singing	2.22	11.11	6.67

(d) Subject 4

Task	Min	Max	Average
Calculation	0.00	8.89	5.56
Breathing	4.44	17.78	7.78
Singing	2.22	15.56	6.67

CHAPTER

5

DISCUSSION

5.1 Classifiers

Classification accuracies for both intra-subject and inter-subject classification tests are generally high for Neural Networks and SVMs compared to Mahalanobis Distance. To understand the reason for differing performances of algorithms, we conducted the Henze-Zirkler's Multivariate Normality Test [16] [10]. The Henze-Zirkler test is based on a non-negative functional distance that measures the distance between two distribution functions. If the data is multivariate normal, the test statistic HZ is approximately lognormally distributed. We calculate the mean, variance and smoothness parameter. Then, the mean and the variance are lognormalized and the p-value is estimated [10]. The detailed description of this test can be found in [17]. If the p-value is greater than certain threshold, the distribution is normal. We found that EEG feature vectors from the data collected failed the Henze-Zirkler's Multivariate Normality Test.

5.2 Intra-Subject Vs Inter-Subject

The intra-subject classification accuracies and TPRs are lower compared to inter-subject classification accuracies. This might be due to similarities in the EEG data for a particular subject. This might also be due to the limitation of the single electrode EEG sensor. For this experiments, the MindWave mobile EEG sensor electrode is placed on the forehead and the ground electrode is placed on the tip of the ear. For this reason, the EEG data from other positions of the human brain are not captured resulting in lack of information to effectively distinguish between different EEG data generated by the same subject.

5.3 Tasks

The classification accuracies and TPR for calculation task was found consistently higher compared to breathing task and song task for all the classifiers used. This might be because the EEG signatures in calculation task are more distinguishable compared the EEG signatures in other tasks. Also note that both breathing task and singing task involved concentrating on breathing and the singing respectively while calculation task involved actual calculation of two digit multiplication. This shows that certain tasks are easily identifiable compared to others.

5.4 Number of classes

It was also found that the classification accuracies drop as we increase the number of classes in case inter-subject classification. The baseline performance is given by Equation 4.3. We can see from Figure 5.1, Figure 5.2 and Figure 5.3 that the baseline performance decreases with increase in the number of classes. Also, we can see that the classification performance of Mahalanobis Distance, Neural Network and SVM classifiers are better than the baseline performance. Note that there is performance decrease even after using classifiers when we increase the number of classes, however the decrease in performance is less compared to baseline performance.

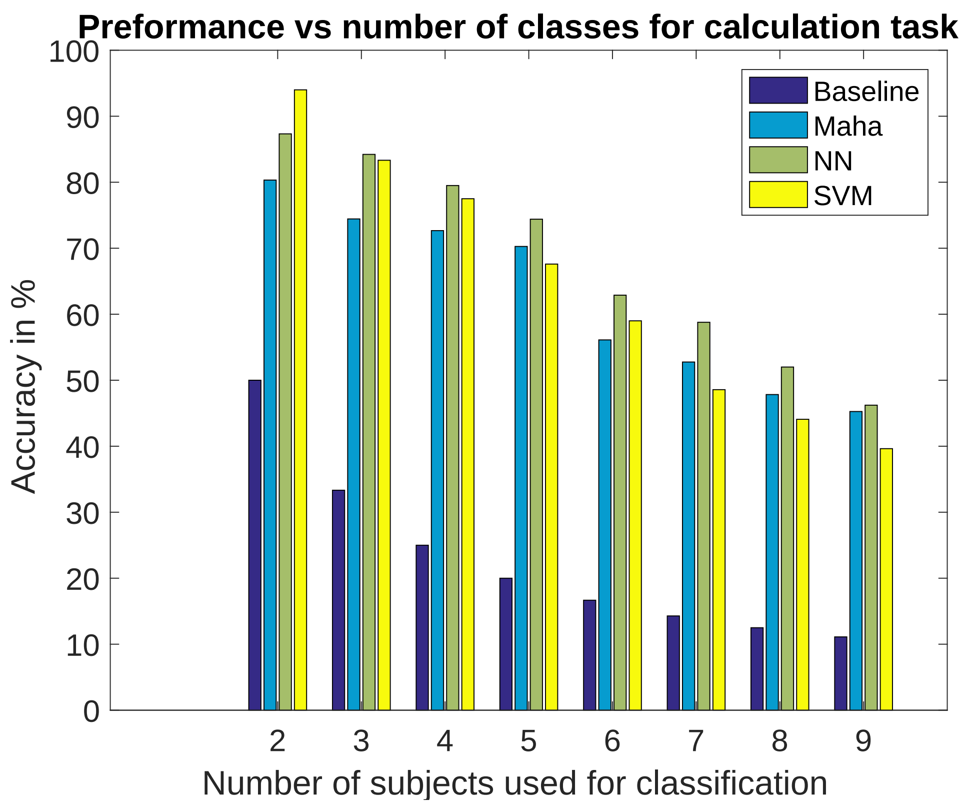

Figure 5.1 Preformance vs number of classes for calculation task

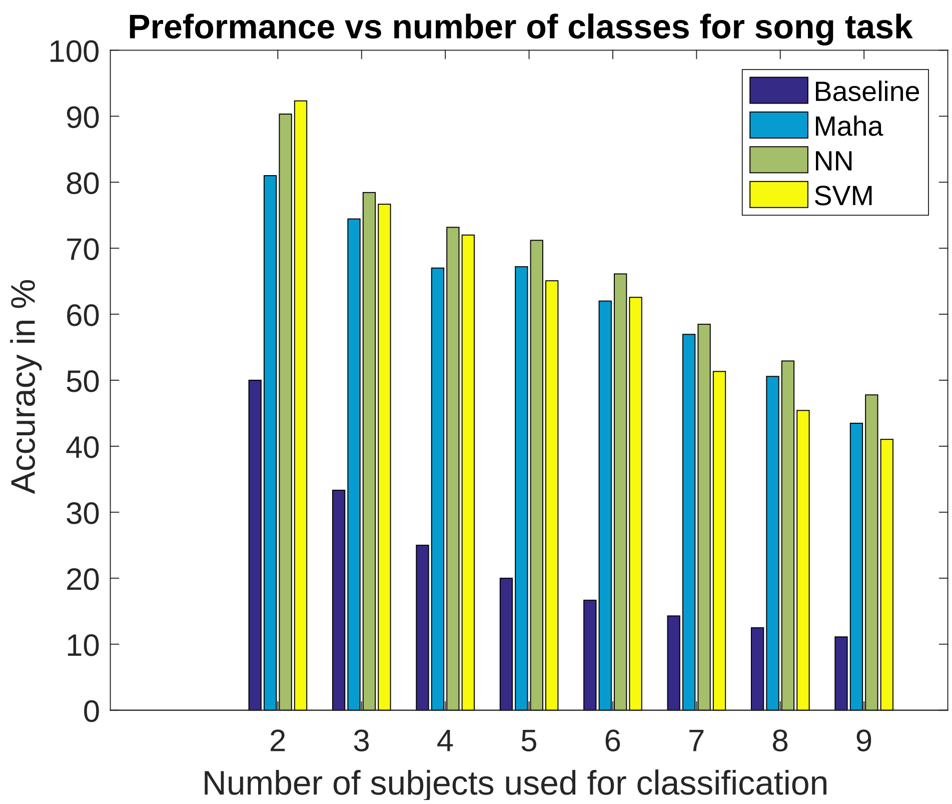

Figure 5.2 Preformance vs number of classes for breathing task

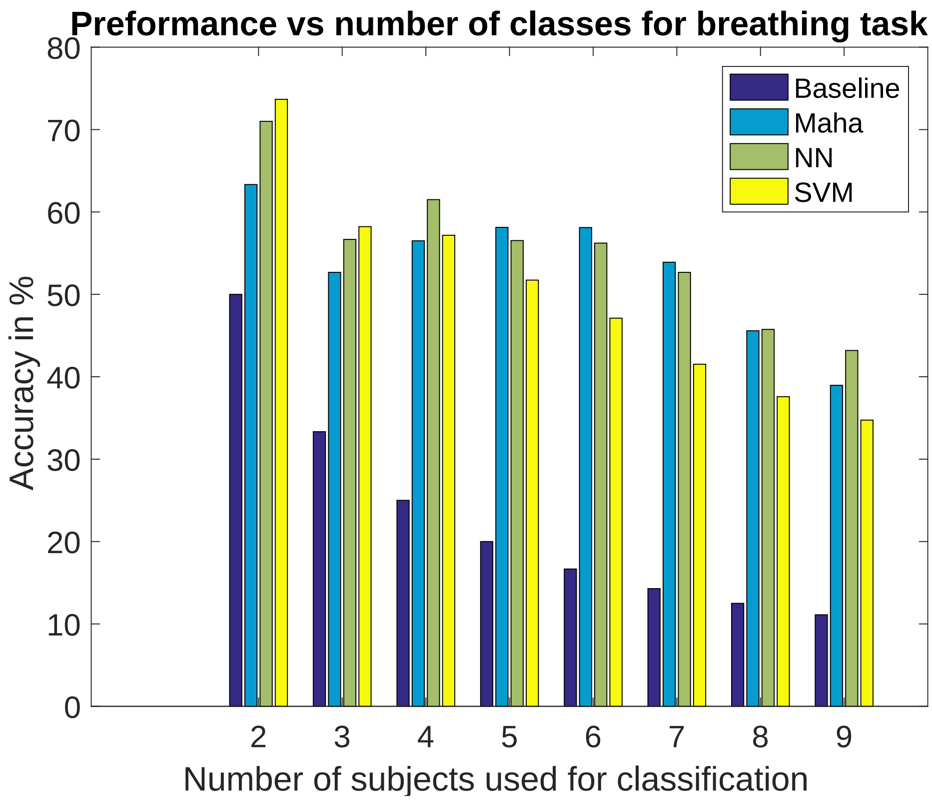

Figure 5.3 Preformance vs number of classes for singing task

BIBLIOGRAPHY

[1] *10-20 System(EEG)*. URL: https://en.wikipedia.org/wiki/10-20_system_(EEG).

[2] *A single Artificial Neuron*. URL: https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png.

[3] Boser, B. E. et al. "A Training Algorithm for Optimal Margin Classifiers". *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: ACM, 1992, pp. 144–152. URL: http://doi.acm.org/10.1145/130385.130401.

[4] Bressler, Y. Z.Y.C.S. L. & Ding, M. "Response preparation and inhibition: The role of the cortical sensorimotor beta rhythm". *Neuroscience* (2008).

[5] Duda, R. et al. *Pattern classification*. Pattern Classification and Scene Analysis: Pattern Classification. Wiley.

[6] *EEG Geodesic Sensor Net*. URL: https://www.egi.com/research-division/geodesic-eeg-system-components/geodesic-sensor-nets.

[7] *Emotive mobile EEG sensor*. URL: http://emotiv.com/epoc/.

[8] Glosser.ca. *A Typical Neural Network*. URL: https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg.

[9] Haykin, S. *Neural Networks and Learning Machines*. Pearson Education, 2009.

[10] Henze, N. & Zirkler, B. "A Class of Invariant Consistent Tests for Multivariate Normality". *Commun. Statist.-Theor. Meth.,* (1990).

[11] H.H.Jasper. "The ten-twenty electrode system of the International Federation." *Physical Review* (1958), pp. 371–375.

[12] Kropotov. "Quantitative EEG, Event-Related Potentials And Neurotherapy" (2009).

[13] McCulloch, W. S. & Pitts, W. "A logical calculus of the ideas immanent in nervous activity". *The bulletin of mathematical biophysics* **5**.4 (1943), pp. 115–133. URL: http://dx.doi.org/10.1007/BF02478259.

[14] *Mindwave EEG sensor*. URL: http://store.neurosky.com/pages/mindwave.

[15] *Multi Layer Perceptron*. URL: https://www.npmjs.com/package/node-neural-network.

[16] *Multivariate normality Test*. URL: http://www.mathworks.com/matlabcentral/fileexchange/17931-hzmvntest.

[17] *Multivariate normality Test*. URL: https://www.researchgate.net/publication/255982280_HZMVNTEST_Henze-Zirkler%27s_Multivariate_Normality_Test.

[18] *Muse EEG sensor*. URL: http://www.choosemuse.com/.

[19] Rosenblatt, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". *Psychological Review* **65(6)** (1958), pp. 65–408.

[20] Snyder, W. & Qi, H. *Machine Vision*. Cambridge University Press, 2010.

[21] Sobotta, J. *Atlas and Text-Book of Human Anatomy: Vascular System, Lymphatic System, Nervous System and Sense Organs*. Nabu Press, 2010.

[22] Therrien, C. W. *Decision Estimation and Classification: An Introduction to Pattern Recognition and Related Topics*. New York, NY, USA: John Wiley & Sons, Inc., 1989.

[23] Werbos, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard Univ., 1974. URL: https://books.google.com/books?id=z81XmgEACAAJ.

APPENDICES

APPENDIX

A

MATLAB CODE

This Appendix includes Matlab code for the EEG security.

Filename : getFeatures.m

```
function [features] = get_features(file_name)
%% This script is used to get the frequency features of the given data stream
% We get the delta, eta, alpha, beta values form frequency domain.
% Delta - 0.1Hz to 3Hz Deep, dreamless sleep, non-REM sleep, unconscious
% Theta - 4Hz to 7Hz Intuitive, creative, recall, fantasy, imaginary, dream
% Alpha - 8Hz to 12Hz Relaxed, but not drowsy, tranquil, conscious
% Low Beta - 13Hz to 17Hz   Formerly SMR, relaxed yet focused, integrated
% High Beta - 18Hz   to 30Hz   Alertness, agitation

STEP_SIZE = 512;
%% Read Data from the file
data = load(file_name);
len = (floor(length(data)/STEP_SIZE)   STEP_SIZE) + 1;
data = data(1: len);
%% Filter Data so that it only has frequency content f < 32Hz
order = 256;
FS = 512;
wc = [0.1 64]/(FS/2);
h = fir1(order, wc);
fil_data = filter(h,1,data);
fil_data = data;

%% Calculating Power spectrun density for each frequency bin
k = 1;
DELTA = 1;
THETA = 2;
L_ALPHA = 3;
H_ALPHA = 4;
L_BETA = 5;
H_BETA = 6;
L_GAMMA = 7;
H_GAMMA = 8;

FFT_LEN_MUL = 1;
DELTA_START = 1 + 1;
DELTA_END = 3;
THETA_START = 4;
THETA_END = 7;
LOW_ALPHA_START = 8;
LOW_ALPHA_END = 9;
HIGH_ALPHA_START = 10;
HIGH_ALPHA_END = 12;
LOW_BETA_START = 13;
LOW_BETA_END = 17;
HIGH_BETA_START = 18;
```

```
HIGH_BETA_END = 30;
LOW_GAMMA_START = 31;
LOW_GAMMA_END = 40;
HIGH_GAMMA_START = 41;
HIGH_GAMMA_END = 48;

delta_range = DELTA_START: (DELTA_END  FFT_LEN_MUL) + 1;
theta_range = ((THETA_START  FFT_LEN_MUL) + 1) : ((THETA_END  FFT_LEN_MUL) + 1);
l_alpha_range = ((LOW_ALPHA_START  FFT_LEN_MUL) + 1) : ((LOW_ALPHA_END  FFT_LEN_MUL) + 1);
h_alpha_range = ((HIGH_ALPHA_START  FFT_LEN_MUL) + 1) : ((HIGH_ALPHA_END  FFT_LEN_MUL) + 1);
l_beta_range = ((LOW_BETA_START  FFT_LEN_MUL) + 1) : ((LOW_BETA_END  FFT_LEN_MUL) + 1);
h_beta_range = ((HIGH_BETA_START  FFT_LEN_MUL) + 1) : ((HIGH_BETA_END  FFT_LEN_MUL) + 1);
l_gamma_range = ((LOW_GAMMA_START  FFT_LEN_MUL) + 1) : ((LOW_GAMMA_END  FFT_LEN_MUL) + 1);
h_gamma_range = ((HIGH_GAMMA_START  FFT_LEN_MUL) + 1) : ((HIGH_GAMMA_END  FFT_LEN_MUL) + 1);


features = zeros(int16(length(data)/STEP_SIZE), 8);
for i = 1 : STEP_SIZE : length(fil_data) − 1
  fil_data_fft= abs(fft(fil_data(i:i+STEP_SIZE),STEP_SIZE  FFT_LEN_MUL));
  features(k,DELTA) = (sum(fil_data_fft(delta_range) . fil_data_fft(delta_range)))/(STEP_SIZE  FFT_LEN_MUL);
  features(k,THETA) = (sum(fil_data_fft(theta_range) . fil_data_fft(theta_range)))/(STEP_SIZE  FFT_LEN_MUL);
  features(k,L_ALPHA) = (sum(fil_data_fft(l_alpha_range) . fil_data_fft(l_alpha_range)))/(STEP_SIZE  FFT_LEN_MUL);
  features(k,H_ALPHA) = (sum(fil_data_fft(h_alpha_range) . fil_data_fft(h_alpha_range)))/(STEP_SIZE  FFT_LEN_MUL);
  features(k,L_BETA) = (sum(fil_data_fft(l_beta_range) . fil_data_fft(l_beta_range)))/(STEP_SIZE  FFT_LEN_MUL);
  features(k,H_BETA) = (sum(fil_data_fft(h_beta_range) . fil_data_fft(h_beta_range)))/(STEP_SIZE  FFT_LEN_MUL);
  features(k,L_GAMMA) = (sum(fil_data_fft(l_gamma_range) . fil_data_fft(l_gamma_range)))/(STEP_SIZE  FFT_LEN_MUL);
  features(k,H_GAMMA) = (sum(fil_data_fft(h_gamma_range) . fil_data_fft(h_gamma_range)))/(STEP_SIZE  FFT_LEN_MUL);
  k = k + 1;
end
```

Filename : prepareData.m

```
function [xTrain, xTest, yTrain, yTest] = prepareData(file_name, ...
  testCases, normalize, divideRatio)
% Takes in filename, number of test cases with that filename,
% and a flag to wheather to normalize of not as input and
% returns feature vector with all the data with that filename
% combined


for i = 1:testCases
  name = strcat(file_name, int2str(i), '.dat');
  if 1 == i
    features = getFeatures(name);
  else
    features = [features ; getFeatures(name)];
  end
end

if 1 == normalize
  mag =  sqrt(sum(abs(features).^2,2));
  data = bsxfun(@rdivide, features, mag);
else
  data = features;
end

ranLoc = randperm(size(data, 1));
data = data(ranLoc,:);
y = ones(size(data, 1), 1);

if (divideRatio == 1.0)
  xTrain = data;
  xTest = data;
  yTrain = y;
  yTest = y;
else
  totalLength = size(data, 1);
  trainLength = floor(totalLength   divideRatio);
  xTrain = data(1:trainLength, :);
  xTest = data(trainLength + 1: end, :);
  yTrain = y(1:trainLength);
  yTest = y(trainLength + 1: end);
end
end
```

Filename : shuffleData.m

```
function [xTrain, xTest, yTrain, yTest] = shuffleData(ixTrain, ixTest, iyTrain,
    iyTest)

ranLoc = randperm(size(ixTrain, 1));
xTrain = ixTrain(ranLoc,:);
yTrain = iyTrain(ranLoc);

ranLoc = randperm(size(ixTest, 1));
xTest = ixTest(ranLoc,:);
yTest = iyTest(ranLoc);
end
```

Filename : performance.m

```
function [TPR, FPR ] = performance( pred, y , class)
TP = 0;
FN = 0;
FP = 0;
TN = 0;
for i = 1:length(pred)
  if((pred(i) == class) && (y(i) == class))
    TP = TP + 1;

  elseif((pred(i) ~= class) && (y(i) == class))
    FN = FN + 1;
  elseif((pred(i) == class) && (y(i) ~= class))
    FP = FP + 1;
  elseif((pred(i) ~= class) && (y(i) ~= class))
    TN = TN + 1;
  end
end

TPR = (TP   100)/(TP+FN);
FPR = (FP   100)/(FP+TN);
end
```

Filename : interTests.m

```
clear; clc;

path = '/Users/pbm/Google Drive/THESIS/DATA/People_data/';
type = 'song';
num_of_sub = 4;
num_of_test_cases = 5;
num_of_it = 10;
maha_accuracy = zeros(num_of_it,1);
nn_accuracy = zeros(num_of_it, 1);
svm_accuracy = zeros(num_of_it, 1);
class = 4;
divide_ratio = 0.7;

for i = 1:num_of_it

  [am, bm, cm] = mahaInter(path, type, num_of_sub, num_of_test_cases, class,
    divide_ratio);
  maha_accuracy(i) = am;
  maha_TPR(i) = bm;
  maha_FPR(i) = cm;
  [am, bm, cm] = nnInter(path, type, num_of_sub, num_of_test_cases, class,
    divide_ratio);
  nn_accuracy(i) = am;
  nn_TPR(i) = bm;
  nn_FPR(i) = cm;
  [am, bm, cm] = svmInter(path, type, num_of_sub, num_of_test_cases, class,
    divide_ratio);
  svm_accuracy(i) = am;
  svm_TPR(i) = bm;
  svm_FPR(i) = cm;
  fprintf('Iteration %d\n', i);
end

maha_accuracy_min = min(maha_accuracy);
maha_accuracy_max = max(maha_accuracy);
maha_accuracy_avg = mean(maha_accuracy);

maha_TPR_min = min(maha_TPR);
maha_TPR_max = max(maha_TPR);
maha_TPR_avg = mean(maha_TPR);

maha_FPR_min = min(maha_FPR);
maha_FPR_max = max(maha_FPR);
maha_FPR_avg = mean(maha_FPR);

```

```

nn_accuracy_min = min(nn_accuracy);
nn_accuracy_max = max(nn_accuracy);
nn_accuracy_avg = mean(nn_accuracy);

nn_TPR_min = min(nn_TPR);
nn_TPR_max = max(nn_TPR);
nn_TPR_avg = mean(nn_TPR);

nn_FPR_min = min(nn_FPR);
nn_FPR_max = max(nn_FPR);
nn_FPR_avg = mean(nn_FPR);

svm_accuracy_min = min(svm_accuracy);
svm_accuracy_max = max(svm_accuracy);
svm_accuracy_avg = mean(svm_accuracy);

svm_TPR_min = min(svm_TPR);
svm_TPR_max = max(svm_TPR);
svm_TPR_avg = mean(svm_TPR);

svm_FPR_min = min(svm_FPR);
svm_FPR_max = max(svm_FPR);
svm_FPR_avg = mean(svm_FPR);

fprintf('&%.2f &%.2f &%.2f \n', maha_accuracy_min, maha_accuracy_max, maha_accuracy_avg);
fprintf('&%.2f &%.2f &%.2f \n', maha_TPR_min, maha_TPR_max, maha_TPR_avg);
fprintf('&%.2f &%.2f &%.2f \n', maha_FPR_min, maha_FPR_max, maha_FPR_avg);
fprintf('-----------------\n');
fprintf('&%.2f &%.2f &%.2f \n', nn_accuracy_min, nn_accuracy_max, nn_accuracy_avg);
fprintf('&%.2f &%.2f &%.2f \n', nn_TPR_min, nn_TPR_max, nn_TPR_avg);
fprintf('&%.2f &%.2f &%.2f \n', nn_FPR_min, nn_FPR_max, nn_FPR_avg);
fprintf('-----------------\n');
fprintf('&%.2f &%.2f &%.2f \n', svm_accuracy_min, svm_accuracy_max, svm_accuracy_avg);
fprintf('&%.2f &%.2f &%.2f \n', svm_TPR_min, svm_TPR_max, svm_TPR_avg);
fprintf('&%.2f &%.2f &%.2f \n', svm_FPR_min, svm_FPR_max, svm_FPR_avg);
```

Filename : intraTests.m

```
clear; clc;

path = '/Users/pbm/Google Drive/THESIS/DATA/People_data/';
sub = '4';
type = {'calc', 'breath', 'song'};
num_of_type = length(type);
num_of_test_cases = 5;
num_of_it = 10;
maha_accuracy = zeros(num_of_it,1);
nn_accuracy =  zeros(num_of_it, 1);
svm_accuracy =  zeros(num_of_it, 1);
class = 3;
divide_ratio = 0.7;

for i = 1:num_of_it
  [am, bm, cm] = mahaIntra(path, sub, type, num_of_type, num_of_test_cases,
    class, divide_ratio);
  maha_accuracy(i) = am;
  maha_TPR(i) = bm;
  maha_FPR(i) = cm;
  [am, bm, cm] = nnIntra(path, sub, type, num_of_type, num_of_test_cases, class,
     divide_ratio);
  nn_accuracy(i) = am;
  nn_TPR(i) = bm;
  nn_FPR(i) = cm;
  [am, bm, cm] = svmIntra(path, sub, type, num_of_type, num_of_test_cases, class
    , divide_ratio);
  svm_accuracy(i) = am;
  svm_TPR(i) = bm;
  svm_FPR(i) = cm;
  fprintf('Iteration %d\n', i);
end

maha_accuracy_min = min(maha_accuracy);
maha_accuracy_max = max(maha_accuracy);
maha_accuracy_avg = mean(maha_accuracy);

maha_TPR_min = min(maha_TPR);
maha_TPR_max = max(maha_TPR);
maha_TPR_avg = mean(maha_TPR);

maha_FPR_min = min(maha_FPR);
maha_FPR_max = max(maha_FPR);
maha_FPR_avg = mean(maha_FPR);

```

```

nn_accuracy_min = min(nn_accuracy);
nn_accuracy_max = max(nn_accuracy);
nn_accuracy_avg = mean(nn_accuracy);

nn_TPR_min = min(nn_TPR);
nn_TPR_max = max(nn_TPR);
nn_TPR_avg = mean(nn_TPR);

nn_FPR_min = min(nn_FPR);
nn_FPR_max = max(nn_FPR);
nn_FPR_avg = mean(nn_FPR);

svm_accuracy_min = min(svm_accuracy);
svm_accuracy_max = max(svm_accuracy);
svm_accuracy_avg = mean(svm_accuracy);

svm_TPR_min = min(svm_TPR);
svm_TPR_max = max(svm_TPR);
svm_TPR_avg = mean(svm_TPR);

svm_FPR_min = min(svm_FPR);
svm_FPR_max = max(svm_FPR);
svm_FPR_avg = mean(svm_FPR);

fprintf('%s &%.2f &%.2f &%.2f \n', sub, maha_accuracy_min, maha_accuracy_max, maha_accuracy_avg);
fprintf('%s &%.2f &%.2f &%.2f \n', sub, maha_TPR_min, maha_TPR_max, maha_TPR_avg);
fprintf('%s &%.2f &%.2f &%.2f \n', sub, maha_FPR_min, maha_FPR_max, maha_FPR_avg);
fprintf('----------------------------------------\n');
fprintf('%s &%.2f &%.2f &%.2f \n', sub, nn_accuracy_min, nn_accuracy_max, nn_accuracy_avg);
fprintf('%s &%.2f &%.2f &%.2f \n', sub, nn_TPR_min, nn_TPR_max, nn_TPR_avg);
fprintf('%s &%.2f &%.2f &%.2f \n', sub, nn_FPR_min, nn_FPR_max, nn_FPR_avg);
fprintf('----------------------------------------\n');
fprintf('%s &%.2f &%.2f &%.2f \n', sub, svm_accuracy_min, svm_accuracy_max, svm_accuracy_avg);
fprintf('%s &%.2f &%.2f &%.2f \n', sub, svm_TPR_min, svm_TPR_max, svm_TPR_avg);
fprintf('%s &%.2f &%.2f &%.2f \n', sub, svm_FPR_min, svm_FPR_max, svm_FPR_avg);
```

Filename : mahaIntra.m

```
function [accuracy, TPR, FPR] = mahaIntra(path, sub, type, num_of_type, num_of_test_cases, class, divide_ratio)

for i = 1:num_of_type
  filePath = strcat(path, sub, '/', type);

  if 1 == i
    [xTrain, xTest, yTrain, yTest] = prepareData(filePath{i}, num_of_test_cases, 1, divide_ratio);
  else
    [xTrainTemp, xTestTemp, yTrainTemp, yTestTemp] = prepareData(filePath{i}, num_of_test_cases, 1, divide_ratio);
    xTrain = [xTrain ; xTrainTemp];
    xTest = [xTest ; xTestTemp];
    yTrain = [yTrain; i    yTrainTemp];
    yTest = [yTest; i    yTestTemp];
  end
end

[xTrain, xTest, yTrain, yTest] = shuffleData(xTrain, xTest, yTrain, yTest);

for i = 1:num_of_type
  [mu(:,i), Kinv(:,:,i)] = get_maha_features(xTrain, yTrain, i);
end


for i = 1 : num_of_type
  for j = 1 : size(xTest,1)
    distance(i, j) = get_maha_dist(xTest(j,:)', mu(:,i), Kinv(:,:,i));
  end
end

[min_distance, pred] = min(distance);

accuracy = sum(pred' == yTest)/size(xTest,1);
accuracy = accuracy    100;
[TPR,FPR] = performance(pred, yTest, class);
end
```

Filename : mahaInter.m

```
function [accuracy, TPR, FPR] = mahaInter(path, type, num_of_sub, num_of_test_cases, class, divide_ratio)


for i = 1:num_of_sub
  if 1 == i
    [xTrain, xTest, yTrain, yTest] = prepareData(strcat(path, int2str(i), '/', type), num_of_test_cases, 1, divide_ratio);
  else
    [xTrainTemp, xTestTemp, yTrainTemp, yTestTemp] = prepareData(strcat(path, int2str(i), '/', type), num_of_test_cases, 1, divide_ratio);
    xTrain = [xTrain ; xTrainTemp];
    xTest = [xTest ; xTestTemp];
    yTrain = [yTrain; i   yTrainTemp];
    yTest = [yTest; i   yTestTemp];
  end
end

[xTrain, xTest, yTrain, yTest] = shuffleData(xTrain, xTest, yTrain, yTest);

for i = 1:num_of_sub
  [mu(:,i), Kinv(:,:,i)] = get_maha_features(xTrain, yTrain, i);
end




for i = 1 : num_of_sub
  for j = 1 : size(xTest,1)
    distance(i, j) = get_maha_dist(xTest(j,:)', mu(:,i), Kinv(:,:,i));
  end
end

[min_distance, pred] = min(distance);

accuracy = sum(pred' == yTest)/size(xTest,1)   100;
[TPR,FPR] = performance(pred, yTest, class);
end
```

Filename : get_maha_features.m

```
function [mu, Kinv] = get_maha_features(data, y, subId)
% Returns mean and Kinv required for mahalanobis
a = y == subId;
mu = mean(data(a,:),1);
Kinv = inv(cov(data(a,:)));
end
```

Filename : get_maha_dist.m

```
function [distance] = get_maha_sit(data, mu, Kinv)
% data and mu are column vectors
distance = (data - mu)'  Kinv  (data - mu);
end
```

Filename : nnIntra.m

```
function [accuracy, TPR, FPR] = nnIntra(path, sub, type, num_of_type,
    num_of_test_cases, class, divide_ratio)
input_layer_size  = 8;
hidden_layer_size = 8;
num_labels = num_of_type;

for i = 1:num_of_type;
  filePath = strcat(path, sub, '/', type);

  if 1 == i
    [xTrain, xTest, yTrain, yTest] = prepareData(filePath{i}, ...
      num_of_test_cases, 1, divide_ratio);
  else
    [xTrainTemp, xTestTemp, yTrainTemp, yTestTemp] = prepareData(...
      filePath{i}, num_of_test_cases, 1, divide_ratio);
    xTrain = [xTrain ; xTrainTemp];
    xTest = [xTest ; xTestTemp];
    yTrain = [yTrain; i   yTrainTemp];
    yTest = [yTest; i   yTestTemp];
  end
end

[xTrain, xTest, yTrain, yTest] = shuffleData(xTrain, xTest, yTrain, yTest);


initial_Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size);
initial_Theta2 = randInitializeWeights(hidden_layer_size, num_labels);

% Unroll parameters
initial_nn_params = [initial_Theta1(:) ; initial_Theta2(:)];

options = optimset('MaxIter', 200);

%  You should also try different values of lambda
lambda = 0.2;

% Create "short hand" for the cost function to be minimized
costFunction = @(p) nnCostFunction(p, ...
  input_layer_size, ...
  hidden_layer_size, ...
  num_labels, xTrain, yTrain, lambda);

% Now, costFunction is a function that takes in only one argument (the
% neural network parameters)
[nn_params, cost] = fmincg(costFunction, initial_nn_params, options);
```

```

Theta1 = reshape(nn_params(1:hidden_layer_size    (input_layer_size + 1)), ...
    hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size    (input_layer_size + 1))):end), ...
    num_labels, (hidden_layer_size + 1));


pred = predict(Theta1, Theta2, xTest);
accuracy =  mean(double(pred == yTest))    100;
[TPR,FPR] = performance(pred, yTest, class);
end
```

Filename : nnInter.m

```
function [accuracy, TPR, FPR] = nnInter(path, type, num_of_sub, ...
  num_of_test_cases, class, divide_ratio)
input_layer_size  = 8;
hidden_layer_size = 8;
num_labels = num_of_sub;

for i = 1:num_of_sub
  if 1 == i
    [xTrain, xTest, yTrain, yTest] = prepareData(strcat(path, ...
      int2str(i), '/', type), num_of_test_cases, 1, divide_ratio);
  else
    [xTrainTemp, xTestTemp, yTrainTemp, yTestTemp] = prepareData(...
      strcat(path, int2str(i), '/', type), num_of_test_cases, 1, divide_ratio);
    xTrain = [xTrain ; xTrainTemp];
    xTest = [xTest ; xTestTemp];
    yTrain = [yTrain; i   yTrainTemp];
    yTest = [yTest; i   yTestTemp];
  end
end

[xTrain, xTest, yTrain, yTest] = shuffleData(xTrain, xTest, yTrain, yTest);

initial_Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size);
initial_Theta2 = randInitializeWeights(hidden_layer_size, num_labels);
initial_nn_params = [initial_Theta1(:) ; initial_Theta2(:)];
options = optimset('MaxIter', 100);

lambda = 1;

costFunction = @(p) nnCostFunction(p, ...
  input_layer_size, ...
  hidden_layer_size, ...
  num_labels, xTrain, yTrain, lambda);

[nn_params, cost] = fmincg(costFunction, initial_nn_params, options);

Theta1 = reshape(nn_params(1:hidden_layer_size   (input_layer_size + 1)), ...
  hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size   (input_layer_size + 1))):end), ...
  num_labels, (hidden_layer_size + 1));

pred = predict(Theta1, Theta2, xTest);
accuracy =  mean(double(pred == yTest))   100;
```

```
[TPR,FPR] = performance(pred, yTest, class);

end
```

Filename : sigmoid.m

```
function g = sigmoid(z)
%SIGMOID Compute sigmoid functoon
%    J = SIGMOID(z) computes the sigmoid of z.
g = 1.0 ./ (1.0 + exp(-z));
end
```

Filename : sigmoidGradient.m

```
function g = sigmoidGradient(z)
%SIGMOIDGRADIENT returns the gradient of the sigmoid function
%evaluated at z
%   g = SIGMOIDGRADIENT(z) computes the gradient of the sigmoid function
%   evaluated at z. This should work regardless if z is a matrix or a
%   vector. In particular, if z is a vector or matrix, you should return
%   the gradient for each element.

g = zeros(size(z));
g = sigmoid(z) .  (1 - sigmoid(z));

end
```

Filename : randInitializeWeights.m

```
function W = randInitializeWeights(L_in, L_out)
%RANDINITIALIZEWEIGHTS Randomly initialize the weights of a layer with L_in
%incoming connections and L_out outgoing connections
%   W = RANDINITIALIZEWEIGHTS(L_in, L_out) randomly initializes the weights
%   of a layer with L_in incoming connections and L_out outgoing
%   connections.
%
%   Note that W should be set to a matrix of size(L_out, 1 + L_in) as
%   the column row of W handles the "bias" terms
%

% You need to return the following variables correctly
W = zeros(L_out, 1 + L_in);

% Randomly initialize the weights to small values
epsilon_init = 1.12;
W = (rand(L_out, 1 + L_in)   2   epsilon_init) - epsilon_init;
end
```

Filename : nnCostFunction.m

```
function [J, grad] = nnCostFunction(nn_params, ...
  input_layer_size, ...
  hidden_layer_size, ...
  num_labels, ...
  X, y, lambda)
%NNCOSTFUNCTION Implements the neural network cost function for a two layer
%neural network which performs classification
%   [J grad] = NNCOSTFUNCTON(nn_params, hidden_layer_size, num_labels, ...
%   X, y, lambda) computes the cost and gradient of the neural network. The
%   parameters for the neural network are "unrolled" into the vector
%   nn_params and need to be converted back into the weight matrices.
%
%   The returned parameter grad should be a "unrolled" vector of the
%   partial derivatives of the neural network.
%
Theta1 = reshape(nn_params(1:hidden_layer_size   (input_layer_size + 1)), ...
  hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size   (input_layer_size + 1))):end), ...
  num_labels, (hidden_layer_size + 1));

% Setup some useful variables
m = size(X, 1);
X = [ones(m,1) X];
% You need to return the following variables correctly
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));
yk_base = (1:num_labels)';

for i = 1:m
  z2 = Theta1   X(i,:)';
  a2 = sigmoid(z2);
  a2 = [1 ;a2];
  z3 = Theta2   a2;
  a3 = sigmoid(z3);
  yk = yk_base == y(i);
  J = J + ((-1   yk'   log(a3)) - ((1 - yk)'   log(1 - a3)));

  delta3 = a3 - yk;
  delta2 = Theta2'   delta3 .  sigmoidGradient([1;z2]);
  delta2 = delta2(2:end);
  Theta2_grad = Theta2_grad + delta3   a2';
  Theta1_grad = Theta1_grad + delta2   X(i,:);
```

```
end
J = J/m;

J = J + ((lambda/(2   m))   (sum(sum(Theta1(:,2:end).^2)) + sum(sum(Theta2(:,2:end).^2))));
Theta1_grad = (1/m)   Theta1_grad;
Theta2_grad = (1/m)   Theta2_grad;
Theta1_grad(:,2:end) = Theta1_grad(:,2:end) + (lambda/m)   Theta1(:,2:end);
Theta2_grad(:,2:end) = Theta2_grad(:,2:end) + (lambda/m)   Theta2(:,2:end);
grad = [Theta1_grad(:) ; Theta2_grad(:)];
end
```

Filename : predict.m

```
function p = predict(Theta1, Theta2, X)
%PREDICT Predict the label of an input given a trained neural network
%   p = PREDICT(Theta1, Theta2, X) outputs the predicted label of X given the
%   trained weights of a neural network (Theta1, Theta2)

m = size(X, 1);
num_labels = size(Theta2, 1);
p = zeros(size(X, 1), 1);

h1 = sigmoid([ones(m, 1) X]   Theta1');
h2 = sigmoid([ones(m, 1) h1]   Theta2');
[dummy, p] = max(h2, [], 2);
end
```

Filename : fmincg.m

```
function [X, fX, i] = fmincg(f, X, options, P1, P2, P3, P4, P5)
% Minimize a continuous differentialble multivariate function. Starting point
% is given by "X" (D by 1), and the function named in the string "f", must
% return a function value and a vector of partial derivatives. The Polack-
% Ribiere flavour of conjugate gradients is used to compute search directions,
% and a line search using quadratic and cubic polynomial approximations and the
% Wolfe-Powell stopping criteria is used together with the slope ratio method
% for guessing initial step sizes. Additionally a bunch of checks are made to
% make sure that exploration is taking place and that extrapolation will not
% be unboundedly large. The "length" gives the length of the run: if it is
% positive, it gives the maximum number of line searches, if negative its
% absolute gives the maximum allowed number of function evaluations. You can
% (optionally) give "length" a second component, which will indicate the
% reduction in function value to be expected in the first line-search (defaults
% to 1.0). The function returns when either its length is up, or if no further
% progress can be made (ie, we are at a minimum, or so close that due to
% numerical problems, we cannot get any closer). If the function terminates
% within a few iterations, it could be an indication that the function value
% and derivatives are not consistent (ie, there may be a bug in the
% implementation of your "f" function). The function returns the found
% solution "X", a vector of function values "fX" indicating the progress made
% and "i" the number of iterations (line searches or function evaluations,
% depending on the sign of "length") used.
%
% Usage: [X, fX, i] = fmincg(f, X, options, P1, P2, P3, P4, P5)
%
% See also: checkgrad
%
% Copyright (C) 2001 and 2002 by Carl Edward Rasmussen. Date 2002-02-13
%
%
% (C) Copyright 1999, 2000 & 2001, Carl Edward Rasmussen
%
% Permission is granted for anyone to copy, use, or modify these
% programs and accompanying documents for purposes of research or
% education, provided this copyright notice is retained, and note is
% made of any changes that have been made.
%
% These programs and documents are distributed without any warranty,
% express or implied.  As the programs were written for research
% purposes only, they have not been tested to the degree that would be
% advisable in any important application.  All use of these programs is
% entirely at the user's own risk.
%
% [ml-class] Changes Made:
```

```
% 1) Function name and argument specifications
% 2) Output display
%

% Read options
if exist('options', 'var') && ~isempty(options) && isfield(options, 'MaxIter')
  length = options.MaxIter;
else
  length = 100;
end


RHO = 0.01;
SIG = 0.5;
INT = 0.1;
EXT = 3.0;
MAX = 20;
RATIO = 100;

argstr = ['feval(f, X'];
for i = 1:(nargin - 3)
  argstr = [argstr, ',P', int2str(i)];
end
argstr = [argstr, ')'];

if max(size(length)) == 2, red=length(2); length=length(1); else red=1; end
S=['Iteration '];

i = 0;
ls_failed = 0;
fX = [];
[f1 df1] = eval(argstr);
i = i + (length<0);
s = -df1;
d1 = -s' s;
z1 = red/(1-d1);

while i < abs(length)
  i = i + (length>0);

  X0 = X; f0 = f1; df0 = df1;
  X = X + z1 s;
  [f2 df2] = eval(argstr);
  i = i + (length<0);
  d2 = df2' s;
  f3 = f1; d3 = d1; z3 = -z1;
```

```
if length>0, M = MAX; else M = min(MAX, -length-i); end
success = 0; limit = -1;
while 1
  while ((f2 > f1+z1 RHO d1) || (d2 > -SIG d1)) && (M > 0)
    limit = z1;
    if f2 > f1
      z2 = z3 - (0.5 d3 z3 z3)/(d3 z3+f2-f3);
    else
      A = 6 (f2-f3)/z3+3 (d2+d3);
      B = 3 (f3-f2)-z3 (d3+2 d2);
      z2 = (sqrt(B B-A d2 z3 z3)-B)/A;
    end
    if isnan(z2) || isinf(z2)
      z2 = z3/2;
    end
    z2 = max(min(z2, INT z3),(1-INT) z3);
    z1 = z1 + z2;
    X = X + z2 s;
    [f2 df2] = eval(argstr);
    M = M - 1; i = i + (length<0);
    d2 = df2' s;
    z3 = z3-z2;
  end
  if f2 > f1+z1 RHO d1 || d2 > -SIG d1
    break;
  elseif d2 > SIG d1
    success = 1; break;
  elseif M == 0
    break;
  end
  A = 6 (f2-f3)/z3+3 (d2+d3);
  B = 3 (f3-f2)-z3 (d3+2 d2);
  z2 = -d2 z3 z3/(B+sqrt(B B-A d2 z3 z3));
  if ~isreal(z2) || isnan(z2) || isinf(z2) || z2 < 0
    if limit < -0.5
      z2 = z1 (EXT-1);
    else
      z2 = (limit-z1)/2;
    end
  elseif (limit > -0.5) && (z2+z1 > limit)
    z2 = (limit-z1)/2;
  elseif (limit < -0.5) && (z2+z1 > z1 EXT)
    z2 = z1 (EXT-1.0);
  elseif z2 < -z3 INT
    z2 = -z3 INT;
  elseif (limit > -0.5) && (z2 < (limit-z1) (1.0-INT))
```

```
      z2 = (limit-z1) (1.0-INT);
    end
    f3 = f2; d3 = d2; z3 = -z2;
    z1 = z1 + z2; X = X + z2 s;
    [f2 df2] = eval(argstr);
    M = M - 1; i = i + (length<0);
    d2 = df2' s;
  end

  if success
    f1 = f2; fX = [fX' f1]';
    s = (df2' df2-df1' df2)/(df1' df1) s - df2;
    tmp = df1; df1 = df2; df2 = tmp;
    d2 = df1' s;
    if d2 > 0
      s = -df1;
      d2 = -s' s;
    end
    z1 = z1  min(RATIO, d1/(d2-realmin));
    d1 = d2;
    ls_failed = 0;
  else
    X = X0; f1 = f0; df1 = df0;
    if ls_failed || i > abs(length)
      break;
    end
    tmp = df1; df1 = df2; df2 = tmp;
    s = -df1;
    d1 = -s' s;
    z1 = 1/(1-d1);
    ls_failed = 1;
  end
  if exist('OCTAVE_VERSION')
    fflush(stdout);
  end
end
```

APPENDIX

B

PYTHON CODE

This Appendix includes Python code for the EEG security.

Filename : main.py

```
import numpy as np
from pre_process import prepare_data , encode_features , decode_features
from tests import intra_sub_tests , inter_sub_tests , verify
import mindwave, time
from pre_process import get_features , normalize


def on_raw(headset , raw):
  #print on_raw.count
  global done
  global count
  global data
  if 0 == count:
    print time.time()
  elif (512    5) >= count:
    count += 1
    return
  elif (512  15 + 2) <= count:
    #print count
    done = 1
  else:
    data.append(raw)
  count += 1

if __name__ == '__main__':
  global done
  global count
  global data
  done = 0
  count = 0
  data = []

  feature_filename = 'features.dat'
  base_filename = '/Users/pbm/Google Drive/THESIS/DATA/People_data/'
  num_sub = 4
  type = ['calc', 'breath', 'song']
  test_cases = 5
  normalize_flag = True
  encode_features(base_filename , feature_filename , num_sub, type, test_cases ,
    normalize_flag)
  features = decode_features(feature_filename)
  #intra_sub_tests(1, features)
  #inter_sub_tests('song', features)

  headset = mindwave.Headset('/dev/tty.MindWaveMobile-DevA')
```

```
time.sleep(2)

headset.connect()
print "Connecting..."
time.sleep(2)
headset.raw_value_handlers.append(on_raw)
while True:
  time.sleep(0.1)
  if 1 == done:
    break
print len(data)

feature_data = get_features(data)
feature_data = normalize(feature_data)
print np.shape(np.array(feature_data))
sub_id = 3
verify('breath', features, feature_data, sub_id)
```

Filename : pre_process.py

```python
import math
import numpy as np
import pickle
from scipy.fftpack import fft


def encode_features(base_filename, feature_filename, num_sub, type, test_cases, normalize_flag):
  features = []
  for i in range(1,num_sub + 1):
    features_per_sub = []
    for item in type:
      filename = base_filename + str(i) + '/' + item
      features_per_sub.append(prepare_data(filename, test_cases, normalize_flag))
    features.append(features_per_sub)

  with open(feature_filename, 'wb') as f:
      pickle.dump(features, f)

def decode_features(feature_filename):
  with open(feature_filename, 'rb') as f:
    features = pickle.load(f)
  return features

def prepare_data(filename, test_cases, normalize_flag):
  """
    Takes in filename, number of test cases associated with the filename,
    and a flag to whether to normalize or not as input and
    returns feature vector with all the data associated with that filename
    combined
  """
  features = []
  for i in range(1, test_cases + 1):
    temp_filename = filename + str(i) + '.dat'
    fp = open(temp_filename, 'r')
    raw_str = fp.readlines()
    raw = []
    raw = [float(i) for i in raw_str ]
    features.extend(get_features(raw))

  if normalize:
    features = normalize(features)

  return features
```

```

def normalize(features):
  """
    Normalizes the feature vectors to make them unit vectors
  """
  np_features = np.array(features)

  features_norm = []
  for i in range(0, np_features.shape[0]):
    features_norm.append(np_features[i,:]/np.sqrt(np.sum(np_features[i,:]
    np_features[i,:])))

  return features_norm


def get_features(raw):
  """
    get_features()
    Used to get the frequency features of the given data stream
    We get the delta, eta, alpha, beta values form frequency domain.
    Delta           0.1Hz  to 3Hz  Deep, dreamless sleep, non-REM sleep,
    unconscious
    Theta           4Hz    to 7Hz    Intuitive, creative, recall, fantasy,
    imaginary, dream
    Alpha           8Hz    to 12Hz   Relaxed, but not drowsy, tranquil,
    conscious
    Low Beta        13Hz   to 17Hz   Formerly SMR, relaxed yet focused,
    integrated
    Midrange Beta                    Thinking, aware of self & surroundings
    High Beta       18Hz   to 30Hz   Alertness, agitation
  """

  STEP_SIZE = 512
  length = int(math.floor(len(raw)/STEP_SIZE)   STEP_SIZE)
  raw = raw[0:length]

  data = []
  for i in range(0, len(raw) - 2, STEP_SIZE):
    data.append(raw[i:i+STEP_SIZE])

  data_fft = []
  for item in data:
    data_fft.append(np.absolute(fft(item)))

  np_data_fft = np.array(data_fft)

```

```
# Since the indexing python is from 0
FFT_LEN_MUL = 1
DELTA_START = 1
DELTA_END = 3
THETA_START = 4
THETA_END = 7
LOW_ALPHA_START = 8
LOW_ALPHA_END = 9
HIGH_ALPHA_START = 10
HIGH_ALPHA_END = 12
LOW_BETA_START = 13
LOW_BETA_END = 17
HIGH_BETA_START = 18
HIGH_BETA_END = 30
LOW_GAMMA_START = 31
LOW_GAMMA_END = 40
HIGH_GAMMA_START = 41
HIGH_GAMMA_END = 48

# +1 to the end range because range() does not consider last element
delta_range = range(DELTA_START, (DELTA_END FFT_LEN_MUL + 1))
theta_range = range((THETA_START FFT_LEN_MUL) , (THETA_END FFT_LEN_MUL + 1))
l_alpha_range = range((LOW_ALPHA_START FFT_LEN_MUL) , (LOW_ALPHA_END FFT_LEN_MUL + 1))
h_alpha_range = range((HIGH_ALPHA_START FFT_LEN_MUL) , (HIGH_ALPHA_END FFT_LEN_MUL + 1))
l_beta_range = range((LOW_BETA_START FFT_LEN_MUL) , (LOW_BETA_END FFT_LEN_MUL + 1))
h_beta_range = range((HIGH_BETA_START FFT_LEN_MUL) , (HIGH_BETA_END FFT_LEN_MUL + 1))
l_gamma_range = range((LOW_GAMMA_START FFT_LEN_MUL) , (LOW_GAMMA_END FFT_LEN_MUL + 1))
h_gamma_range = range((HIGH_GAMMA_START FFT_LEN_MUL) , (HIGH_GAMMA_END FFT_LEN_MUL + 1))

features = []
for item in np_data_fft:
    item_sq = np.array(item item)
    delta = np.sum(item_sq[delta_range])/(STEP_SIZE FFT_LEN_MUL)
    theta = np.sum(item_sq[theta_range])/(STEP_SIZE FFT_LEN_MUL)
    l_alpha = np.sum(item_sq[l_alpha_range])/(STEP_SIZE FFT_LEN_MUL)
    h_alpha = np.sum(item_sq[h_alpha_range])/(STEP_SIZE FFT_LEN_MUL)
    l_beta = np.sum(item_sq[l_beta_range])/(STEP_SIZE FFT_LEN_MUL)
    h_beta = np.sum(item_sq[h_beta_range])/(STEP_SIZE FFT_LEN_MUL)
    l_gamma = np.sum(item_sq[l_gamma_range])/(STEP_SIZE FFT_LEN_MUL)
```

```
        h_gamma = np.sum(item_sq[h_gamma_range])/(STEP_SIZE  FFT_LEN_MUL)
        features.append([delta, theta, l_alpha, h_alpha, l_beta, h_beta, l_gamma, h_gamma])

    return features
```

Filename : mindwave.py

```
import select, serial, threading

# Byte codes
CONNECT              = '\xc0'
DISCONNECT           = '\xc1'
AUTOCONNECT          = '\xc2'
SYNC                 = '\xaa'
EXCODE               = '\x55'
POOR_SIGNAL          = '\x02'
ATTENTION            = '\x04'
MEDITATION           = '\x05'
BLINK                = '\x16'
HEADSET_CONNECTED    = '\xd0'
HEADSET_NOT_FOUND    = '\xd1'
HEADSET_DISCONNECTED = '\xd2'
REQUEST_DENIED       = '\xd3'
STANDBY_SCAN         = '\xd4'
RAW_VALUE            = '\x80'

# Status codes
STATUS_CONNECTED     = 'connected'
STATUS_SCANNING      = 'scanning'
STATUS_STANDBY       = 'standby'

class Headset(object):
    """
    A MindWave Headset
    """

    class DongleListener(threading.Thread):
        """
        Serial listener for dongle device.
        """
        def __init__(self, headset, args, kwargs):
            """Set up the listener device."""
            self.headset = headset
            super(Headset.DongleListener, self).__init__( args, kwargs)

        def run(self):
            """Run the listener thread."""
            s = self.headset.dongle

            # Re-apply settings to ensure packet stream
            s.write(DISCONNECT)
            d = s.getSettingsDict()
```

```
        for i in xrange(2):
            d['rtscts'] = not d['rtscts']
            s.applySettingsDict(d)

        while True:
            # Begin listening for packets
            try:
                if s.read() == SYNC and s.read() == SYNC:
                    # Packet found, determine plength
                    while True:
                        plength = ord(s.read())
                        if plength != 170:
                            break
                    if plength > 170:
                        continue

                    # Read in the payload
                    payload = s.read(plength)

                    # Verify its checksum
                    val = sum(ord(b) for b in payload[:-1])
                    val &= 0xff
                    val = ~val & 0xff
                    chksum = ord(s.read())

                    #if val == chksum:
                    if True: # ignore bad checksums
                        self.parse_payload(payload)
            except (select.error, OSError):
                break
            except serial.SerialException:
                s.close()
                break

    def parse_payload(self, payload):
        """Parse the payload to determine an action."""
        while payload:
            # Parse data row
            excode = 0
            try:
                code, payload = payload[0], payload[1:]
            except IndexError:
                pass
            while code == EXCODE:
                # Count excode bytes
                excode += 1
```

```
            try:
                code, payload = payload[0], payload[1:]
            except IndexError:
                pass
        if ord(code) < 0x80:
            # This is a single-byte code
            try:
                value, payload = payload[0], payload[1:]
            except IndexError:
                pass
            if code == POOR_SIGNAL:
                # Poor signal
                old_poor_signal = self.headset.poor_signal
                self.headset.poor_signal = ord(value)
                if self.headset.poor_signal > 0:
                    if old_poor_signal == 0:
                        for handler in \
                            self.headset.poor_signal_handlers:
                            handler(self.headset,
                                    self.headset.poor_signal)
                else:
                    if old_poor_signal > 0:
                        for handler in \
                            self.headset.good_signal_handlers:
                            handler(self.headset,
                                    self.headset.poor_signal)
            elif code == ATTENTION:
                # Attention level
                self.headset.attention = ord(value)
                for handler in self.headset.attention_handlers:
                    handler(self.headset, self.headset.attention)
            elif code == MEDITATION:
                # Meditation level
                self.headset.meditation = ord(value)
                for handler in self.headset.meditation_handlers:
                    handler(self.headset, self.headset.meditation)
            elif code == BLINK:
                # Blink strength
                self.headset.blink = ord(value)
                for handler in self.headset.blink_handlers:
                    handler(self.headset, self.headset.blink)
        else:
            # This is a multi-byte code
            try:
                vlength, payload = ord(payload[0]), payload[1:]
            except IndexError:
```

```
                continue
            value, payload = payload[:vlength], payload[vlength:]
            # Multi-byte EEG and Raw Wave codes not included
            # Raw Value added due to Mindset Communications Protocol
            if code == RAW_VALUE:
                raw=ord(value[0]) 256+ord(value[1])
                if (raw>=32768):
                    raw=raw-65536
                #print raw
                self.headset.raw_value = raw
                for handler in self.headset.raw_value_handlers:
                    handler(self.headset, self.headset.raw_value)
            if code == HEADSET_CONNECTED:
                # Headset connect success
                run_handlers = self.headset.status != STATUS_CONNECTED
                self.headset.status = STATUS_CONNECTED
                self.headset.headset_id = value.encode('hex')
                if run_handlers:
                    for handler in \
                        self.headset.headset_connected_handlers:
                        handler(self.headset)
            elif code == HEADSET_NOT_FOUND:
                # Headset not found
                if vlength > 0:
                    not_found_id = value.encode('hex')
                    for handler in \
                        self.headset.headset_notfound_handlers:
                        handler(self.headset, not_found_id)
                else:
                    for handler in \
                        self.headset.headset_notfound_handlers:
                        handler(self.headset, None)
            elif code == HEADSET_DISCONNECTED:
                # Headset disconnected
                headset_id = value.encode('hex')
                for handler in \
                    self.headset.headset_disconnected_handlers:
                    handler(self.headset, headset_id)
            elif code == REQUEST_DENIED:
                # Request denied
                for handler in self.headset.request_denied_handlers:
                    handler(self.headset)
            elif code == STANDBY_SCAN:
                # Standby/Scan mode
                try:
                    byte = ord(value[0])
```

```
                    except IndexError:
                        byte = None
                    if byte:
                        run_handlers = (self.headset.status !=
                                        STATUS_SCANNING)
                        self.headset.status = STATUS_SCANNING
                        if run_handlers:
                            for handler in self.headset.scanning_handlers:
                                handler(self.headset)
                    else:
                        run_handlers = (self.headset.status !=
                                        STATUS_STANDBY)
                        self.headset.status = STATUS_STANDBY
                        if run_handlers:
                            for handler in self.headset.standby_handlers:
                                handler(self.headset)


    def __init__(self, device, headset_id=None, open_serial=True):
        """Initialize the headset."""
        # Initialize headset values
        self.dongle = None
        self.listener = None
        self.device = device
        self.headset_id = headset_id
        self.poor_signal = 255
        self.attention = 0
        self.meditation = 0
        self.blink = 0
        self.raw_value = 0
        self.status = None

        # Create event handler lists
        self.poor_signal_handlers = []
        self.good_signal_handlers = []
        self.attention_handlers = []
        self.meditation_handlers = []
        self.blink_handlers = []
        self.raw_value_handlers = []
        self.headset_connected_handlers = []
        self.headset_notfound_handlers = []
        self.headset_disconnected_handlers = []
        self.request_denied_handlers = []
        self.scanning_handlers = []
        self.standby_handlers = []

```

```
        # Open the socket
        if open_serial:
            self.serial_open()

    def connect(self, headset_id=None):
        """Connect to the specified headset id."""
        if headset_id:
            self.headset_id = headset_id
        else:
            headset_id = self.headset_id
            if not headset_id:
                self.autoconnect()
                return
        self.dongle.write(''.join([CONNECT, headset_id.decode('hex')]))

    def autoconnect(self):
        """Automatically connect device to headset."""
        self.dongle.write(AUTOCONNECT)

    def disconnect(self):
        """Disconnect the device from the headset."""
        self.dongle.write(DISCONNECT)

    def serial_open(self):
        """Open the serial connection and begin listening for data."""
        # Establish serial connection to the dongle
        if not self.dongle or not self.dongle.isOpen():
            self.dongle = serial.Serial(self.device, 115200)

        # Begin listening to the serial device
        if not self.listener or not self.listener.isAlive():
            self.listener = self.DongleListener(self)
            self.listener.daemon = True
            self.listener.start()

    def serial_close(self):
        """Close the serial connection."""
        self.dongle.close()
```

Filename : classifiers.py

```
import numpy as np
from sklearn import svm, grid_search
from sklearn.neighbors import NearestNeighbors
#from sklearn.neural_network import MLPClassifier

def svm_classifier(x_train, y_train, x_test, y_test):
  '''
  clf = svm.SVC()
  clf.fit(x_train, y_train)
  '''
  parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
  svr = svm.SVC(probability=True)
  clf = grid_search.GridSearchCV(svr, parameters)
  clf.fit(x_train, y_train)

  dec = clf.predict_proba(x_test)
  #dec = clf.decision_function(x_test)
  print np.array(dec)
  print np.shape(np.array(dec))
  pred = clf.predict(x_test)
  return pred


def maha_classifier(x_train, y_train, x_test, y_test):
  print lol


def ann_classifier(x_train, y_train, x_test, y_test):
  print lol
  '''
  clf = MLPClassifier(algorithm='l-bfgs', alpha=1e-5, hidden_layer_sizes=(5, 2),
    random_state=1)
  clf.fit(x_train, y_train)
  clf.predict(x_test)
  '''

def k_nn(x_test, y_test, number_of_neighbours):
  nbrs = NearestNeighbors(n_neighbors=number_of_neighbours, algorithm='ball_tree
    ').fit(X)
```

Filename : tests.py

```
import numpy as np
from classifiers import svm_classifier, ann_classifier, maha_classifier

def intra_sub_tests(test_sub, features):
  intra_features = features[test_sub - 1]
  x_train = []
  x_test = []
  y_train = []
  y_test = []
  i = 0
  for test_type in intra_features:
    for case_num in test_type:
      x_train.append(case_num)
      x_test.append(case_num)
      y_train.append(i)
      y_test.append(i)
    i = i + 1

  pred = svm_classifier(x_train, y_train, x_test, y_test)
  np_acc = np.array(np.array(pred) == np.array(y_test))
  print float(np_acc.sum())/float(len(y_test))

def inter_sub_tests(type, features):
  x_train = []
  x_test = []
  y_train = []
  y_test = []

  if 'calc' == type:
    type_num = 0
  elif 'breath' == type:
    type_num = 1
  elif 'song' == type:
    type_num = 2
  else:
    print 'Type Error'
  i = 0
  for sub in features:
    case = sub[type_num]
    for item in case:
      x_train.append(item)
      x_test.append(item)
      y_train.append(i)
      y_test.append(i)
    i = i + 1
```

```
  pred = svm_classifier(x_train, y_train, x_test, y_test)
  np_acc = np.array(np.array(pred) == np.array(y_test))
  print float(np_acc.sum())/float(len(y_test))

def verify(type, features, data, sub_id):
  x_train = []
  x_test = []
  y_train = []
  y_test = []

  if 'calc' == type:
    type_num = 0
  elif 'breath' == type:
    type_num = 1
  elif 'song' == type:
    type_num = 2
  else:
    print 'Type Error'
  i = 0
  for sub in features:
    case = sub[type_num]
    for item in case:
      x_train.append(item)
      y_train.append(i)
    i = i + 1
  x_test = data
  for i in range(0, len(x_test)):
    y_test.append(sub_id)

  pred = svm_classifier(x_train, y_train, x_test, y_test)
  np_acc = np.array(np.array(pred) == np.array(y_test))
  print pred, y_test
  print float(np_acc.sum())/float(len(y_test))
  if .6 < float(np_acc.sum())/float(len(y_test)):
    print 'Verified Used'
  else:
    print 'Verification failed'
```