Classification and Feature Extraction of Different Hand Movements from the EMG Signal using Machine Leaning based Algorithms

A Thesis

SUBMITTED TO THE FACULTY OF THE

UNIVERSITY OF MINNESOTA

BY

Bipasha Kundu

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

ADVISOR: PROFESSOR DESINENI SUBBARAM NAIDU, PHD

AUGUST 2021

Acknowledgement

I wish to express my sincere gratitude to my advisor Prof. Desineni Subbaram Naidu for his constant guidance and support through my MS journey. Throughout the program, he helped me in different ways: by giving advice on research and setting program goals, allowing me to study independent research via online courses, and finally with a wonderful project to implement my knowledge. It was an honor for me to work as one of his Graduate Student.

I would like to thank the Department of Electrical Engineering, University of Minnesota, Duluth for funding me throughout my MS program and providing me the opportunities to work as a Teaching Assistant for different courses.

I want to thank my committee members, Dr. Arshia Khan(Professor, Dept. of Computer Science) and Dr. Peng Fang (Assistant Professor, Dept. of Electrical Engineering) for their service on my Thesis Committee.

Finally, Special thanks to my parents, husband, and Sister for providing me continuous, unparalleled, and the most sincere love, inspiration, and support.

Abstract

Prosthetic plays an important role for the amputees to improve the ability and mobility of their regular activities. Electromyography(EMG) has been used for decades in the control of the motorized upper-limb prosthesis. Processed EMG can imitate human movements. Mayo armband is a wireless sensor of low power, Bluetooth, and small interference which provides a good quality EMG signal. The Myo armband measures the EMG from the upper-limb. In this thesis work, the statistical time-domain features have been considered to classify different hand movements. The classification and comparison has been performed by 4 different Machine Learning based algorithms i.e. Support Vector Machine(SVM), Naïve Bayes(NB), Random Forest(RF), and K-Nearest Neighbor(KNN). The data has been collected from subjects (males and females) of different ages. The classifier model has used 80% data as a training set and the remaining 20% of data as the test set. The result shows that Random Forest and SVM outperforms the other two algorithms with an accuracy of 98%. Referring to the accuracy here, this classification model serves as a promising candidate for the input of prosthetic hand control systems.

Table of contents

List of Tables

List of Figures

Abbreviations

EMG - Electromyography

ML – Machine Learning

KNN - k-nearest neighbors

NB- Naïve Bayes

RF – Random Forest

SVM - Support Vector Machines

AI - Artificial Intelligence

CNN - Convolutional Neural Network

RNN - Recurrent Neural Network

MAV- Mean Absolute Value

RMS- Root Mean Square

SD- Standard Deviation

SSI- Simple Square Integral

Chapter 1

Introduction

1.1 Introduction

Upper-limb amputations are traumatic occurrences for individuals. In the United States, overall, approximately 1.7 million people or approximately 1 of every 200 people are living with a limb loss. According to the National Center for Health Statistics, every year 50,000 new amputation cases are added. Among them, the most common is partial hand amputations with loss of one or more fingers. There are several causes for amputations. The most common causes are poor circulation because of damage or narrowing of the arteries, called peripheral arterial disease. The body's cells cannot get the oxygen and nutrients they need without adequate blood flow. The affected tissue begins to die if they don't get proper oxygen, and several infections may set in. The other causes are severe injuries from road accidents, war, serious burns, fireworks, cancerous tumors in the muscle or bone of the limb, infections that do not get better with treatment, and many more [1].

Human Bio-electric signals are extensively studied and applied in various clinical and psychophysiological studies. The Bioelectrical signal means an electrical signal obtained from any organ that exhibits a physical variable of interest. This signal is commonly a function of time and is definable in terms of its amplitude, frequency, and phase. In the recent past, EMG has found its use in the rehabilitation of patients with amputations in the form of Robotic prostheses. EMG proves to be a valuable tool as it provides a natural way of sensing and classifying different movements of the body [2].

An Electromyography (EMG) signal is a biomedical signal to measure muscle responses or electrical activity produced by skeletal muscles. The nerves control the muscles by electrical signals called an impulse; these impulses can be measured and analyzed with the help of EMG sensors. The attributes (i.e., amplitude and spectrum) of an EMG depend on several factors, including thickness and temperature of the skin, the thickness of the fat between the muscle and the skin, the velocity of the blood flow, and location of the sensors. Factors like fatigue, aging, and neuromuscular diseases degrade muscle performance as well as EMG patterns [3].

There are two types of EMG depending on the type of sensors. One of them is the surface and another one is intramuscular. In surface EMG, non-invasive surface sensors are placed on the

skin to record the electrical activity of the muscles under it. In intramuscular EMG, an invasive sensor (i.e., needle) is introduced into the muscle.

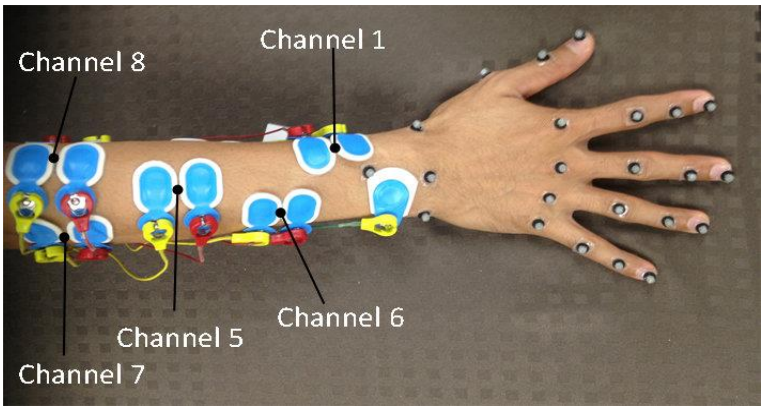


Fig 1.1: The surface EMG electrodes placed on the Hand [18]

1.2 Literature Survey

To a person with an upper limb amputation or congenital defect, a well-functioning prosthesis can open the door to many work and life opportunities. A fundamental component of many modern prostheses is the myoelectric control system, to control prosthetic movements. In recent years, EMG and gesture classification has become a very popular topic. Many researchers have done research on feature extraction, classification of gestures/movements at offline and real-time, comparison of time domain, frequency domain features, and others.

According to [2], five different hand gestures were classified using KNN and dynamic wrapping algorithms. Then accuracy was compared with the accuracy of the MYO system. They found that their classification model (accuracy 86%) performs better than an MYO system (accuracy 83%).

Nikita Anil at [4], used a signal processing technique, wavelet decomposition, where the signals are fragmented into wavelet coefficients that are localized in the time domain, as well as in the frequency domain. These coefficients make up the dataset and are classified using Support Vector Machines (an ML technique).

In [5], feature extraction and classification of EMG signals were performed using Principal Component Analysis (PCA) and uncorrelated linear discriminant analysis (ULDA) for feature reduction purposes and then, applies SVM to recognize different gestures in real-time.

The authors in [6] extracted five eigenvalues in a time domain and applied Neural Network (NN) to classify six gestures. They obtained an accuracy of 93% in their proposed model.

Ahsan et al. (2011)[8] proposed a paper which uses an EMG signal for recognition of gestures using Artificial Neural Network (ANN). It mentions a detailed study about EMG signals and building a human-computer interface to aid aged or disabled people. A dataset of 204 samples was taken and a hidden layer of 10 neurons produced the best result with a success rate of 88.4%

In [10], the author has used the k-nearest neighbor and the dynamic time warping algorithms to classify the hand gestures given by the MYO armband. They included a detector of muscle activity that speeds the time of processing up and improves the accuracy of the recognition. Finally, they acquired an accuracy of 89.5% and proposed that their model outperforms the MYO systems, as well as the other systems.

According to [11], they implemented two modes of hand movements: relax hand and closing hand. They used KNN and SVM classifiers for the recognition by extracting statistical time domain features (mean, variance, kurtosis, and skewness). They finally achieved an accuracy 96.58%.

The availability of good quality signals for the prosthetic hand control system is an important issue in classifying different hand gestures of hand movements. As Processed EMG can imitate human movements, the main concern about EMG signal is the processing. MYO armband is a wireless sensor that was developed by Thalmic Labs in 2014. MYO is an armband that can be worn on the forearm below the elbow, which is controlled by our gestures and movements. It is a sensor of low power, Bluetooth, and small interference, which provides a good quality EMG signal. Many researchers have used MYO armbands to collect and record the EMG signal and later implemented different features from that EMG signal.

1.3 Objectives of the Thesis

Statement of the Problem:

Machine learning has become a hot topic in the recent world. The main objective of this thesis is to work with EMG signals and different Machine Learning Algorithms. Firstly, the EMG signal has been processed by extracting four different time-domain features, i.e. Mean Absolute Value (MAV), Variance (VA), Standard Deviation (SD), and Simple Square Integral (SSI) and then four different Machine Learning Algorithms have been used here to classify five different hand movements to provide as a promising candidate for the input of prosthetic hand control systems and real-time classification of hand movements.

1.4 Overview of the Work

In this work, the EMG signal has been collected and recorded using the MYO armband. The MYO armband is a gesture recognition wireless sensor developed by Thalmic Labs. It is a wearable gesture and motion control device. Five hand movements have been predicted using four different Machine Learning Algorithms, and finally, the accuracy has been compared. The ML Algorithms were feed with processed EMG that was acquired by extracting four time-domain features, i.e. MAV, VA, SD, SSI.

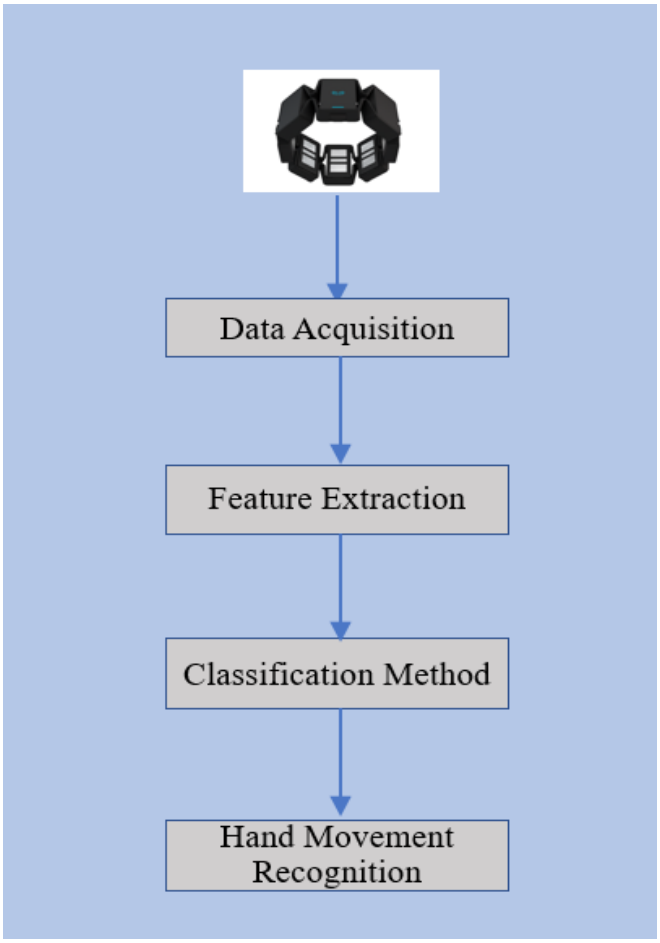


Figure 1.2: Flow chart of the Research

1.5 Chapters Review

Chapter 2 addresses Machine Learning and Artificial Intelligence. It also discusses the classification of different ML algorithms and their applications in real life.

Chapter 3 discusses how the materials (MYO armband) and how the data(EMG signals) were collected using the MYO armband.

Chapter 4 shows different steps of data pre-processing. It addresses how the features were extracted as a part of pre-processing of the data and then proceeded to feed the different Machine Learning Algorithms.

Chapter 5 provides information's about the results and analysis of the study. A comparison and visual analysis of different ML algorithms also have been shown here.

Chapter 6 ends with the Conclusion and the Future work.

A part of this thesis work has been accepted and published in the 3rd International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Kuala Lumpur, Malaysia, on June 12, 2021.

Chapter 2

Machine Learning

This chapter introduces Artificial Intelligence and Machine Learning Algorithms. It discusses the classification of ML Algorithms and their applications.

2.1 Introduction to Machine Learning

The term machine learning was coined in 1959 by Arthur Samuel, an American IBMer and pioneer in the field of computer gaming and artificial intelligence (AI). Machine learning is an area of AI with a concept that a computer program can learn and adapt to new data without human intervention [9]. Figure 2.1 shows how AI and ML are connected to each other.

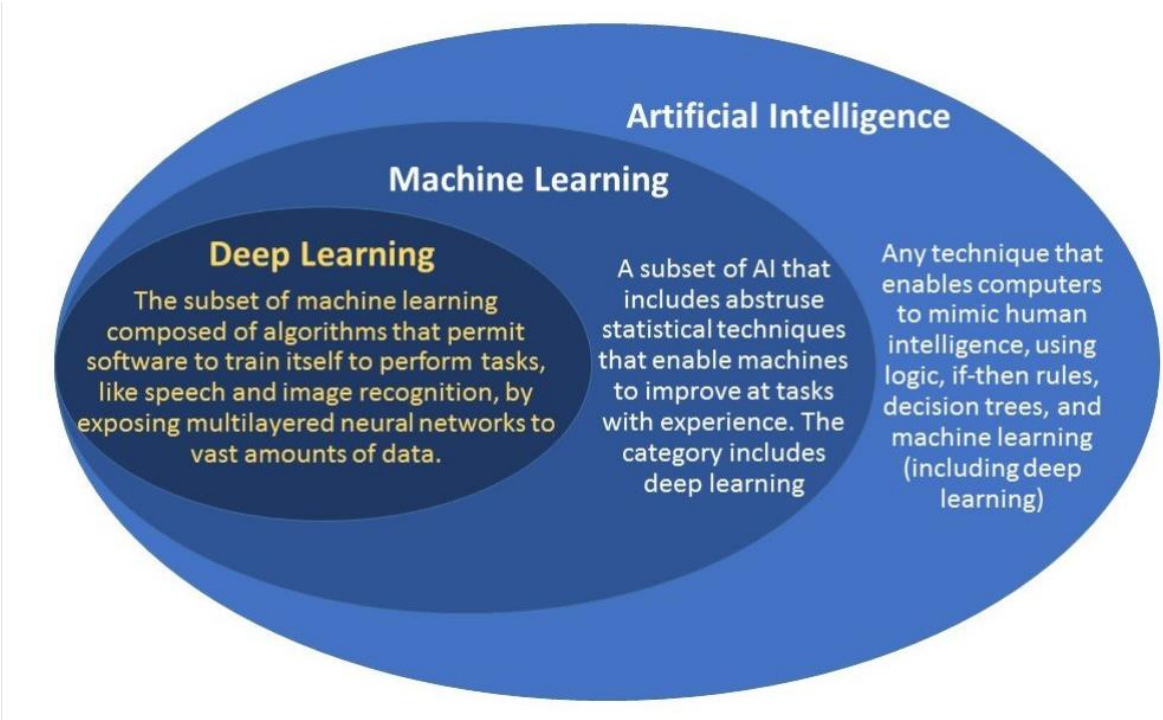


Figure 2.1: Relation between AI, ML and Deep Learning [9]

Machine learning is a subset of AI. The theory is simple: machines take data and 'learn' for themselves. It is currently the most promising tool in the AI pool for businesses. Machine learning systems can quickly apply knowledge and training from large datasets to excel at facial recognition, speech recognition, object recognition, translation, and many other tasks. Machine

learning allows a system to learn to recognize patterns on its own and make predictions, contrary to hand-coding a software program with specific instructions to complete a task.

The types of Machine Learning Algorithms differ in their approach, the type of data they use as input and output, and the type of task or problem that they are intended to solve. Most common and used learning algorithms are :

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

2.1.1 Supervised Learning

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task [9].

2.1.1.1 Examples of Supervised Machine Learning

Supervised Machine Learning Algorithm can be grouped into the following categories:

- **Classification**: Classification algorithms are used when the outputs are restricted to a limited set of values. In classification problems, the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification [10]. Most popular classification problems are Support Vector Machine (SVM), K Nearest Neighbor (KNN), Decision Tree (DT), Naïve Bayes (NB), and Artificial Neural Network (ANN).

- **Regression**: Regression algorithms are used when the outputs may have any numerical value within a range. Here, the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign [10]. Linear and Logistic Regressions are the two regression supervised learning techniques.

Supervised ML is one of the most powerful engines that enable AI systems to make business decisions faster and more accurately than humans.

2.1.2 Unsupervised Learning

Unsupervised learning algorithms take a set of data that contains only inputs and find structure in the data, like grouping or clustering of data points. The algorithms, therefore, learn from test data that has not been labeled, classified or categorized [9]. Unsupervised learning is where you only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning above there are no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data [11].

2.1.2.1 Examples of Unsupervised Learning

Unsupervised learning problems can be further grouped into clustering and association problems.

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior. An example of a clustering problem is K-means Clustering.

- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y. A prior algorithm is popularly known as the Association learning.

2.1.3 Semi -Supervised Learning

Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Some of the training examples are missing training labels, yet many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce a considerable improvement in learning accuracy. A good example is a photo archive where only some of the images are labeled (e.g. dog, cat, person), and the majority are unlabeled [9].

2.1.4 Reinforcement Learning

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Due to its generality, the field is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics, and genetic algorithms. In machine learning, the environment is typically represented as a Markov Decision Process (MDP). Many reinforcement learning algorithms use dynamic programming techniques. Reinforcement learning algorithms do not assume knowledge of an exact mathematical model of the MDP, and are used when exact models are infeasible. Reinforcement learning algorithms are used in autonomous vehicles or in learning to play a game against a human opponent [9].

2.2 Overview of Supervised Learning

Supervised Machine learning algorithms are 2 types: Classification and Regression. The Classification and Regression Techniques are further grouped into several types. A short overview of them has been discussed here.

The most used supervised learning techniques are :

- Linear Regression
- Logistical Regression
- Decision Trees
- Random Forest
- Support Vector Machines (SVM)
- Naive Bayes
- K Nearest Neighbor
- Neural Networks

2.2.1 Linear Regression:

Linear regression is one of the most basic types of regression in machine learning. The linear regression model consists of a predictor variable and a dependent variable related linearly to each other. In case the data involves more than one independent variable, then linear regression is called multiple linear regression models.

The below-given equation is used to denote the linear regression model:

$$Y = mx + c + e \quad \text{(2.1)}$$

where m is the slope of the line, c is an intercept, and e represents the error in the model.

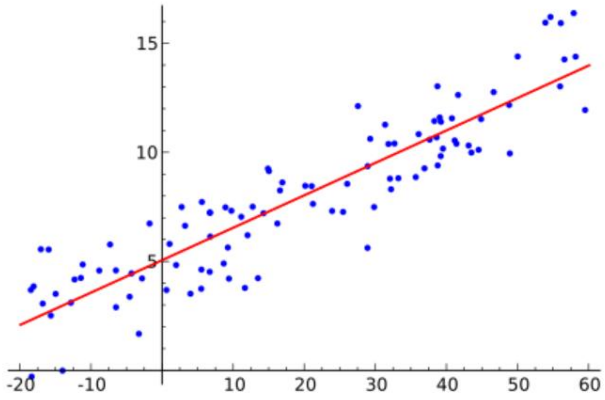


Figure 2.2.1: Linear Regression [20]

The best fit line is determined by varying the values of m and c. The predictor error is the difference between the observed values and the predicted value. The values of m and c get selected in such a way that it gives the minimum predictor error. It is important to note that a simple linear regression model is susceptible to outliers. Therefore, it should not be used in case of big size data [12].

2.2.2 Logistic Regression

Logistic regression is one of the types of regression analysis techniques, which gets used when the dependent variable is discrete, e.g. 0 or 1, true or false, etc. This means the target variable can have only two values, and a sigmoid curve denotes the relation between the target variable and the independent variable.

Logit function is used in Logistic Regression to measure the relationship between the target variable and independent variables. Below is the equation that denotes the logistic regression.

$$\text{logit}(p) = \ln(p/(1-p)) + b0 + b1X1 + b2X2 + b3X3 \ldots + bk.Xk \quad \text{(2.2)}$$

Where, p is the probability of occurrence of the feature.

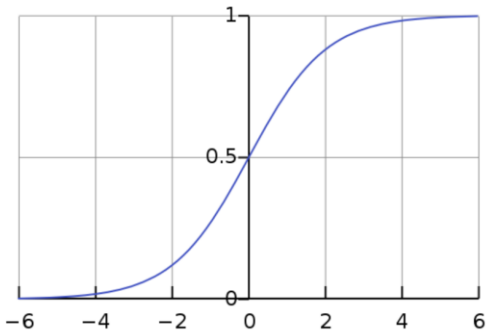


Figure 2.2.2: Logistic Regression

2.2.3 Decision Trees

A Decision Tree (DT) is the most powerful and popular tool for classification and prediction. A Decision Tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. The following figure represents the DT very well [21].

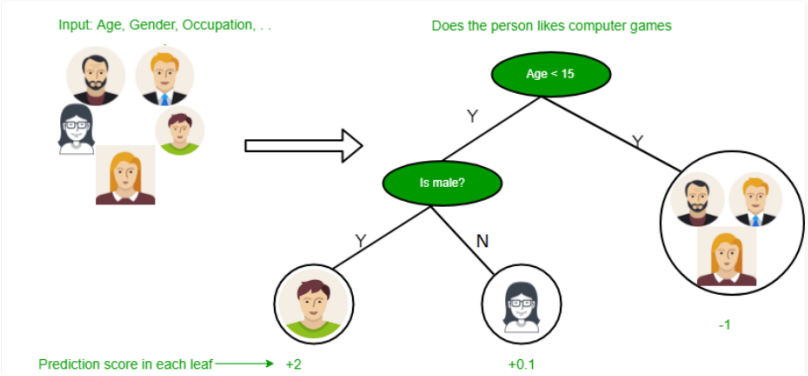


Figure 2.2.3: Example of Decision Tree [21]

Decision Trees are able to generate understandable rules. They also perform classification without requiring much computation and provide a clear indication of which fields are most important for prediction or classification. Despite all of these strengths, Decision Trees can be computationally expensive to train. The process of growing a Decision Tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared [21].

2.2.4 Random Forest

A Random Forest (RF), like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction, and the class with the most votes becomes our model's prediction. The following figure given below perfectly represents the mechanism of RF very well [22].

The fundamental concept behind RF is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is "A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models" [22].

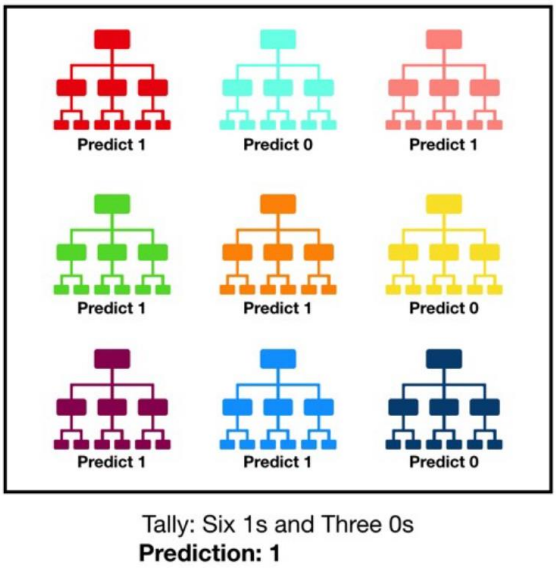


Figure 2.2.4: Visualization of RF making Predictions [22]

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors**.** While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

2.2.5 Support Vector Machine

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence [14].

Figure 2.2.5.1 Possible hyperplanes [14]

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

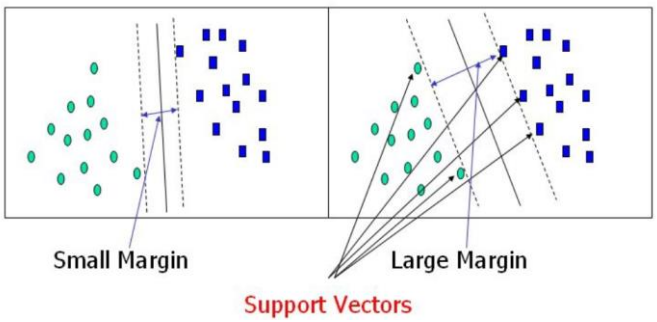


Figure 2.2.4: Support Vector Machine [14]

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the

hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM [14].

2.2.5.1 Kernel Trick Of SVM

If there is a way to map the data from 2-dimensional space to 3-dimensional space, a decision surface that clearly divides between different classes can be found easily. The thought of this data transformation process is to map all the data points to a higher dimension (in this case, 3 dimensions), find the boundary, and make the classifications.

However, when there are more and more dimensions, computations within that space become more and more expensive. This is when the kernel trick comes in. It allows to operate in the original feature space without computing the coordinates of the data in a higher dimensional space [15].

There are different kernels. The most popular ones are:

- Polynomial kernel
- Radial basis function (RBF) kernel

Polynomial kernel:

“Intuitively, the polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these, just like the example above. With n original features and d degrees of polynomial, the polynomial kernel yields n^d expanded features.

$$k\ (x, y) = (x^T + 1)^d \quad \text{.....(2.3)}$$

RBF Kernel:

The RBF kernel is also called the Gaussian kernel. There is an infinite number of dimensions in the feature space because it can be expanded by the Taylor Series. In the format below, The γ parameter defines how much influence a single training example has. The larger it is, the closer other examples must be to be affected.

$$K(x, y) = e^{-}\ \gamma\ ||x - y||^2\ ,\ \gamma\ > 0 \text{.....(2.4)}$$

The kernel trick sounds like a "perfect" plan. However, one critical thing to keep in mind is that when we map data to a higher dimension, there are chances that we may overfit the model. Thus choosing the right kernel function (including the right parameters) and regularization are of great importance.

2.2.6 Naïve Bayes Theorem

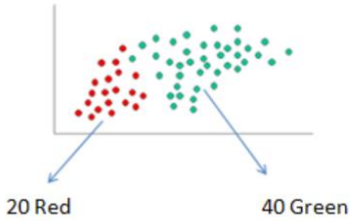
Naïve Bayes (NB) Algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. This is naive because it is (almost) never true. Here is why NB works anyway. It is one of the simplest and most effective Classification Algorithms which helps in building the fast machine learning models that can make quick predictions. It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. For example, if the fruit is identified on the bases of color, shape, and taste, then a red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other. It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

$$P_{(y|x_{1,\ldots\ldots},x_n)} = \frac{(P_y)P_{(x_1,\ldots\ldots,x_n|y)}}{P_{(x_1,\ldots\ldots x_n)}} \text{.....(2.5)}$$

Where y is the class variable and x is the feature vector of the data.

Below is one simple way to explain the Bayes rule. The task is to identify the color of a newly-observed dot.



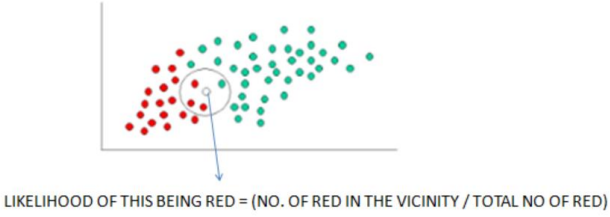
Since there are twice as many GREEN objects as RED, it is reasonable to believe that a new case (which hasn't been observed yet) is twice as likely to have membership with GREEN rather than RED. In the Bayesian analysis, this belief is known as the prior probability. Prior probabilities are based on previous experience, in this case the percentage of GREEN and RED objects, and often used to predict outcomes before they actually happen.

Since there is a total of 60 objects, 40 of which are GREEN and 20 RED, our prior probabilities for class membership can be written as below:

$$\text{Prior Probability of GREEN} = \frac{No.Of\ GREEN\ objects}{Total\ No\ of\ objects} = \frac{40}{60}$$

$$\text{Prior Probability of RED} = \frac{No.Of\ RED\ objects}{Total\ No\ of\ objects} = \frac{20}{60}$$

Having formulated our prior probability, we are now ready to classify a new object (WHITE circle in the diagram below). Since the objects are well clustered, it is reasonable to assume that the more GREEN (or RED) objects in the vicinity of X, the more likely that the new cases belong to that particular color. To measure this likelihood, we draw a circle around X which encompasses a number (to be chosen a priori) of points irrespective of their class labels. Then we calculate the number of points in the circle belonging to each class label. From this we calculate the likelihood**:**



From the illustration above, it is clear that the likelihood of X given GREEN is smaller than Likelihood of X given RED, since the circle encompasses 1 GREEN object and 3 RED ones.

Although the prior probabilities indicate that X may belong to GREEN (given that there are twice as many GREEN compared to RED) the likelihood indicates otherwise; that the class membership of X is RED (given that there are more RED objects in the vicinity of X than GREEN). In the Bayesian analysis, the final classification is produced by combining both sources of information, i.e., the prior and the likelihood, to form a posterior probability using Bayes' rule. Finally, we classify X as RED since its class membership achieves the largest posterior probability [16].

It is a classification technique based on Bayes' theorem with an assumption of independence between predictors.

In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Yes, it is really Naïve.

2.2.7 K-Nearest Neighbor

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

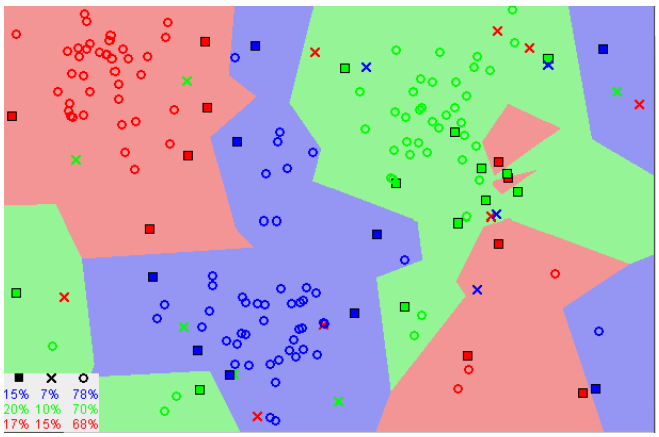


Figure 2.2.7.1: Image showing how similar data points typically exist close to each other [17]

In the following image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice. KNN runs this formula to compute the distance between each data point and the test data. It then finds the probability of these points being similar to the test data and classifies it based on which points share the highest probabilities [17].

To visualize this formula, it would look something like this:

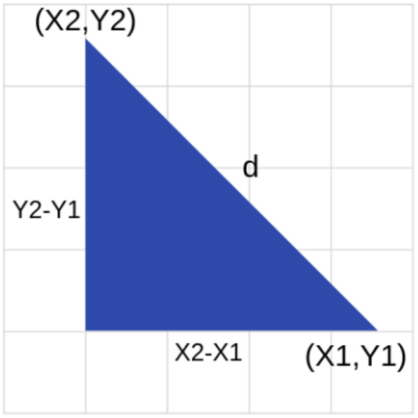


Figure 2.2.7.2: Calculation of Euclidian Distance

The KNN algorithm basically follows the steps given below:

- Load the data
- Initialize K to your chosen number of neighbors
- 3. For each example in the data
    - Calculate the distance between the query example and the current example from the data.
    - Add the distance and the index of the example to an ordered collection
- Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
- Pick the first K entries from the sorted collection

- Get the labels of the selected K entries

- If regression, return the mean of the K labels

- If classification, return the mode of the K labels

Choosing the Value of K

Choosing the proper value of is most important for KNN. To select the K that's right for the data, the KNN algorithm should be run several times with different values of K and choose the K that reduces the number of errors while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

KNN is very easy to use and implement. There's no need to build a model, tune several parameters, or make additional assumptions. The algorithm is versatile. It can be used for classification, regression, and search. The only disadvantage of using this algorithm is that it gets significantly slower as the number of examples and/or predictors/independent variables increase.

2.2.8 Neural Network

Artificial Neural Networks (ANN) is a supervised learning system built of a large number of simple elements, called neurons or perceptron. Each neuron can make simple decisions and feeds those decisions to other neurons, organized in interconnected layers. Together, the neural network can emulate almost any function and answer practically any question, given enough training samples and computing power. A "shallow" neural network has only three layers of neurons:

- An input layer that accepts the independent variables or inputs of the model
- One hidden layer
- An output layer that generates predictions

A Deep Neural Network (DNN) has a similar structure, but it has two or more "hidden layers" of neurons that process inputs. Goodfellow, Bengio and Courville showed that while shallow neural networks are able to tackle complex problems, deep learning networks are more accurate and improve in accuracy as more neuron layers are added. Additional layers are useful up to a

limit of 9-10, after which their predictive power starts to decline. Today most neural network models and implementations use a deep network of between 3-10 neuron layers [18].

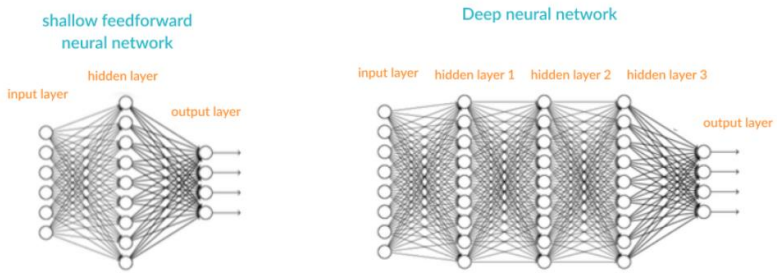


Figure 2.2.8.1: Difference of Shallow vs deep Neural Network [23]

Types of Neural Network:

There are different types of Neural network. Among them 3 are discussed below along with their applications.

Convolution Neural Network:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. An illustration of CNN is provided in the Fig 2.2.8.2.

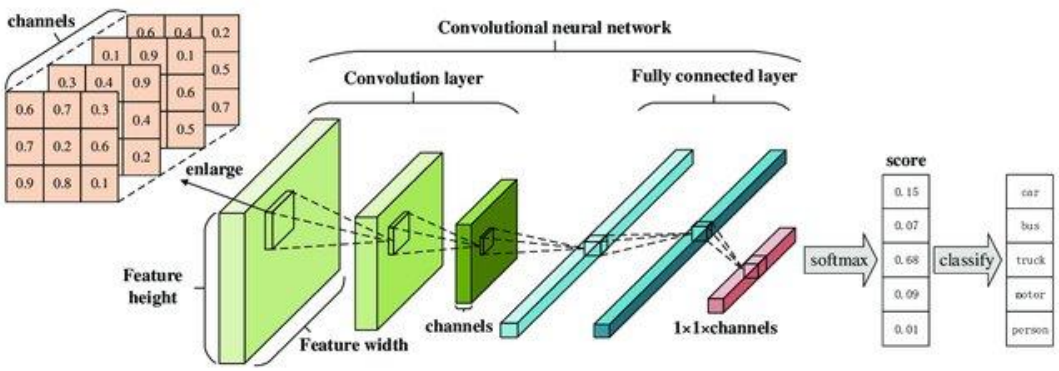


Figure 2.2.8.2: A CNN sequence

Recurrent Neural Network:

Recurrent neural networks (RNN) are the state of the art algorithm for sequential data and are used by Apple's Siri and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. It is one of the algorithms behind the scenes of the amazing achievements seen in deep learning over the past few years. The figure 2.2.8.3 shows the RNN architecture very well [17].

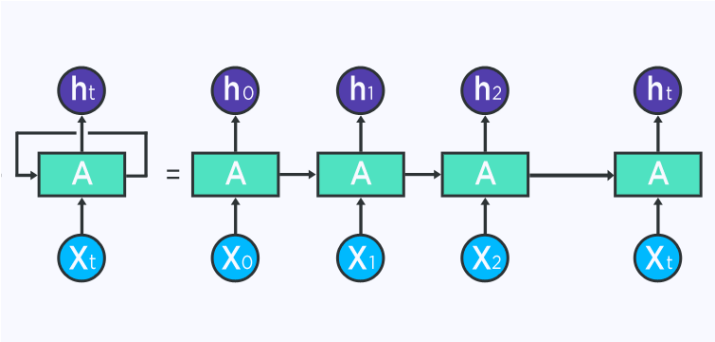


Figure 2.2.8.3: Basic structure of RNN [17]

Autoencoder:

Auto Encoders are an unsupervised machine learning algorithm. The input and output layers are the same. Autoencoder tries to make the hidden layers learn the input to successfully recreate it.

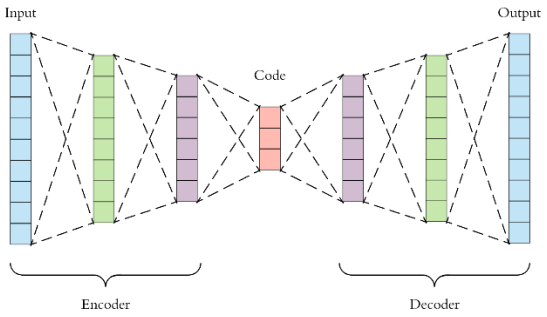


Figure 2.2.8.4: Structure of Autoencoder

Auto Encoders are used for anomaly detection as well as compression purposes. Besides the above mentioned types of neural networks, there are others such as Generative Adversarial Networks, Long Short Term Models, etc.

2.2.9 F- Score

The F-score, also called the F1-score, is a measure of a model's accuracy on a dataset. It is used to evaluate binary classification systems, which classify examples into 'positive' or 'negative'.

The F-score is a way of combining the precision and recall of the model, and it is defined as the harmonic mean of the model's precision and recall.

The F-score is commonly used for evaluating information retrieval systems, such as search engines, and also for many kinds of machine learning models, in particular in natural language processing.

It is possible to adjust the F-score to give more importance to precision over recall, or vice-versa. Common adjusted F-scores are the F0.5-score and the F2-score, as well as the standard F1-score.

The formula for the standard F1-score is the harmonic mean of the precision and recall. A perfect model has an F-score of 1. [give the number of equations]

$$F1 = \frac{2}{\frac{1}{recall} * \frac{1}{precision}}$$

$$= 2 * \frac{precision * recall}{precision + recall}$$

$$= \frac{tp}{tp + \frac{1}{2}(fp + fn)}$$

Where,

Precision is the fraction of true positive examples among the examples that the model classified as positive. In other words, the number of true positives divided by the number of false positives plus true positives.

Recall, also known as sensitivity, is the fraction of examples classified as positive, among the total number of positive examples. In other words, the number of true positives divided by the number of true positives plus false negatives.

tp is the number of true positives classified by the model.

fn is the number of false negatives classified by the model.

fp is the number of false positives classified by the model.

Calculation F-score vs Accuracy

The accuracy is defined as the ratio of correctly classified examples of all cases. So the mathematical definition of accuracy from f1 score is defined as:

$$\text{Accuracy} = \frac{\text{tp+tn+}}{\text{tp+tn+fp+fn}} \quad \text{.....(2.6)}$$

So, F1 is an overall measure of a model's accuracy that combines precision and recall, in that weird way that addition and multiplication just mix two ingredients to make a separate dish altogether. That is, a good F1 score means that you have low false positives and low false negatives, so you're correctly identifying real threats and you are not disturbed by false alarms. An F1 score is considered perfect when it's 1, while the model is a total failure when it's 0.

Chapter 3

Experimentation and Data Collection

3.1 MYO Arm Band

In this work, the data have been collected from the MYO armband. The MYO armband in Fig. 3.1, is a gesture recognition wireless sensor developed by Thalmic Labs. It is a wearable gesture and motion control device that uses a set of 8 sensors, combined with IMU (Inertial Measurement Unit) sensors, including gyroscope, accelerometer, and magnetometer, to recognize gestures.

These 8 sensors acquire the data from the forearm. The MYO armband has a signal frequency of 200Hz. This device requires the user to wear it and synchronize it with the hand movements before it can be used. This process can take a few minutes. Then the data is generated as a Bluetooth packet. MYO transmits the data through a Bluetooth Dongle connected to a PC. The generated packet has two types of data. One is EMG data value, and the other is IMU data values; however, only the EMG data has been used in this work.

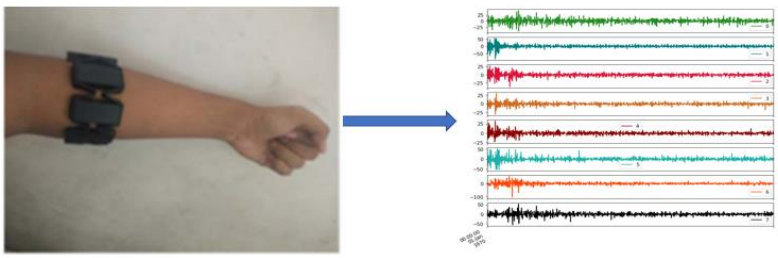


Figure 3.1: The MYO armband worn in Hand and the signal from 8 sensors

The MYO armband's connection with a computer uses the MYO Connect application provided by the Thalmic Labs company, as shown in Fig. 3.2. The connection sequence follows the standard instructions that are available in the application.

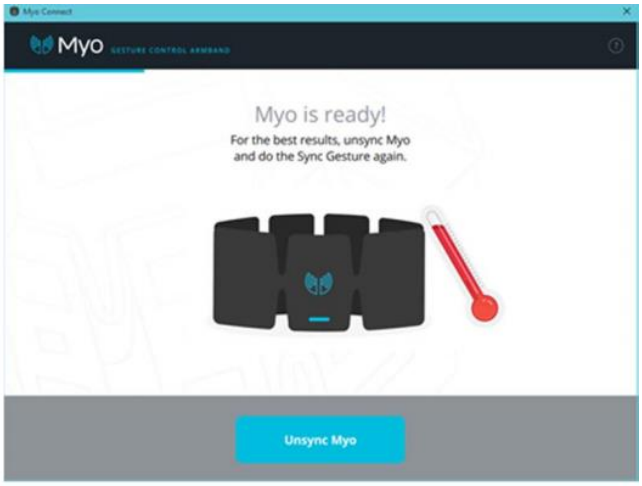


Figure 3.2: Myo Application Window

This application requires synchronization of our hand muscles with the MYO armband that is used so that the EMG signal produced by the hand muscles can be read properly by the EMG sensor that is installed in a circle inside the MYO armband. Some standard movements are used in this synchronization process. The process of displaying the results of EMG signals that are read by EMG sensors online is done using the Simulink program developed in this study.

3.2 Data Collection

Electromyography (EMG) is an experimental technique concerned with the development, recording and analysis of myoelectric signals. Myoelectric signals are formed by physiological variations in the state of muscle fiber membranes. Applications of EMG are mainly seen in Medical Research, Rehabilitation, Ergonomics and Sports Science. Our study focuses on application of EMG for movement analysis related to Sports Science.

The dataset was created by recording EMG signals using MYO armbands from different healthy people. The EMG signal was collected from six abled-body subjects (three females/ three males, ages: 22-30 yrs.), and they had no accident history on their dominant hand. The subjects performed 5 hand movements including thumb flexion (TF), index finger down (ID), middle finger down (MD), ring finger down (RD), little finger down (LD), and each movement was maintained for 5s. The hand movement has been recorded using an acquisition software developed in Python [13].

Figure 3.1: Five Hand Movements

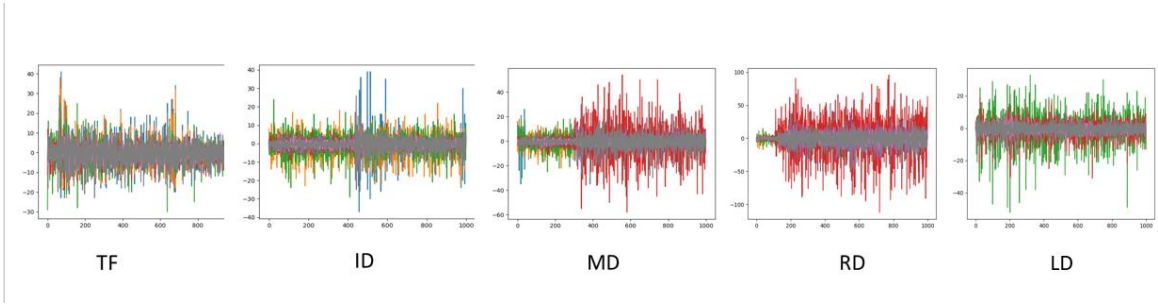


Figure 3.2: EMG signal for all 5 movements from One subject

The dataset was created after collecting EMG signals from all six people. 1000 EMG samples were collected for every 8 sensors and each movement. The EMG signal's amplitude lies in between 1-100 mV, making it a considerably weak signal. The signal lies in the frequency range from 0-500 Hz and most dominant in between 50-150 Hz. Later, the EMG signal was converted into a CSV file format for feature extraction and feeding to the Machine Learning Algorithm for classification and recognition.

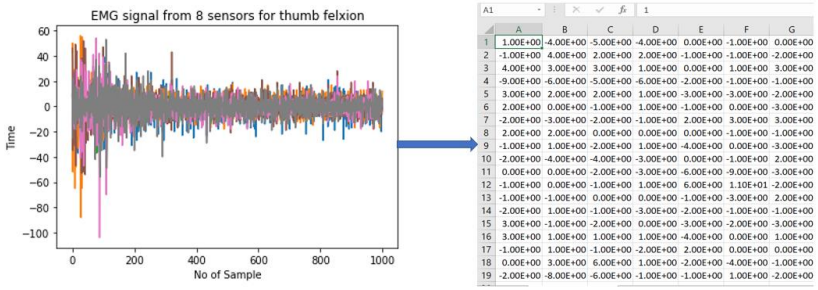


Figure 3.3: Dataset Format

Chapter 4

Data Processing and Feature Extraction

In this chapter, the processing of data has been discussed. As part of pre-processing, the features have been extracted, and hence it was used to classify the EMG data from different hand movements. The features can be 2 types:

- Frequency Domain Features [mention why you didn't use FD]
- Time Domain Features

The common frequency domain features are Autoregressive Coefficients, Frequency Median, Modified Median Frequency, Modified Frequency Mean, Wavelet Transform, Fast Fourier Transform, Power Spectrum, and so on.

The most used time-domain features are Mean Absolute Value (MAV), Standard Deviation (SD), Variance (VA), Slope Sign Integral (SSI), Root Mean Square (RMS), and many more. In this study, we have considered the time domain features to extract. Four time-domain features have been extracted here. To work with this huge dataset and without GPU, we have taken the windowed average of the collected 1000 EMG data points from the 8 sensors. The window we took was 20 per sensor.

4.2 Feature Extraction

Feature extraction is the method of taking the features from the EMG signal. Mean Absolute Value (MAV), Root Means Square (RMS), Variance (VA), and Simple Square Integral (SSI) has been analyzed for each movement. They are called time-domain (TD) features, as the EMG signal is represented in time. The value of this feature extraction will be the input for the classification model.

4.2.1 Mean Absolute Value

MAV is defined as the average of absolute of the EMG signal. The MAV is the computer calculated equivalent of the average rectified value (ARV). The MAV is known as a time domain variable because it is measured as a function of time. It represents the area under the EMG signal once it has been rectified, meaning that all of the negative voltage values have been made positive. The MAV is used as a measure of the amplitude of the EMG signal. It is calculated as follows:

$$MAV = \frac{1}{N} \sum_{k=1}^{N} |X_k| \quad \text{..... (4.1)}$$

Where N is the total length of the Signal and X_k represents the EMG Signal.

4.2.2 Root Mean Square

RMS defines the square root of the average power of the EMG signal. It is represented as amplitude modulated Gaussian random process whose RMS is related to the constant force and non-fatiguing contraction. Its value has been used to quantify the electric signal because it reflects the physiological activity in the motor unit during contraction. It is expressed as follows:

$$\text{RMS} = \sqrt{\frac{1}{N}} \sum_{k=1}^{N} X_k^2 \quad \text{.....(4.2)}$$

4.2.3 Variance

EMG Signal Variance is the measure of the power density of the signal. The value of EMG variance can be zero because EMG signals are handed based on a white Gaussian Noise. It can be calculated as:

$$\text{VAR} = \frac{1}{N} \sum_{k=1}^{N} (X_k - \bar{X})^2 \quad \text{..... (4.3)}$$

4.2.4 Simple Square Integral

SSI gives a measure of the energy of the EMG signal. It is similar to ZC and another method to represent the frequency information of EMG signal. It is defined as follows:

$$\text{SSI} = \sum_{k=1}^{N} \left(\left|X_k^2\right|\right) \quad \text{.....(4.4)}$$

We used a sliding window of step 5s for each of the 8 sensors. Therefore, for 1000 samples, we got 200 data points for each hand movement, which are then fed into four different algorithms for classification.

Chapter 5

Result Analysis

The sample raw EMG signal from 8 sensors has been shown in Fig.3.2. The figure illustrates that the amplitude is different for each sensor. Similarly, amplitude and shape are different for different movements and subjects. In this study, the raw EMG signal has been used for feature extraction. The small interference and Bluetooth of MYO provide a good quality of the signal, so the feature has been extracted from the raw EMG signal. The time-domain features extracted here are MAV, RMS, VA, and SSI. We used a sliding window of step 5s for each of the 8 sensors. Therefore, for 1000 samples, we got 200 data points for each hand movement, which are then fed into four different algorithms for classification.

5.1 Experimental Analysis and Observations

The whole process was an offline experiment. The EMG data was collected from MYO armbands. The data were recorded for each movement. Then the EMG data were processed, features were extracted, and finally the data were fed to different Machine Learning Algorithms. The purpose of this work was to work with the EMG signal and Machine Learning Algorithms.

The EMG data collection was performed using software written in Python. Then the feature extraction and preprocessing were performed using Python. Later, different machine learning python libraries, such as Pandas, NumPy, Sklearn, TensorFlow, were used for implementing the ML part.

The data was collected in a duration of 5s for every 8 sensors for five different hand movements. Total 33 files were created for each hand movement. Later these files were converted to a .csv format and stacked all together to make the dataset a single one. It contained 32 features and five classes (5 hand movements). The classifier model used 80% of the data as the training set and 20% of the data as the test set. The EMG data from the 8 sensors and for the 5 movements were very random. A clear of the raw EMG signal is given below:

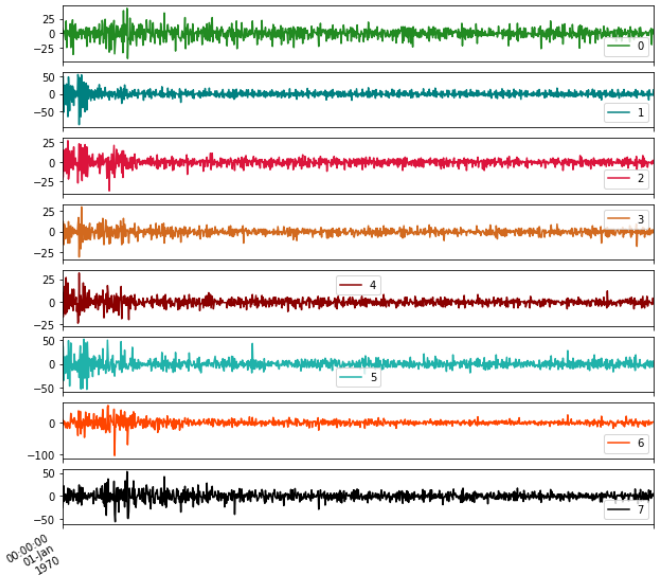


Figure 5.1: Raw EMG signal for TF from 8 sensors

5.2 Observations without pre-processing

5.2.1 Observation 1

Initially, the experiment was stated from using raw EMG data. The ML algorithms were trained using these raw EMG signal. And the average results were following: mention about the algorithms described in … 2.11….

Algorithm Name	Accuracy
Naïve Bayes	79.04%
Random Forest Classifier	71.13%
Support Vector Machine	73.02%
K Nearest Neighbor	58.88%

Table 5.1: Accuracy before pre-processing

It was clearly noticed that the accuracy is very poor when we use the data as the input before pre-processing. It took a lot of time to run KNN as we were using CPU. The accuracy is also very poor for this algorithm as compared to other three algorithms.

5.2.2 Observation 2

After getting poor accuracy, the correlation operation was implemented to get better accuracy. Correlation is a measure of how strongly one variable depends on another. If the features are correlated, they can be dropped. After performing correlation coefficients, the obtained average accuracy were as follows:

Algorithm Name	Accuracy
Naïve Bayes	79.875%
Random Forest Classifier	78.38%
Support Vector Machine	81.88%
K Nearest Neighbor	74.65%

Table 5.2: Accuracy after correlation coefficients

We also plotted the EMG data into a correlation coefficient plot to see if the features are related to each other. After plotting the correlation graph, it was the following figures:

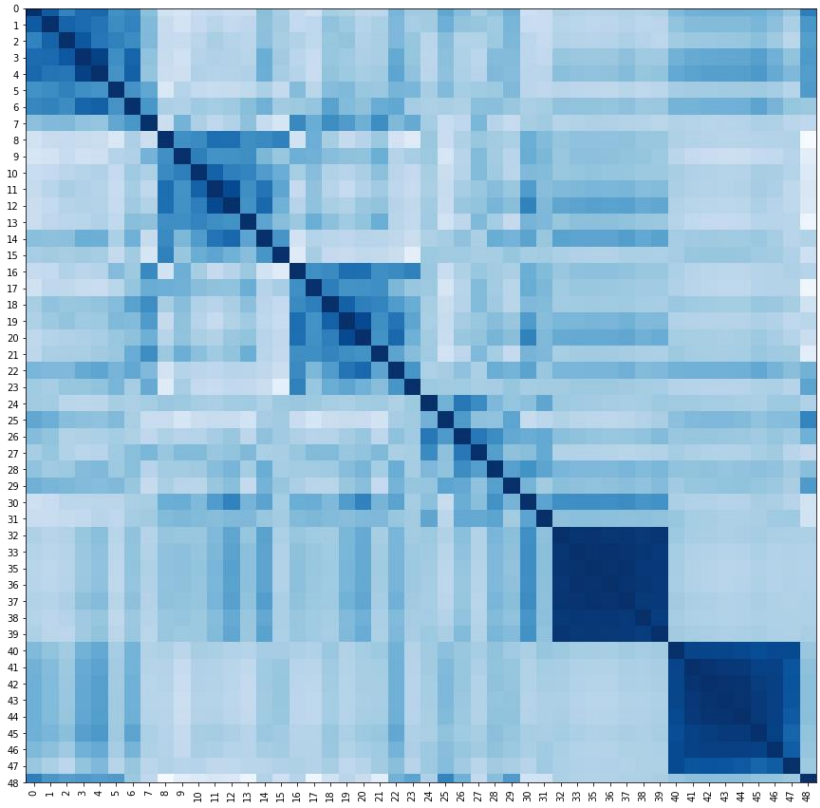


Figure 5.2: Correlation before pre-processing

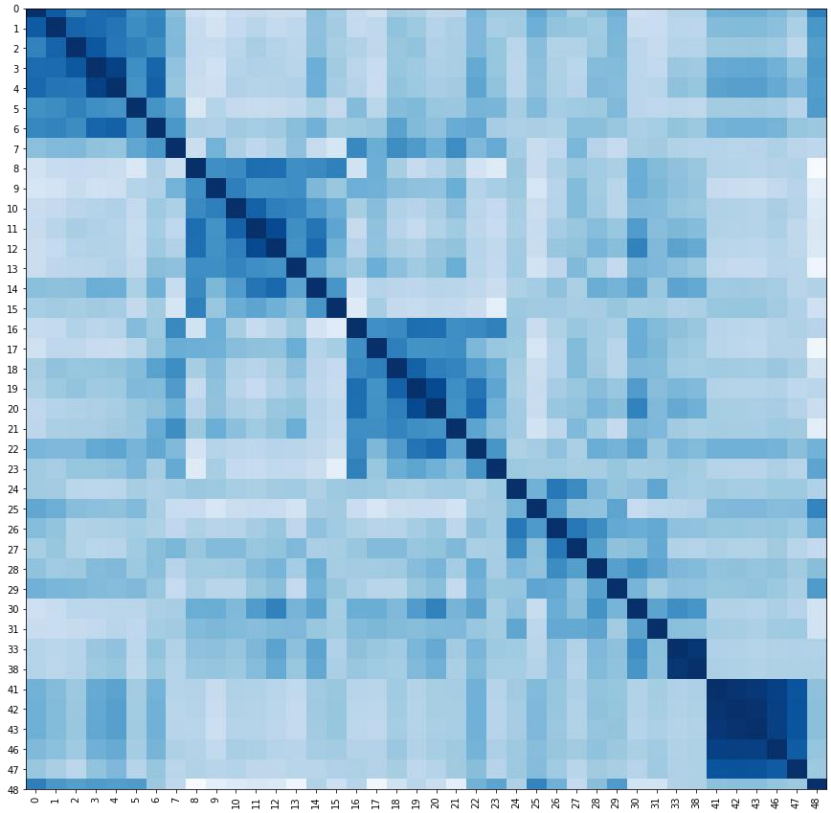


Figure 5.3: Correlation after deleting the correlated features

5.3 Observation with Feature Extraction

After all these observations and poor accuracy, the feature extraction part was done to achieve better accuracy. The feature extraction was an important part of data pre-processing. We performed four time-domain feature extraction. The features were Mean Absolute Value, Root Mean Square, Variance, and the Single Sign Integral. After extracting the features, the data were used to train the ML algorithms. After passing these processed data, the following average accuracy was achieved. In Table 3, the comparison of average accuracy for each feature and classification technique has been shown.

Algorithm	**Features**			
	MAV	***RMS***	***VA***	***SSI***
SVM	98%	97%	76%	94%
NB	89%	94%	92%	88%
KNN	95%	94%	89%	87%
RF	96%	98%	94%	97%

Table 5.3: Comparison of the Percentage of the Average Accuracy of 5 Hand Movements

From Table 1, the classifier output shows that SVM and RF have achieved the highest accuracy 98% whereas the NB and KNN have a lower performance. They have an accuracy of 94% and 95%. These are the average accuracy for all movements. A bar chart of the accuracy has been shown in Fig. 5.4.

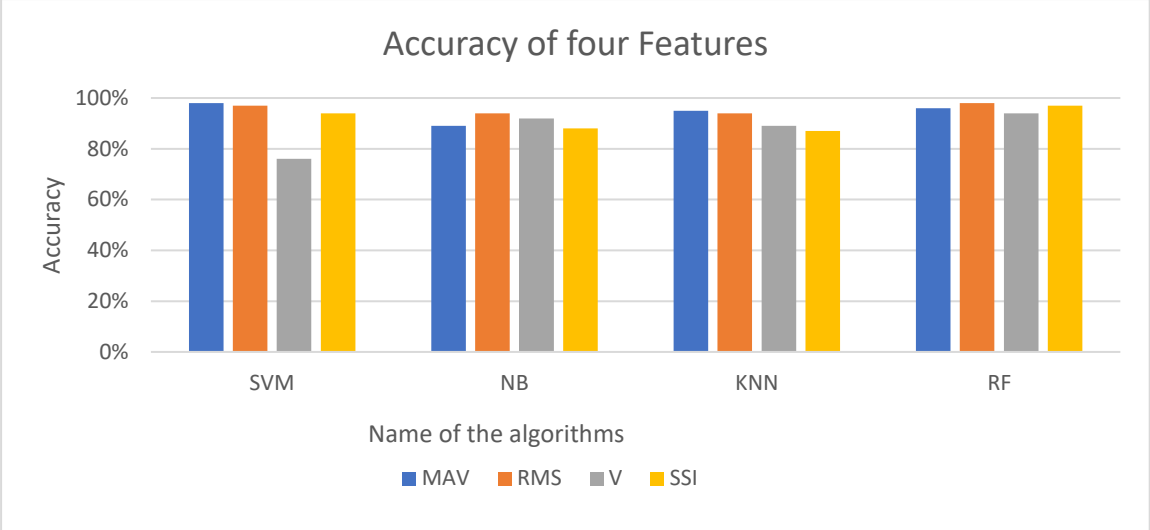


Figure 5.4: Classification Accuracy

Fig. 5.4 illustrates that SVM obtains the highest accuracy of 98% for MAV, NB has the highest accuracy of 97% for RMS, and RF has the accuracy of 98% RMS. However, KNN has poor performance compared to other models. It has an accuracy of 95%. Among all features which have been used as the input for classification, MAV and RMS have provided a better accuracy.

We again plotted the EMG data to see after pre-processing if there are any features that are correlated or not. If we find any features related to each other we can again remove those features to obtain a better accuracy. But after pre-processing the correlation graph was as follows:

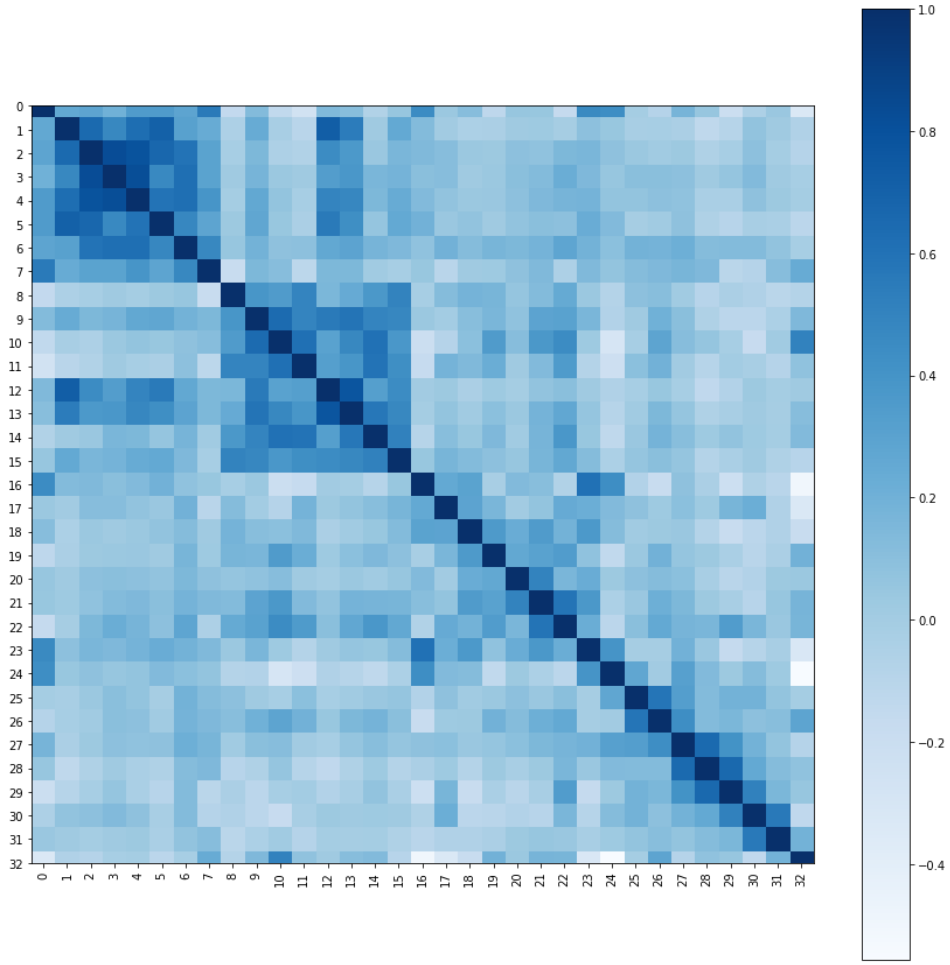


Figure 5.5: Correlation graph after pre-processing

From Figure 5.5, it is clearly observed that there is no features which are corelated with each other, Each feature was individual. This figure shows that the data have been processed perfectly.

F-score is the measure of test accuracy. The F1-score for each movements and each features have been shown on Fig. 5.6 to Fig. 5.9.

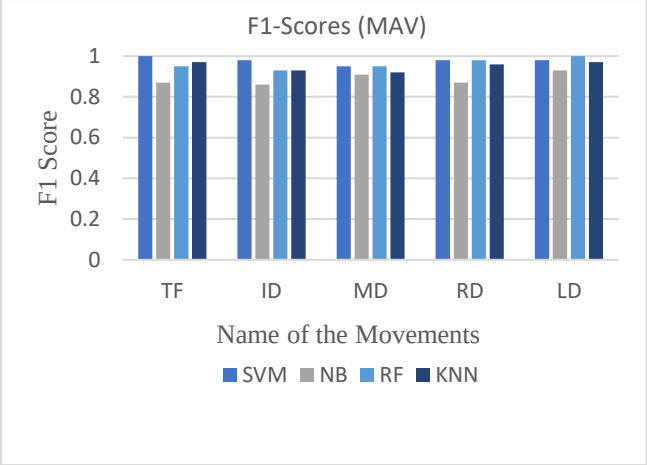


Figure 5.6. F1-Score for MAV

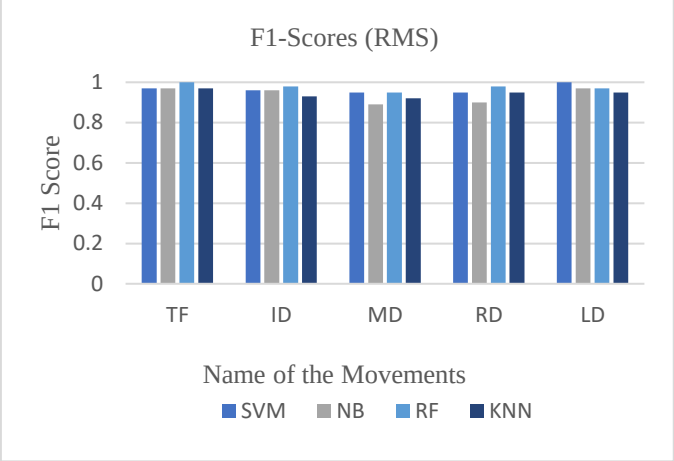


Figure 5.7: F1-Score for RMS

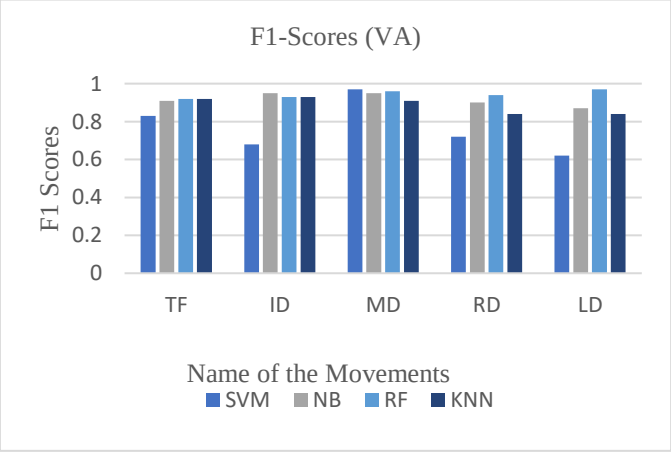


Figure 5.8: Fig. 9. F1-Score for VA

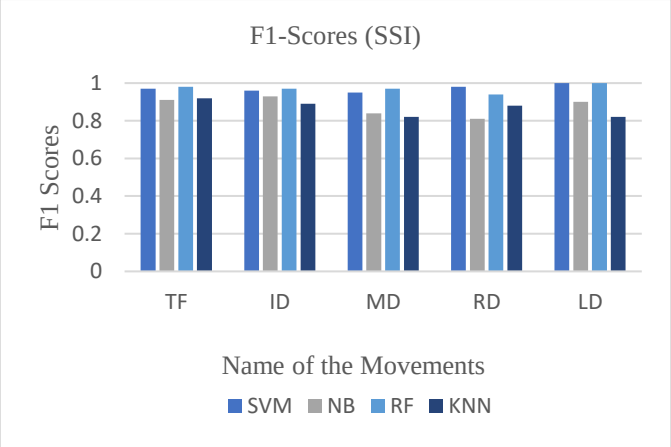


Figure 5.9: Fig. 9. F1-Score for SSI

From Fig. 5.6 to Fig 5. 9, it is noticed that TF, RD, and LD have a better F1 score for MAV and RMS in comparison to other two movements. Among all the movements, TF has the best F1 scores compared to the rest of the algorithms. Overall, referring to the analysis and results, these classification model can be the best approach as the input for the prosthetic hand.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Artificial Intelligence, Machine Learning Algorithms and its classification were addressed in chapter 2.

Machine learning is one modern innovation that has helped people enhance not only many industrial and professional processes but also advances everyday living. In the recent world, Machine Learning is a hot topic. It can be used in the techniques and tools that can help in the diagnosis of diseases. It is used for the analysis of the clinical parameters and their combination for the prognosis example prediction of disease progression for the extraction of medical knowledge for the outcome research, for therapy planning and patient monitoring, and many more. This research work aims to implement different Machine Learning methods to classify five different hand movements from the EMG signal, which has been collected from the human body. The most important part was the data processing. The approach of using raw EMG signal to classify the movements was not very satisfying. We obtained poor accuracy. Some more approaches were implemented to get a better result such as correlation and feature extraction. The most fruitful approach was the feature extraction, where we extracted four time-domain features, i.e. Mean Absolute Value, Root Mean Square, Variance, and Single Sign Integral. The average accuracy has been shown for five movements. Among the four features, MAV and RMS outperform VA, SSI. The SVM and RF perform the best in terms of accuracy. The other two algorithms' performance is also noticeable but a bit lower than these two; however, it can be a better candidate for the input of the prosthetic hand control.

6.2 Publication

Paper Titled ,"Classification and Feature Extraction of Different Hand Movements from the EMG Signal using Machine Leaning based Algorithms", jointly prepared by Bipasha Kundu and Dr. D Subbaram Naidu & presented by Bipasha Kundu at 3rd International Conference on Electrical, Communication and Computer Engineering (ICECCE), Kuala Lumpur, Malaysia, June 12,2021.

6.3 Future Work

In this work four ML algorithms, i.e. SVM, RF, NB, and KNN, have been used. For future work, this can be done using different types of Neural Networks. If the EMG signal is not used and the pictures of each movement from different people could be collected to create a dataset, then the identification and classification of the hand movements could be performed using Convolution Neural Network, which uses the image for classification and prediction. There is another type of Neural Network which works with time-series data, i.e. Recurrent Neural Network. RNN can be also implemented to see if it provides better accuracy. In that case, a GPU will be needed to train these algorithms as Neural Network works very slow on CPU. These Machine Learning algorithms also took a long time to run because of the use of the CPU instead of GPU. Most importantly, as this study was performed offline, for future work, this classification model and result can be used for real-time movement classification. Finally, the data were collected from six healthy people. If data could be collected data from more participants, better accuracy and results could be achieved for this research work.

References

[1] Diane W. Braza, Jennifer N. Yacub Martin, “Upper Limb Amputations,” Essentials of Physical Medicine and Rehabilitation (Fourth Edition), Elsevier, 2020, page 651-657

[2] Marco E. Benalcázar, Andrés G. Jaramillo, Jonathan A. Zea, and Andrés Páez, “Hand Gesture Recognition Using Machine Learning and the Myo Armband”, 25th European Signal Processing Conference (EUSIPCO), 2017.

[3] Tony Chau “Machine Learning for Gesture Recognition with Electromyography,” , Norwegian University of Science and Technology,Dept. of CS, June 2017,unpublished.

[4] Nikitha Anil, Sreeletha S.H “EMG based Gesture Recognition using Machine Learning,” International Conference on Intelligent Computing and Control Systems (ICICCS 2018).

[5] SACHIN NEGI, YATINDRA KUMAR, V. M. MISHRA,“Feature Extraction and Classification for EMG Signals Using Linear Discriminant Analysis ,” IEEE Trans, 2016.

[6] Shunzhan He, Chenguang Yang, Min Wang, Long Cheng, Zedong Hu, “Hand Gesture Recognition using MYO Armband”, IEEE Trans, 2017.

[7] https://github.com/NiklasRosenstein/myo-python/tree/master/myo

[8] Md. R. Ahsan, Muhammad I. Ibrahimy, Othman O. Khalifa,”The Use of Artificial Neural Network in the Classification of EMG Signals”,IEEE Trans, 2011

[9] https://en.wikipedia.org/

[10] “https://www.datarobot.com/wiki/supervised-machine-learning/”

[11] https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[12] https://www.upgrad.com/blog/types-of-regression-models-in-machine-learning/

[13] https://github.com/NiklasRosenstein/myo-python/tree/master/myo

[14] https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47”

[15] https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d

[16] https://blog.floydhub.com/naive-bayes-for-machine-learning/#:~:text=And%20the%20Machine%20Learning%20%E2%80%93%20The,presence%20of%20any%20other%20feature.

[17] https://builtin.com/data-science/recurrent-neural-networks-and-lstm

[18] https://wiki.pathmind.com/neural-network

[19] Jimson Ngeo , Tomoya Tamei , Tomohiro Shibata ,"Estimation of continuous multi-DOF finger joint kinematics from surface EMG using a multi-output Gaussian Process", July 2014

[20] "https://en.wikipedia.org/wiki/Linear_regression"

[21] https://www.geeksforgeeks.org/decision-tree-introduction-example/

[22] https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[23] https://medium.com/@EskoKilpi/neural-networks-as-the-architecture-of-human-work-3f9d20f019a3

Appendices

A.1: Code for Connecting Myo Armband

```
from __future__ import print_function
from collections import deque
from threading import Lock, Thread
import matplotlib
matplotlib.use("TkAgg")
import matplotlib.pyplot as plt


import myo
import time
import sys
import psutil
import os
import numpy as np
import pandas as pd


number_of_samples = 1000
data_array=[]


Sensor1 = np.zeros((1,number_of_samples))
Sensor2 = np.zeros((1,number_of_samples))
Sensor3 = np.zeros((1,number_of_samples))
Sensor4 = np.zeros((1,number_of_samples))
Sensor5 = np.zeros((1,number_of_samples))
Sensor6 = np.zeros((1,number_of_samples))
Sensor7 = np.zeros((1,number_of_samples))
Sensor8 = np.zeros((1,number_of_samples))


index_open_training_set = np.zeros((8,number_of_samples))
middle_open_training_set = np.zeros((8,number_of_samples))
thumb_open_training_set = np.zeros((8,number_of_samples))
ring_open_training_set = np.zeros((8,number_of_samples))
little_open_training_set = np.zeros((8,number_of_samples))
all_fingers_closed_training_set = np.zeros((8,number_of_samples))
resttraining_set = np.zeros((8,number_of_samples))

thumb_open_label = 0
index_open_label = 1
middle_open_label = 2
ring_open_label = 3
little_open_label = 4


def find_one_hot(labels,classes):
 # = tf.constant(C)
 output = tf.one_hot(labels,classes,axis=0)
 sess = tf.Session()
 out = sess.run(output)
```

```
 sess.close
 return out

import psutil
name = input("Enter name of Subject")
```

 # This process checks if Myo Connect.exe is running

```
def check_if_process_running():
    try:
        for proc in psutil.process_iter():
            if proc.name()=='Myo Connect.exe':
                return True
        return False


    except (psutil.NoSuchProcess,psutil.AccessDenied, psutil.ZombieProcess):
        print (PROCNAME, " not running")
```

 # If the process Myo Connect.exe is not running then we restart that process

```
def restart_process():
    PROCNAME = "Myo Connect.exe"

    for proc in psutil.process_iter():
        # check whether the process name matches
        if proc.name() == PROCNAME:
            proc.kill()
            # Wait a second
            time.sleep(1)


    while(check_if_process_running()==False):
        path = 'C:\\Program Files (x86)\\Thalmic Labs\\Myo Connect\\Myo Connect.exe'
        os.startfile(path)
        time.sleep(1)


    print("Process started")
    return True
```

 # This is Myo-python SDK's listener that listens to EMG signal

```
class Listener(myo.DeviceListener):
    def __init__(self, n):
        self.n = n
        self.lock = Lock()
        self.emg_data_queue = deque(maxlen=n)


    def on_connected(self, event):
        print("Hello, Myo!")
        self.started = time.time()
        event.device.stream_emg(True)
```

```
    def get_emg_data(self):
        with self.lock:
            print("H")


    def on_emg(self, event):
        with self.lock:
            self.emg_data_queue.append((event.emg))

            if len(list(self.emg_data_queue))>=number_of_samples:
                data_array.append(list(self.emg_data_queue))
                self.emg_data_queue.clear()
                return False


def main():

    index_open_training_set = np.zeros((8,number_of_samples))
    middle_open_training_set = np.zeros((8,number_of_samples))
    thumb_open_training_set = np.zeros((8,number_of_samples))
    ring_open_training_set = np.zeros((8,number_of_samples))
    little_open_training_set = np.zeros((8,number_of_samples))
    rest_training_set=np.zeros((8,number_of_samples))
    wave_out__training_set=np.zeros((8,number_of_samples))
    all_fingers_closed_training_set=np.zeros((8,number_of_samples))


    verification_set = np.zeros((8,number_of_samples))
    training_set = np.zeros((8,number_of_samples))
    # This function kills Myo Connect.exe and restarts it to make sure it is running
    # Because sometimes the application does not run even when Myo Connect process is
running
    while(restart_process()!=True):
        pass
    # Wait for 3 seconds until Myo Connect.exe starts
    time.sleep(3)


    # Initialize the SDK of Myo Armband
    myo.init('C:\\Program Files\\Python37\\myo64.dll')
    hub = myo.Hub()
    listener = Listener(number_of_samples)
    legend = ['Sensor 1','Sensor 2','Sensor 3','Sensor 4','Sensor 5','Sensor 6','Sensor 7','Sensor
8']


    ##################TRAINING DATA FOR THUMB FINGER OPEN ########

    while True:
        try:
            hub = myo.Hub()
            listener = Listener(number_of_samples)
            input("Open THUMB ")
            hub.run(listener.on_event,20000)
            thumb_open_training_set = np.array((data_array[0]))
            print(thumb_open_training_set,thumb_open_training_set.shape)
            data_array.clear()
```

```
            np.savetxt("thumb_open_training_set.csv", thumb_open_training_set, delimiter=",")
            plt.plot(thumb_open_training_set)
            plt.show()
            break
        except:
            while(restart_process()!=True):
                pass
            # Wait for 3 seconds until Myo Connect.exe starts
            time.sleep(3)
            np.savetxt("thumb_open_training_set.csv", thumb_open_training_set, delimiter=",")


    ################## TRAINING DATA FOR INDEX FINGER OPEN ########

    while True:
        try:
            input("Open index finger")
            start_time = time.time()
            hub = myo.Hub()
            listener = Listener(number_of_samples)

            hub.run(listener.on_event,20000)
            index_open_training_set = np.array((data_array[0]))
            print(index_open_training_set,index_open_training_set.shape)
            data_array.clear()
            np.savetxt("index_open_training_set.csv", index_open_training_set, delimiter=",")
            plt.plot(index_open_training_set)
            plt.show()
                    break

        except:

            while(restart_process()!=True):
                pass
     # Wait for 3 seconds until Myo Connect.exe starts
            time.sleep(3)


    ################## TRAINING DATA FOR MIDDLE FINGER OPEN
################

    while True:
        try:
            input("Open MIDDLE finger")
            hub = myo.Hub()
            listener = Listener(number_of_samples)
            hub.run(listener.on_event,20000)
            middle_open_training_set = np.array((data_array[0]))
            print(middle_open_training_set,middle_open_training_set.shape)
            data_array.clear()
            np.savetxt("middle_open_training_set.csv", middle_open_training_set, delimiter=",")
            plt.plot(middle_open_training_set)
            plt.show()
            break
        except:
            while(restart_process()!=True):
                pass
```

```
            # Wait for 3 seconds until Myo Connect.exe starts
            time.sleep(3)
            np.savetxt("middle_open_training_set.csv", middle_open_training_set, delimiter=",")


    ################## TRAINING DATA FOR RING FINGER OPEN ##########

    while True:
        try:
            input("Open Ring finger")
            hub = myo.Hub()
            listener = Listener(number_of_samples)
            hub.run(listener.on_event,20000)
            ring_open_training_set = np.array((data_array[0]))
            print(ring_open_training_set,ring_open_training_set.shape)
            data_array.clear()
            np.savetxt("ring_open_training_set.csv", ring_open_training_set, delimiter=",")
            plt.plot(ring_open_training_set)
            plt.show()
            break
        except:
            while(restart_process()!=True):
                pass
            # Wait for 3 seconds until Myo Connect.exe starts
            time.sleep(3)
            np.savetxt("ring_open_training_set.csv", ring_open_training_set, delimiter=",")



    ################### TRAINING DATA FOR LITTLE FINGER
####################

    while True:
        try:
            input("Open Little Finger")
            hub = myo.Hub()
            listener = Listener(number_of_samples)
            hub.run(listener.on_event,20000)
            little_open_training_set = np.array((data_array[0]))
            print(little_open_training_set,little_open_training_set.shape)
            data_array.clear()
            np.savetxt("little_open_training_set.csv", little_open_training_set, delimiter=",")
            plt.plot(little_open_training_set)
            plt.show()
            break
        except:
            while(restart_process()!=True):
                pass
            # Wait for 3 seconds until Myo Connect.exe starts
            time.sleep(3)
            np.savetxt("little_open_training_set.csv", little_open_training_set, delimiter=",")

if __name__ == '__main__':

    main()
```

A 2: Code for Pre-Processing, Training and Test

```
import time
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import os
import math
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.svm import SVC
from sklearn.metrics import classification_report,confusion_matrix
col_names= ['emg1','emg2','emg3','emg4','emg5','emg6','emg7','emg8','Class']
data11=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bipasha\\thumb_open_training_set.csv', header=None)
data21=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bipasha\\index_open_training_set.csv', header=None)
data31=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bipasha\\middle_open_training_set.csv', header=None)
data41=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bipasha\\ring_open_training_set.csv', header=None)
data51=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bipasha\\little_open_training_set.csv', header=None)
data61=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bipasha\\all_fingers_closed_training_set.csv', header=None)
data12=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Rupak\\thumb_open_training_set.csv', header=None)
data22=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Rupak\\index_open_training_set.csv', header=None)
data32=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Rupak\\middle_open_training_set.csv', header=None)
data42=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Rupak\\ring_open_training_set.csv', header=None)
```

```
data52=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Rupak\\little_open_training_set.csv', header=None)
data62=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Rupak\\all_fingers_closed_training_set.csv', header=None)
#data=pd.read_csv('C:\\Users\\Bipasha\\Final Data Copy\\Previous data from 4
person\\thumb_open_test_set.csv',header=None)
data13=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bohota\\thumb_open_training_set.csv',header=None)
data23=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bohota\\index_open_training_set.csv',header=None)
data33=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bohota\\middle_open_training_set.csv',header=None)
data43=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bohota\\ring_open_training_set.csv',header=None)
data53=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Bohota\\little_open_training_set.csv',header=None)
data14=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Newaz\\thumb_open_training_set.csv',header=None)
data24=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Newaz\\index_open_training_set.csv',header=None)
data34=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Newaz\\middle_open_training_set.csv',header=None)
data44=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Newaz\\ring_open_training_set.csv',header=None)
data54=pd.read_csv('C:\\Users\\Bipasha\\New data
collected\\Newaz\\little_open_training_set.csv',header=None)


print(data12.head())
data11.shape

d0=np.zeros((1000,1))
type(d0)

d1=np.ones((1000,1))
d2=2*np.ones((1000,1))
d3=3*np.ones((1000,1))
```

```
d4=4*np.ones((1000,1))
#d5=np.ones((1000,1))
type(d1)

# This will contain the column names of the Data frame
columns = list(data21.columns)[1:]
print(data21.columns)
df=data21.shape
print('There are {} rows and {} columns.\n'.format(
      df[0], df[1]))

#Extracting each signal from the dataset
colors=["forestgreen","teal","crimson","chocolate","darkred","lightseagreen","orangered"
,"black"]
time_thumb_open=data11.iloc[:,0:8]
time_thumb_open.index=pd.to_datetime(time_thumb_open.index)
time_thumb_open.iloc[:1000,:].plot(subplots=True,figsize=(10,10),color=colors);
plt.plot(data11)
#plt.subplot(246)
plt.title("EMG signal from 8 sensors for thumb felxion", loc='center')
plt.xlabel("No of Sample")
plt.ylabel("Time")
data21=data31.values.tolist()
```

#Feature Extraction MAV,RMS,VAR,SSC

#Extracting each signal from the dataset

```
colors=["forestgreen","teal","crimson","chocolate","darkred","lightseagreen","orangered","bl
ack"]
time_thumb_open=data52.iloc[:,0:8]
time_thumb_open.index=pd.to_datetime(time_thumb_open.index)
time_thumb_open.iloc[:1000,:].plot(subplots=True,figsize=(10,10),color=colors);

df1 = np.concatenate([data11,data12,data13,data14],axis=1)
df2 = np.concatenate([data21,data22,data23,data24],axis=1)
df3 = np.concatenate([data31,data32,data33,data34],axis=1)
df4 = np.concatenate([data41,data42,data43,data44],axis=1)
df5 = np.concatenate([data51,data52,data53,data54],axis=1)
#df6 = np.concatenate([data61,data62,d6],axis=1)

df5.shape
df1 = np.absolute(df1)
```

```
df2 =np.absolute(df2)
df3 = np.absolute(df3)
df4 = np.absolute(df4)
df5 =np.absolute(df5)
print(type(df1))
df1=pd.DataFrame(data=df5)
```

#Extracting each signal from the dataset

```
    colors=["forestgreen","teal","crimson","chocolate","darkred","lightseagreen","orangered"
    ,"black"]
    time_thumb_open=data41.iloc[:,0:8]
    time_thumb_open.index=pd.to_datetime(time_thumb_open.index)
    time_thumb_open.iloc[:1000,:].plot(subplots=True,figsize=(10,10),color=colors);
```

```
dff=np.concatenate([df1,df2,df3,df4,df5],axis=0)

div = 5
number_of_samples=1000
averages = int(number_of_samples/div)#200 ta average value
thumb_open_averages = np.zeros((int(averages),32))
index_open_averages = np.zeros((int(averages),32))
middle_open_averages = np.zeros((int(averages),32))
ring_open_averages = np.zeros((int(averages),32))
little_open_averages = np.zeros((int(averages),32))

averages = int(number_of_samples/div)
averages
dff=pd.DataFrame(data=dff)
#variance
div = 5
number_of_samples=1000
N = int(number_of_samples/div)
thumb_open_VAR = np.zeros((int(N),32))
index_open_VAR = np.zeros((int(N),32))
middle_open_VAR = np.zeros((int(N),32))
ring_open_VAR = np.zeros((int(N),32))
little_open_VAR = np.zeros((int(N),32))
```

```
for i in range(1,N+1):
   thumb_open_VAR[i-1,:] = np.sum(df1[(i-1)*div:i*div,:]**2,axis=0) /(N-1)
   index_open_VAR[i-1,:] = np.sum(df2[(i-1)*div:i*div,:]**2,axis=0)/(N-1)
   middle_open_VAR[i-1,:] = np.sum(df3[(i-1)*div:i**2*div,:]**2,axis=0)/(N-1)
   ring_open_VAR[i-1,:] = np.sum(df4[(i-1)*div:i*div,:]**2,axis=0)/(N-1)
   little_open_VAR[i-1,:] = np.sum(df5[(i-1)*div:i*div,:]**2,axis=0)/(N-1)

for i in range(1,N+1):
   thumb_open_VAR[i-1,:] = np.sum(df1[(i-1)*div:i*div,:]**2/(N-1),axis=0)
   index_open_VAR[i-1,:] = np.sum(df2[(i-1)*div:i*div,:]**2,axis=0)/(N-1)
   middle_open_VAR[i-1,:] = np.sum(df3[(i-1)*div:i**2*div,:]**2/(N-1) ,axis=0)
   ring_open_VAR[i-1,:] = np.sum(df4[(i-1)*div:i*div,:]**2/(N-1),axis=0)
   little_open_VAR[i-1,:] = np.sum(df5[(i-1)*div:i*div,:]**2/(N-1),axis=0)

d0=np.zeros((200,1))
d1=np.ones((200,1))
d2=2*np.ones((200,1))
d3=3*np.ones((200,1))
d4=4*np.ones((200,1))

thumb_open_VAR=np.concatenate([thumb_open_VAR,d0],axis=1)
thumb_open_VAR.shape
index_open_VAR=np.concatenate([index_open_VAR,d1],axis=1)
middle_open_VAR=np.concatenate([middle_open_VAR,d2],axis=1)
ring_open_VAR=np.concatenate([ring_open_VAR,d3],axis=1)
little_open_VAR=np.concatenate([little_open_VAR,d4],axis=1)

conc_array1 = np.concatenate([thumb_open_VAR,index_open_VAR,middle_open_VAR,
ring_open_VAR,little_open_VAR],axis=0)
print(conc_array1.shape)

np.savetxt("conc_array1.csv", conc_array1, delimiter=",")
df_VAR=pd.read_csv("conc_array1.csv",header=None)
```

Convertig them into numpy array

```
df1=df1.to_numpy()
df2=df2.to_numpy()
df3=df3.to_numpy()
df4=df4.to_numpy()
df5=df5.to_numpy()
```

Calculating the mean values of all finger data set

```
for i in range(1,averages+1):
   thumb_open_averages[i-1,:] = np.mean(df1[(i-1)*div:i*div,:],axis=0)
   index_open_averages[i-1,:] = np.mean(df2[(i-1)*div:i*div,:],axis=0)

   middle_open_averages[i-1,:] = np.mean(df3[(i-1)*div:i*div,:],axis=0)
   ring_open_averages[i-1,:] = np.mean(df4[(i-1)*div:i*div,:],axis=0)
   little_open_averages[i-1,:] = np.mean(df5[(i-1)*div:i*div,:],axis=0)
#return all
thumb_open_averages=np.concatenate([thumb_open_averages,d0],axis=1)
index_open_averages=np.concatenate([index_open_averages,d1],axis=1)
```

```
middle_open_averages=np.concatenate([middle_open_averages,d2],axis=1)
ring_open_averages=np.concatenate([ring_open_averages,d3],axis=1)
little_open_averages=np.concatenate([little_open_averages,d4],axis=1)


conc_array =
np.concatenate([thumb_open_averages,index_open_averages,middle_open_averages,
ring_open_averages,little_open_averages],axis=0)
print(conc_array.shape)

np.savetxt("conc_array.csv", conc_array, delimiter=",")
#df_MAV=pd.read_csv("conc_array.csv",header=None)
df_MAV=pd.read_csv("conc_array.csv",header=None)
```

Calculating Root Mean Square

```
 thumb_open_RMS = np.zeros((int(N),32))
index_open_RMS = np.zeros((int(N),32))
middle_open_RMS = np.zeros((int(N),32))
ring_open_RMS = np.zeros((int(N),32))
little_open_RMS = np.zeros((int(N),32))

for i in range(1,N+1):
    thumb_open_RMS[i-1,:] = np.sqrt(np.sum(df1[(i-1)*div:i*div,:]**2/N,axis=0))
    index_open_RMS[i-1,:] = np.sqrt(np.sum(df2[(i-1)*div:i*div,:]**2/N,axis=0))

    middle_open_RMS[i-1,:] = np.sqrt(np.sum(df3[(i-1)*div:i*div,:]**2/N,axis=0))
    ring_open_RMS[i-1,:] = np.sqrt(np.sum(df4[(i-1)*div:i*div,:]**2/N,axis=0))
    little_open_RMS[i-1,:] = np.sqrt(np.sum(df5[(i-1)*div:i*div,:]**2/N,axis=0))
#return all

thumb_open_RMS=np.concatenate([thumb_open_RMS,d0],axis=1)
index_open_RMS=np.concatenate([index_open_RMS,d1],axis=1)
middle_open_RMS=np.concatenate([middle_open_RMS,d2],axis=1)
ring_open_RMS=np.concatenate([ring_open_RMS,d3],axis=1)
little_open_RMS=np.concatenate([little_open_RMS,d4],axis=1)

conc_array3 = np.concatenate([thumb_open_RMS,index_open_RMS,middle_open_RMS,
ring_open_RMS,little_open_RMS],axis=0)
print(conc_array3.shape)

np.savetxt("conc_array3.csv", conc_array3, delimiter=",")
df_RMS=pd.read_csv("conc_array3.csv",header=None)
```

Calculation SSI (Slope sign Change)

```
thumb_open_SSI = np.zeros((int(N),32))
index_open_SSI = np.zeros((int(N),32))
middle_open_SSI = np.zeros((int(N),32))
ring_open_SSI = np.zeros((int(N),32))
little_open_SSI = np.zeros((int(N),32))

for i in range(1,N+1):
```

```
    thumb_open_SSI[i-1,:] = np.sum(df1[(i-1)*div:i*div,:]**2,axis=0)
    index_open_SSI[i-1,:] = np.sum(df2[(i-1)*div:i*div,:]**2,axis=0)

    middle_open_SSI[i-1,:] = np.sum(df3[(i-1)*div:i*div,:]**2,axis=0)
    ring_open_SSI[i-1,:] = np.sum(df4[(i-1)*div:i*div,:]**2,axis=0)
    little_open_SSI[i-1,:] = np.sum(df5[(i-1)*div:i*div,:]**2,axis=0)
#return all

thumb_open_SSI=np.concatenate([thumb_open_SSI,d0],axis=1)
index_open_SSI=np.concatenate([index_open_SSI,d1],axis=1)
middle_open_SSI=np.concatenate([middle_open_SSI,d2],axis=1)
ring_open_SSI=np.concatenate([ring_open_SSI,d3],axis=1)
little_open_SSI=np.concatenate([little_open_SSI,d4],axis=1)

conc_array4 = np.concatenate([thumb_open_SSI,index_open_SSI,middle_open_SSI,
ring_open_SSI,little_open_SSI],axis=0)
print(conc_array4.shape)

np.savetxt("conc_array4.csv", conc_array4, delimiter=",")
df_SSI=pd.read_csv("conc_array4.csv",header=None)
```

Pre_processing the Data

#Replacing NaN values with '0'

```
df=df.replace('0.0',np.NaN)
df = df.dropna(axis=1)
#df[abs(df)< .1]= NaN
#isnan(dff).sum(0)
```

#Dropping Missing Values

```
print(df.shape)
df.dropna(how = "any", inplace = True)
print(df.shape)

print(df.shape)
df.drop_duplicates(keep = "first",inplace = True)
print(df.shape)
```

Plotting the correation Graph

```
def plot_corr(df, size =15, colorMap= 'Blues'):
    corr = df.corr()
    fig, ax = plt.subplots(figsize=(size, size))
    im = ax.imshow(corr, cmap = colorMap)

    plt.xticks(range(len(corr.columns)), corr.columns, rotation = 90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    fig.colorbar(im,   orientation='vertical')
    plt.show()
plot_corr(df_MAV)
plot_corr(df_VAR)
plot_corr(df_RMS)
```

```
plot_corr(df_SSI)
```

#Splitting Training and Test Data

```
X=df_VAR.drop([32],axis=1)
y=df_VAR[32]

from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.10,random_state=42)
```

#Using the linear kernel for SVM

```
from sklearn import svm
from sklearn import metrics
```

#We now Normalize the data to make sure different features take on similar range of values, For this purpose we use StandarScaler().

```
from sklearn import svm
#X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=38)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_transformed= scaler.transform(X_train)
clf = svm.SVC(C=1, gamma = .1).fit(X_train_transformed, y_train)
X_test_transformed = scaler.transform(X_test)
clf.score(X_test_transformed, y_test)*100


clf = svm.SVC(C=10, gamma = .01).fit(X_train_transformed, y_train)
X_test_transfored = scaler.transform(X_test_transformed)
clf.score(X_test_transformed, y_test)*100
```

#Naive Bayes with Scaling(Scaling not Necessray for Naive Bayes)

```
naive=GaussianNB().fit(X_train_transformed,y_train)
naive
y_pred=naive.predict(X_test_transformed)
print("Accuracy:",accuracy_score(y_test,y_pred)*100)

print(classification_report(y_test,y_pred))
```

RBF SVM with GridSearchCV

 # Choosing the best hyperparameters for RBf kernel

```
param_grid = {'C': [0.1, 1, 10, 100,1000,10000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001,
.00001]}     # select a set of parameters
clf_grid = GridSearchCV(SVC(kernel = 'rbf'), param_grid)   # pass the set of parameters to
the classifier
```

```
# select the best parameters
clf_grid.fit(X_train, y_train)
print("Best Parameters:\n", clf_grid.best_params_)
svclassifier =SVC(kernel = 'rbf', C = 10, gamma =.01)
# C = 10.0, gamma = 0.1 - selected using GridSearchCV
#svclassifier.fit(X_train,y_train)
clf=svclassifier.fit(X_train_transformed,y_train)
y_pred = clf.predict(X_test_transformed)
accuracy_score(y_pred, y_test)*100

print(classification_report(y_test,y_pred))
```

Random forest classifier with raw data

```
from sklearn.ensemble import RandomForestClassifier
maxdepth_space = [6,8,10,12,14,16,18,20]
n_estimator_space =[10,11,12,13,14,15,16,17,18,19,10]
dict_params = {'criterion':['gini','entropy'], 'max_depth':maxdepth_space,
"n_estimators":n_estimator_space}

model = RandomForestClassifier()
grid_cv = GridSearchCV(model, param_grid = dict_params, cv = 10)
grid_cv.fit(X_train,y_train)
test_acc = grid_cv.score(X_test, y_test)
y_pred = grid_cv.predict(X_test)
conf_mat = confusion_matrix(y_test, y_pred)
clr = classification_report(y_test, y_pred)
print(grid_cv.best_params_)

print(classification_report(y_test,y_pred))
```

KNN classifier with Scaling

```
from sklearn.neighbors import KNeighborsClassifier
kneigh=KNeighborsClassifier(n_neighbors=39)
k_model=kneigh.fit(X_train,y_train)
k_model
y_pred=k_model.predict(X_test)
print("Accuracy:",accuracy_score(y_test,y_pred)*100)
```

 # getting the best parameters for KNN classifier_tunning

```
params={"n_neighbors": np.arange(1,40)}
knn=KNeighborsClassifier()
knn_cv=GridSearchCV(knn,params,cv=5)
knn_cv.fit(X_train,y_train)

knn_cv.best_params_
print(knn_cv.best_score_)
print(classification_report(y_test,y_pred))
```

Running KNN fro Neighbor(1-40) with Scaled Data

```
neighbors = np.arange(1, 10)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
```

 # Loop over different values of k

```
for i, k in enumerate(neighbors):
    # Setup a k-NN Classifier with k neighbors: knn
    knn_cv = KNeighborsClassifier(n_neighbors=k)
 knn_cv.fit(X_train_transformed,y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn_cv.score(X_train_transformed,y_train)

    #Compute accuracy on the testing set
    test_accuracy[i] = knn_cv.score(X_test_transformed,y_test)

# plot Generation

plt.title('k-NN: Varying Number of Neighbors')
plt.plot(neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
print(knn_cv.score(X_test_transformed,y_test))
print(knn_cv.score(X_train_transformed,y_train))

print(np.mean(train_accuracy))
print(np.mean(test_accuracy))
```